PARALLEL-PARALLEL SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates memory-bandwidth-bound inference of large autoregressive target models using a small, fast drafter model. Target-latent-conditioned parallel drafters, such as DFlash, currently offer the best accuracy-latency tradeoff, leading to the highest reported speedups. However, latent conditioning still requires executing drafting and verification sequentially, placing drafting latency on the critical path. This limits achievable speedups and prevents effective use of emerging hardware that can support concurrent drafting and verification. We introduce **pa**rallel-**pa**rallel **spec**ulative (PaPaSpec) decoding, which runs a multi-layer latent-conditioned parallel drafter concurrently with verification, hiding drafting latency entirely. The key enabler is a prefix and depth-causal latent conditioning scheme that restricts each drafter layer to only access the target latents computed so far. Hiding drafting behind verification not only removes it from the critical latency path, but also buys the drafter additional time to refine its draft. We exploit this by looping over intermediate layers until new conditioning information arrives, which increases the acceptance length while keeping the drafter size constant. We evaluate PaPaSpec in single-batch serving across three different target models on NVIDIA H100 GPUs. Across seven benchmarks covering math, coding, and chat, PaPaSpec accelerates autoregressive generation by 3.26–5.50× on average, a 13–22% improvement over DFlash.

1 INTRODUCTION

Autoregressive (AR) generation (Vaswani et al., 2017; Brown et al., 2020) remains the dominant generation paradigm in large language models (LLMs). While AR training can be efficiently parallelized via teacher forcing (Williams & Zipser, 1989), AR inference is inherently sequential, generating tokens one at a time. Predicting a single token requires loading the model parameters and the key-value (KV) cache from off-chip memory to the on-chip compute units. In low-concurrency serving, where only a few requests are processed at once, AR generation is hence often memory-bandwidth-bound, leaving spare compute unutilized. As a viable solution, speculative de-

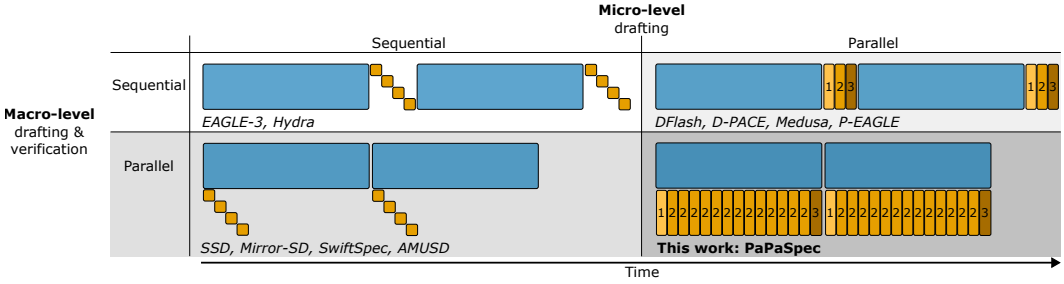


Figure 1: Verification (■) and drafting (■) timeline for different approaches. Performing verification and drafting macro-sequentially puts drafting into the critical latency path, whether the drafter itself is micro-level sequential or parallel. Instead, performing verification and drafting in macro-level parallel hides the drafting latency, but so far only supports micro-level sequential drafters. This work presents PaPaSpec, which performs macro-level parallel verification with micro-level parallel drafting, hiding the drafting latency while further exploiting spare compute via looping.

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.

coding (Leviathan et al., 2023; Chen et al., 2023) accelerates inference by leveraging this spare compute to verify, in parallel, a sequence of possible next tokens proposed by a cheaper drafter model.

While effective, speculative decoding faces two distinct forms of sequentiality that negatively impact the achievable decoding speedup (see Figure 1). First, drafting itself is often sequential, generating candidate tokens autoregressively one token after another. We refer to this as *micro-level* sequentiality. Second, drafting and verification are typically executed synchronously, such that verification must complete before the next drafting can begin. We refer to this as *macro-level* sequentiality. Eliminating both forms of sequentiality is essential for maximizing speculative decoding speedups.

Recent progress in resolving micro-level sequentiality is largely driven by conditioning drafters on the target model's latent states. Because these latents encode information about future tokens (Samragh et al., 2025), they allow the drafter to predict multiple tokens in parallel. While target-model conditioning improves the acceptance rates of sequential drafters (Li et al., 2024b;a; 2025; Ankner et al., 2024), it acts as a critical enabler for parallel drafters (Stern et al., 2018; Cai et al., 2024), with conditioning depth ranging from a single hidden state to multiple target layers. DFlash (Chen et al., 2026) pushes this paradigm even further: it conditions on the accumulated *history* of projected multi-layer target latents rather than just the last-token hidden states. While such parallel drafting, optionally enhanced with improved training objectives (Wu et al., 2026) or light-weight AR correction (Cheng et al., 2026; Huang et al., 2026), achieves state-of-the-art (SoTA) speedups, the depth of the drafter, and hence its acceptance length, is limited due to being on the critical latency path.

Concurrently, another line of work has sought to eliminate macro-level sequentiality by executing drafting and verification asynchronously in parallel (McDanel, 2025; Kumar et al., 2026). Indeed, dedicated accelerators for speculative drafting are increasingly available (Bartlett et al., 2026; Bhendawade et al., 2025), such as the recently announced Groq 3 LPX on the NVIDIA Vera Rubin platform (Weidman et al., 2026). These devices allow launching drafting before the target model's forward pass completes, thereby hiding drafting latency behind verification. However, current macro-parallel approaches rely on standard, decoupled sequential drafters rather than tightly conditioned parallel ones: multi-layer latent conditioning as used by SoTA micro-parallel drafters requires target latents, which are not yet available when drafting occurs concurrently with verification.

In this work, we propose ***pa**rallel-**pa**rallel **spec**ulative (PaPaSpec) decoding*, which removes both levels of sequentiality: a tightly conditioned parallel drafter (micro-level) operates in parallel with the target model (macro-level), hiding drafting behind verification (see Figure 1). The key innovation is a new latent conditioning that attends only to already computed target model layers and prefix, making conditioning available incrementally as the target verifies. Hiding drafting behind verification goes beyond merely removing the drafting latency: it buys the drafter time, which we can spend on additional compute by increasing the drafter's depth. We show that looping internal layers in the drafter can effectively increase acceptance length while keeping the number of drafter parameters constant. This work makes the following contributions:

- We propose prefix and depth-causal conditioning for parallel drafters, where each drafter's layer only accesses the target model latents that have already been computed during verification. This effectively removes the requirement in prior target-conditioned drafters (e.g., DFlash) that the target model's forward pass complete before drafting can begin.
- We introduce looping to parallel drafting, where intermediate layers are looped to increase acceptance length while keeping the number of parameters constant: a single drafter can thus adapt to varying acceptance length, compute, and latency demands at inference time.
- Combining our prefix and depth-causal conditioning, looping, and macro-parallelism yields PaPaSpec, a parallel drafter that drafts in parallel with the target model verification, combining micro- and macro-level parallelism in speculative decoding.
- We evaluate PaPaSpec on seven benchmarks covering math reasoning, coding, and chat when serving single-batch requests on H100 GPUs. For Qwen3-8B in greedy decoding, PaPaSpec achieves an average speedup of 5.50× over AR generation, which is a 21% improvement over DFlash and 8% better than our improved D-PACE baseline. PaPaSpec's advantage transfers to Qwen3-4B (21% gain over DFlash) and Gemma-4-26B-A4B-it (17%), and holds under temperature-1 sampling across all three models (13–22%).

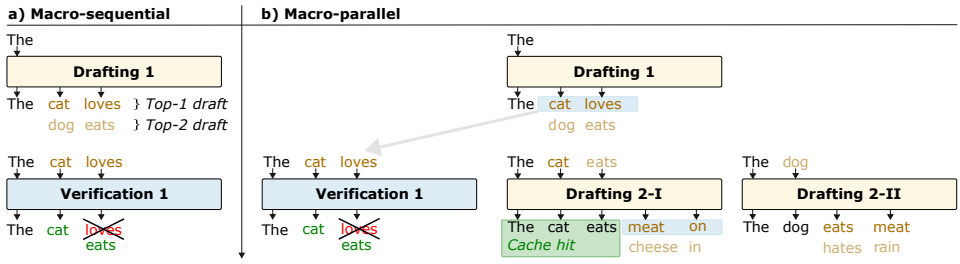


Figure 2: **(a) Macro-sequential decoding.** Drafting is followed by verification, where the top-1 draft ("cat loves") is verified. The target model accepts the first token ("cat"), while rejecting the second ("loves") and resampling a bonus token ("eats"). **(b) Macro-parallel decoding.** While the target model verifies the top-1 draft, the drafter speculates on two possible next prefixes ("The cat eats" and "The dog"), and generates drafts concurrently. If the verified sequence matches one of the speculated prefixes, the corresponding draft is already available and can be verified next.

2 PRELIMINARIES

2.1 SPECULATIVE DECODING

Speculative decoding accelerates AR inference of a large target model p by using a faster draft model q. Given a prefix $x_{1:n} := (x_1, x_2, ..., x_n)$ of length n, where each token x_i is in the vocabulary $\mathcal{V}$, the drafter proposes a continuation of γ tokens $\hat{x}_{n+1:n+\gamma} \sim q(\cdot|x_{1:n})$. The target model then verifies the entire draft in one pass. In a memory-bandwidth-bound regime, the verification/target latency t_T is largely insensitive to the drafting length, i.e., is assumed to equal the target model's per-token AR latency. The drafted tokens are verified from left to right, either via exact-match acceptance in the greedy setting (Stern et al., 2018), or via acceptance sampling in the stochastic setting (Leviathan et al., 2023). When a token is rejected, a bonus token is sampled from the normalized residual distribution between the target and drafter. This approach is lossless with respect to the target distribution. We denote the average acceptance length (including the bonus token) as $\tau \in [1, \gamma + 1]$.

Macro-sequential verification and drafting As shown in Figure 2a, the standard speculative decoding framework issues drafting and verification sequentially. The expected speedup is $\eta = \frac{\tau \cdot t_T}{t_D + t_T}$, where t_T is the target model latency and t_D the drafting latency. Since the drafter is on the critical latency path, naïvely aiming for acceptance length improvements by scaling up the drafter, which may increase drafting latency, does not necessarily translate into a wall-clock speedup improvement.

Macro-parallel verification and drafting with SSD One viable approach is to hide the drafting latency behind the verification, as proposed by Kumar et al. (2026) in speculative speculative decoding (SSD). Figure 2b illustrates this concept. Here, the next drafting starts while the previous draft is still being verified. The main challenge is that the result of the current verification (the accepted length and the bonus token) is not yet known. To resolve this, the next drafting round is based on a *speculated prefix*, which consists of the first $\hat{\tau} - 1$ tokens of the verification sequence, followed by a bonus token that differs from the corresponding token in the submitted verification sequence (e.g., the top-2 token from the previous draft). Here, $\hat{\tau} \in [1, \gamma + 1]$. To increase the chance of speculating the correct prefix, a *cache* containing multiple prefixes is used to draft in parallel. If the verification outcome matches one of the speculated prefixes (cache hit), the corresponding draft can be verified without any additional latency. If none of the speculated prefixes match or the entire draft was accepted, an additional sequential draft has to be performed, which again falls on the critical latency path. Hence, even though predominantly hidden, drafting latency still plays a role in SSD.

2.2 MICRO-PARALLEL DRAFTING WITH DFLASH

In contrast to SSD, which parallelizes on the system level, micro-parallel drafting parallelizes the drafting step itself, generating all draft tokens in a single forward pass. DFlash (Chen et al., 2026) realizes this with a single-step masked diffusion model (Austin et al., 2021a; Arriola et al., 2025),

denoising all γ positions simultaneously and yielding a drafting latency independent of γ. Key to its accuracy is conditioning the drafter on the target model's latents from the previous verification step.

Recall that during verification, the target model computes $p(x_{n+1} \mid x_{1:n}) = \big(\mathrm{LM}_{\mathrm{head}} \circ \mathcal{T}^{(L_T)} \circ \cdots \circ \mathcal{T}^{(1)} \circ \mathrm{E}\big)(x_{1:n})$, where $\circ$ denotes function composition, $\mathrm{E}(\cdot)$ the input embedding, $\mathcal{T}^{(l)}(\cdot)$ the l-th Transformer decoder layer, and $\mathrm{LM}_{\mathrm{head}}(\cdot)$ the projection (including softmax) from the model's final latents to the simplex over the vocabulary. In addition, the target also provides the latents

$$h^{(l)}_{1:n} = \big(\mathcal{T}^{(l)} \circ \cdots \circ \mathcal{T}^{(1)} \circ \mathrm{E}\big)\big(x_{1:n}\big) \in \mathbb{R}^{n \times d} \text{ for all } l \in [1, L_T].$$

As $\mathcal{T}^{(l)}$ uses causal attention, all latents from the previous verifications (e.g., $h^{(l)}_{1:n-1}$) can be cached. DFlash conditions on a subset $\{l_1, \cdots l_m\}$ of the target latents via concatenation and projection:

$$c_{1:n} = W\big[h^{(l_1)}_{1:n} \| \cdots \| h^{(l_m)}_{1:n}\big] = \textstyle\sum_{i=1}^{m} W^{(i)} h^{(l_i)}_{1:n}, \tag{1}$$

where $c_{1:n} \in \mathbb{R}^{n \times d}$ and $W^{(i)} \in \mathbb{R}^{d \times d}$. These projected latents act as a conditioning signal via injection into the drafter's attention. Additionally, the input to the drafter is a bonus token x_{n+1} and γ mask tokens $z = \mathrm{E}\big(x_{n+1}, [M], \ldots, [M]\big) \in \mathbb{R}^{(\gamma+1) \times d}$. The drafter thus computes the marginals

$$q\big(x_{n+2:n+\gamma+1} \mid x_{1:n+1}\big) = \big(\mathrm{LM}_{\mathrm{head}} \circ \mathcal{D}^{(L_D)}(\,\cdot\,; c_{1:n}) \circ \cdots \circ \mathcal{D}^{(1)}(\,\cdot\,; c_{1:n})\big)\big(z\big),$$

where L_D is the number of layers and $\mathcal{D}^{(l)}(\cdot; c_{1:n})$ is a Transformer layer with bidirectional attention in the positions of z and causal attention to RMSNorm-scaled $c_{1:n}$. E and $\mathrm{LM}_{\mathrm{head}}$ are borrowed from the target (i.e., parameter-sharing). Two elements prohibit straightforward macro-parallel decoding in DFlash. Firstly, the drafter needs the bonus token x_{n+1}, which again requires speculation as in SSD. Second, and more detrimentally, the conditioning vector $c_{1:n}$ depends on all the chosen verifier hidden states, some of which might only be available towards the end of verification.

3 Parallel-parallel speculative decoding (PaPaSpec)

This section presents PaPaSpec, a tightly target-latent-conditioned micro-parallel drafter that operates in a macro-parallel speculative decoding framework, illustrated in Figure 3. We first introduce our prefix and depth-causal conditioning that removes the dependency on fully completed target verification, enabling concurrent drafting and verification. We then show how the gained spare latency can be converted into improved draft quality through layer-wise looping. Finally, we describe the integration into SSD and present a novel dynamic budget-allocation strategy.

3.1 Prefix and depth-causal conditioning

DFlash natively requires the availability of all selected target hidden layers. Consequently, the conditioning signal $c_{1:n}$ becomes available only after the final (selected) target layer has been computed. Hence, naïvely applying macro-parallel SSD with such conditioning would hardly provide any wall-clock time improvement, as drafting would still depend on, and wait for, the completion of verification. To remove this bottleneck, we break the depth-bidirectional dependency between target and drafter by instead introducing a *prefix and depth-causal conditioning mechanism*.

Architecture Instead of using the same conditioning for every drafter layer, we follow Zhang et al. (2026a) and modify the DFlash architecture with layer-specific conditioning. Our modification differs in that we retain the projection matrices $W^{(i)}$. More decisively, we treat previously verified tokens and currently verified tokens differently. For tokens belonging to completed verification rounds (i.e., the prefix), all verifier layers are available and full depth-bidirectional conditioning can be applied. For tokens in the block currently undergoing verification, only a subset of verifier layers has been computed; therefore, conditioning can only access latents completed so far. For a token index t, assuming tokens up to position n have already been verified and speculating that we will have $\hat{\tau}$ tokens accepted, we define the prefix and depth-causal conditioning for drafter layer l as

$$c^{(l)}_t = \begin{cases} \sum_{i=1}^{m} \alpha_{i,l}\, W^{(i)} h^{(l_i)}_t, & 1 \le t \le n, \qquad \text{(prefix)} \\ \sum_{i=1}^{l} \beta_{i,l}\, W^{(i)} h^{(l_i)}_t, & n < t \le n + \hat{\tau} - 1 \qquad \text{(tokens under verification).} \end{cases}$$

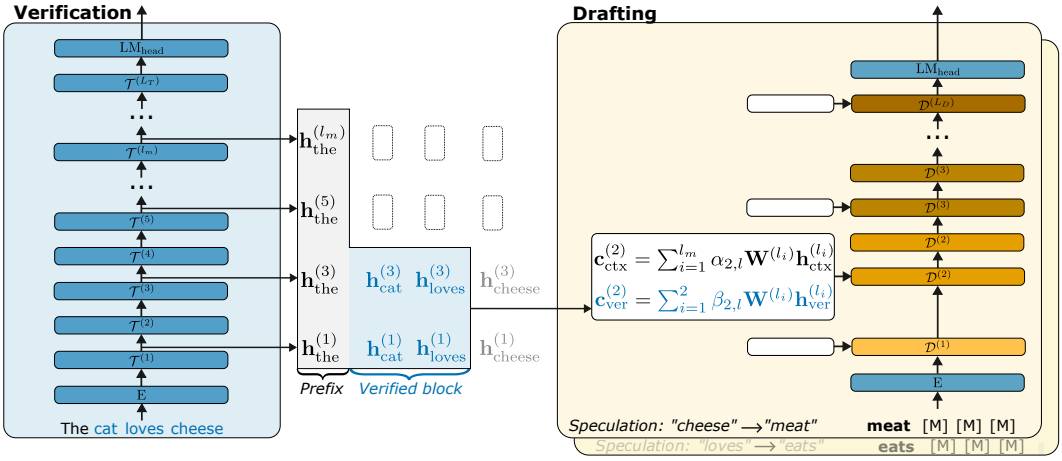


Figure 3: PaPaSpec architecture. The target model verifies the previous draft (e.g., "cat loves cheese"), while the drafter concurrently generates drafts based on speculated acceptance lengths and bonus tokens (e.g., "meat" or "eats"). The drafter is conditioned on layer-specific latents following a prefix and depth-causal scheme in which each drafter layer only accesses target latents computed so far. Depending on verification progress, latents from all layers (h_{the}) or only the so far completed layers (h_{cat} and h_{loves}) enter the conditioning. Finally, the drafter can leverage the spare time before the next latents arrive to increase drafting quality by looping over intermediate layers.

Training procedure By design of SSD, the drafter operates under varying acceptance lengths $\hat{\tau} \in [1, \gamma + 1]$ of the previous block. Since we only condition depth-causally to unverified token positions, there is a shift in the conditioning distribution depending on the value of $\hat{\tau}$. We model the acceptance length as a geometric distribution under the assumption of i.i.d. acceptance probability ρ per position (Leviathan et al., 2023). To cover all $\hat{\tau}$ during training, we thus sample $\hat{\tau}$ from $\Pr(\hat{\tau}{=}k) = (1-\rho)\,\rho^{k-1}$ and truncate it to $[1, \gamma]$ to match the bonus token availability at inference.

3.2 Looped parallel drafting

Because PaPaSpec removes drafting from the critical latency path, drafting can consume up to the same latency budget as verification without increasing the wall-clock time. A natural way to improve the draft quality of masked diffusion models would be to increase the number of denoising steps. However, multi-step denoising requires access to the full verifier conditioning signals and therefore would reintroduce the synchronization constraints between drafting and verification. Instead, we increase the effective depth of the drafter by reusing intermediate layers, a technique known as looping, which has been shown to achieve depth-scaling behavior similar to multi-step diffusion (Lee et al., 2026; Carballo-Castro et al., 2026). We repeatedly apply the intermediate layers of the drafter, e.g., the layers $\mathcal{D}^{(2)}$, $\mathcal{D}^{(3)}$, and $\mathcal{D}^{(4)}$ in a 5-layer drafter. Crucially, rather than looping blocks of layers, we loop each layer individually. This allows every drafter layer to immediately consume newly available conditioning signals as verification progresses (see Figure 4).

Looping allows the drafting compute to be scaled dynamically, for example based on the available drafter compute budget, and does not increase the number of parameters. In particular, the drafter can still be executed without any loops, which is important for latency-sensitive drafting in case of an SSD cache miss. Unlike the original SSD framework, which instantiates both an accurate, high-latency primary drafter and a cheap, low-latency backup drafter, PaPaSpec covers both modes with a *single model*. Accordingly, we train the drafter to operate reliably at both compute budgets: for every training batch, we sample the number of loops as $r = 1 + b(R-1)$, with $b \sim \text{Bernoulli}(0.5)$, where R is the maximum number of loops.

3.3 Integration into speculative speculative decoding (SSD)

Figure 3 illustrates the adoption of PaPaSpec into SSD. During verification, the drafter concurrently generates drafts for the next SSD iteration. These drafts are conditioned on all of the prefix latents

and incrementally available depth-causal latents of the currently verified block, as well as the speculated bonus tokens sampled from the previous drafting round. In the event of an SSD cache miss, drafting falls back to macro-sequential execution, disabling looping to minimize drafting latency.

Confidence-based budget allocation In SSD, the drafter generates candidate continuations from a set of speculated pairs (acceptance length, bonus token), with the bonus token derived from the drafter. A key design choice is how many bonus tokens f_k should be considered per speculated accept length k, given a fixed drafting budget $B = \sum_{k=1}^{\gamma} f_k$. Prior work used fixed allocation strategies, such as uniform or geometric distributions (Kumar et al., 2026). However, the confidence of a draft has been shown to correlate with the acceptance length (Li et al., 2024a; Wu et al., 2026). We therefore propose a confidence-based budget allocation scheme that dynamically distributes the available drafting budget according to the drafter's predicted token distribution q_i. More precisely, from the set $\mathcal{C} = \{k, b\}$ of all possible initial positions k and bonus tokens b, we select the subset

$$\mathcal{C}^\star = \text{top-B}_{(k,b)\in\mathcal{C}}\ \sigma(k,b) \quad \text{with} \quad \sigma(k,b) = \log q_k(b) + \textstyle\sum_{i=1}^{k-1} \log q_i(\hat{x}_i).$$

I.e., we select the branches with maximal log-probability $\sigma(k, b)$ under the drafter's marginals, a surrogate for the verifier. The induced drafting budget per position k is therefore $f_k = |b : (k, b) \in \mathcal{C}^\star|$. Using this method, when the drafter is uncertain about its early predictions, more budget will be allocated towards the start of the block. Likewise, if the drafter is highly confident in early positions but is uncertain around a mid point, the drafting budget will be concentrated around that point.

Multi-device deployment By design, PaPaSpec executes the verifier and drafter concurrently on separate compute devices. In our current implementation, we deploy the target model and drafter on two H100 GPUs. The primary additional cost introduced by PaPaSpec is the transfer of verifier latents to the drafter. For a drafted block of length γ currently under verification, the target model transmits the selected latents from the m layers with dimensionality d. This amounts to a total communication volume of $m \cdot \gamma \cdot d$ values. For a practical configuration with $m = 5$, $\gamma = 16$, $d = 4096$, and BF16 precision, the communication overhead is only 655 kB per verification. Compared to the cost of executing the verifier, this memory-bandwidth overhead is negligible. On NVIDIA GPUs, the execution of the drafter and verifier layers does not have to be synchronized, e.g., by using the NCCL LL (Zhang et al., 2026b). In our implementation, both models communicate over non-blocking point-to-point NCCL and are executed through CUDA Graph replays using the HuggingFace Transformers (Wolf et al., 2019) backend with `torch.compile` (see Appendix C.1). While we evaluate PaPaSpec using a two-GPU setup, this deployment model naturally aligns with emerging accelerator architectures that provide dedicated hardware for draft generation alongside target verification, such as the d-Matrix Corsair (Bartlett et al., 2026), the NPU in Apple's M-series (Bhendawade et al., 2025), or the recently announced Groq 3 LPX on the NVIDIA Vera Rubin platform (Weidman et al., 2026). Architectures with unified memory, such as Apple's M-series, would further eliminate the communication overhead entirely.

4 Experiments

4.1 Setup

Target models and evaluations We evaluate on Qwen3-4B and Qwen3-8B (Yang et al., 2025), as well as Gemma-4-26B-A4B-it (Gemma Team, 2026). Our ablations focus on Qwen3-8B. We evaluate on a diverse range of benchmarks across three domains. For *math reasoning*, we use GSM8K (Mirzadeh et al., 2025), MATH (Lightman et al., 2024), and AIME26 through MathArena (Dekoninck et al., 2026); for *coding*, we use HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021b), and LiveCodeBench (v1-v6) (Jain et al., 2025); and for *chat*, we use MT-bench (Zheng et al., 2023). For each method, we report the speedup with respect to standard AR decoding. We benchmark the macro-sequential drafters DFlash and D-PACE on a single H100, while PaPaSpec uses two (one each for the target model and drafter). All methods run on the Hugging Face Transformers backend with `torch.compile` and CUDA Graph replays, which is competitive to vLLM.

Drafter details Following DFlash, we train our PaPaSpec with a block size of $\gamma = 16$ and $R = 3$ loops using a mixture of 800k samples from NVIDIA's Nemotron Post-Training Dataset

Table 1: Speedup over AR generation for different speculative decoding methods on H100 GPUs. All approaches are compiled into a single CUDA-graph and operate at block size $\gamma = 16$ with a maximum sequence length of 2048. PaPaSpec uses a dynamic budget of 32 for all target models. † indicates a public checkpoint rather than training reproduction.

Model	**Method**	MATH			CODE			CHAT	*Avg.*
		GSM8K	MATH-500	AIME26	HumanEval	MBPP	LCB	MT-Bench	
Temperature = 0									
Qwen3-8B	DFlash†	4.66	5.65	5.24	4.64	4.29	4.85	2.48	4.55
	D-PACE	5.22	6.41	6.01	5.05	4.74	5.35	2.75	5.08
	PaPaSpec	**5.80**$^{+24\%}$	**6.72**$^{+19\%}$	**6.30**$^{+20\%}$	**5.63**$^{+21\%}$	**5.25**$^{+22\%}$	**5.78**$^{+19\%}$	**2.98**$^{+20\%}$	**5.50**$^{+21\%}$
Qwen3-4B	DFlash†	4.44	5.43	4.93	4.42	4.19	4.35	2.40	4.31
	PaPaSpec	**5.68**$^{+28\%}$	**6.38**$^{+17\%}$	**5.63**$^{+14\%}$	**5.37**$^{+21\%}$	**5.25**$^{+25\%}$	**5.37**$^{+23\%}$	**2.92**$^{+22\%}$	**5.23**$^{+21\%}$
Gemma-4 26B-A4B	DFlash†	3.20	3.68	3.33	3.77	2.70	2.82	1.78	3.04
	PaPaSpec	**3.57**$^{+12\%}$	**4.26**$^{+16\%}$	**4.11**$^{+23\%}$	**4.48**$^{+19\%}$	**3.15**$^{+17\%}$	**3.52**$^{+25\%}$	**1.92**$^{+8\%}$	**3.57**$^{+17\%}$
Temperature = 1									
Qwen3-8B	DFlash†	4.23	4.56	3.13	3.80	3.84	4.46	2.26	3.75
	D-PACE	4.64	4.88	3.52	4.09	4.17	4.85	2.43	4.08
	PaPaSpec	**5.23**$^{+24\%}$	**5.24**$^{+15\%}$	**3.75**$^{+20\%}$	**4.59**$^{+21\%}$	**4.67**$^{+22\%}$	**5.37**$^{+20\%}$	**2.65**$^{+17\%}$	**4.50**$^{+20\%}$
Qwen3-4B	DFlash†	4.09	4.37	3.17	4.01	3.85	3.95	2.24	3.67
	PaPaSpec	**5.24**$^{+28\%}$	**5.15**$^{+18\%}$	**3.86**$^{+22\%}$	**4.84**$^{+21\%}$	**4.75**$^{+23\%}$	**4.94**$^{+25\%}$	**2.67**$^{+19\%}$	**4.49**$^{+22\%}$
Gemma-4 26B-A4B	DFlash†	3.11	3.53	3.00	3.60	2.63	2.59	1.74	2.88
	PaPaSpec	**3.35**$^{+8\%}$	**3.92**$^{+11\%}$	**3.47**$^{+16\%}$	**4.12**$^{+14\%}$	**3.02**$^{+15\%}$	**3.08**$^{+19\%}$	**1.83**$^{+5\%}$	**3.26**$^{+13\%}$

V2 (Nathawani et al., 2025) and CodeAlpaca (Luo et al., 2024). For our conditioning ablations, we used a subset of 100k samples. To obtain drafters tailored to each target model, we generate on-policy ground-truth responses with each target model. We use the D-PACE (Wu et al., 2026) loss and rely, except for depth-causal conditioning and looping, on the exact same architectural configuration as DFlash. Besides testing against publicly released DFlash checkpoints, we train an additional DFlash drafter with the D-PACE loss as a stronger baseline. Appendix B provides more details.

4.2 MAIN RESULTS

We report our main results in Table 1. On Qwen3-8B in greedy generation, PaPaSpec reaches an average speedup of 5.50×, which is 8.3% higher than our strong D-PACE baseline and 21% higher than DFlash. PaPaSpec achieves a throughput of 706 tokens per second with our compiled Hugging-Face Transformers setup, which is competitive with inference-optimized frameworks such as vLLM (see Appendix A.1). The advantage transfers to Qwen3-4B with a 5.23× speedup and 21% gain over DFlash, as well as to Gemma-4-26B-A4B-it, which features an overall lower speedup of 3.57× but still provides a 17% gain over DFlash. When the target models sample with a temperature of 1, the overall speedups decrease, but we maintain our relative advantage over DFlash (13–22%). Figure 4 depicts timing measurements of PaPaSpec for one decoding step with Qwen3-8B. The verification step takes 8.5 ms in total, whereas a single loop drafter decoder layer takes 0.55 ms and a layer looped 3 times takes 1.49 ms. As we can see, the next draft tokens are available *before* the target finishes the verification; hence, the drafting latency is effectively hidden. We report a detailed timing analysis for different drafting and looping budgets in Appendix A.2.

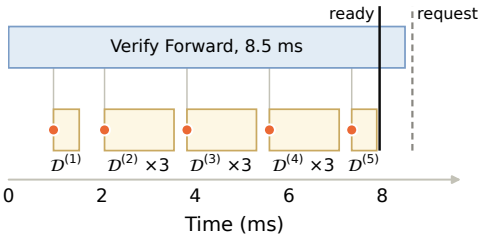


Figure 4: PaPaSpec timeline for one macro-parallel decoding step with Qwen3-8B, three loops, and a cache budget of 32.

4.3 Ablations

We ablate our method on Qwen3-8B, greedy decoding, on MATH500, HumanEval, and MT-Bench.

Target conditioning Figure 5 shows how different conditionings affect the average acceptance length as the speculated accepted position of the previous verification ($\hat{\tau}$) grows. The drafters shown here are all trained on the 100k subset. DFlash/D-PACE's rich conditioning on all verifier hidden states yields the highest acceptance length, but requires the verifier's forward pass to complete first and therefore only supports macro-sequential decoding. Conditioning only on the prefix, i.e., the latents available before the current verification round began (prefix-only), supports macro-parallel decoding but degrades sharply as $\hat{\tau}$ grows. Conditioning only on the layers that have so far been verified (depth-causal, i.e., causally restricting conditioning across both prefix and verification window) increases the acceptance length. PaPaSpec with full hybrid conditioning further reduces the gap to D-PACE, which can be closed with looping, while supporting macro-parallel decoding.

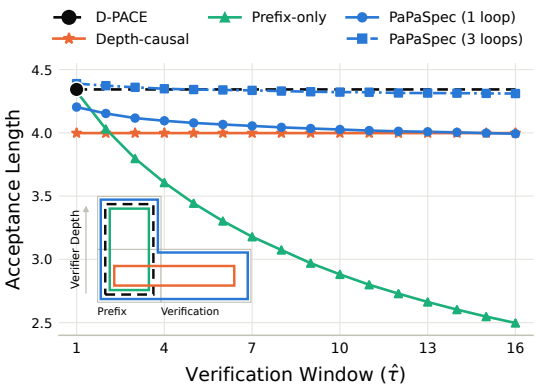


Figure 5: Target conditioning ablation.

Looping. We have already demonstrated looping's positive impact on the average acceptance length. Figure 6a shows that the benefit transfers to actual speedup improvements. We analyze the speedup as a function of the cache budget B. Increasing the budget improves the speedup as it reduces cache misses, but benefits degrade at large budgets that push the drafting into a compute-bound regime. This transition happens earlier for looped drafters due to their increased per-token compute. Despite saturating earlier, the peak speedup of looped drafters still exceeds that of single-loop drafters. Overall, this result shows that *investing compute in looping is more beneficial than naïvely increasing the draft budget with a single loop.*

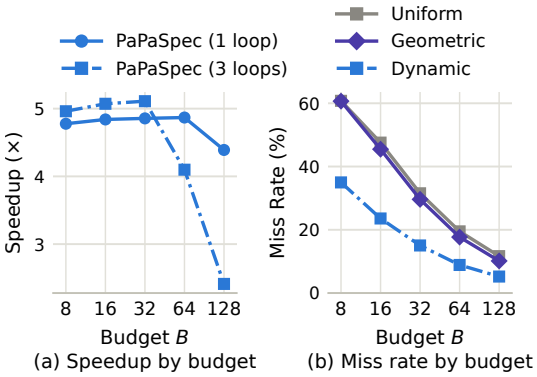


Figure 6: Budget allocation ablation.

SSD cache. Figure 6b further compares our dynamic confidence-based cache allocation with the uniform and geometric baselines. Our method reduces the miss rate for all cache budgets, with particularly pronounced gains in low budget regime: For example, at a cache budget of 8, our method reduces the miss rate from 60.7% to 35.0%. By extending the cache budget from 8 to 32, we can further reduce the miss rate to 15%. More detailed results are in Appendix A.3.

5 Related Work

Speculative drafters Even though early approaches to multi-token prediction (MTP) (Stern et al., 2018; Gloeckle et al., 2024) already use the target's final-layer hidden state, initial speculative decoding frameworks used disconnected target and draft models (Leviathan et al., 2023; Chen et al., 2023). To avoid the need for a separate drafter, self-speculative decoding either adopts sub-networks of the target model (Liu et al., 2024; Zhang et al., 2024), turns the target model itself into a parallel sampler via mask tokens at the input (Monea et al., 2023), or generates candidate continuations in parallel via Jacobi forcing (Fu et al., 2024). Despite requiring little or no adaptation of the target model, these methods yield limited speedups. Hence, efforts turned to designing tailored drafters with variable levels of target integration. While some approaches maintained

loose conditioning (An et al., 2026a;b) to stay target agnostic, major efforts went into drafters conditioned on the target's final hidden state, including Medusa (Cai et al., 2024), Hydra (Ankner et al., 2024), ParallelSpec (Xiao et al., 2024), Falcon (Gao et al., 2025), EAGLE (Li et al., 2024b), and EAGLE-2 (Li et al., 2024a). EAGLE-3 (Li et al., 2025) tightened conditioning by accessing target latents from multiple layers, achieving significantly higher acceptance rates. SpecDiff (Christopher et al., 2025) introduced discrete diffusion to drafters, while SpecDiff-2 (Sandler et al., 2026) improved their alignment to the target with new loss functions; however, the diffusion drafters are fully decoupled. P-EAGLE (Hui et al., 2026) parallelizes target-latent-conditioned AR drafting, using block diffusion modeling with causal attention and implementing target-latent conditioning by superposing token embeddings with conditioning states at the already decoded token positions. DFlash (Chen et al., 2026) uses block diffusion modeling for parallel generation with multi-layer target conditioning. Instead of feeding the same projected latents to each drafting layer, DFlare (Zhang et al., 2026a) increases the flexibility by generating drafter-layer-specific conditionings. All prior latent-conditioned drafters require the target's full latent computation before drafting can begin. Instead, our PaPaSpec introduces depth-causal conditioning that allows drafting to proceed during verification, effectively hiding the drafting latency.

Macro-parallel verification and drafting AMUSD (McDanel, 2025) asynchronously performs drafting and verification on separate devices, rolling back the draft from the point of disagreement whenever the verifier rejects it. SSD (Kumar et al., 2026) drafts possible alternative outcomes for each position, allocating a budget based on a geometric distribution over acceptance length. Motivated by different GPU count scaling behavior of target and draft models, SwiftSpec (Zhang et al., 2026b) distributes them across separate GPUs. Mirror-SD (Bhendawade et al., 2025) speculates the next prefix by reading out intermediate verifier states; however, the subsequent drafting itself is performed with a micro-sequential drafter and without any target conditioning. All these macro-parallel drafters remain micro-sequential and lack target-latent conditioning. Yet, tight conditioning is known to yield the highest acceptance length, a property PaPaSpec preserves even under macro-level parallel drafting.

Looping Introduced by Dehghani et al. (2019), looping has shown benefits in inference-compute scaling at constant parameter count (Geiping et al., 2025; Wang et al., 2025; Jolicoeur-Martineau, 2025; Zhu et al., 2026; Prairie et al., 2026; McLeish et al., 2026). The success sparked interest in accelerating looped architectures themselves, as shown in contemporary works on pipelined self-speculative decoding (Cho et al., 2026; Ha & Kim, 2026). However, these works can only accelerate looped Transformers, while our approach is target-architecture invariant. Recently, looping has been proposed as a depth-scaling mechanism in diffusion models (Lee et al., 2026; Carballo-Castro et al., 2026). However, none of these works considered looping for dedicated speculative drafters. Instead, PaPaSpec leverages looping to improve acceptance length in latent-conditioned parallel drafters.

6 Conclusion

We presented PaPaSpec, a speculative decoding approach featuring a target-latent-conditioned parallel drafter that operates concurrently with the target verification. To achieve this, our depth-causal conditioning allows the drafter to consume the target latents that have been computed so far during verification. Thanks to removing drafting from the critical latency path, the drafter can dynamically increase its compute budget via looping. When integrated into SSD and enhanced with a novel dynamic drafting-budget allocation, PaPaSpec achieves up to 22% higher speedups than DFlash.

Limitations and future work In our experimental setup, we follow the established practice of operating the drafter on an additional GPU of the same size as the target host (Kumar et al., 2026). With the planned future general availability of accelerators for drafting (Bartlett et al., 2026; Weidman et al., 2026), future work could demonstrate PaPaSpec's performance at lower hardware cost. Moreover, while our dynamic drafting-budget allocation notably reduces cache misses, future work may explore estimating the missing bonus token (at position $\hat{\tau} = \gamma + 1$) in the case of full block acceptance. This would further resolve the remaining macro-sequentiality of PaPaSpec. Ideally, removing PaPaSpec's reliance on the bonus token at the drafter input would entirely resolve any cache issues. Finally, our evaluation is scoped to single-batch serving; PaPaSpec could be extended to concurrent serving, e.g., by exploring budget allocation across multiple concurrent requests.

AI USE STATEMENT

In this work, we used generative AI tools to implement parts of the software code and suggestions in experimental design and feedback on our research methodology. We have not used generative AI tools for data analysis and interpreting results, and its use in assistance in proofs or mathematical claims are not applicable to this work. Additionally, we used generative AI tools for formatting and discovering existing literature. We have reviewed all AI-assisted work by manually verifying all artifacts. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our method accelerates LLM inference while provably preserving the output distribution of the target model, a property we inherit from speculative decoding. It changes only how fast tokens are generated, not which tokens are generated. It therefore introduces no new capabilities, biases, or risks, and the ethical concerns associated with LLMs (e.g., harmful content, bias, privacy) remain properties of the target model and fall outside the scope of this work. Our experiments use only publicly available models and datasets in accordance with their licenses.

REPRODUCIBILITY STATEMENT

All training datasets, benchmarks, and models are public. We describe our experimental setup and implementations in Appendix B. We report the computational resources required in B.1 to reproduce the training of all drafters. We then describe specifics on training details in Appendix B.2. We summarize the architectural parameters and statistics of all the models used in this work in Appendix B.3. For key concepts of PaPaSpec, we provide code snippets in Appendix C. As indicated by ICLR's author guidelines, we are available to share the code anonymously during the rebuttal period.

REFERENCES

Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. PARD: Accelerating LLM Inference with Low-Cost PARallel Draft Model Adaptation. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026a. URL https://openreview.net/forum?id=XbOyv7iVGL.

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*, May 2026b. doi: 10.48550/arXiv.2605.08632. URL http://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. In *First Conference on Language Modeling (COLM)*, August 2024. URL https://openreview.net/forum?id=FbhjirzvJG.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=tyEyYT267x.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured Denoising Diffusion Models in Discrete State-Spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, November 2021a. URL https://openreview.net/forum?id=h7-XixPCAL.

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

James Bartlett, Natalie Serrino, Zain Asgar, Sudeep Bhoja, Prashant Nair, Nikhil Ghanathe, and Nithesh Kurella. Low-latency inference with speculative decoding on d-Matrix Corsair and GPU. Gimlet Labs Blog, March 2026. URL https://gimletlabs.ai/blog/low-latency-spec-decode-corsair. Accessed: 2026-09-09.

Nikhil Bhendawade, Kumari Nishu, Arnav Kundu, Chris Bartels, Minsik Cho, and Irina Belousova. Mirror Speculative Decoding: Breaking the Serial Barrier in LLM Inference. *arXiv preprint arXiv:2510.13161*, December 2025. doi: 10.48550/arXiv.2510.13161. URL http://arxiv.org/abs/2510.13161.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

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 *Forty-first International Conference on Machine Learning (ICML)*, June 2024. URL https://openreview.net/forum?id=PEpbUobfJv.

Alba Carballo-Castro, Julianna Piskorz, Paulius Rauba, Mihaela van der Schaar, and Pascal Frossard. Scaling with Recursion in Masked Discrete Diffusion Models. In *ICML 2026 Workshop on Structured Probabilistic Inference & Generative Modeling*, 2026. URL https://openreview.net/forum?id=zI2IeuDcoS.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating Large Language Model Decoding with Speculative Sampling. *arXiv preprint arXiv:2302.01318*, February 2023. doi: 10.48550/arXiv.2302.01318. URL http://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. In *Forty-third International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=Oz335dV48X.

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. *arXiv preprint arXiv:2107.03374*, July 2021. doi: 10.48550/arXiv.2107.03374. URL http://arxiv.org/abs/2107.03374.

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

SangLyul Cho, Langqing Cui, Sehoon Kim, Dongsu Han, and Insu Han. LoopSpec: Pipelined Self-Speculative Decoding for Looped Transformers. *arXiv preprint arXiv:2609.17184*, September 2026. doi: 10.48550/arXiv.2609.17184. URL http://arxiv.org/abs/2609.17184.

Jacob K Christopher, Brian R. Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative Diffusion Decoding: Accelerating Language Generation through Diffusion. 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. 12042–12059, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.601. URL https://aclanthology.org/2025.naacl-long.601/.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal Transformers. In *International Conference on Learning Representations (ICLR)*, March 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. *arXiv preprint arXiv:2605.00674*, May 2026. doi: 10.48550/arXiv.2605.00674. URL https://arxiv.org/abs/2605.00674.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the Sequential Dependency of LLM Inference Using Lookahead Decoding. In *Forty-first International Conference on Machine Learning (ICML)*, June 2024. URL https://openreview.net/forum?id=eDjvSFOkXw.

Xiangxiang Gao, Weisheng Xie, Yiwei Xiang, and Feng Ji. Falcon: Faster and Parallel Inference of Large Language Models Through Enhanced Semi-Autoregressive Drafting and Custom-Designed Decoding Tree. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(22):23933–23941, April 2025. ISSN 2374-3468. doi: 10.1609/aaai.v39i22.34566. URL https://ojs.aaai.org/index.php/AAAI/article/view/34566.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025. doi: 10.52202/085713-1380. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Paper-Conference.pdf.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, July 2026. doi: 10.48550/arXiv.2607.02770. URL https://arxiv.org/abs/2607.02770.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & Faster Large Language Models via Multi-token Prediction. In *Forty-first International Conference on Machine Learning (ICML)*, June 2024. URL https://openreview.net/forum?id=pEWAcejiU2.

Hyeongju Ha and Jae-Joon Kim. WaveFront Decoding: Parallelized Self-Speculative Decoding for Looped Language Models. *arXiv preprint arXiv:2609.23033*, September 2026. doi: 10.48550/arXiv.2609.23033. URL http://arxiv.org/abs/2609.23033.

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*, May 2026. doi: 10.48550/arXiv.2605.29707. URL http://arxiv.org/abs/2605.29707.

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. *arXiv preprint arXiv:2602.01469*, February 2026. doi: 10.48550/arXiv.2602.01469. URL http://arxiv.org/abs/2602.01469.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Alexia Jolicoeur-Martineau. Less is More: Recursive Reasoning with Tiny Networks. *arXiv preprint arXiv:2510.04871*, October 2025. doi: 10.48550/arXiv.2510.04871. URL http://arxiv.org/abs/2510.04871.

Tanishq Kumar, Tri Dao, and Avner May. Speculative Speculative Decoding. In *The Fourteenth International Conference on Learning Representations (ICLR)*, May 2026. URL https://openreview.net/forum?id=aL1Wnml9Ef.

Sanghyun Lee, Chunsan Hong, Seungryong Kim, Jonghyun Lee, Jongho Park, and Dongmin Park. Looped Diffusion Language Models. *arXiv preprint arXiv:2605.26106*, May 2026. doi: 10.48550/arXiv.2605.26106. URL http://arxiv.org/abs/2605.26106.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. In *Fortieth International Conference on Machine Learning (ICML)*, 2023. URL https://openreview.net/forum?id=C9NEblP8vS.

Shenggui Li, Chao Wang, Yikai Zhu, Yubo Wang, Fan Yin, Shuai Shi, Yefei Chen, Xiaomin Dong, Qiaoling Chen, Jin Pan, et al. SpecForge: A flexible and efficient open-source training framework for speculative decoding. *arXiv preprint arXiv:2603.18567*, 2026.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7421–7432, Miami, Florida, USA, November 2024a. 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. In *Forty-first International Conference on Machine Learning (ICML)*, June 2024b. URL https://openreview.net/forum?id=1NdN7eXyb4.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test. In *Advances in Neural Information Processing Systems (NeurIPS)*, October 2025. URL https://openreview.net/forum?id=4exx1hUffq.

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 *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless Self-Speculative Decoding for Accelerating LLMs via Double Early Exiting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*, November 2024. URL https://openreview.net/forum?id=lT3oc04mDp.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=UnUwSIgK5W.

Bradley McDanel. AMUSD: Asynchronous Multi-Device Speculative Decoding for LLM Acceleration. In *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2025. doi: 10.1109/ISCAS56072.2025.11043575. URL https://ieeexplore.ieee.org/abstract/document/11043575.

Sean Michael McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching Pretrained Language Models to Think Deeper with Retrofitted Recurrence. In *Third Conference on Language Modeling (COLM)*, August 2026. URL https://openreview.net/forum?id=PXVQTHYwgt.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=AjXkRZIvjB.

Giovanni Monea, Armand Joulin, and Edouard Grave. PaSS: Parallel Speculative Sampling. *arXiv preprint arXiv:2311.13581*, November 2023. doi: 10.48550/arXiv.2311.13581. URL http://arxiv.org/abs/2311.13581.

Dhruv Nathawani, Shuoyang Ding, Vitaly Lavrukhin, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Boris Ginsburg, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v2, August 2025. URL https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v2.

Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y. Fu. Parcae: Scaling Laws For Stable Looped Language Models. *arXiv preprint arXiv:2604.12946*, April 2026. doi: 10.48550/arXiv.2604.12946. URL http://arxiv.org/abs/2604.12946.

Red Hat. Speculators: A unified library for speculative decoding algorithms in llm serving. https://github.com/vllm-project/speculators, 2025.

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your LLM Knows the Future: Uncovering Its Multi-Token Prediction Potential. *arXiv preprint arXiv:2507.11851*, July 2025. doi: 10.48550/arXiv.2507.11851. URL http://arxiv.org/abs/2507.11851.

Jameson Sandler, Jacob Christopher, Tom Hartvigsen, and Ferdinando Fioretto. SpecDiff-2: Scaling Diffusion Drafter Alignment For Faster Speculative Decoding. *Proceedings of Machine Learning and Systems (MLSys)*, 8:1128–1147, May 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/041dad5ed2191b44ba3ed0e00cdc3187-Abstract-Conference.html.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise Parallel Decoding for Deep Autoregressive Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/hash/c4127b9194fe8562c64dc0f5bf2c93bc-Abstract.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical Reasoning Model. *arXiv preprint arXiv:2506.21734*, August 2025. doi: 10.48550/arXiv.2506.21734. URL http://arxiv.org/abs/2506.21734.

Seth Weidman, Kirthi Devleker, and Andrew Ling. How NVIDIA Groq 3 LPX unlocks ultrafast interactivity at long context on NVIDIA vera rubin. NVIDIA Technical Blog, August 2026. URL https://developer.nvidia.com/blog/how-nvidia-groq-3-lpx-unlocks-ultrafast-interactivity-at-long-context-on-nvidia-vera-rubin/. Accessed: 2026-09-09.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989. doi: 10.1162/neco.1989.1.2.270.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

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*, May 2026. doi: 10.48550/arXiv.2605.18810. URL http://arxiv.org/abs/2605.18810.

Zilin Xiao, Hongming Zhang, Tao Ge, Siru Ouyang, Vicente Ordonez, and Dong Yu. ParallelSpec: Parallel Drafter for Efficient Speculative Decoding. *arXiv preprint arXiv:2410.05589v1*, October 2024. URL https://arxiv.org/abs/2410.05589v1.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, May 2025. doi: 10.48550/arXiv.2505.09388. URL http://arxiv.org/abs/2505.09388.

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. *arXiv preprint arXiv:2606.02091*, June 2026a. doi: 10.48550/arXiv.2606.02091. URL http://arxiv.org/abs/2606.02091.

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, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.607. URL https://aclanthology.org/2024.acl-long.607/.

Ziyi Zhang, Ziheng Jiang, Chengquan Jiang, Menghan Yu, Size Zheng, Haibin Lin, Xin Liu, and Henry Hoffmann. SwiftSpec: Disaggregated Speculative Decoding and Fused Kernels for Low-Latency LLM Inference. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 2197–2211, Pittsburgh PA USA, March 2026b. ACM. ISBN 979-8-4007-2359-9. doi: 10.1145/3779212.3790246. URL https://dl.acm.org/doi/10.1145/3779212.3790246.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling Latent Reasoning via Looped Language Models. *arXiv preprint arXiv:2510.25741*, July 2026. doi: 10.48550/arXiv.2510.25741. URL http://arxiv.org/abs/2510.25741.

A Additional Results and Ablations

A.1 Throughput comparison to vLLM

Table 2 benchmarks the official DFlash drafters with two backends. The first is vLLM (v0.21.0) and the second is HuggingFace Transformers under `torch.compile`, as used in all experiments in this work. We benchmark the same checkpoints, for 128 prompts per dataset, across Math500, HumanEval and MT-Bench, with a maximum of 2048 new tokens, on H100 GPUs. Qwen3-4B and Qwen3-8B run on FlashAttention-2 in both vLLM and our framework. Gemma-4 runs on vLLM's Triton attention for both. Our backend achieves a lower decoding throughput than the optimized inference with vLLM on DFlash, about 19% fewer tokens per second for Qwen3-8B. However, the speedup gains of PaPaSpec are significant enough to close the gap between our simple Transformers implementation and vLLM, narrowing the gap to 4%. As PaPaSpec's speedup over DFlash already closes most of the gap to vLLM even within our simple Transformers implementation, we expect a native vLLM integration to carry this speedup over, yielding the fastest solution overall.

Table 2: The released DFlash drafters served by vLLM and by our compiled HuggingFace backend, and PaPaSpec (the configuration of the main table: dynamic fan-out at budget 32, three loops, the drafter on its own GPU) in the same backend: decode throughput (tok/s), acceptance length τ and speedup over autoregressive decoding; greedy, 128 prompts, 2048 new tokens, one request at a time. The Qwen3 targets run on FlashAttention-2 in both stacks; Gemma-4's target runs on vLLM's Triton attention backend due to vLLM refusing FlashAttention for it.

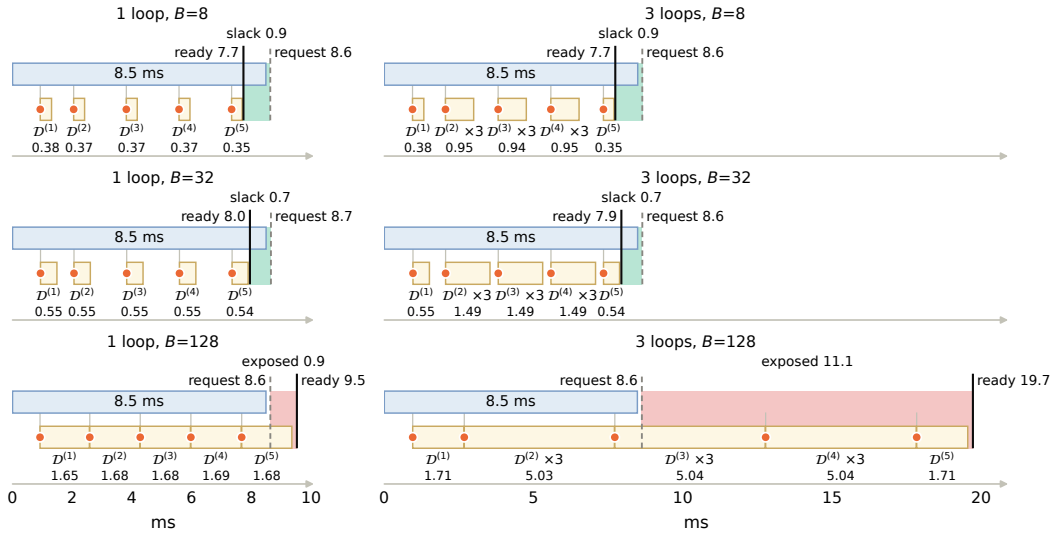
Model	**Stack**	MATH-500			HumanEval			MT-Bench		
		tok/s	τ	Speedup	tok/s	τ	Speedup	tok/s	τ	Speedup
Qwen3-8B	vLLM (DFlash)	901	8.05	6.22	751	6.52	5.15	392	4.23	2.67
	HF (DFlash)	731	8.14	5.65	602	6.62	4.64	321	4.25	2.48
	HF (PaPaSpec)	863	8.70	6.72	721	7.09	5.63	384	4.58	2.98
Qwen3-4B	vLLM (DFlash)	1354	8.11	6.24	1140	6.67	5.23	608	4.19	2.78
	HF (DFlash)	1054	8.23	5.43	860	6.68	4.42	460	4.35	2.40
	HF (PaPaSpec)	1201	8.53	6.38	1010	7.03	5.37	552	4.71	2.92
Gemma-4 26B-A4B	vLLM (DFlash)	633	7.32	2.97	644	8.32	3.04	315	4.49	1.48
	HF (DFlash)	581	7.31	3.68	594	7.80	3.77	280	4.19	1.78
	HF (PaPaSpec)	666	7.93	4.26	702	9.03	4.48	302	4.37	1.92

A.2 Latency breakdown of PaPaSpec

Figure 7 covers the latencies of key events in a single step of parallel verification and drafting in PaPaSpec across cache allocation budgets. The latency events are invariant to the dataset being benchmarked; they depend only on the drafter, the target model, and key parameters to our method such as the number of loops and budget. For PaPaSpec with a single loop, the drafter prepares all draft tokens 0.9ms before the target requests the next batch of draft tokens for a budget of 8, and 0.7ms for a budget of 32. However, with a budget of 128, the compute is saturated, and the drafts are ready 0.9ms after the target requests them. This is especially pronounced for PaPaSpec with 3 loops, where the cache is only ready 11.1 ms after being requested by the verifier, more than the verification time itself. Hence, the cache budget and number of loops have to be jointly considered given the compute resources.

A.3 Budget allocation

Cache strategy ablation The detailed results of our ablation of the three drafter cache allocation strategies are in Table 3. The dynamic cache strategy consistently outperforms all other cache strategies in miss rate and speedup. Looping further improves speedup through two factors: increasing the acceptance length and decreasing the miss rate. The speedup eventually saturates after a budget



(a) 1 loop (b) 3 loops

Figure 7: Event timeline across different cache budgets for the Qwen3-8B target, operating our drafter with **(a)** 1 loop and **(b)** 3 loops. The green area indicates the slack where the draft is ready before the target model requests the next draft tokens. Red indicates the drafting latency is exposed on the critical path, with the draft being ready only after it is requested by the target.

Table 3: Miss rate (MR, %: the share of rejected blocks whose correction was not cached) and speedup (×) over autoregressive generation by number of loops, strategy, and budget. The target model is Qwen3-8B, the drafter is our PaPaSpec model from Table 1. Each value is an average over Math500, HumanEval and MT-Bench.

		Budget 8		Budget 16		Budget 32		Budget 64		Budget 128	
Loops	Strategy	MR	Speedup	MR	Speedup	MR	Speedup	MR	Speedup	MR	Speedup
1	Uniform	60.9	4.58	48.4	4.66	32.0	4.71	19.9	4.77	12.2	4.31
	Geometric	61.1	4.68	46.1	4.75	30.0	4.83	18.4	4.72	10.7	3.96
	Dynamic	35.7	4.78	24.0	4.84	15.2	4.86	9.1	**4.87**	5.3	**4.39**
3	Uniform	60.8	4.71	47.6	4.83	31.5	4.93	19.5	4.02	11.7	2.39
	Geometric	60.7	4.88	45.5	5.00	29.6	**5.14**	17.7	3.85	10.1	2.17
	Dynamic	**35.0**	**4.96**	**23.6**	**5.07**	**15.0**	5.11	**8.9**	4.10	**5.2**	2.41

of 32 for PaPaSpec with 3 loops and after a budget of 64 for 1 loop, mainly due to the drafter taking more time than the verifier to prepare the drafts, exposing drafting along the critical path.

Miss Rate decomposition A limitation of our method, which we inherit from SSD (Kumar et al., 2026), is the case where the full block is accepted from the target model ($\hat{\tau} = 16$). The drafter has not predicted any bonus token since γ is capped, and the bonus token from a full block acceptance is emitted at position $\gamma + 1$. Figure 8 shows the decomposition of cache misses into full block acceptances and blocks whose bonus token was not cached. Looping, across the uniform and dynamic cache strategies, increases the full block acceptance rate and slightly reduces the rejection rate.

B Experimental Details

B.1 Compute Resources

We used a full node of 8xH100 80GB HBM3 GPUs for all our training runs. The training of our ablations on the 100k dataset, all of which we were using Qwen3-8B as the target model, took 6.5 hours, for a total of 52 GPU hours. As for our PaPaSpec models reported in the main results, the

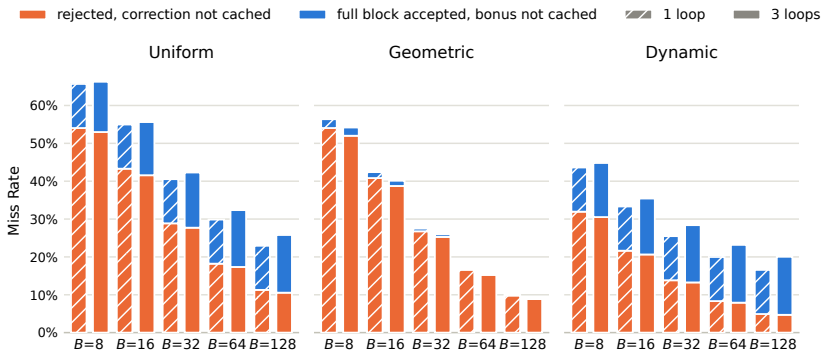


Figure 8: Breakdown of cache miss due to rejection and full block acceptance.

drafter for Gemma-4-26B-A4B took 66 hours to train (528 GPU hours), the drafter for Qwen3-4B took 36 hours (288 GPU hours), and the drafter for Qwen3-8B took 61 hours (488 GPU hours).

B.2 Training Implementation

Framework We adapted the official implementation of DFlash's training recipe from vLLM's Speculators (Red Hat, 2025), for both the reconstruction of the D-PACE baseline, reproducing the released checkpoint, and the PaPaSpec checkpoints. We cap the sequence length of each prompt in our data mixture to 3072. We sample 512 anchors per sequence and use a block size γ of 16.

Optimizer and batch size We train up to 6 epochs and use AdamW with a learning rate of 6e-4, a linear warmup of 4% of steps, a clipping threshold of 1 and a cosine scheduler. We use a global batch size of 8 sequences per optimizer step, one sequence per 8 GPUs and aggregate with Distributed Data Parallel. To reproduce the reported acceptance length metrics reported in DFlash (Chen et al., 2026), we additionally needed to cast the FP16 weights up to FP32 for AdamW, with the forward pass and the checkpoint being kept in FP16. This was borrowed from SpecForge's implementation (Li et al., 2026).

Training dataset For the ablation experiments, in Figure 5, we use a mixture of 50k prompts from CodeAlpaca (Luo et al., 2024) and 50k prompts from NVIDIA's Nemotron Post-Training Dataset V2 (Nathawani et al., 2025). All 50k samples from CodeAlpaca are code-related, whereas for the Nemotron Post-Training Dataset we sample with equal probability from their chat, STEM, math, and code. We withheld 10% of the training data for validation metrics during training. Our main PaPaSpec results, as well as our D-PACE baseline and a reproduction of DFlash matching the reported acceptance lengths, were trained on all of CodeAlpaca (111k rows) and 800k samples from the Nemotron Post-Training dataset. More specifically, 200k samples in STEM, 200k in math, 175k in code, and 225k in chat.

Hidden states Generating the verifier hidden states for training can be done either offline, where all hidden states are generated and stored for later use during training, or generated online when needed during training. The total amount of hidden states depends multiplicatively on the sequence length, number of samples in the training dataset, the verifier hidden state dimension and the amount of hidden states m we want to condition on. In our experiments, we have a maximum sequence length of 3072, 820 144 training sequences, and a hidden dimension of 4 096 for Qwen3-8B, 5 target hidden layers and FP16 precision (2 bytes per value) it amounts to $3072{\cdot}820,144{\cdot}4,096{\cdot}5{\cdot}2 \approx 103.2$ TB. This was prohibitively expensive to store, so we generated the hidden states online during training by launching a vLLM instance on each of our training GPUs to prefill and retrieve the hidden states. We allocated 30% of each H100's 80 GB VRAM to vLLM extraction's and the rest for training. For Gemma-4-26B-A4E we had to host the vLLM servers on separate GPUs from training, allocating 4 GPUs to each, since both would not fit in the 80 GB of the H100's VRAM.

	Qwen3-4B		Qwen3-8B		Gemma-4-26B-A4B	
	Target	Drafter	Target	Drafter	Target	Drafter
Parameters	4.0B	0.54B	8.2B	1.05B	25.8B (4B act.)	0.42B
Layers	36	5	36	5	30	5
Hidden size	2560	2560	4096	4096	2816	2816
FFN width	9728	9728	12288	12288	704×128 exp. (top-8)	5632
FFN type	SwiGLU	SwiGLU	SwiGLU	SwiGLU	MoE, GeGLU	SwiGLU
Attention heads (Q / KV)	32 / 8	32 / 8	32 / 8	32 / 8	16 / 8 (global: 16 / 2)	32 / 8
Head dimension	128	128	128	128	256 (global: 512)	128
Vocabulary	151,936 (shared)		151,936 (shared)		262,144 (shared)	
$\{l_1 \cdots l_m\}$	—	1, 9, 17, 25, 33	—	1, 9, 17, 25, 33	—	6, 11, 17, 22, 27
Draft block size γ	—	16	—	16	—	16

Table 4: Architectural hyperparameters for the three target models and the DFlash drafters.

Depth-causal conditioning During training, we sampled $\hat{\tau}$ from $\Pr(\hat{\tau}{=}k) = (1-\rho)\,\rho^{k-1}$ using a ρ of 0.8 for the smaller scale ablations trained on the 100k dataset and with a ρ of 0.85 for our main results on the 800k dataset.

D-PACE loss The D-PACE (Wu et al., 2026) loss function is borrowed from the official implementation, set with a weight smoothing hyperparameter of 0.5. We use this for both the strong D-PACE baseline, as well as our PaPaSpec drafters.

B.3 Model Hyperparameters

Table 4 summarizes key architectural differences across the three target models and drafters. The drafters are identical to the official DFlash releases[1] [2] [3] for the corresponding target models. Our PaPaSpec inherits the same hyperparameters, with only the lightweight addition of the α and β parameters for our depth-causal conditioning. Looping extends the test-time compute at inference time, but keeps the parameters fixed.

C Implementation

We highlight more specific details of the implementation of two key parts of PaPaSpec: the target-drafter communication through NCCL and the conditioning mechanism.

C.1 Verifier-drafter communication

Listings 1 and 2 give the per-step protocol. The two ranks run on two GPUs of the same node, joined by a dedicated NCCL process group, and exchange only fixed-shape point-to-point messages. `link` denotes this group as seen from one rank with `send`/`recv` the blocking and `isend`/`irecv` the non-blocking point-to-point primitives of `torch.distributed` addressed to the peer rank.

Each step opens with a request from the verifier, the pair $(\hat{\tau}, x_{n+1})$ of acceptance length and bonus token, and a fixed-size reply from the drafter (a cache-hit flag and the drafted tokens $\hat{x}_{n+2:n+\gamma}$).

The verification forward is captured as $m{+}1$ CUDA graphs cut at the selected layers $\{l_1, \ldots, l_m\}$, so each latent $h^{(l_k)} \in \mathbb{R}^{\gamma \times d_T}$ is sent with a non-blocking `end` as soon as its segment has been replayed.

On the drafter rank, the receive of each latent is posted right before the drafter layer that consumes it, since NCCL matches point-to-point messages by issue order and `wait()` is a stream-level wait. $\mathcal{D}^{(k)}$ runs while the verifier is still computing $h^{(l_{k+1})}$. Receives are posted one at a time (a pending receive kernel contends with the drafter's compute), and the drafter never synchronizes its device, which would serialize it behind the verifier's forward. Once all m latents have arrived, the drafter

[1] https://huggingface.co/z-lab/Qwen3-8B-DFlash-b16
[2] https://huggingface.co/z-lab/Qwen3-4B-DFlash-b16
[3] https://huggingface.co/z-lab/gemma-4-26B-A4B-it-DFlash

runs $\mathrm{LM}_{\text{head}}$ over its cached branches while polling for the next request, so a cache hit is answered with no GPU work on the critical path.

Listing 1: Verifier rank: one decoding step. The m selected layers $l_1, \ldots, l_m$ cut the verifier into $m{+}1$ CUDA graphs; each latent $h^{(l_k)}$ is sent non-blocking as soon as its segment has been replayed.

```
link.send([τ̂, x_{n+1}])                          # request
x̂_{n+2:n+γ} = link.recv()                         # reply: cached or jit draft
for k in range(1, m + 2):
    h = graphs[k].replay()                              # layers l_{k-1}+1 ... l_k
    if k <= m:
        link.isend(h)                                    # latent h^{(l_k)} ∈ R^{γ×d_T}
verify(LM_head(h), x̂_{n+2:n+γ})
```

Listing 2: Drafter rank: one decoding step. The receive of each latent is posted right before the drafter layer that consumes it; `wait()` is a stream-level wait, so $\mathcal{D}^{(k)}$ runs while the verifier computes $h^{(l_{k+1})}$.

```
τ̂, x_{n+1} = link.recv()                          # request
link.send(lookup_or_jit_draft(τ̂, x_{n+1}))        # reply
for k in range(1, m + 1):
    h^{(l_k)} = link.irecv().wait()                  # posted only now
    D^{(k)}(h^{(l_k)})                             # branch drafts, overlapped
```

C.2 Conditioning Mechanism

Listing 3 gives the prefix and depth-causal conditioning as computed. The $W^{(i)}$ are the column blocks of DFlash's fusion matrix W, so the projected latents are formed once per position and mixed per drafter layer. Each drafter decoder layer $\mathcal{D}^{(l)}$ adds two softmax rows on top of DFlash: $\alpha_{\cdot,l}$ over all m latents for the verified prefix, and $\beta_{\cdot,l}$ over $l_1, \ldots, l_l$ only for the $\hat{\tau} - 1$ positions under verification, where the weigths for $i > l$ are masked to $-\infty$. Both tables are zero-initialised (i.e. uniform) so α starts as DFlash's shared conditioning. At inference the α-rows of the prefix are cached across steps; the β-rows are rebuilt for each block as its latents arrive.

Listing 3: Prefix and depth-causal conditioning for drafter layer $\mathcal{D}^{(l)}$. $a, b \in \mathbb{R}^{L_D \times m}$ are the learned mixing weights; the $-\infty$ mask makes $\beta_{i,l} = 0$ (weight and gradient) for every selected layer l_i not yet computed.

```
u_{t,i} = W^{(i)} h_t^{(l_i)}                    # projected latents, i = 1..m
α_{·,l} = softmax(a_l)                                  # all m latents
β_{·,l} = softmax(b_l.masked_fill(i > l, −∞))    # depth-causal: i ≤ l only
for t in 1..n:                                    # prefix (previously verified)
    c_t^{(l)} = RMSNorm(Σ_{i=1}^{m} α_{i,l} u_{t,i})
for t in n+1..n+τ̂−1:                             # block under verification
    c_t^{(l)} = RMSNorm(Σ_{i=1}^{l} β_{i,l} u_{t,i})
```