SPECULATIVE PIPELINE DECODING: HIGHER-ACCURACY DRAFTING WITH HIDDEN LATENCY VIA PIPELINE PARALLELISM

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative Decoding (SD) accelerates low-concurrency LLM inference with a draft-then-verify paradigm. Mainstream methods, however, rely on multi-token prediction, which incurs compounding prediction difficulty and exposed draft latency. We propose Speculative Pipeline Decoding (SPD), which partitions the target LLM into n pipeline stages so that n tokens of a single sequence advance in parallel. To keep the pipeline saturated, a Pipeline Draft Module (PDM) aggregates multi-depth target features to predict the next token and runs concurrently with each pipeline step, yielding bounded prediction difficulty, higher acceptance, and hidden draft latency. Experiments show that SPD achieves higher theoretical and wall-clock speedup than EAGLE-3 at moderate pipeline width, while more aggressive widths still leave room for further gains.

1 INTRODUCTION

The inference process of Large Language Models (LLMs) is bottlenecked by its autoregressive nature, which fundamentally restricts performance to memory bandwidth rather than compute capacity. Speculative Decoding (SD) (Leviathan et al., 2023) has emerged as a prominent solution to mitigate this memory-bound latency. Traditional SD employs a smaller, independent draft model to predict a sequence of future tokens, which are subsequently verified together by the target LLM. Recent advancements, such as EAGLE (Li et al., 2024), bypass the need for an external draft model by appending a lightweight feature-extrapolation head to the target model, directly utilizing the LLM's internal hidden states to improve draft acceptance rates.

Despite these improvements, existing speculative decoding methods are fundamentally anchored to a **multi-token prediction paradigm**. This paradigm inherently suffers from two critical, structural limitations that prevent further acceleration.

First, multi-token drafting suffers from **compounding feature drift**: after leaving the last target-verified token, later draft steps are increasingly conditioned on the draft module's own shallow hidden states rather than the target's, so acceptance decays sharply with draft length. This is why EAGLE-3 (Li et al., 2026) typically keeps the draft length small (e.g., k=3) in practice.

Second, the multi-token paradigm introduces **exposed draft latency**. Serial drafting leaves the target idle and offsets the verify-side speedup. Recent methods such as DFlash (Chen et al., 2026) reduce but do not fully remove this cost, and may trade off draft quality.

To overcome these limitations, we propose **Speculative Pipeline Decoding (SPD)**. We restructure the target LLM into an n-stage pipeline so that n tokens of a single sequence advance at distinct depths concurrently. This use of pipeline parallelism differs from the conventional setting in training and multi-request serving, where stages are filled with *micro-batches* to hide pipeline bubbles and maximize throughput. SPD instead pipelines *tokens of a single sequence* to reduce decode latency, and idle stages arise from a missing next token rather than a missing micro-batch. A **Pipeline Draft Module (PDM)** therefore predicts a single next token each step to keep the pipeline saturated. SPD has two key designs:

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.

Table 1: Paradigm comparison. MTP = multi-token prediction (e.g., EAGLE-3); PP = pipeline parallelism.

	MTP	**PPSD**	**SPD (ours)**
Target execution	single-path / batch verify	n-stage PP	n-stage PP
Draft horizon	k future tokens	1 token/step	1 token/step
Draft features	target + draft HS	first-stage only	multi-depth target HS
Draft timing	before verify (serial)	after PP step	with PP step
Draft latency	often exposed	exposed	masked ($L_s \approx L/n-1$)

First, SPD uses **Multi-Depth Feature Aggregation** and does not condition on draft-generated features (Figure 1). Each step predicts one token from target-LLM hidden states aggregated across pipeline depths, including partial states of in-flight tokens, so prediction stays in the target's feature space. Difficulty is bounded by the fixed width n rather than an open-ended draft horizon, yielding higher acceptance without choosing a draft length k.

Second, SPD **eliminates exposed serial drafting latency** by fully overlapping the PDM with the target pipeline step. We shift the PDM forward so that it conditions on *pre-step* features (intermediate states available before the current pipeline forward) rather than *post-step* features (states after that forward), enabling true concurrency with the stage compute. Pre-step features are slightly less informative, but multi-depth aggregation compensates, so a deeper PDM can still have its latency masked by the pipeline step.

We conduct extensive evaluations using Qwen3.5-4B and Qwen3.5-9B (Qwen Team, 2026) across three representative benchmarks: MT-Bench (Zheng et al., 2023), GSM8K (Cobbe et al., 2021), and HumanEval (Chen et al., 2021). Empirical results show that SPD surpasses EAGLE-3 (Li et al., 2026) in wall-clock speedup across both model sizes at moderate pipeline width. At larger widths (e.g., n=16), theoretical gains remain strong while measured speedup is currently tempered by per-cycle systems costs. This gap points to further headroom from systems-level optimizations rather than a limit of the decoding paradigm itself.

Our contributions are summarized as follows:

- We formulate a pipeline-parallel speculative decoding paradigm that replaces unbounded multi-token draft horizons with a fixed pipeline width, structurally bounding draft difficulty while accelerating single-sequence decode.
- We show that conditioning a single-token drafter solely on multi-depth target features, and overlapping it with the pipeline step, yields higher acceptance while removing exposed serial drafting latency relative to drafting baselines.
- We provide distributed wall-clock evidence on multi-GPU hardware, and characterize when theory and practice diverge, pointing to concrete systems opportunities for closing the remaining gap at aggressive pipeline widths.

2 Related Work

Speculative Decoding. Standard speculative decoding (Leviathan et al., 2023) accelerates autoregressive generation by drafting several future tokens with a cheaper model and verifying them in one parallel target forward. The systems intuition is that verification amortizes a single weight load over multiple candidates, raising arithmetic intensity on the target pass. Self-speculative methods such as EAGLE and EAGLE-3 (Li et al., 2024; 2026) remove the external draft LM by attaching a lightweight head that extrapolates from the target's internal hidden states. Despite higher acceptance than independent draft models, they remain multi-token predictors: later draft steps condition on the draft's own features, so acceptance decays with draft length and serial drafting still idles the target. Table 1 contrasts this multi-token prediction (MTP) paradigm with PPSD and SPD.

Parallel and Diffusion Drafting. A complementary line of work attacks draft latency itself. P-EAGLE (Hui et al., 2026) emits multiple draft tokens in parallel at quadratic training cost; Speculative Speculative Decoding (Kumar et al., 2026) overlaps drafting with anticipated verification

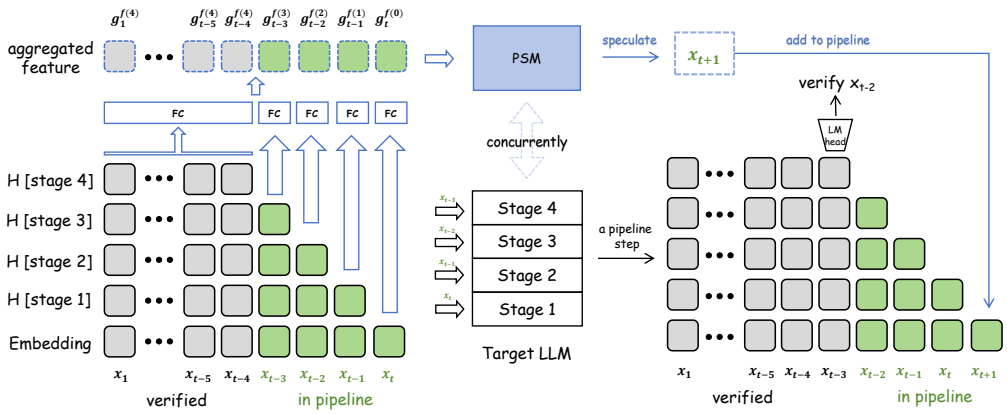


Figure 1: Speculative Pipeline Decoding with n=4 stages. The target LLM is partitioned into four stages. At the beginning of a saturated cycle, in-flight tokens $x_{t-3}, \ldots, x_t$ occupy the pipeline at varying depths, while $x_1, \ldots, x_{t-4}$ are already fully processed. For each token, hidden states at a fixed set of layer checkpoints (selected by how far that token has progressed) are concatenated and projected into one aggregated feature for the Pipeline Draft Module (PDM). The PDM drafts $\hat{x}_{t+1}$ concurrently with the target pipeline forward. After the forward, the oldest in-flight token x_{t-3} exits and yields logits that verify x_{t-2}; if accepted, $\hat{x}_{t+1}$ enters stage 1 for the next cycle.

outcomes at the price of branching compute. More recently, diffusion-style drafters push parallelism further inside the draft stage: DFlash (Chen et al., 2026) replaces autoregressive drafting with a lightweight block-diffusion model that proposes an entire token block in one forward pass, while DSpark (Cheng et al., 2026) pairs a parallel draft backbone with a lightweight sequential head and confidence-scheduled verification to curb suffix decay and verification waste under load. These approaches improve how drafts are *generated*, yet they still verify against a conventionally executed target. SPD instead restructures the target's own execution; diffusion or semi-autoregressive drafters are therefore orthogonal ingredients that could, in principle, be composed with a pipeline-parallel target.

Pipeline Drafting. Pipeline parallelism is standard in distributed training and serving, where micro-batches typically fill stages to hide bubbles and raise throughput. Using the same stage partitioning to accelerate *single-sequence* decode instead requires drafting the next token so that stages stay occupied without batching independent requests. SpecPipe (Yin et al., 2025) combines pipelines with speculative decoding mainly as a system optimization for draft trees. PPSD (Li et al., 2025) also uses pipeline parallelism for single-sequence decode, but drafts from first-stage features alone, so acceptance remains low, and still runs drafting after each pipeline step, leaving draft latency exposed. SPD addresses both issues with multi-depth target aggregation and a pre-step schedule that overlaps the PDM with the pipeline step.

3 Methodology

We now detail Speculative Pipeline Decoding. The design follows a simple narrative: pipeline-parallel target execution creates a need to supply the next token before earlier tokens finish (§3.1); filling that need without leaving the target's feature space motivates multi-depth aggregation (§3.2); an early, parallel draft schedule then hides draft cost behind each pipeline step (§3.3); and streaming verification, phase-shifted from drafting by the pipeline width, keeps the schedule correct under accept/reject (§3.4). Figure 1 summarizes the end-to-end loop.

3.1 Pipeline Execution and the Need for Drafting

Standard autoregressive decoding feeds one token through all L layers before the next token can begin. Pipeline parallelism instead partitions the target LLM into n stages and advances n tokens of the *same* sequence at different depths in lockstep (Figure 1). Each cycle then costs roughly one

stage forward rather than a full model forward, which is the source of SPD's latency reduction. This benefit, however, is available only if the pipeline stays full: every cycle must pop a finished token from the last stage *and* push a new token into the first stage. In single-sequence decoding that creates a circular dependency: the next token is not yet known because its predecessor has not produced final logits. SPD breaks the circle by **drafting** the missing token while earlier tokens are still in flight.

Concretely, we attach a **Pipeline Draft Module (PDM)**: an L_s-layer Transformer decoder f_θ with an independent LM head W_{lm}. We fix the PDM to Qwen3-style standard attention even when the target (e.g., Qwen3.5) mixes attention types, avoiding linear-attention complexity in the drafter. We deliberately predict only the *single* next token per cycle, since multi-token drafting would reintroduce compounding feature drift inside the pipeline.

At step t, let $x_1, \ldots, x_t$ be the current sequence (verified prefix plus in-flight tokens). Position i carries a pipeline depth $k_i \in \{0, \ldots, n\}$ equal to how many stages x_i has completed when drafting begins (§3.3). Rather than feeding raw token IDs, the PDM consumes a depth-conditioned feature sequence whose entries are derived from each token's already-computed target states (construction deferred to §3.2):

$$\begin{aligned} \mathcal{G}_t &= \big(g_1^{k_1},\, g_2^{k_2},\, \ldots,\, g_t^{k_t}\big), \quad \mathbf{h}_t = \big[f_\theta(\mathcal{G}_t)\big]_t \\ \hat{x}_{t+1} &\sim \mathrm{softmax}\big(W_{\text{lm}}\mathbf{h}_t\big) \end{aligned} \tag{1}$$

where $[\cdot]_t$ denotes the last-position representation. Once the pipeline is saturated, verified positions all have depth $k{=}n$, while the n in-flight tokens form a *depth staircase* from almost-finished to newly embedded:

$$\mathcal{G}_t = \big(g_1^n, \ldots, g_{t-n}^n,\, g_{t-n+1}^{n-1}, \ldots, g_{t-1}^1,\, g_t^0\big). \tag{2}$$

Eq. equation 2 only records how far each token has progressed: larger k means a richer (later-defined) feature, and $k{=}0$ is the newest token's embedding-level input.

3.2 Multi-Depth Feature Aggregation

We now define the features g_i^k used in Eqs. equation 1–equation 2. PPSD uses only first-stage features, which is cheap but information-poor and degrades as n grows (each stage becomes shallower). EAGLE-3 already fuses multi-layer target hidden states on the verified prefix, but after leaving the last verified token it extrapolates from the draft module's own hidden states, which is precisely the path that causes compounding feature drift. SPD takes a different route: for *every* position, including in-flight tokens, use **as much of the target LLM as is already available**, and never condition on draft-generated features.

Let H_t^l denote the hidden state of token x_t after layer l ($l{=}0$ is the embedding). If x_t has completed k stages, the deepest available layer is $l_{\max}(k) = \min(L,\, k \cdot L/n)$. Maintaining a separate fusion rule for every k would be cumbersome, so we fix m shared anchors $\mathcal{B} = [b_0, \ldots, b_{m-1}]$ with $b_0{=}0$ (e.g. for $L{=}32$ we use $[0, 8, 16, 24, 31]$). A token at depth k may consume every anchor that lies no deeper than $l_{\max}(k)$:

$$f(k) = \max\big\{\, i \;\big|\; b_i \le l_{\max}(k) \,\big\}, \tag{3}$$

$$g_t^k = \mathrm{FC}_{f(k)}\Big(\mathrm{Concat}\Big(H_t^{b_0}, \ldots, H_t^{b_{f(k)}}\Big)\Big). \tag{4}$$

Thus g_t^k is always built from x_t's own target hidden states; the superscript k selects *which* anchors are eligible. Verified positions in Eq. equation 2 therefore share only the full-depth rule $k{=}n$ (the richest anchor set), while each g_i^n remains distinct. As a token advances through the pipeline, new anchors become eligible and g_t^k grows strictly richer; at $k{=}0$ it reduces to a projection of the embedding.

Before the pipeline is full, only $a{<}n$ tokens are in flight. Earlier positions, including the prefill prefix, are already fully processed, so they receive depth $k{=}n$; the trailing a tokens form a shortened staircase:

$$\mathcal{G}_t = \big(g_1^n, \ldots, g_{t-a}^n,\, g_{t-a+1}^{a-1}, \ldots, g_t^0\big). \tag{5}$$

When $a{=}n$, this reduces to the steady-state layout in Eq. equation 2.

The incompleteness of any feature is therefore capped by the constant pipeline width n, not by an open-ended draft length. This is why SPD need not tune a sensitive multi-token horizon: difficulty

is structurally bounded, and drafting stays inside the target's feature geometry. Training-time alignment with these layouts (sequence replication across aggregation types, simulated fill levels, and the structural attention mask) is detailed in Appendix A.

3.3 Simultaneous Execution: Hiding Draft Latency Behind the Pipeline

Even an accurate drafter can offset its own gains if it runs serially with the target. Classical speculative decoding must draft and then verify the *same* candidates within one round, so the two stages are data-dependent and cannot overlap. Under pipeline execution the two are phase-shifted by about n steps (§3.4): a cycle verifies a token drafted earlier while the PDM proposes a token that will be checked only later. With no intra-cycle draft→verify dependency, drafting can proceed in parallel with the target forward.

To start with that forward, the PDM must use features already available at the beginning of the step. We therefore condition it on **pre-step** features (intermediate hidden states present *before* the current pipeline forward) rather than **post-step** features obtained after the forward completes. Concretely, as soon as a newly drafted token enters stage 1, we launch the PDM on the current depth staircase: the newest token contributes only an embedding (g^0), and older in-flight tokens contribute their pre-step intermediates. Relative to a post-step schedule, every in-flight position is one stage shallower (e.g., g_t^0 instead of g_t^1 for the newest token), so draft accuracy is slightly lower (e.g., $\sim 5\%$ on Qwen3.5-4B). The modest drop is more than repaid by latency masking (§4.5).

True concurrency requires a dedicated draft rank: each of the n stages occupies one GPU, and the PDM occupies an additional GPU that receives intermediate hidden states and runs aggregation plus decoding while stage workers compute. This extra rank is the systems cost of keeping drafting off the critical path: trading one device for removing exposed serial draft latency.

For the overlap to be useful, drafting must finish no later than one stage forward. Under equal per-layer cost this suggests $L_s \leq L/n$; accounting for aggregation and the independent LM head, we adopt the practical rule $L_s \approx L/n - 1$, which still allows a multi-layer drafter far deeper than a single-layer head.

3.4 Streaming Verification and Rollback

Lossless decoding still requires verifying drafted tokens against the target distribution. Because the pipeline finishes one token per cycle, verification is **streaming**: we check one in-flight successor at a time rather than a long draft block.

We keep the indexing of Eq. equation 2: after a saturated step, $x_{1:t-n}$ is the verified prefix, $x_{t-n+1:t}$ are the in-flight tokens (oldest to newest), and the PDM is about to propose $\hat{x}_{t+1}$. An in-flight successor becomes checkable only after its predecessor exits the last stage, so the token under verification was drafted about n cycles earlier. This is the phase offset presupposed in §3.3.

After each forward, the oldest in-flight token x_{t-n+1} exits, is committed (it was accepted in the previous cycle), and yields H_{t-n+1}^L. The target LM head forms $P(x_{t-n+2})$, which judges x_{t-n+2}. Under greedy decoding we require an exact argmax match; under sampling we apply standard rejection sampling.

If x_{t-n+2} is accepted, $\hat{x}_{t+1}$ enters stage 1 and the steady-state flow continues. If it is rejected, we truncate target KV caches to length $t-n+1$ (discarding $x_{t-n+2}, \ldots, x_t$), flush intermediate pipeline activations, and reseed stage 1 from $x_{t-n+2} \sim P(x_{t-n+2} \mid x_{1:t-n+1})$.

4 Experiments

4.1 Experimental Setup

Training. Following EAGLE-3, we freeze all of the target LLM's parameters and train only the PDM. Supervision is knowledge distillation: at every position the PDM predicts next-token logits from aggregated target features (Eq. equation 4), and we minimize KL divergence against the frozen teacher's (i.e., target LLM's) logits of the same token. The PDM's LM head is initialized from the

teacher's but with a smaller vocabulary to keep the drafter compact. We mix common instruction-following datasets: ShareGPT-70k, UltraChat-200k (Ding et al., 2023), SmolTalk (Allal et al., 2025), and SmolTalk-Chinese (Yu et al., 2025), filter out sequences longer than 2,048 tokens, and obtain about 1.2M samples. Optimization uses learning rate $1e{-}4$ with linear decay for a single epoch. How training matches inference's varying pipeline occupancy is described in Appendix A.

Evaluation. We evaluate on Qwen3.5-4B and Qwen3.5-9B (Qwen Team, 2026) with thinking mode disabled (standard non-chain-of-thought decoding). Following EAGLE-3, we report MT-Bench (chat), GSM8K (math), and HumanEval (code), with a maximum generation length of 512. Sampling runs ($T{=}1$) use top-$k{=}50$ and top-$p{=}1.0$. Baselines are EAGLE-3 (Li et al., 2026) with draft length $k{\in}\{3, 7, 15\}$ and $L_s{=}1$, and PPSD (Li et al., 2025) with $n{\in}\{4, 8, 16\}$ and $L_s{=}1$. Neither baseline hides draft latency: EAGLE-3 drafts serially before verify, and PPSD drafts after each pipeline step. Thus $L_s{>}1$ would multiply exposed draft cost and typically slow wall-clock time; following common practice we therefore keep their drafters at one layer. For SPD, latency masking makes deeper drafters practical: we use uniform L/n stage splits and select L_s near $L/n - 1$. Distributed wall-clock measurements run on multiple 8×H20 GPU servers: SPD places each of the n stages and the PDM on a separate rank ($n{+}1$ GPUs), so configurations such as $n{=}8$ (9 ranks) and $n{=}16$ (17 ranks) span more than one node; EAGLE-3 and the single-GPU autoregressive baseline use one H20 on the same cluster. For wall-clock speedup, we run EAGLE-3 with the existing SGLang (Zheng et al., 2024) serving stack, while PPSD and SPD use our own PyTorch inference framework implemented from scratch for pipeline-parallel speculative decoding.

4.2 Metrics

Unless noted, all speedups are relative to standard autoregressive decoding of the same target LLM (single-GPU, batch size 1). Speedup claims mix drafting quality with systems cost, so we report both theoretical and wall-clock speedup to show that SPD is stronger on the ideal upper bound and on real hardware. Let N be the number of generated tokens and K the number of executed decoding steps (pipeline forwards for SPD; draft–verify rounds for EAGLE-3). Rejections and flushes imply $K > N$ in general; we write $\alpha{=}N/K$ for the *step acceptance ratio* (fraction of steps that ultimately commit a token).

Ignoring draft cost, one SPD cycle advances the equivalent of n parallel stage tokens at $1/n$ of a full forward, yielding the *oracle* (zero-draft-cost) theoretical speedup

$$\mathcal{S}^0_{\mathrm{spd}} = \frac{N}{K \cdot (1/n)} = \alpha \cdot n, \tag{6}$$

which coincides with the usual acceptance length when drafting is free. When draft latency is *not* hidden, we apply a *draft-cost-adjusted* factor for EAGLE-3 (serial k-token drafting before verify) and PPSD (draft after each pipeline step):

$$\begin{aligned} \mathcal{S}_{\mathrm{eagle}} &= \mathcal{S}^0_{\mathrm{eagle}} \cdot \frac{L}{L_s \cdot k + L}, \\ \mathcal{S}_{\mathrm{ppsd}} &= \mathcal{S}^0_{\mathrm{ppsd}} \cdot \frac{L}{L_s \cdot n + L}. \end{aligned} \tag{7}$$

Under the practical masking rule, SPD sets $\mathcal{S}_{\mathrm{spd}} = \mathcal{S}^0_{\mathrm{spd}}$. Wall-clock speedup then asks whether these idealizations survive real hardware: we measure decode-phase tokens/s against the same single-GPU autoregressive baseline on the H20 cluster above (prefill excluded; batch size 1), and report the ratio as wall-clock speedup.

4.3 Main Results and Analysis

Table 2 presents the comprehensive comparison of theoretical and wall-clock speedups across Qwen3.5-4B and Qwen3.5-9B under both greedy decoding ($T{=}0$) and stochastic sampling ($T{=}1$).

Behavior of Multi-Token Drafting Baselines (EAGLE-3). As a representative mainstream speculative decoding approach, EAGLE-3 exhibits stable acceleration across various workloads at moderate draft lengths. Under $k{=}3$ and $k{=}7$, it reliably achieves $1.88\times$–$1.97\times$ wall-clock speedup on Qwen3.5-4B and $2.24\times$–$2.54\times$ on Qwen3.5-9B at $T{=}0$. However, when scaling the draft horizon

Table 2: Single-path drafting on Qwen3.5-4B/9B (L=32). Format: theoretical / wall-clock speedup. MT = MT-Bench; HE = HumanEval.

	Qwen3.5-4B				Qwen3.5-9B			
Method	**Overall**	**MT**	**GSM8K**	**HE**	**Overall**	**MT**	**GSM8K**	**HE**
Temperature T=0								
EAGLE-3 (k=3)	2.47 / 1.88	2.11 / 1.66	2.71 / 2.04	2.58 / 1.95	2.69 / 2.24	2.28 / 1.95	2.88 / 2.36	2.92 / 2.40
EAGLE-3 (k=7)	2.72 / 1.97	2.14 / 1.65	3.10 / 2.17	2.92 / 2.10	3.17 / 2.54	2.35 / 1.99	3.44 / **2.69**	3.71 / 2.94
EAGLE-3 (k=15)	2.39 / 1.62	1.81 / 1.30	2.73 / 1.72	2.63 / 1.83	2.88 / 2.18	2.01 / 1.62	3.13 / 2.22	3.49 / 2.69
PPSD (n=4, L_s=1)	1.54 / 1.28	1.41 / 1.17	1.52 / 1.26	1.70 / 1.41	1.90 / 1.53	1.57 / 1.28	1.85 / 1.49	2.27 / 1.82
PPSD (n=8, L_s=1)	1.68 / 1.25	1.38 / 1.04	1.60 / 1.20	2.05 / 1.52	1.96 / 1.44	1.51 / 1.13	1.82 / 1.34	2.55 / 1.85
PPSD (n=16, L_s=1)	1.40 / 0.89	1.18 / 0.75	1.36 / 0.86	1.65 / 1.04	1.56 / 0.98	1.23 / 0.78	1.48 / 0.92	1.98 / 1.23
Ours (n=4, L_s=4)	2.64 / 2.27	2.19 / 1.90	2.61 / 2.24	3.11 / 2.65	2.70 / 2.32	2.24 / 1.94	2.68 / 2.30	3.19 / 2.73
Ours (n=8, L_s=4)	3.51 / 2.41	2.63 / 1.84	3.32 / 2.30	4.59 / 3.10	3.62 / 2.44	2.69 / 1.85	3.48 / 2.35	4.70 / 3.10
Ours (n=8, L_s=3)	3.39 / **2.53**	2.55 / **1.93**	3.22 / **2.43**	4.40 / **3.23**	3.61 / **2.67**	2.65 / **2.01**	3.43 / 2.55	4.74 / **3.44**
Ours (n=8, L_s=2)	3.32 / 2.47	2.48 / 1.88	3.15 / 2.36	4.34 / 3.17	3.46 / 2.56	2.59 / 1.96	3.26 / 2.42	4.54 / 3.31
Ours (n=16, L_s=2)	3.70 / 1.43	2.59 / 1.04	3.35 / 1.32	5.17 / 1.94	3.90 / 1.49	2.71 / 1.08	3.52 / 1.37	5.48 / 2.02
Temperature T=1								
EAGLE-3 (k=3)	2.03 / 1.47	1.80 / 1.31	2.10 / 1.53	2.19 / 1.58	2.13 / 1.81	1.86 / 1.60	2.23 / 1.88	2.29 / 1.96
EAGLE-3 (k=7)	2.06 / 1.46	1.73 / 1.26	2.15 / 1.55	2.29 / 1.56	2.26 / 1.78	1.89 / 1.53	2.35 / 1.91	2.53 / 1.89
EAGLE-3 (k=15)	1.80 / 1.17	1.45 / 0.98	1.87 / 1.21	2.08 / 1.33	1.95 / 1.40	1.55 / 1.22	2.00 / 1.47	2.30 / 1.51
PPSD (n=4, L_s=1)	1.39 / 1.06	1.30 / 0.99	1.37 / 1.05	1.50 / 1.14	1.71 / 1.27	1.44 / 1.08	1.67 / 1.24	2.03 / 1.49
PPSD (n=8, L_s=1)	1.44 / 0.95	1.27 / 0.84	1.40 / 0.92	1.66 / 1.09	1.68 / 1.10	1.36 / 0.90	1.57 / 1.03	2.12 / 1.37
PPSD (n=16, L_s=1)	1.20 / 0.65	1.06 / 0.58	1.19 / 0.64	1.33 / 0.72	1.32 / 0.72	1.10 / 0.61	1.27 / 0.69	1.60 / 0.86
Ours (n=4, L_s=4)	2.48 / 1.95	2.09 / **1.67**	2.49 / 1.96	2.85 / 2.21	2.57 / 2.01	2.13 / 1.72	2.54 / 1.99	3.03 / 2.33
Ours (n=8, L_s=4)	3.13 / 1.84	2.41 / 1.45	3.02 / 1.80	3.96 / 2.28	3.34 / 1.93	2.52 / 1.51	3.18 / 1.86	4.31 / 2.43
Ours (n=8, L_s=3)	3.05 / **2.05**	2.40 / 1.66	2.91 / **1.99**	3.84 / **2.50**	3.33 / **2.21**	2.50 / **1.73**	3.16 / **2.13**	4.32 / **2.78**
Ours (n=8, L_s=2)	2.96 / 2.03	2.29 / 1.62	2.86 / **1.99**	3.73 / 2.47	3.16 / 2.15	2.36 / 1.68	3.01 / 2.07	4.11 / 2.69
Ours (n=16, L_s=2)	3.19 / 1.06	2.37 / 0.81	2.96 / 1.00	4.24 / 1.37	3.46 / 1.13	2.45 / 0.83	3.11 / 1.03	4.83 / 1.53

Table 3: Effect of PDM layer count L_s on step acceptance ratio $\alpha = N/K$ (Qwen3.5-4B, single-path drafting; averaged over MT-Bench, GSM8K, and HumanEval).

n	L_s	T=0	T=1
8	1	0.2977	0.2708
8	2	0.3589	0.3276
8	3	0.3721	0.3396
8	4	0.3799	0.3435
4	1	0.5352	0.5007
4	4	0.6119	0.5752

to k=15, both the theoretical and wall-clock speedups degrade substantially across all benchmarks (e.g., overall wall-clock speedup drops from $1.97\times$ to $1.62\times$ on 4B, and from $2.54\times$ to $2.18\times$ on 9B). This performance degradation stems from two compounding factors: (1) drafting 15 tokens sequentially creates substantial exposed drafting latency that stalls the target model, and (2) due to compounding feature drift, later tokens in an extended draft sequence suffer from severely diminished acceptance rates. Consequently, the marginal increase in accepted tokens cannot compensate for the heightened serial drafting overhead.

Performance of Shallow-Feature Pipeline Baseline (PPSD). PPSD remains the weakest baseline across all evaluated settings, rarely exceeding $1.5\times$ wall-clock speedup. That is because PPSD conditions drafting only on shallow first-stage hidden states, while draft quality further deteriorates as pipeline width n increases (making them shallower). Furthermore, executing the draft step serially after each pipeline cycle leaves drafting latency entirely exposed on the critical path.

Efficacy of Speculative Pipeline Decoding (SPD). In contrast, our proposed SPD consistently delivers superior performance across model sizes and temperatures. Under the balanced configuration (n=8, L_s=3), SPD achieves the highest overall wall-clock speedup on both models: reaching **2.53**$\times$ on Qwen3.5-4B and **2.67**$\times$ on Qwen3.5-9B at T=0. The advantage of SPD is even more pronounced on structured generation tasks such as HumanEval, where SPD reaches **3.23**$\times$ and **3.44**$\times$ wall-clock

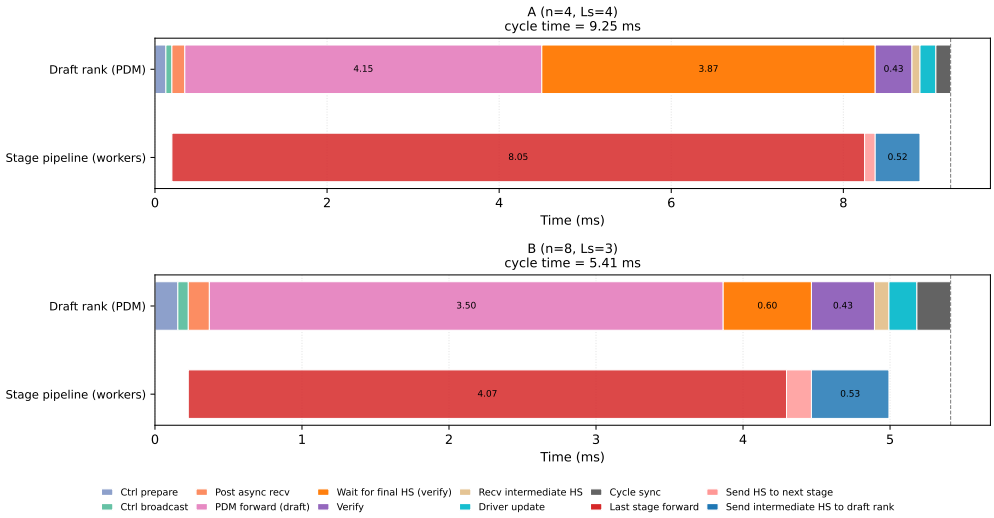


Figure 2: Steady-state (pipeline full) per-cycle wall-time breakdown on Qwen3.5-4B for (A) n=4, L_s=4 and (B) n=8, L_s=3. Drafting on the dedicated PDM GPU overlaps worker-side stage forward; fixed verification/transfer/control costs grow relatively as stages get shallower.

speedup on 4B and 9B models, respectively. Under stochastic sampling (T=1), SPD maintains robust acceleration (**2.05**× on 4B and **2.21**× on 9B), outperforming EAGLE-3 by a significant margin.

Balancing Drafter Capacity (L_s) and Latency Masking. Fixing the pipeline width to n=8, sweeping $L_s \in \{2, 3, 4\}$ highlights the fundamental trade-off between draft prediction accuracy and latency hiding. As detailed in Table 3, increasing L_s from 1 to 3 yields substantial gains in step acceptance ratio α (e.g., from 33.4% to 42.4% on 4B), which translates to higher theoretical speedup. However, as illustrated in the breakdown of Figure 2(B), setting L_s=3 pushes draft execution time (4.41 ms) very close to the worker stage forward time (4.72 ms). Further increasing drafter depth to L_s=4 exceeds the stage forward boundary, placing drafting back onto the critical path. As a result, while L_s=4 achieves slightly higher theoretical speedup (3.51× vs. 3.39×), its wall-clock speedup drops to 2.41×. Thus, our design rule $L_s \approx L/n - 1$ represents the practical sweet spot for maximizing acceptance while ensuring complete latency masking.

Pipeline Scaling vs. Per-Cycle Overhead. While theoretical speedup scales monotonically with wider pipelines (e.g., reaching 3.70× on 4B and 3.90× on 9B at n=16), wall-clock speedup peaks at n=8 and degrades at n=16. To identify the cause of this gap, we profile the running time of each action in SPD when the pipeline is full. As revealed by Figure 2, each pipeline cycle carries non-negligible fixed system overheads (e.g., inter-GPU tensor transfers, synchronization, and streaming verification). As n grows, stage computation decreases while fixed overheads remain constant, occupying an increasingly dominant portion of cycle time (∼12.9% at n=4 vs. ∼24.3% at n=8), leading to the increasing gap to the ideal speedup. However, for larger target models (e.g. Qwen3.5-9B), the longer per-stage computation will amortize these fixed costs more effectively, allowing measured speedups to align more closely with theoretical projections.

4.4 Draft Trees

EAGLE-3 can sample multiple candidates per step into a *draft tree* to raise acceptance; SPD admits the same branching. We find that tree drafting (width=4) clearly improves acceptance and theoretical speedup relative to single-path drafting (Appendix B). However, wall-clock speedup for draft trees is omitted because it requires complex system implementations such as branched KV caches, tree attention masks, and cross-stage synchronization, which we leave for future work.

Table 4: Qwen3.5-4B results when the PDM uses *post-step* features (i.e. serial drafting), compared to using *pre-step* features (our method). Format: **zero-draft-cost** theoretical speedup / **draft-cost-adjusted** theoretical speedup, averaged over MT-Bench, GSM8K, and HumanEval.

Setting	post-step		pre-step	
	T=0	T=1	T=0	T=1
$n = 4, L_s = 2$	2.78 / 2.22	2.65 / 2.12	2.34 / 2.34	2.18 / 2.18
$n = 8, L_s = 2$	3.64 / 2.43	3.28 / 2.19	3.32 / 3.32	2.96 / 2.96
$n = 16, L_s = 2$	4.07 / 2.03	3.47 / 1.74	3.70 / 3.70	3.19 / 3.19

4.5 Ablation: Pre-step vs. Post-step Features

The schedule in §3.3 drafts from shallower *pre-step* features so that drafting can start with the pipeline forward to mask its latency. Waiting for deeper *post-step* features can improve the draft's acceptance rate, thus raises the zero-draft-cost theoretical speedup (as shown in Table 4), especially at large n, but re-serializes drafting behind the pipeline step. Once that unhidden drafting latency is counted, the draft-cost-adjusted theoretical speedup collapses, e.g., to 2.43 at $n{=}8$ and 2.03 at $n{=}16$, much lower than 3.32 and 3.70 of the *pre-step* strategy, when $T = 0$. The ablation supports the design choice in §3.3: a modest acceptance drop is preferable to re-exposing draft latency.

5 Conclusion

We presented Speculative Pipeline Decoding, a shift from multi-token drafting to pipeline-parallel target execution. By predicting one token at a time from multi-depth target features, SPD bounds draft difficulty by pipeline width; by overlapping a pre-step PDM with each pipeline step, it hides draft latency. Distributed measurements show that moderate configurations ($n{=}8$, $L_s{\approx}L/n{-}1$) outperform EAGLE-3 in wall-clock speedup on Qwen3.5-4B and 9B, while overly large n is limited by fixed per-cycle overhead. We hope this perspective encourages a new paradigm of speculative decoding for more aggressive single-sequence speedups.

6 Limitations

Arithmetic intensity. Unlike conventional speculative decoding, which mainly raises per-device arithmetic intensity by verifying multiple candidates in one target forward, SPD accelerates a single sequence by using more GPUs in parallel and does not itself improve the compute-to-memory ratio on each device. However, the two approaches are largely orthogonal: combining SPD with intensity-raising drafters (e.g., EAGLE-style or diffusion methods) or with tensor parallelism is a promising route to further speedup.

Hardware footprint. SPD requires n stage ranks plus a dedicated PDM rank, so it uses more GPUs than single-device speculative decoding. This cost is most acceptable precisely where large models are already multi-GPU deployed; on such targets, longer stage forwards also amortize fixed per-cycle overhead better, aligning practical speedup more closely with theory. A related engineering direction is *non-uniform* stage partitioning, which co-locates the PDM with a shallower first stage to keep the rank count within a single 4- or 8-GPU node. We leave this as exploratory future work.

AI USE STATEMENT

In this work, we used generative AI tools to assist with method implementation and paper editing. Specifically, we used Composer 2.5 to assist in writing and editing the software code for our experimental implementation. Additionally, we used GPT-5.5 to polish and edit the manuscript text to improve readability. We have not used generative AI tools for other tasks with required disclosure, such as generating synthetic datasets, developing theoretical models, formulating hypotheses, or interpreting results.

All AI-assisted outputs were thoroughly examined and verified. All generated code was manually inspected, refactored, and tested for correctness and reproducibility by the authors. All edited text was reviewed to ensure scientific accuracy and proper attribution. The authors take full responsibility for the final content of this submission, including all text, claims, and code artifacts.

REFERENCES

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. Smollm2: When smol goes big – data-centric training of a small language model, 2025. URL https://arxiv.org/abs/2502.02737.

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.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, et al. 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, et al. Training verifiers to solve math word problems, 2021. *URL https://arxiv.org/abs/2110.14168*, 9, 2021.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023.

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*, 2026.

Tanishq Kumar, Tri Dao, and Avner May. Speculative speculative decoding. *arXiv preprint arXiv:2603.03251*, 2026.

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

Ruanjun Li, Ziheng Liu, Yuanming Shi, Jiawei Shao, Chi Zhang, and Xuelong Li. Pipeline parallelism is all you need for optimized early-exit based self-speculative decoding. *arXiv preprint arXiv:2509.19368*, 2025.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 28935–28948, 2024.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *Advances in Neural Information Processing Systems*, 38:136737–136756, 2026.

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

Haofei Yin, Mengbai Xiao, Tinghong Li, Xiao Zhang, Dongxiao Yu, and Guanghui Zhang. Specpipe: Accelerating pipeline parallelism-based llm inference with speculative decoding. *arXiv preprint arXiv:2504.04104*, 2025.

Yijiong Yu, Ziyun Dai, Zekun Wang, Wei Wang, Ran Chen, and Ji Pei. Opencsg chinese corpus: A series of high-quality chinese datasets for llm training, 2025. URL https://arxiv.org/abs/2501.08197.

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

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

A Training Alignment with Pre-step Pipeline Drafting

The main text defines multi-depth aggregation (Eqs. equation 3–equation 4) and the steady / warm-up feature layouts (Eqs. equation 2, equation 5). At inference the PDM always consumes *pre-step* features, so the newest in-flight token contributes only g^0, and the depth staircase depends on how many tokens currently occupy the pipeline. Training must therefore expose the same layout variation; this appendix describes the three mechanisms that achieve that alignment: sequence replication across aggregation types, randomized pipeline-fill simulation, and the structural attention mask. We keep the Qwen3.5 anchors fixed at $\mathcal{B}{=}[0, 8, 16, 24, 31]$ ($m{=}5$); exact choices matter little ($\sim$2% acceptance) as long as shallow, middle, and deep layers are represented, while dropping the embedding anchor ($b_0{=}0$) prevents stable convergence.

A.1 Sequence Replication for Multi-Depth Supervision

A single fused feature per token cannot represent coexisting aggregation types or the cross-depth visibility that the staircase requires. We therefore **replicate** every position into m contiguous blocks (deepest to shallowest), expanding sequence length from N to mN. The frozen target is run once per sequence; each block applies Eq. equation 4 with the corresponding anchor prefix. The LM head reads only the shallowest block (g^0) at each position, matching the pre-step inference schedule. Completed full-depth features g_i^n remain valid across steps and may be KV-cached; any g_i^k with $k{<}n$ is refreshed on the next advance, so caching appends only the trailing $(n{+}1)$ refreshed features each step.

A.2 Warm-up Layouts and Simulated Pipeline Fill

When the pipeline is saturated ($a{=}n$), the PDM sees the full depth staircase of Eq. equation 2. During warm-up, and briefly after a flush, only $a{<}n$ tokens are in flight, so the layout collapses to the shortened staircase of Eq. equation 5: the prefix uses depth $k{=}n$, and only the trailing a positions form a staircase. These two families of layouts are not interchangeable; a PDM trained only on the saturated pattern would be miscalibrated whenever the pipeline is partially filled. We therefore train under a *simulated* fill level that randomly reproduces both regimes. For each sequence, with probability 0.5 we set $\mathrm{fill}{=}n$ (saturated); otherwise fill is drawn uniformly from $\{1, \ldots, n{-}1\}$ (warm-up). The structural mask below then forces every attention pattern to obey the corresponding layout, so the PDM learns to draft under both depth staircases.

A.3 Structural Attention Mask

Every replicated block participates in attention; the mask alone encodes which cross-depth interactions are legal under the sampled fill. A query at time t and depth d attending to key T requires key

Table 5: Theoretical speedup with draft trees on Qwen3.5-4B/9B. MT = MT-Bench; HE = HumanEval.

	Qwen3.5-4B				Qwen3.5-9B			
Method	**Overall**	**MT**	**GSM8K**	**HE**	**Overall**	**MT**	**GSM8K**	**HE**
Temperature $T{=}0$								
EAGLE-3 ($k{=}3$)	2.75	2.15	3.14	2.97	3.16	2.82	3.36	3.30
EAGLE-3 ($k{=}7$)	3.61	**2.86**	**4.15**	3.81	**4.14**	**3.19**	**4.58**	4.66
EAGLE-3 ($k{=}15$)	3.33	2.55	3.80	3.66	4.07	2.86	4.37	4.97
PPSD ($n{=}4$, $L_s{=}1$)	1.54	1.41	1.56	1.70	1.80	1.54	1.88	2.22
PPSD ($n{=}8$, $L_s{=}1$)	1.49	1.28	1.53	1.83	1.66	1.36	1.71	2.27
PPSD ($n{=}16$, $L_s{=}1$)	1.18	1.04	1.22	1.38	1.28	1.08	1.31	1.64
Ours ($n{=}4$, $L_s{=}4$)	2.70	2.28	2.68	3.13	2.77	2.32	2.76	3.23
Ours ($n{=}8$, $L_s{=}4$)	3.59	2.69	3.45	4.64	3.72	2.77	3.60	4.80
Ours ($n{=}8$, $L_s{=}3$)	3.46	2.60	3.37	4.43	3.69	2.72	3.55	4.79
Ours ($n{=}8$, $L_s{=}2$)	3.40	2.54	3.26	4.40	3.55	2.65	3.38	4.62
Ours ($n{=}16$, $L_s{=}2$)	**3.75**	2.62	3.42	**5.22**	4.00	2.78	3.64	**5.58**
Temperature $T{=}1$								
EAGLE-3 ($k{=}3$)	2.63	2.38	2.76	2.75	2.63	2.39	2.68	2.83
EAGLE-3 ($k{=}7$)	2.79	2.32	2.84	3.20	2.99	2.47	3.15	3.34
EAGLE-3 ($k{=}15$)	2.44	2.01	2.52	2.78	2.71	2.22	2.79	3.12
PPSD ($n{=}4$, $L_s{=}1$)	1.62	1.50	1.65	1.73	1.89	1.63	1.97	2.22
PPSD ($n{=}8$, $L_s{=}1$)	1.67	1.44	1.74	1.97	1.88	1.55	1.93	2.45
PPSD ($n{=}16$, $L_s{=}1$)	1.35	1.16	1.41	1.56	1.47	1.22	1.52	1.84
Ours ($n{=}4$, $L_s{=}4$)	2.85	2.53	2.79	3.25	2.88	2.56	2.83	3.26
Ours ($n{=}8$, $L_s{=}4$)	3.93	3.22	3.69	4.89	4.08	3.28	3.88	5.10
Ours ($n{=}8$, $L_s{=}3$)	3.84	3.15	3.66	4.72	4.03	3.21	3.81	5.07
Ours ($n{=}8$, $L_s{=}2$)	3.74	3.06	3.53	4.64	3.88	3.14	3.65	4.84
Ours ($n{=}16$, $L_s{=}2$)	**4.30**	**3.38**	**3.80**	**5.73**	**4.53**	**3.48**	**3.96**	**6.16**

depth $d'{=}\min(n,\, d{+}t{-}T)$ in block $f(d')$. If $t{-}T$ exceeds fill, the key is redirected to the deepest block, matching the depth-maximized prefix of warm-up Eq. equation 5. Causal order is preserved ($T{\le}t$). Together with sequence replication, this mask makes training occupancy match the pre-step inference layouts without running an actual pipeline during training.

B Draft-Tree Theoretical Speedup

As noted in §4.4, SPD can expand each draft step into a width-4 draft tree. Table 5 reports the resulting **theoretical** speedup only (wall-clock for branched drafting is not yet implemented). Relative to single-path drafting in the main tables, trees raise acceptance and thus $\mathcal{S}$ for SPD (e.g., overall $3.05{\to}3.84$ on Qwen3.5-4B at $T{=}1$ for $n{=}8$, $L_s{=}3$). For EAGLE-3 and PPSD the numbers use the draft-cost-adjusted $\mathcal{S}$ of Eq. equation 7; for SPD, latency masking keeps $\mathcal{S}{=}\mathcal{S}^0$.

Under trees, EAGLE-3 often leads on greedy decoding for the larger model (overall and MT-Bench / GSM8K at $T{=}0$), while SPD is stronger under sampling and on HumanEval. PPSD remains weakest: trees do not fix shallow first-stage features and can hurt when incorrect branches crowd out shorter correct paths. These figures characterize draft quality under branching, not deployable wall-clock speedups.