LATENTSPEC: TWO-STAGE THINKER–DRAFTER PARALLEL DRAFTING WITH SHARED LATENT REASONING FOR SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates LLM inference by using a lightweight draft model to generate multiple candidate tokens, which are then verified in parallel by the target model. Recent parallel drafting methods reduce drafting latency by predicting an entire candidate block in a single forward pass. However, although existing methods allow information exchange across candidate positions, they typically lack an explicit shared block-level representation of the future, making it difficult to globally coordinate candidate-block generation and potentially limiting both the overall quality of the candidate block and the length of its accepted prefix. Inspired by recent advances in latent reasoning, we propose LatentSpec, a parallel drafting model with a two-stage Thinker–Drafter architecture. The Thinker first updates a small set of latent queries using intermediate features from the frozen target model, thereby constructing a shared latent reasoning state in a continuous hidden space. Conditioned on this state, the Drafter then coordinates and predicts the candidate block as a whole in parallel within a single forward pass. Experiments on multiple benchmarks with Qwen3 models show that, while preserving or improving average acceptance length relative to DFlash, LatentSpec reduces drafting FLOPs by 8.26–9.41% and increases end-to-end throughput by up to 9.33%.

1 INTRODUCTION

Large language models have achieved remarkable performance across a wide range of tasks (Yang et al., 2025). However, the sequential nature of autoregressive decoding has emerged as a primary bottleneck limiting inference efficiency (Stern et al., 2018; Xia et al., 2024). Speculative decoding mitigates this issue by employing a lightweight drafter to generate multiple candidate tokens at low cost, which are then verified by the target model in a single parallel forward pass (Leviathan et al., 2023; Chen et al., 2023). This reduces the number of target model invocations and accelerates inference while preserving the target model's output distribution under the appropriate verification rule.

The practical speedup of speculative decoding is primarily constrained by two factors: the acceptance length of draft tokens and the drafting latency of the draft model (Leviathan et al., 2023). Acceptance length determines how many tokens can be committed per verification round. The higher the acceptance length, the more tokens are produced by a single parallel verification of the target model, and the fewer decoding rounds are required. Conversely, if the draft model and the target model differ substantially in their predictions, the candidate sequence may be rejected at an early position, and the subsequent tokens usually cannot be accepted either, thereby wasting the computational effort of draft generation and target verification (Zhou et al., 2024). Therefore, even if a relatively long candidate sequence is drafted at once, a low acceptance length can still substantially diminish the speedup gains (Zhou et al., 2024; Wu et al., 2026). Drafting latency, in turn, determines the additional time required to generate candidate tokens. Increasing draft-model capacity may improve prediction quality but also increases drafting cost, creating an acceptance–latency trade-off (Zhang et al., 2026; Sadhukhan et al., 2025). In particular, autoregressive drafters still generate candidates token by token, and their sequential latency grows with draft length (Li et al., 2024a; 2025). Among recent methods, blockwise parallel drafters have gained increasing attention (Hui et al., 2026): DFlash predicts an entire candidate block in a single draft forward pass (Chen et al.,

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.

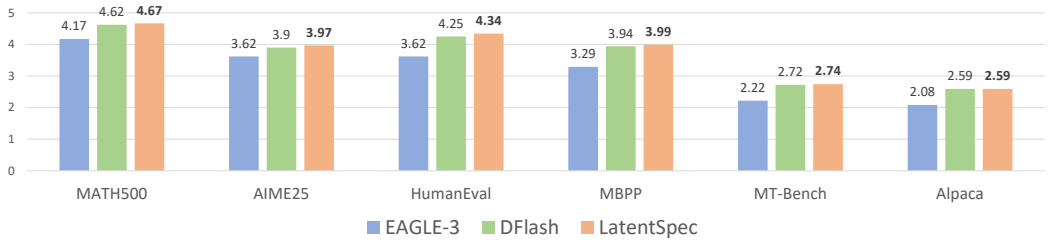

Figure 1: Average acceptance length of EAGLE-3, DFlash, and LatentSpec on Qwen3-4B under sampling ($T = 1$).

2026), while Domino and DSpark add lightweight intra-block dependency modeling (Huang et al., 2026; Cheng et al., 2026).

However, parallel candidate-block generation must balance efficient position-wise prediction against the need for coherent block-level conditioning. DFlash enables information exchange among masked draft positions (Chen et al., 2026), and follow-up work strengthens target-feature fusion or introduces prefix-dependent corrections (Zhang et al., 2026; Huang et al., 2026; Cheng et al., 2026); these approaches do not explicitly construct a shared latent workspace before predicting the block. Related masked-block training objectives instead focus on position weighting and train–inference mismatch (Wu et al., 2026). Chain-of-thought prompting shows that explicit intermediate steps can support complex reasoning (Wei et al., 2022), while latent-reasoning approaches explore computation in continuous hidden states instead of emitting a complete textual reasoning chain (Hao et al., 2024; Zhang et al., 2025). Moreover, chain-of-thought prompting can effectively raise a model's capability ceiling, whereas for a draft model in speculative decoding, the capability ceiling is the target model itself. This raises the question of whether endowing the draft model with latent-reasoning capabilities can raise its own ceiling and thereby bring it closer to the target model. Inspired by this observation, we investigate whether a compact shared latent representation can coordinate parallel predictions across a speculative block.

To this end, we propose LatentSpec, a two-stage Thinker–Drafter parallel draft model based on shared latent states. Specifically, the Thinker takes as input intermediate features of a frozen target model and iteratively updates a small set of latent queries, forming a round-local state for the candidate block. Subsequently, the Drafter conditions on both the cached target features and this shared state to predict the entire block in one forward pass. Unlike methods that add sequential processing of sampled draft tokens, LatentSpec retains parallel token prediction; compared with stacking full-block draft layers, its Thinker applies part of the computation to fewer latent positions. We evaluate acceptance length, draft cost, and end-to-end latency on Qwen3 models (Yang et al., 2025) to determine whether this allocation improves the quality–cost trade-off relative to DFlash. Our contributions are summarized as follows:

1. We identify that parallel drafting methods allow information interaction among candidate positions but do not explicitly build a shared block-level latent workspace before token prediction.
2. We propose LatentSpec, which introduces a shared latent state into parallel drafting through a two-stage Thinker–Drafter architecture: the Thinker constructs a block-level state in continuous hidden space, and the Drafter jointly predicts the entire candidate block in a single forward pass conditioned on this state.
3. We evaluate LatentSpec on Qwen3 across math, coding, and dialogue benchmarks. Compared with DFlash, LatentSpec reduces draft FLOPs and drafting latency while achieving higher average acceptance length in most benchmark–model–decoding settings and higher end-to-end throughput in most evaluated workloads.

2 Related Work

Speculative Decoding. Speculative decoding uses a lightweight drafter to propose several tokens and verifies them together with a target model; rejection sampling with residual correction preserves

the target distribution (Leviathan et al., 2023; Chen et al., 2023; Xia et al., 2024). Earlier blockwise decoding also exploited parallel verification of candidate continuations (Stern et al., 2018). SpecInfer verifies candidate trees, Medusa attaches multiple prediction heads, and EAGLE drafts from target features (Miao et al., 2024; Cai et al., 2024; Li et al., 2024a). EAGLE-2 uses context-dependent draft trees, while EAGLE-3 improves feature-conditioned proposals with training-time test (Li et al., 2024b; 2025). Cached target keys and values or distillation can further improve proposal quality (Du et al., 2024; Zhou et al., 2024); retrieval-based drafting offers a model-free alternative (He et al., 2024). ConFu conditions the drafter on future-oriented contemplate tokens, but this increases verification overhead since each draft node requires an extra token during target-model verification (Qin et al., 2026).These methods offer different quality–cost trade-offs, but extending a neural draft autoregressively still incurs sequential draft steps.

Parallel Block Drafting. Parallel drafters predict multiple future positions without feeding previously sampled draft tokens back into the model. Medusa uses independent prediction heads, while multi-token prediction provides related future-position supervision (Cai et al., 2024; Gloeckle et al., 2024). P-EAGLE and PARD develop parallel draft architectures; DART constructs candidate trees from diffusion-inspired parallel proposals (Hui et al., 2026; An et al., 2026; Liu et al., 2026). DFlash conditions masked draft positions on cached target features and predicts an entire block in one pass (Chen et al., 2026). DFlare strengthens target-feature fusion, Domino adds a prefix-dependent causal head, and DSpark couples lightweight intra-block dependencies with confidence-scheduled verification (Zhang et al., 2026; Huang et al., 2026; Cheng et al., 2026). Parallel-generation research highlights the difficulty of modeling dependencies between simultaneously predicted positions (Gu et al., 2018; Huang et al., 2022; Arriola et al., 2025). D-PACE instead changes the position weights used to train parallel drafters (Wu et al., 2026).

Latent Reasoning. Chain-of-thought prompting generates explicit intermediate steps (Wei et al., 2022), which can increase generation latency. Latent-reasoning methods allocate computation to hidden or continuous states: pause tokens allow additional computation before output (Goyal et al., 2024); Coconut feeds hidden states back as continuous thoughts (Hao et al., 2024); and Soft Thinking forms continuous concepts from probability-weighted token embeddings (Zhang et al., 2025). PonderLM-2 generates an intermediate continuous hidden state before each output token, learning latent thoughts from general-domain pretraining to refine token prediction (Zeng et al., 2025). Unlike these methods, we do not apply supervised fine-tuning to the draft model using explicit CoT data; instead, the draft model acquires latent reasoning ability through end-to-end training.

3 PRELIMINARIES

We introduce the necessary preliminaries. We first describe the inference procedure and notation of speculative decoding (Leviathan et al., 2023; Chen et al., 2023), then review recent approaches to parallel block drafting, and finally discuss evaluation metrics for speculative decoding.

3.1 SPECULATIVE DECODING

Let y_t be the latest committed token (the *anchor*), $\mathbf{y}_{1:t}$ the committed prefix, and $\mathcal{T}$ the autoregressive target model. A lightweight drafter proposes K tokens $\hat{\mathbf{y}}_{t+1:t+K}$ with distributions q_i for offsets $i = 1, \ldots, K$. A single causal target pass evaluates each proposal against the appropriate proposed prefix:

$$p_i(v) = p_{\mathcal{T}}(v \mid \mathbf{y}_{1:t}, \hat{\mathbf{y}}_{t+1:t+i-1}), \quad i = 1, \ldots, K+1. \tag{1}$$

Here p_{K+1} is evaluated after all K proposals. Under greedy decoding, the verifier commits the longest prefix for which $\hat{y}_{t+i} = \arg\max_v p_i(v)$ at every accepted position. Under sampling, a proposal $\hat{y}_{t+i} \sim q_i$ is accepted, conditional on reaching position i, with probability

$$\alpha_i = \min\left(1, \frac{p_i(\hat{y}_{t+i})}{q_i(\hat{y}_{t+i})}\right). \tag{2}$$

At the first rejection, the verifier draws a correction from the normalized positive residual $[p_i - q_i]_+$; if all proposals are accepted, it draws a bonus token from p_{K+1}. Greedy matching preserves the target's greedy output, and rejection sampling with residual correction preserves its sampling distribution (Leviathan et al., 2023; Chen et al., 2023). In both cases, only the accepted prefix and the correction or bonus token are committed.

3.2 PARALLEL BLOCK DRAFTING

DFlash (Chen et al., 2026) predicts an entire block in one draft-model forward pass rather than rolling out its proposals autoregressively; it builds on the broader idea of parallel masked and block-wise prediction (Ghazvininejad et al., 2019; Arriola et al., 2025). Given selected target layers $\mathcal{L} = \{\ell_1, \ldots, \ell_m\}$, it fuses their cached features at positions preceding the anchor into a context memory:

$$\mathbf{C}_{<t} = \mathrm{RMSNorm}\Big(\mathbf{W}_c \left[\mathbf{H}_{<t}^{(\ell_1)}; \ldots; \mathbf{H}_{<t}^{(\ell_m)}\right]\Big). \tag{3}$$

The brackets concatenate target-layer features along the channel dimension; $\mathbf{W}_c$ projects them to the drafter width. The anchor y_t is supplied separately as the first draft input, and the remaining $K-1$ inputs are mask embeddings. Draft positions attend bidirectionally within the block and read $\mathbf{C}_{<t}$ as persistent Key–Value memory, yielding $q_1, \ldots, q_K$ simultaneously. These are position-wise proposal distributions generated in a single parallel pass, rather than an autoregressive rollout. Each sampled token is verified against the proposed prefix to its left.

3.3 ACCEPTANCE LENGTH AND LATENCY OBJECTIVE

Let A be the number of consecutively accepted draft tokens. Apart from termination at the end of generation, a round commits $\tau_r = A + 1$ tokens, including its bonus token; the average acceptance length is $\tau = \mathbb{E}[\tau_r]$. For mean per-round draft and verification costs T_{draft} and T_{verify}, respectively, the amortized latency is approximately $(T_{\mathrm{draft}} + T_{\mathrm{verify}})/\tau$ per committed token. Thus, higher acceptance alone does not guarantee greater speedup if the drafter becomes comparably slower (Sadhukhan et al., 2025); batching and serving conditions can further change the throughput trade-off (Kwon et al., 2023).

4 LATENTSPEC

4.1 DESIGN MOTIVATION AND OVERVIEW

In block-parallel drafting (Chen et al., 2026; An et al., 2026), candidate positions can exchange information through bidirectional attention, but none observes the tokens actually sampled at earlier positions. This dependency challenge is also studied in non-autoregressive generation (Huang et al., 2022). We therefore seek a shared representation that coordinates the block *before* candidate selection, without introducing token-by-token draft execution. This is distinct from conditioning on a realized draft prefix: a shared state can convey context about the future block, but cannot encode the as-yet-unknown sampled tokens.

LatentSpec implements this idea in two sequential stages. In a speculative round, let y_t be the latest committed token and $\mathbf{C}_{<t} \in \mathbb{R}^{(t-1)\times d}$ the cached target features preceding it (Equation 3). Stage 1, the *Thinker* $\mathcal{G}_\phi$, reads $\mathbf{C}_{<t}$ and updates M latent slots to form a round-local shared state $\mathbf{S}_t$. Stage 2, the *Drafter* $\mathcal{D}_\theta$, reads both $\mathbf{C}_{<t}$ and $\mathbf{S}_t$ to predict K future-token distributions in parallel. The stages run in order, but neither iterates over sampled draft tokens. The target model, its copied token embedding $\mathbf{E}$, and its copied language-modeling head $\mathbf{W}_{\mathrm{lm}}$ remain frozen. Figure 2 summarizes the pipeline; $\mathbf{S}_t$ is neither a textual reasoning trace nor part of the committed target sequence.

4.2 STAGE 1: THINKER

Latent-slot initialization. Motivated by computation in continuous latent states (Hao et al., 2024; Goyal et al., 2024), the first slot represents the anchor; the other $M-1$ slots start from a dedicated latent-token embedding:

$$\mathbf{U}^{(0)} = \left[\mathbf{E}(y_t); \underbrace{\mathbf{E}(\langle\mathrm{latent}\rangle); \ldots; \mathbf{E}(\langle\mathrm{latent}\rangle)}_{M-1 \text{ slots}}\right] \in \mathbb{R}^{M\times d}. \tag{4}$$

Distinct rotary positions $t, \ldots, t+M-1$ distinguish the slots despite their identical non-anchor embeddings. At each layer, the slots supply the queries, while the cached target context and the slots

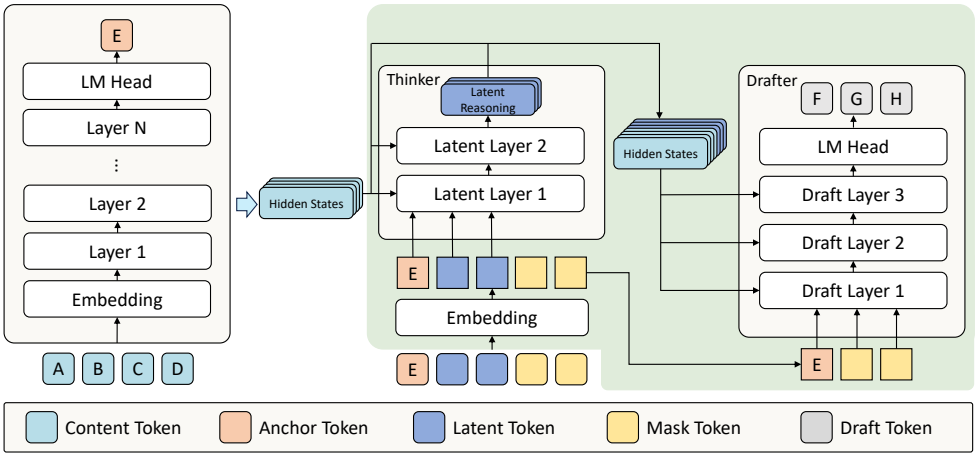

Figure 2: LatentSpec's two-stage drafting pipeline. Stage 1 updates an anchor slot and $M-1$ latent slots from target hidden states to generate a latent reasoning state. Stage 2 then has the Drafter generate drafts additionally conditioned on this latent reasoning state, where all K draft positions are conditioned on the resulting shared state and their token distributions are predicted in parallel using the frozen target LM head.

have separate key–value projections:

$$\begin{aligned}\mathbf{Q}_{\mathrm{T}}^{(\ell)} &= \mathbf{U}^{(\ell-1)}\mathbf{W}_{Q,\mathrm{T}}^{(\ell)},\\ \mathbf{K}_{\mathrm{T}}^{(\ell)} &= \left[\mathbf{C}_{<t}\mathbf{W}_{K,c}^{(\ell)}; \mathbf{U}^{(\ell-1)}\mathbf{W}_{K,u}^{(\ell)}\right],\\ \mathbf{V}_{\mathrm{T}}^{(\ell)} &= \left[\mathbf{C}_{<t}\mathbf{W}_{V,c}^{(\ell)}; \mathbf{U}^{(\ell-1)}\mathbf{W}_{V,u}^{(\ell)}\right].\end{aligned} \tag{5}$$

Here and below, brackets concatenate sequences along the attention key–value axis, unlike the channel-wise feature concatenation in Equation 3. Each slot attends to the cached prefix and bidirectionally to the other slots. A pre-normalized Transformer block with attention, an MLP, and residual connections updates only the slots:

$$\mathbf{U}^{(\ell)} = \mathrm{ThinkerLayer}^{(\ell)}\left(\mathbf{U}^{(\ell-1)}, \mathbf{C}_{<t}\right), \quad \ell = 1, \ldots, L_{\mathrm{T}}. \tag{6}$$

After L_{T} layers, the shared memory is $\mathbf{S}_t = \mathrm{RMSNorm}(\mathbf{U}^{(L_{\mathrm{T}})}) \in \mathbb{R}^{M\times d}$. It is computed once per round, before any draft token is selected, and is available to every candidate position in Stage 2. No explicit target is imposed on its contents.

4.3 Stage 2: Drafter

The Drafter predicts K future-token distributions from K input positions. As in the shifted alignment of DFlash (Chen et al., 2026), the first input is the anchor embedding and the remaining $K-1$ inputs are mask embeddings:

$$\mathbf{X}^{(0)} = \left[\mathbf{E}(y_t); \underbrace{\mathbf{E}(\langle\mathrm{mask}\rangle); \ldots; \mathbf{E}(\langle\mathrm{mask}\rangle)}_{K-1 \text{ positions}}\right] \in \mathbb{R}^{K\times d}. \tag{7}$$

These inputs use rotary positions $t, \ldots, t+K-1$. The output at input position $t+i-1$ predicts the token at $t+i$; thus y_t is a conditioning input, not one of the K proposals.

At each draft layer, the K draft states query three sources: the cached target context $\mathbf{C}_{<t}$, the same shared memory $\mathbf{S}_t$ for all positions, and the current block states. Each source has separate key–value

projections:

$$\begin{aligned}\mathbf{Q}_{\mathrm{D}}^{(\ell)} &= \mathbf{X}^{(\ell-1)}\mathbf{W}_{Q,\mathrm{D}}^{(\ell)},\\ \mathbf{K}_{\mathrm{D}}^{(\ell)} &= \left[\mathbf{C}_{<t}\mathbf{W}_{K,c}^{(\ell)}; \mathbf{S}_t\mathbf{W}_{K,s}^{(\ell)}; \mathbf{X}^{(\ell-1)}\mathbf{W}_{K,x}^{(\ell)}\right],\\ \mathbf{V}_{\mathrm{D}}^{(\ell)} &= \left[\mathbf{C}_{<t}\mathbf{W}_{V,c}^{(\ell)}; \mathbf{S}_t\mathbf{W}_{V,s}^{(\ell)}; \mathbf{X}^{(\ell-1)}\mathbf{W}_{V,x}^{(\ell)}\right].\end{aligned} \tag{8}$$

Only $\mathbf{X}$ is updated across the L_{D} pre-normalized Transformer layers; target features and latent memory remain fixed during this stage:

$$\mathbf{X}^{(\ell)} = \mathrm{DraftLayer}^{(\ell)}\left(\mathbf{X}^{(\ell-1)}, \mathbf{C}_{<t}, \mathbf{S}_t\right), \quad \ell = 1, \ldots, L_{\mathrm{D}}. \tag{9}$$

Bidirectional block attention lets draft positions exchange information within this stage. The frozen target LM head yields the position-wise proposal distributions

$$q_i(\cdot) = \mathrm{softmax}\left(\mathbf{W}_{\mathrm{lm}}\,\mathrm{RMSNorm}(\mathbf{x}_{i-1}^{(L_{\mathrm{D}})})\right), \quad i = 1, \ldots, K. \tag{10}$$

Each proposal $\hat{y}_{t+i}$ is selected or sampled from q_i without feeding previously sampled proposals back into the Drafter. Thus, positions interact through hidden states and shared memory, not through realized token-level causal conditioning. The Thinker and Drafter require two sequential stages, but all K proposal distributions are obtained in one Drafter pass.

Training objective. We train both stages on target-generated trajectories with a frozen target model. Because verification stops at the first rejection, we weight earlier draft positions more heavily (Wu et al., 2026) and combine position-weighted next-token cross-entropy with the ℓ_1 distance between draft and aligned target distributions (Zhou et al., 2024). The loss weights are 0.1 and 0.9, respectively; gradients reach the Thinker through the Drafter without directly supervising the latent state. Appendix A.1 details the packed training masks, alignment, and full objective.

4.4 Computation Allocation and Efficiency

LatentSpec applies L_{T} layers to M latent slots and L_{D} layers to all K draft positions. By default, $M = 4$, $K = 7$, $L_{\mathrm{T}} = 2$, and $L_{\mathrm{D}} = 3$. Relative to a five-layer full-block drafter, two layers operate on four rather than seven query positions. This saves position-dependent work, but the Thinker must finish before the Drafter starts; separate projections and the stage boundary also incur costs. Consequently, a reduction in draft FLOPs does not by itself establish a reduction in wall-clock latency, and neither metric alone establishes an end-to-end speedup (Sadhukhan et al., 2025; Zheng et al., 2024). We measure these quantities separately in Section 5.

Draft–verify inference. After target prefill and the first anchor token, the Thinker builds $\mathbf{S}_t$ once and the Drafter computes the K proposal distributions in parallel. The target verifies the anchor and proposed block in one causal pass, using greedy matching or the sampling and residual-correction rule in Section 3. Only the accepted draft prefix and the correction or bonus token are committed. Verified target features for newly committed positions enter the next round's context cache, while features beyond the accepted boundary are discarded. The correction or bonus token becomes the next anchor, for which a new latent state is computed. The two-stage drafter therefore adds no target-model pass and leaves the standard lossless verification rule unchanged (Leviathan et al., 2023; Chen et al., 2023).

5 Experiments

5.1 Experimental Setup

Models and training. We evaluate LatentSpec with frozen Qwen3-4B and Qwen3-8B target models (Yang et al., 2025). Our acceptance-length baselines are EAGLE-3 (Li et al., 2025) and DFlash (Chen et al., 2026); the end-to-end throughput comparison additionally includes target-only decoding. On Qwen3-4B, we compare against both three-layer (DFlash-3L) and five-layer (DFlash-5L) full-block drafters. LatentSpec instead allocates two layers to its Thinker over $M = 4$ latent slots and three layers to its Drafter over $K = 7$ positions. Thus, it matches DFlash-5L in total Transformer depth. We train all compared speculative decoding methods on the same target-regenerated

Table 1: Draft latency (ms) across eight context lengths under matched CUDA Graph conditions.

Context length	DFlash (ms)	LatentSpec (ms)	Reduction
128	0.8487	**0.8393**	1.10%
256	1.0783	**1.0048**	6.82%
512	1.3904	**1.3264**	4.60%
1024	1.9769	**1.9227**	2.74%
2048	3.2384	**3.1647**	2.28%
4096	5.5372	**5.2439**	5.30%
8192	9.7297	**9.4893**	2.47%
16384	18.4186	**17.9280**	2.66%

Open-PerfectBlend dataset with cached target features (Xu et al., 2024); further training and checkpoint details appear in Appendix A.2.

Evaluation protocol. We measure average acceptance length τ on six benchmarks spanning mathematics, code, and dialogue (Table 2), under greedy decoding ($T = 0$) and sampling ($T = 1$). The benchmark families include AIME25 and MATH-500 for mathematics (Zhang & Math-AI, 2025; Hendrycks et al., 2021), HumanEval and MBPP for code (Chen et al., 2021; Austin et al., 2021), MT-Bench and Alpaca for dialogue and instruction following (Zheng et al., 2023; Taori et al., 2023); these are the six datasets listed in Table 2. The metric counts the tokens committed per verification round, including a correction or bonus token when applicable. We also report end-to-end aggregate throughput in generated tokens/s at different request concurrencies (Table 3), since batch size and scheduling affect speculative serving efficiency (Zheng et al., 2024). Within each comparison, methods share the target model and evaluation prompts. Appendix A.2 specifies the sampling and timing protocols, including the different generation limits used for acceptance and throughput measurements.

5.2 Main Results

Drafting cost. Table 1 measures draft-backbone latency under matched CUDA Graph conditions; we distinguish this microbenchmark from end-to-end throughput, because acceptance gains alone need not translate to end-to-end speedups (Sadhukhan et al., 2025). The Thinker architecture applies two layers to four latent slots rather than all seven draft positions, reducing position-dependent computation. Correspondingly, LatentSpec is faster than DFlash-5L at all eight tested context lengths (128–16,384 tokens), with latency reductions ranging from 1.10% to 6.82%. A detailed analytical comparison of draft-proposal FLOPs is provided in Appendix A.2.

Acceptance length. Table 2 compares draft quality across target sizes and decoding modes. On Qwen3-4B, LatentSpec achieves an average τ of 4.15 under greedy decoding and 3.72 under sampling, compared with 4.12 and 3.67 for DFlash-5L. Under greedy decoding on Qwen3-4B, it also attains the highest acceptance length on each of the six individual benchmarks. On Qwen3-8B, the corresponding averages are 4.17 versus 4.16 at $T = 0$ and 3.70 versus 3.71 at $T = 1$. Taken together with the cost results above, these measurements show that the two-stage LatentSpec architecture achieves acceptance lengths comparable to or better than DFlash-5L while requiring less computation during both training and inference. They further suggest that the Thinker learns useful latent reasoning states that improve the acceptance–computation trade-off.

End-to-end throughput. Table 3 reports aggregate throughput on AIME25 and MATH-500 at five request concurrencies. LatentSpec exceeds DFlash in 18 of the 20 model–dataset–concurrency comparisons. At concurrency 16, its throughput improves over DFlash by 9.19% and 2.23% on Qwen3-4B AIME25 and MATH-500, respectively (3893.26 versus 3565.71 and 4956.51 versus 4848.30 tokens/s), and by 9.33% and 4.40% on the corresponding Qwen3-8B datasets. The exceptions are Qwen3-4B on AIME25 at concurrency 8 (0.05% lower; 3185.79 versus 3187.47 tokens/s) and on MATH-500 at concurrency 2 (1.23% lower; 1717.68 versus 1738.99 tokens/s). Together, these experiments show that lower drafting cost, combined with comparable or higher average acceptance length, gives LatentSpec an end-to-end throughput advantage in most evaluated settings.

Table 2: Average acceptance length τ for Qwen3-4B and Qwen3-8B under greedy decoding ($T = 0$) and sampling ($T = 1$). The six benchmarks cover mathematics, coding, and dialogue.

		Math		Code		Chat		Overall
Target Model	**Draft Model**	**MATH-500**	**AIME25**	**HumanEval**	**MBPP**	**MT-Bench**	**Alpaca**	**Avg.**
Temperature $T = 0$ (greedy)								
Qwen3-4B	EAGLE-3	4.99	4.43	4.27	3.89	2.60	2.41	3.77
	DFlash-3L	5.15	4.51	4.36	4.12	2.85	2.65	3.94
	DFlash-5L	5.36	4.73	4.59	4.33	2.96	2.77	4.12
	LatentSpec	**5.38**	**4.77**	**4.62**	**4.35**	**2.99**	**2.78**	**4.15**
Qwen3-8B	EAGLE-3	5.30	4.72	4.53	4.22	2.90	2.72	4.07
	DFlash-5L	**5.44**	**4.81**	4.63	4.32	2.97	2.79	4.16
	LatentSpec	5.43	**4.81**	**4.64**	**4.34**	**2.98**	**2.80**	**4.17**
Temperature $T = 1$ (sampling)								
Qwen3-4B	EAGLE-3	4.17	3.62	3.62	3.29	2.22	2.08	3.17
	DFlash-3L	4.50	3.75	4.03	3.77	2.64	2.46	3.53
	DFlash-5L	4.62	3.90	4.25	3.94	2.72	**2.59**	3.67
	LatentSpec	**4.67**	**3.97**	**4.34**	**3.99**	**2.74**	**2.59**	**3.72**
Qwen3-8B	EAGLE-3	4.54	3.81	4.04	3.62	2.52	2.40	3.49
	DFlash-5L	4.67	**3.93**	**4.28**	**4.02**	**2.75**	2.60	**3.71**
	LatentSpec	**4.70**	3.87	4.26	**4.02**	**2.75**	**2.62**	3.70

Table 3: End-to-end aggregate throughput (tokens/s, including first-token latency) on AIME25 and MATH-500 at different request concurrencies.

Target	**Dataset**	**Method**	**Concurrency**				
			1	**2**	**4**	**8**	**16**
Qwen3-4B	AIME25	Baseline	273.32	519.23	931.16	1688.78	2969.99
		EAGLE-3	509.86	981.74	1684.54	2532.98	3112.40
		DFlash	730.86	1370.14	2254.01	**3187.47**	3565.71
		LatentSpec	**761.13**	**1376.77**	**2418.29**	3185.79	**3893.26**
	MATH-500	Baseline	275.18	524.57	1000.36	1844.82	3288.74
		EAGLE-3	652.39	1247.25	2259.70	3293.03	4249.76
		DFlash	919.65	**1738.99**	3041.86	4143.49	4848.30
		LatentSpec	**920.46**	1717.68	**3050.88**	**4167.28**	**4956.51**
Qwen3-8B	AIME25	Baseline	180.23	345.60	625.27	1164.86	2051.16
		EAGLE-3	373.76	668.49	1167.63	1648.85	2087.10
		DFlash	500.06	918.32	1544.16	2080.53	2223.86
		LatentSpec	**522.05**	**955.40**	**1556.89**	**2193.99**	**2431.45**
	MATH-500	Baseline	181.44	348.99	674.36	1271.20	2279.78
		EAGLE-3	454.16	858.27	1563.01	2300.45	2799.03
		DFlash	623.31	1168.52	2021.03	2732.72	3070.79
		LatentSpec	**639.44**	**1194.42**	**2085.50**	**2846.54**	**3205.84**

5.3 Ablation Study

To examine the benefit of providing shared latent reasoning to the Drafter, we compare LatentSpec with DFlash-3L on Qwen3-4B in Table 2. Both use three layers to process the draft positions, while LatentSpec additionally runs a two-layer Thinker on four latent slots and supplies its output to each Drafter layer. Under greedy decoding, LatentSpec improves average acceptance length from 3.94 to 4.15; under sampling, it improves from 3.53 to 3.71. The gains hold on all six benchmarks at both temperatures, suggesting that the Thinker's shared latent reasoning output provides an informative conditioning signal for parallel draft prediction.

Table 4: Average acceptance length τ on Qwen3-4B for DSpark and DSpark+LatentSpec under greedy decoding ($T = 0$) and sampling ($T = 1$).

	Math		Code		Chat		Overall
Drafter	**MATH-500**	**AIME25**	**HumanEval**	**MBPP**	**MT-Bench**	**Alpaca**	**Avg.**
Temperature $T = 0$ (greedy)							
DSpark	**5.62**	**5.01**	**4.85**	**4.60**	**3.15**	**2.95**	**4.36**
DSpark+LatentSpec	**5.62**	**5.01**	4.81	**4.60**	3.14	**2.95**	4.35
Temperature $T = 1$ (sampling)							
DSpark	**5.02**	4.24	4.59	4.32	2.94	2.78	3.98
DSpark+LatentSpec	5.01	**4.32**	**4.60**	**4.36**	**2.95**	**2.81**	**4.01**

5.4 Compatibility with DSpark

We further evaluated LatentSpec in combination with DSpark (Cheng et al., 2026) on Qwen3-4B (Table 4). Under greedy decoding, the average acceptance length remained nearly unchanged, decreasing from 4.36 to 4.35. At the reported precision, four of the six benchmark results were unchanged, while HumanEval and MT-Bench decreased by 0.04 and 0.01, respectively. Under sampling, the average acceptance length increased from 3.98 to 4.01, with improvements on five of the six benchmarks. These results show that LatentSpec maintains comparable draft quality when combined with DSpark and provides a small improvement under sampling in this setting. Since DSpark's backbone is based on DFlash, combining LatentSpec with DSpark similarly reduces the computational cost required for both training and inference, while maintaining comparable acceptance length on the benchmarks. However, the improvement is modest and does not persist under greedy decoding. For completeness and transparency, we report these results. How best to combine the parallel construction of latent-reasoning hidden states with autoregressive correction mechanisms remains an open question for further study.

6 Conclusion

We propose LatentSpec, a draft model based on a two-stage Thinker-Drafter architecture. To the best of our knowledge, this work is the first to introduce latent reasoning into draft model training for speculative decoding. It achieves this without introducing additional autoregressive computation, while maintaining a fully parallel, single-pass forward computation for drafting. Experiments across multiple benchmarks with Qwen3 models show that LatentSpec achieves higher average acceptance lengths than DFlash on most datasets. It also reduces drafting FLOPs by 8.26–9.41% on Qwen3-4B and improves end-to-end throughput in 18 of the 20 evaluated model–dataset–concurrency settings. These results suggest that introducing Latent Reasoning into the draft model to construct latent reasoning states, which serve as conditioning for the subsequent Drafter, can effectively coordinate the generation direction of the entire candidate block, thereby improving the quality of draft tokens and increasing the acceptance rate. How to more effectively combine LatentSpec's parallel latent-reasoning-state construction with autoregressive correction mechanisms, remains an open direction for future work.

AI use statement

During the conduct of this study and the writing of the manuscript, we used generative AI tools solely as an auxiliary means to identify and correct errors in grammar, expression, and content, and to assist in checking for potential plagiarism or textual similarity risks. For all work involving or assisted by AI, we have conducted human review, verification, and necessary revisions to ensure its accuracy, originality, and academic integrity. We take full responsibility for the final content of this study, including the text, viewpoints, claims, analyses, interpretations, and outputs generated with the assistance of generative AI.

Reproducibility statement

All results in this work are fully reproducible. The code is available at the following anonymous link: `https://anonymous.4open.science/r/LatentSpec`, where the README.md file provides detailed instructions for reproduction. The datasets and evaluation protocols used in our experiments are described in detail in the Appendix.

References

Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. PARD: Accelerating llm inference with low-cost parallel draft model adaptation. In *International Conference on Learning Representations*, 2026.

Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, volume 2025, pp. 50726–50753, 2025.

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

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. *arXiv preprint arXiv:2602.06036*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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. *arXiv preprint arXiv:2607.05147*, 2026.

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

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 6112–6121, 2019.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations*, volume 2024, pp. 27896–27923, 2024.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=B1l8BtlCb`.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2024. URL https://aclanthology.org/2024.naacl-long.88/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Fei Huang, Tianhua Tao, Hao Zhou, Lei Li, and Minlie Huang. On the learning of non-autoregressive transformers. In *International conference on machine learning*, pp. 9356–9376, 2022.

Jianuo Huang, Yaojie Zhang, Qituan Zhang, Hao Lin, Hanlin Xu, and Linfeng Zhang. Domino: Decoupling causal modeling from autoregressive drafting in speculative decoding. *arXiv preprint arXiv:2605.29707*, 2026.

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

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

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.

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*, ICML'24, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 7421–7432, 2024b.

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

Fuliang Liu, Xue Li, Ketai Zhao, Yinxi Gao, Ziyan Zhou, Zhonghui Zhang, Zhibin Wang, Wanchun Dou, Sheng Zhong, and Chen Tian. Dart: Diffusion-inspired speculative decoding for fast llm inference. *arXiv preprint arXiv:2601.19278*, 2026.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949, 2024.

Zongyue Qin, Raghavv Goel, Mukul Gagrani, Risheek Garrepalli, Mingu Lee, and Yizhou Sun. Confu: Contemplate the future for better speculative sampling. *arXiv preprint arXiv:2603.08899*, 2026.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian Yen, Avner May, Tianqi Chen, and Beidi Chen. MagicDec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *International Conference on Learning Representations*, volume 2025, pp. 6835–6850, 2025.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 10107–10116, 2018.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, 2022. ISBN 9781713871088.

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

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

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

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

Boyi Zeng, He Li, Shixiang Song, Yixuan Wang, Zitong Wang, Ziwei He, Xinbing Wang, and Zhouhan Lin. Ponderlm-2: Pretraining llm with latent thoughts in continuous space. *arXiv preprint arXiv:2509.23184*, 2025.

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

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, and Xin Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 168990–169012. Curran Associates, Inc., 2025. doi: 10.52202/085713-5629. URL `https://proceedings.neurips.cc/paper_files/paper/2025/file/f7396d1c54d51416958d63e285377103-Paper-Conference.pdf`.

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.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. Distillspec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations*, volume 2024, pp. 32011–32050, 2024.

A Appendix

A.1 Training Objective for the Thinker and Drafter

Given a target-generated training trajectory $\mathbf{y}_{1:N}$, we sample valid anchors a and supervise the K positions following each anchor. For every block, features at positions before a form $\mathbf{C}_{<a}$; y_a initializes both the Thinker and the Drafter. When several anchors are packed into one training pass, an attention mask lets each block read its own prefix and internal slots while disallowing cross-block attention and access to future target features. Target features and final hidden states are precomputed with a frozen target. We optimize the context projection, Thinker, and Drafter, while keeping the copied token embedding and LM head frozen.

Position-weighted prediction. Because verification stops at the first rejection, errors at early positions shorten the accepted prefix (Wu et al., 2026). With validity indicator $m_{a,i} \in \{0, 1\}$, decay parameter $\gamma > 0$, and normalizer $Z = \sum_{a,i} m_{a,i} w_i$, we set

$$w_i = \exp\left(-\frac{i-1}{\gamma}\right), \qquad i = 1, \ldots, K, \tag{11}$$

and minimize the position-weighted cross-entropy

$$\mathcal{L}_{\mathrm{CE}} = -\frac{1}{Z} \sum_{a} \sum_{i=1}^{K} m_{a,i} w_i \log q_{a,i}(y_{a+i}). \tag{12}$$

Here $q_{a,i}$ is the Drafter's distribution at offset i when initialized from anchor a and the corresponding cached target prefix; the subsequent gold tokens are labels, not additional draft inputs.

Target-distribution alignment. Let $\mathbf{h}^{\mathcal{T}}_{a+i-1}$ be the frozen target's final state computed under the teacher-forced prefix $\mathbf{y}_{1:a+i-1}$. Its aligned next-token distribution is

$$p_{a,i}(\cdot) = \mathrm{softmax}\big(\mathbf{W}_{\mathrm{lm}} \mathbf{h}^{\mathcal{T}}_{a+i-1}\big). \tag{13}$$

For any two distributions p and q evaluated at the *same* prefix, sampling from q and applying the rejection rule of Equation 2 yields the single-position acceptance probability

$$\sum_{v \in \mathcal{V}} q(v) \min\left(1, \frac{p(v)}{q(v)}\right) = 1 - \frac{1}{2}\|p - q\|_1. \tag{14}$$

This single-step total-variation identity underlies distribution-alignment approaches to speculative drafting (Zhou et al., 2024); it motivates aligning the position-wise training distributions using

$$\mathcal{L}_{\mathrm{L1}} = \frac{1}{Z} \sum_{a} \sum_{i=1}^{K} m_{a,i} w_i \|p_{a,i} - q_{a,i}\|_1. \tag{15}$$

The teacher-forced $p_{a,i}$ is only a training signal: inference-time verification conditions on the *drafted* prefix, which may differ from the teacher-forced prefix. Thus, Equation 14 is a single-position identity, not a guarantee of whole-block acceptance under this objective. The combined loss is

$$\mathcal{L}_{\mathrm{LatentSpec}} = \lambda_{\mathrm{CE}} \mathcal{L}_{\mathrm{CE}} + \lambda_{\mathrm{L1}} \mathcal{L}_{\mathrm{L1}}. \tag{16}$$

We use $\gamma = 4$, $\lambda_{\mathrm{CE}} = 0.1$, and $\lambda_{\mathrm{L1}} = 0.9$. Both losses train the Thinker through the Drafter, without direct supervision of $\mathbf{S}_t$.

A.2 Implementation Details

Training details. We trained the draft models on Open-PerfectBlend[1], a general-purpose instruction dataset containing approximately 1.3 million examples spanning chat (17.6%), mathematics (39.4%), code (38.9%), and instruction following (4.1%). We held out 5% of the original dataset and used only the prompts from the remaining examples: the original assistant responses were

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

discarded and regenerated by the corresponding target model using SGLang with thinking mode disabled. For each regenerated conversation, we ran a single forward pass through the frozen target model to extract and cache the hidden states required for draft-model training. We implemented LatentSpec within the DeepSpec training framework and retrained the draft-model baselines on the same regenerated data to avoid differences attributable to training corpora. All training runs used random seed 42, matching the DSpark training configuration. We used AdamW with a peak learning rate of 6×10^{-4}, a 4% linear warm-up followed by cosine decay, zero weight decay, a global batch size of 512, and a maximum sequence length of 4,096. The training objective combined next-token cross-entropy and the ℓ_1 distance between the draft and target output distributions with weights of 0.1 and 0.9, respectively. Unless otherwise specified, we evaluated the checkpoint saved after the first training epoch.

Acceptance evaluation. We evaluate acceptance on MATH-500, AIME 2025, HumanEval, MBPP, MT-Bench, and Alpaca, with sample caps of 500, 30, 164, 256, 80, and 500, respectively. For each example, we retain the first user turn, disable the target model's thinking mode, and generate at most 2,048 new tokens with batch size one under both greedy decoding ($T = 0$) and sampling ($T = 1$). We use evaluation seed 980406 for both sample selection and generation. This seed matches the DSpark evaluation configuration. Under greedy decoding, a proposed token is accepted only when it matches the target model's greedy prediction. Verification stops at the first mismatch, where the target prediction is committed as the correction token; if all proposals are accepted, the target model's greedy bonus token is committed. Under sampling, a proposed token x_i is accepted with probability $\min\{1, p_i(x_i)/q_i(x_i)\}$, where p_i and q_i are the target and draft distributions; acceptance stops at the first rejection. We then sample a correction token from the normalized positive part of $p_i - q_i$, or sample a bonus token from the target distribution if all proposals are accepted. The reported acceptance length τ is the total number of committed tokens divided by the number of verification steps, including the correction or bonus token except when an accepted stop token terminates generation. For position i, we report the number of verification steps in which the accepted draft prefix reaches i, divided by the number of steps that propose a token at that position. Counts are aggregated across evaluation workers before computing each metric.

End-to-end throughput evaluation. We measured end-to-end generation throughput on a single NVIDIA H20 GPU with 141 GB of memory. The serving implementation was based on SGLang v0.5.19, which we extended with LatentSpec speculative decoding. We launched each target–draft pair with tensor parallelism set to one and benchmarked its OpenAI-compatible streaming chat-completion endpoint. For LatentSpec and DFlash, the speculative configuration used eight verification tokens per round, corresponding to one existing context token and up to seven draft tokens; the EAGLE-3 baseline used seven speculative steps, top-$k = 1$ for its draft tree, and eight draft tokens. We used the first-epoch draft checkpoints for the reported comparisons.

The benchmark comprised MATH-500, AIME 2025. Each prompt was submitted as a single user message; all available prompts were used at every tested concurrency, without further subsampling. The concurrency limits were 1, 2, 4, 8, and 16 simultaneous requests. We disabled Qwen3's explicit thinking mode and set the maximum completion length to 512 tokens, temperature to 1.0, top-p to 1.0, top-k to -1 (disabled), and min-p to 0.0. Frequency and presence penalties were 0, the repetition penalty was 1, and the request-level sampling seed was 980406, this seed matches the DSpark evaluation configuration. We did not enable `ignore_eos` or specify additional stop sequences.

Before measurement, we issued 32 warm-up requests at concurrency 16 and flushed the server KV cache; we also flushed the KV cache between dataset–concurrency groups. Throughput was computed from successful streaming requests as the sum of their completion-token counts divided by the duration of the union of their intervals from request submission to the last generated token. Thus, the reported tokens-per-second values include first-token latency. Table 3 reports the AIME 2025 and MATH-500 results under this protocol.

Draft FLOPs. We compare LatentSpec with the depth-matched DFlash-5L drafter on Qwen3-4B. Let C denote the cached context length, $R = 8$ the number of newly committed target states, $K = 7$ the draft block size, and $M = 4$ the number of latent slots. Counting each multiply–add as two FLOPs, the cost of one decoder layer with q query positions, e newly projected external key/value

Table 5: Comparison of analytical FLOPs per draft proposal between DFlash-5L and LatentSpec on Qwen3-4B with $K = 7$ and $M = 4$.

Context length	DFlash-5L (GFLOPs)	LatentSpec (GFLOPs)	Reduction
512	13.752	**12.616**	8.260%
1,024	14.045	**12.859**	8.444%
2,048	14.632	**13.346**	8.789%
4,096	15.807	**14.319**	9.414%

positions, and a visible key positions per query is

$$\mathcal{L}(q, e, a) = 2q(dd_q + d_q d + 2dd_{\mathrm{kv}} + 3df) + 4edd_{\mathrm{kv}} + 4qad_q,$$

where d and f are the hidden and MLP dimensions, and d_q and d_{kv} are the query and key/value projection dimensions. Writing $F_{\mathrm{fusion}} = 2Rn_f d^2$ for the shared target-feature fusion cost and $F_{\mathrm{head}} = 2KdV$ for the draft vocabulary head, the two proposal costs are

$$F_{\mathrm{DFlash}} = F_{\mathrm{fusion}} + 5\mathcal{L}(K, R, C + K) + F_{\mathrm{head}},$$
$$F_{\mathrm{LatentSpec}} = F_{\mathrm{fusion}} + 2\mathcal{L}(M, R, C + M) + 3\mathcal{L}(K, R + M, C + M + K) + F_{\mathrm{head}}.$$

As shown in Table 5, LatentSpec reduces draft-proposal FLOPs by 8.260%–9.414% over contexts of 512–4,096 tokens. This computational reduction is expected to facilitate faster training and lower inference-time drafting latency.