DRAFT IN PARALLEL, CONDITION THROUGH DEPTH: ADJACENT CAUSAL INJECTION FOR SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Parallel speculative drafting generates multiple candidates in one backbone pass, but independent token selection can produce inconsistent continuations that shorten the accepted prefix. Existing methods mostly leave conditional decoding to a lightweight module after the backbone, which limits the flow of predecessor information to successors. Our analysis of DFlash shows that early positions already form recoverable predictions in shallow layers, and that accurate adjacent predecessors help successors more when they enter earlier. We therefore propose DSpine, a drafter with causal conditioning injection throughout the backbone: at every layer, gated adjacent injection writes each predecessor's predicted feature into its successor, so the causal conditioning chain unfolds over network depth while all positions update in parallel. A unified transfer space built from the target model's output embeddings unifies layer-wise injection with final conditional decoding, and layer-wise output-embedding supervision promotes the formation of predicted features in shallow layers. Fused kernels and a transition cache execute both efficiently in parallel within SGLang. Across mathematics, coding, and conversation on Qwen3-4B and Qwen3-8B, DSpine raises the mean acceptance length of Qwen3-8B under greedy decoding from 3.77 with DFlash to 4.82 (+27.8%) and delivers 23.3% higher SGLang throughput than DFlash.

1 INTRODUCTION

Autoregressive decoding is a major source of latency in large language models. Speculative decoding accelerates it without changing the target model: a lightweight drafter proposes multiple candidates, and the target model verifies them in a single forward pass (Leviathan et al., 2023). Its speedup depends on how many candidates are accepted per round and how cheaply they are drafted. Autoregressive drafters such as EAGLE-3 reuse target-model features to improve predictions but still require one sequential drafting step per draft position (Li et al., 2025). Inspired by diffusion language modeling, DFlash instead predicts a masked block in parallel within one forward pass conditioned on target context features, so a longer draft adds no sequential drafting steps (Chen et al., 2026).

This shift also changes how token dependencies are established. Autoregressive drafting conditions each successor on its already selected predecessors, whereas positions in a parallel draft must form predictions before their predecessors commit to tokens, so individually plausible candidates can be inconsistent when combined. Because the target accepts candidates in prefix order, a single inconsistency truncates the rest of the round's draft, leaving much of the cheaply drafted block unaccepted. Reducing drafting cost must therefore go hand in hand with establishing effective predecessor conditioning inside parallel computation.

Recent work improves token consistency in parallel drafts through lightweight conditioning modules. Inside the draft backbone, DFlash2 mixes adjacent positions through local convolutions, but these blend neighboring hidden states without separating the predecessor's predictive information from the rest of its state (Inco AI, 2026). Domino, DSpark, and DFlash2's path selector instead condition on the predecessor after the backbone to adjust token distributions or candidate selection (Huang et al., 2026; Cheng et al., 2026; Inco AI, 2026). Conditional decoding in these methods is thus

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.

left to a lightweight module after the backbone, which limits the flow of predecessor information to successors. Moreover, the predecessor information these modules pass is confined to the final-layer vocabulary space, so it can neither be combined with in-layer features nor exploit the rich intermediate features formed across the backbone.

Our empirical study yields two key insights. **(1) Earlier positions form predictions at shallower depth:** layer-wise readouts already recover substantial token-predictive information at early positions in shallow layers (Figure 1(a)), so the prefix could condition successors whose computation is still in progress. **(2) Successor prediction depends on both the content of the adjacent predecessor and when it enters the network:** even with correct earlier context, replacing only the adjacent predecessor with its draft prediction sharply lowers successor accuracy, and injecting the same correct predecessor features earlier clearly improves it (Figure 1(c)).

Guided by these insights, we propose **DSpine**, a drafter with causal conditioning injection throughout the backbone. At every layer, a gated adjacent injection module writes each predecessor's predicted feature into its successor, so that the causal conditioning chain unfolds over network depth while all positions update in parallel. We further build a unified transfer space from the output embeddings of the target model, which unifies layer-wise injection with the information transfer in final conditional decoding. We also design layer-wise output-embedding supervision that aligns the predicted features with this space, promoting their formation in shallow layers and reducing their discrepancy from token embeddings. Finally, we introduce a transition cache that precomputes the scores of all adjacent candidate pairs in parallel, turning sequential conditional decoding into parallel computation. Our contributions are:

- We identify recoverable shallow prefix predictions and the effect of predecessor-conditioning timing, motivating dependency formation across network depth.
- We propose DSpine, which injects causal conditioning throughout the backbone and links it to final conditional decoding through a unified output-embedding space with layer-wise supervision.
- We demonstrate that DSpine surpasses baselines trained on the same data in both acceptance length and SGLang throughput: on Qwen3-8B, it raises the mean acceptance length from 3.77 for DFlash to 4.82, and its SGLang throughput exceeds that of DFlash by 23.3%.

2 Preliminaries

2.1 Speculative Decoding

Given a confirmed context $x_{\leq 0}$, a drafter proposes M candidates $\hat{x}_{1:M}$, which the target model verifies in one forward pass (Leviathan et al., 2023). Let $q_{\mathrm{draft},t}$ be the actual proposal distribution given this context and preceding candidates; both it and the target distribution p_t include their respective sampling transformations. Conditional on acceptance of all preceding candidates, candidate t is accepted with probability

$$\min\left(1, \frac{p_t(\hat{x}_t)}{q_{\mathrm{draft},t}(\hat{x}_t)}\right), \qquad p_t(v) = p_{\mathrm{target}}(v \mid x_{\leq 0}, \hat{x}_{1:t-1}). \tag{1}$$

At the first rejection, the remaining draft is discarded and a replacement is sampled from the corrected distribution (Equation (11)); if all candidates pass, the target supplies an additional token. Greedy decoding accepts candidates matching the target's choice under the same prefix.

The average acceptance length $\tau = \mathbb{E}[N_{\mathrm{acc}}] + 1$ is the number of tokens advanced per round, where N_{acc} counts consecutively accepted draft tokens and the extra one is the correction or additional target token. For a fixed runtime configuration, ignoring prefill and termination costs, the average time per output token (TPOT) is approximately

$$\mathrm{TPOT}_{\mathrm{spec}} \approx \frac{T_{\mathrm{draft}} + T_{\mathrm{verify}}}{\tau}, \tag{2}$$

where T_{draft} and T_{verify} are mean per-round costs, including sampling and cache management. Later candidates contribute only when the preceding prefix is accepted, and acceptance gains must compensate for added drafting cost.

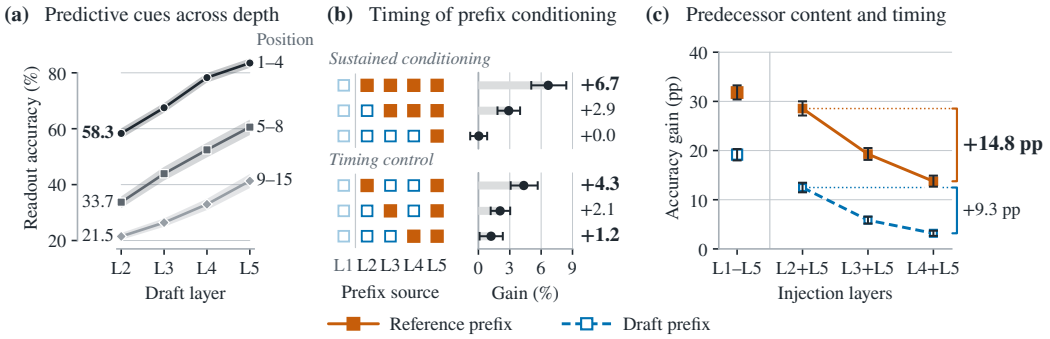

Figure 1: (a) Readout accuracy of each layer, grouped by block position. (b) Gain in consecutive correct tokens when the drafted prefix is replaced by the correct prefix at the filled (orange) layers; top: from one layer onward; bottom: at two layers, moving the earlier one. (c) Gain in successor accuracy when, at the specified layers, the adjacent predecessor's features come from the correct token rather than its draft prediction (L1–L5: all layers); line color marks whether the history before the predecessor is correct (orange) or drafted (blue).

2.2 Parallel Drafting

DFlash-style parallel drafters process a block of $M + 1$ positions, the known anchor x_0 followed by M masked candidate positions (Chen et al., 2026; Zhang et al., 2026). In DFlash, fused target-model features over $x_{<0}$ supply K/V at every layer. The backbone updates all positions in one pass with bidirectional block attention, and the LM head shared with the target produces candidate distributions.

An autoregressive drafter generates each token conditioned on the predecessors it has drafted, whereas a DFlash-style drafter predicts every position from the confirmed context alone, so the block proposal factorizes over positions:

$$q_{\text{draft}}(\hat{x}_{1:M} \mid x_{\leq 0}) = \prod_{t=1}^{M} q_{\text{draft},t}(\hat{x}_t \mid x_{\leq 0}). \tag{3}$$

Each position thus marginalizes over its possible predecessors instead of conditioning on the one actually drafted, and individually likely tokens can form inconsistent combinations. Because verification accepts candidates in prefix order, such inconsistencies cut the accepted prefix, and acceptance decays rapidly at later positions (Cheng et al., 2026).

3 When and How Prefix Predictions Condition Parallel Drafting

This section asks whether evolving prefix predictions can condition successor computation in parallel drafting. We first examine when predictive information becomes recoverable and when reliable prefix conditions are most useful, then turn to the immediate predecessor: its contribution beyond earlier history and the timing of its use. We use the frozen official Qwen3-8B-DFlash-b16 with greedy Qwen3-8B continuations as the correct tokens; datasets, metrics, and full results are in Appendix B.

3.1 Early Predictive Cues and the Timing of Conditioning

To examine when predictive information emerges, we read out predictions from each layer. Intermediate DFlash states lie outside the input space of the target LM head and cannot be decoded directly, so for each layer we fit, on separate data, an equal-capacity lightweight low-rank projection that maps its frozen states into the space decoded by the frozen LM head. At the same shallow depth, early positions are read out much more accurately than distant ones, while all position groups improve with depth (Figure 1(a)). Although positions update in parallel, predictive information is not equally available: prefixes already carry recoverable cues while successor predictions are still forming, so prefixes could condition successor computation before selecting their own tokens.

We next examine when more accurate prefix conditions are most useful. With a correct token as the anchor, we fill the preceding prefix with draft predictions or correct tokens, which the target model encodes for DFlash's existing context pathway. Sustained access to correct-prefix features from an early layer yields a 6.7% relative gain in consecutive correct tokens over the all-draft-prefix baseline, and delaying it reduces the gain (Figure 1(b), sustained conditioning).

Earlier access, however, also means that more layers receive correct features. To isolate timing, we fix the total number of injection layers and vary only their placement: earlier injection raises the relative gain from 1.2% to 4.3% (Figure 1(b), timing control). Thus, even at a fixed number of uses, earlier access to reliable conditions improves successor prediction: a reliable prefix is valuable not only for more accurate content but also for timely use as successor representations form.

3.2 Adjacent Predecessors Shape Successor Predictions

We next examine how the immediate predecessor affects successor prediction. Keeping the target-encoded earlier history correct, we use the predecessor as the anchor of a new draft block: replacing only the correct predecessor with its draft prediction lowers successor accuracy by 31.8 percentage points on average over three block positions (L1–L5 in Figure 1(c)). Correct earlier history thus does not fully compensate for an inaccurate predecessor: the adjacent token supplies additional predictive cues for its successor.

The predecessor's effect also depends on when it enters successor computation. Keeping the draft-predicted anchor, we replace its K/V at selected layers with the same-layer K/V cached from a forward pass with the correct predecessor. With correct earlier history, the same number of injection layers, and the same predecessor K/V at the final layer, earlier injection improves successor accuracy over later injection by 14.8 points on average (Figure 1(c)), indicating that adjacent conditions affect how successor representations form.

These two observations are complementary: shallow prefix states already carry recoverable predictive cues, and accurate predecessor features help successors more when supplied earlier. Motivated by this, we explore transmitting predecessor prediction features to successors from shallow layers, letting token dependencies build up progressively through network depth (Section 4).

4 DSpine: Adjacent Causal Injection and Output-Embedding Supervision

DSpine establishes predecessor conditioning at every backbone layer (Figure 2): gated adjacent injection writes each predecessor's predicted feature into its successor, and layer-wise output-embedding supervision promotes the formation of these features in shallow layers. We design a unified transfer space built from the target LM head, so that the embedding of a decoded predecessor token serves as ideal causal information and completes final conditional decoding through the same mechanism. A fused transition cache precomputes the final-layer features and scores for all adjacent candidate pairs in parallel, leaving only selection or sampling sequential.

4.1 Adjacent Causal Injection

Section 3.2 shows that the adjacent predecessor carries predictive cues that earlier context cannot replace (Figure 1(c)). Inspired by this finding, DSpine establishes an explicit pathway from each predecessor to its immediate successor after every Transformer layer. Unlike DFlash and other parallel drafters with bidirectional block attention, DSpine uses *block-causal attention* in all L backbone layers: position t attends only to target-context K/V and block positions $0, \ldots, t$. Backbone computation and explicit injection thus share the predecessor-to-successor direction while retaining single-pass, block-parallel drafting.

Unified transfer space. The target LM head scores token v as $E_v^\top h$, so its output embeddings naturally encode which token a state predicts. We build the unified transfer space from them: we ℓ_2-normalize the rows of the LM head, apply PCA whitening, keep r principal directions, and normalize each row again to obtain a fixed embedding table C (Appendix A). All causal information passes through this space. Each layer projects its pre-injection state $u_{\ell,t}$ into a predicted feature $z_{\ell,t} =$

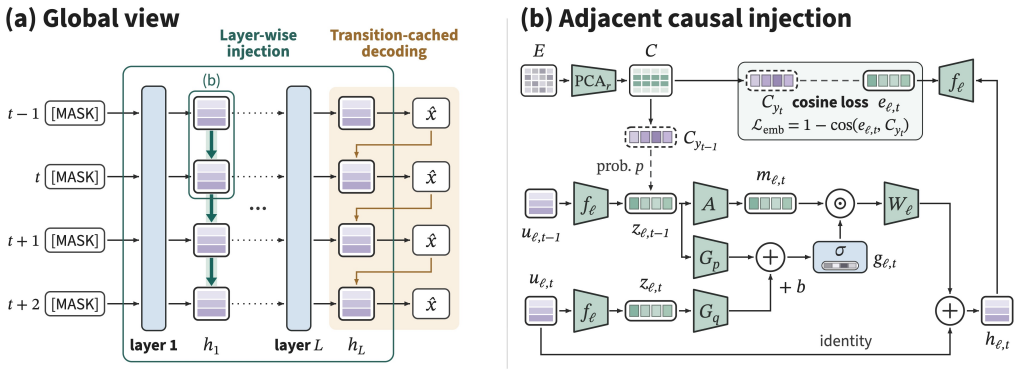

Figure 2: Overview of DSpine. (a) Injection passes predecessor features to successors at every layer (green); after the final layer, transition-cached decoding selects each token conditioned on its selected predecessor (orange). (b) One injection layer: predecessor and successor features jointly gate how much of the predecessor's message enters the successor's residual stream; training adds cosine alignment to C and, with probability p, substitutes $C_{y_{t-1}}$ for the predecessor feature.

$f_\ell(u_{\ell,t}) = \sqrt{r}\ \mathrm{normalize}(R_\ell u_{\ell,t})$ without intermediate vocabulary decoding, where R_ℓ is a layer-specific learned projection to r dimensions and normalize denotes ℓ_2 normalization; known tokens, namely the anchor and decoded predecessors, enter directly as $\sqrt{r}\, C_x$.

Gated adjacent conditioning. Because shallow predecessor features are still forming and vary in reliability, each successor receives a message only from its immediate predecessor and gates how much of it to absorb. At position t, the receiver feature $z_{\ell,t}$ and predecessor feature $z_{\ell,t-1}$ jointly determine the gate, while the predecessor feature is projected into a message that is gated and written back to the residual stream:

$$g_{\ell,t} = \sigma(G_q z_{\ell,t} + G_p z_{\ell,t-1} + b)\,, \tag{4}$$

$$m_{\ell,t} = A z_{\ell,t-1},$$

$$h_{\ell,t} = u_{\ell,t} + \mathrm{RMS}(u_{\ell,t})\, W_\ell\, (g_{\ell,t} \odot m_{\ell,t})\,. \tag{5}$$

Here σ is the sigmoid function; A, G_q, G_p, and b are shared across layers, and each layer has its own write matrix W_ℓ. All messages use pre-update features, so injection within a layer runs in parallel across positions; the anchor remains unchanged.

Propagation through depth. Because earlier access to accurate predecessor features benefits successor prediction (Figure 1(c)), injection starts at the first layer and its effects propagate through depth: messages received at one layer enter the next layer's computation, shaping the prediction features that successors send onward. For example, information that $t-1$ receives from $t-2$ at layer ℓ can influence the message sent to t at layer $\ell+1$ (Figure 2(a)). The causal conditioning chain thus unfolds over network depth while positions within each layer update in parallel; earlier context remains accessible through block-causal attention.

Final conditional decoding. The final-layer output $h_{L,t}$ passes through the final RMSNorm and shared LM head to determine each position's initial top-K candidates; final decoding then conditions each position on the token actually decoded at its predecessor. A decoded predecessor is a known token, so, like the anchor, it enters the unified transfer space directly: for a token s at position $t-1$, we replace the predicted feature $z_{L,t-1}$ with $\sqrt{r}\, C_s$, changing both gate and message, and redo the final-layer write (Equation (5)) from the cached pre-injection state $u_{L,t}$; we denote the output by $h_t^{\circ}(s)$ and call this step *last-write refinement*. The shared LM head maps $h_t^{\circ}(s)$ to conditional scores $S_t(s, v)$ over the initial candidates, which can be precomputed in parallel for all predecessor candidates (Section 4.4).

4.2 Layer-Wise Output-Embedding Supervision

The predicted features transmitted by injection already carry recoverable cues in shallow layers but are still forming (Figure 1(a)), and the final-layer loss shapes them only indirectly through later computation. DSpine therefore aligns the predicted features at every layer directly with the correct token's embedding in the unified transfer space. This supervision promotes their formation in shallow layers and narrows the gap between in-layer predicted features and the token embeddings used in final decoding, so that the two behave consistently when passed through the same injection.

Directional alignment. With the projection f_ℓ used for injection, we read $e_{\ell,i} = f_\ell(h_{\ell,i})$ from post-injection states and align it with the correct token's embedding C_{y_i}:

$$\mathcal{L}_{\text{emb}} = \sum_{\ell=1}^{L} \sum_{i} w_i \left[1 - \cos(e_{\ell,i}, C_{y_i})\right]. \quad (6)$$

where $w_i = \exp(-(t_i - 1)/\gamma)$ is the decay weight for block position t_i. Because the readout follows injection, this loss trains both the formation of each feature and the successor's use of the received message; it constrains only the direction of a low-dimensional projection, leaving the full hidden state free for contextual computation.

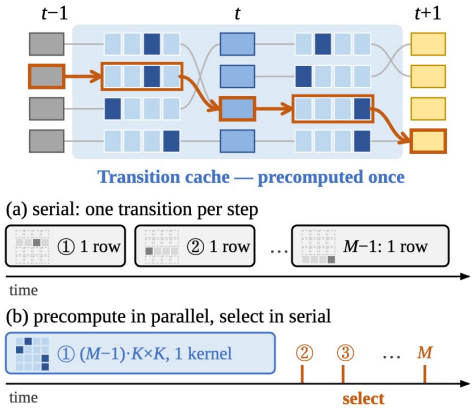

Figure 3: Transition cache. (a) Serial conditional scoring. (b) Parallel transition precomputation followed by sequential selection; greedy implementation shown.

4.3 Training

We sample draft blocks from training sequences and train the drafter with the target model, shared input embeddings, and LM head frozen. Training supervises two outputs. The first is the ordinary final-layer output, which predicts over the full vocabulary and determines the top-K candidates. The second is the output of last-write refinement: as at inference, it replaces the predicted predecessor feature with a token embedding and redoes the final-layer write, except that the correct predecessor token is used, and predicts only over those candidates. Both receive the same loss below, jointly training candidate generation and final conditional decoding.

Let q_i and p_i denote the draft and target distributions at position i under the correct prefix. Following the loss design of DSpark (Cheng et al., 2026), the loss is a weighted mix of token cross-entropy and probability ℓ_1 distance:

$$\mathcal{L}_{\text{dist}} = \sum_{i} w_i \Big[-\alpha \log q_i(y_i) + (1 - \alpha) \left\| q_i - p_i \right\|_1\Big], \quad (7)$$

where α is the mixing weight. The output of last-write refinement is also included in $\mathcal{L}_{\text{dist}}$, except that its distributions are restricted to the top-K candidates, with the target renormalized to that set.

Because predicted features and token embeddings share the unified transfer space, early in training we replace shallow-layer predecessor features with correct-token embeddings with some probability, keeping receiver features model-derived (Bengio et al., 2015), and gradually phase this replacement out; last-write refinement always uses correct predecessors. Block-causal attention ensures that the injected correct information cannot flow backward. The overall training objective weights layer-wise supervision by λ (coefficients and schedules in Appendix A):

$$\mathcal{L} = \mathcal{L}_{\text{dist}} + \lambda \mathcal{L}_{\text{emb}}. \quad (8)$$

4.4 Efficient Inference

Fused layer-wise injection. Layer-wise injection consists of small normalization, projection, gating, and residual-write operations, which a direct implementation executes as many kernel launches with intermediate tensor traffic. We merge the three projections into one matrix multiplication and

Table 1: Average acceptance length τ on Qwen3 models. Each cell gives temperature 0 / 1; bold marks the best result for each model and temperature, and underline marks the second best.

Model	Method	MATH		CODE			CHAT		OVERALL
		GSM8K	MATH-500	HumanEval	MBPP	LCB	MT-Bench	Arena-Hard	*Avg.*
Qwen3-8B	DFlash	4.25 / 3.96	4.23 / 3.86	3.92 / 3.59	3.93 / 3.56	3.53 / 2.99	3.27 / 2.97	3.30 / 2.80	3.77 / 3.39
	Domino	4.83 / 4.52	4.81 / 4.30	4.21 / 3.79	4.31 / 3.85	3.90 / 3.23	3.58 / 3.18	3.61 / 2.99	4.18 / 3.69
	DSpark	4.99 / 4.61	4.90 / 4.48	4.43 / 4.07	4.50 / 4.12	4.01 / 3.49	3.69 / 3.37	3.69 / 3.26	4.32 / 3.92
	DFlash2	5.03 / 4.56	5.10 / 4.47	4.43 / 4.04	4.52 / 4.06	4.09 / 3.50	3.71 / 3.35	3.82 / 3.29	4.39 / 3.90
	DSpine (ours)	**5.67 / 5.31**	**5.49 / 5.05**	**4.94 / 4.63**	**4.98 / 4.61**	**4.46 / 3.91**	**4.11 / 3.79**	**4.12 / 3.62**	**4.82 / 4.42**
Qwen3-4B	DFlash	4.16 / 3.98	4.14 / 3.82	3.77 / 3.62	3.92 / 3.65	3.42 / 2.99	3.29 / 3.16	3.26 / 2.89	3.71 / 3.44
	Domino	4.79 / 4.58	4.73 / 4.31	4.22 / 4.00	4.39 / 4.09	3.79 / 3.25	3.66 / 3.42	3.61 / 3.16	4.17 / 3.83
	DSpark	5.00 / 4.64	4.90 / 4.47	4.38 / 4.12	4.59 / 4.20	3.93 / 3.48	3.80 / 3.51	3.71 / 3.33	4.33 / 3.96
	DFlash2	5.14 / 4.63	5.15 / 4.51	4.59 / 4.10	4.72 / 4.20	4.10 / 3.50	3.90 / 3.51	3.91 / 3.44	4.50 / 3.98
	DSpine (ours)	**5.53 / 5.22**	**5.38 / 4.94**	**4.79 / 4.59**	**5.04 / 4.67**	**4.29 / 3.79**	**4.19 / 3.84**	**4.02 / 3.64**	**4.75 / 4.38**

use two fused kernels, one for the gated message and one for the residual write together with the next layer's input normalization, all executed within the draft backbone's CUDA graph.

Transition-cached decoding. Final conditional decoding makes each position's scores depend on the predecessor's choice, so computing them directly proceeds sequentially across positions. The transition cache removes this sequential computation: once the backbone has fixed each position's pre-injection state $u_{L,t}$ and top-K candidate set $\mathcal{V}_t = \{v_{t,1}, \ldots, v_{t,K}\}$, last-write refinement depends only on the predecessor's token, so before selection we compute the refined final-layer features for all predecessor candidates in parallel and score the successor candidates, organized as $T_t[i, j] = S_t(v_{t-1,i}, v_{t,j})$ (Figure 3). Enumerating all K^2 pairs per transition trades additional scoring work for a shorter sequential dependency chain; in practice, $K = 16$ suffices at a small additional cost (Section 5.4). Selection then proceeds through the cache from the first position, without any sequential matrix operations.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Models and evaluations. We conduct experiments on Qwen3-4B and Qwen3-8B (Yang et al., 2025) with thinking mode disabled, covering three task categories: *Math*: GSM8K (Cobbe et al., 2021) and MATH-500 (Lightman et al., 2023); *Code*: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (LCB) (Jain et al., 2024); *Chat*: MT-Bench (Zheng et al., 2023a) and Arena-Hard (Li et al., 2024a). On all seven benchmarks, we measure the average acceptance length τ (Section 2.1) in SGLang (Zheng et al., 2023b) at temperatures 0 and 1; serving throughput is measured on GSM8K, MATH-500, HumanEval, and MBPP at temperature 0 and concurrency 2–32, with the same fixed verification budget for all methods (Appendix C).

Baselines. We compare with four representative state-of-the-art parallel drafters: DFlash (Chen et al., 2026), Domino (Huang et al., 2026), DSpark (Cheng et al., 2026), and DFlash2 (Inco AI, 2026). DFlash selects tokens independently across positions; the other three build on the DFlash backbone to model intra-block dependencies.

Implementation. For each target model, all drafters are retrained on the same ShareGPT data, with responses regenerated by the target; backbones use five layers and a block size of 16 and are trained for six epochs with the same global batch size and learning-rate schedule (Appendix A).

5.2 MAIN RESULTS

As shown in Table 1, DSpine achieves the longest acceptance length on all benchmarks for both models at both temperatures. On Qwen3-8B, it raises the average τ of DFlash from 3.77 to 4.82 and leads the strongest baseline by 10.0% with greedy decoding (DFlash2) and by 12.8% under

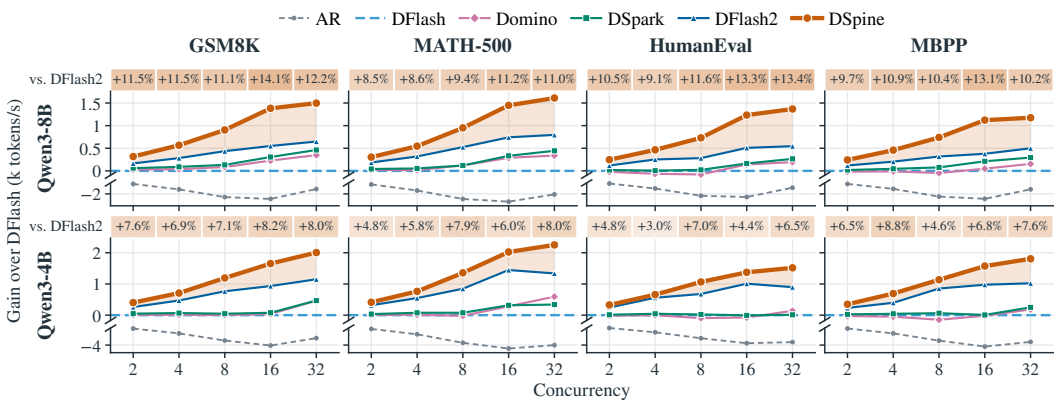

Figure 4: Serving throughput gain over DFlash on SGLang, in thousands of tokens per second (temperature zero). The dashed zero line is DFlash; shading and the strip above each panel give the lead of DSpine over DFlash2, the strongest baseline.

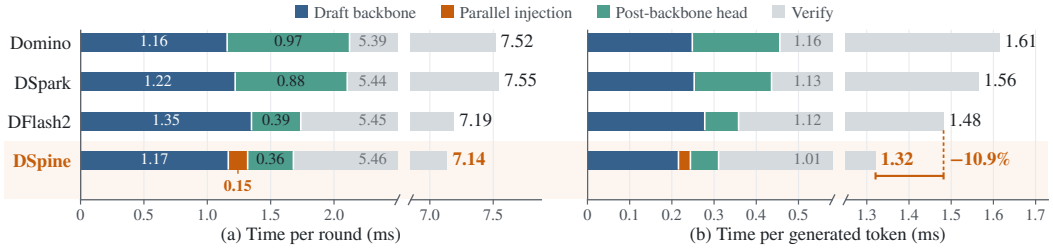

Figure 5: Module-level breakdown of Qwen3-8B decoding time on SGLang.

sampling (DSpark); on Qwen3-4B, its lead over the strongest baseline, DFlash2, is 5.4% and 10.1%, respectively. Domino, DSpark, and DFlash2, which condition on predecessors after the backbone, already improve substantially over DFlash, while DSpine, which conditions on predecessors at every layer and again at token selection, extends the gain further, with a larger lead under sampling (see Appendix C for each method's proposal policy at temperature one).

5.3 Serving Throughput

As shown in Figure 4, DSpine achieves the highest throughput at every task and concurrency level on both models: on Qwen3-8B it outperforms the strongest baseline, DFlash2, by 11.1% on average and reaches up to 4.0× speedup over autoregressive decoding (Table 8); on Qwen3-4B it leads DFlash2 by 6.5% on average. The gain in acceptance length thus carries over to serving, and the extra cost of layer-wise injection and transition-cached decoding (Section 5.4) does not offset it.

5.4 Decoding Latency Breakdown

To examine where decoding time goes, we time each decoding round of Qwen3-8B in SGLang, split into a draft stage and a verification stage. As shown in Figure 5, verification takes 72–76% of round time and costs about the same for all methods. DSpine's parallel injection adds at most 0.15 ms per round across all five layers, and its final conditional decoding, which scores candidates and selects through the transition cache (Section 4.4), costs less than half as much as the post-backbone heads of DSpark and Domino. Overall, a DSpine round costs about as much as a DFlash2 round (7.14 vs. 7.19 ms), and because more tokens are accepted per round, DSpine spends 10.9% less decoding time per generated token than DFlash2.

Table 2: Ablation of module

Dataset	Full	w/o last-write refinement	w/o refinement & injection
MATH-500	**5.49**	5.11	4.93
MBPP	**4.98**	4.65	4.54
MT-Bench	**4.11**	3.82	3.67

Table 3: Ablation of supervision loss

Dataset	None (DFlash)	Token CE	Embedding cosine
MATH-500	3.75	3.67	**3.85**
MBPP	3.48	3.40	**3.63**
MT-Bench	2.91	2.84	**2.99**

5.5 Ablation Study

Effect of each component. We train each variant in Table 2 independently. Removing last-write refinement reduces τ by 6.6–7.1%, and further removing parallel injection enlarges the drop to 8.8–10.7%. Both components improve acceptance length, and their contributions are complementary.

Supervision target. Table 3 compares intermediate-layer supervision: directional supervision toward compact output embeddings improves acceptance length, whereas per-layer full-vocabulary cross-entropy falls below the unsupervised baseline. This supports the design of Section 4.2, as forcing intermediate states to classify tokens may interfere with later layers' contextual computation.

6 Related Work

Speculative decoding lets a lightweight drafter propose tokens that the target model verifies losslessly (Leviathan et al., 2023; Chen et al., 2023). Follow-ups mainly improve the drafter: Medusa predicts several positions with independent heads, Hydra makes these heads depend on drafted tokens (Cai et al., 2024; Ankner et al., 2024), the EAGLE series drafts autoregressively from target features (Li et al., 2024b;c; 2025), and tree verification checks multiple continuations per round (Miao et al., 2024; Li et al., 2024c). Autoregressive drafting conditions each token on its predecessors but costs one sequential step per token.

Parallel decoding methods predict multiple positions at once (Stern et al., 2018; An et al., 2026; Liu et al., 2026; Christopher et al., 2025; Li et al., 2026; Chen et al., 2025). DFlash drafts a block in one pass of a lightweight block-diffusion model conditioned on target features (Chen et al., 2026); follow-ups add layer-wise target conditioning, position-weighted losses, or draft trees (Zhang et al., 2026; Wu et al., 2026; Ringel & Romano, 2026). Since positions predict before their predecessors commit, the resulting tokens can be inconsistent (Stern et al., 2018; Cheng et al., 2026). Most remedies restore this dependency after the backbone, through a causal correction branch (Domino), a sequential head (DSpark), or a path selector over top-k candidates (DFlash2) (Huang et al., 2026; Cheng et al., 2026; Inco AI, 2026), as well as previous-token conditioning, block refinement, or adjacent-candidate scoring (Rheinboldt et al., 2026; Wang et al., 2026; Rusanovsky et al., 2026). Inside the backbone, DFlash2 adds causal two-tap convolutions, and DART and JetSpec use causal block attention (Inco AI, 2026; Liu et al., 2026; Hu et al., 2026). DSpine also uses block-causal attention but additionally injects each predecessor's predicted feature, supervised in a unified transfer space, into its successor at every layer; the same injection carries decoded tokens in final conditional decoding.

7 Conclusion

DSpine injects causal conditioning throughout the backbone, building intra-block token dependencies over network depth within a single parallel drafting pass. Gated adjacent injection passes predecessor features to successors at every layer, and a unified transfer space with layer-wise supervision links this injection to final conditional decoding. A transition cache precomputes the scores of all adjacent candidate pairs in parallel, turning sequential conditional decoding into parallel computation. Across math, code, and chat tasks, DSpine improves acceptance length over representative parallel drafters and translates these gains into higher SGLang serving throughput. These results motivate further study of in-layer information transfer, including more flexible communication ranges and interactions in other parallel decoding architectures.

AI USE STATEMENT

Generative AI tools were used only to polish the writing and to implement parts of the code, not for research design or analysis. All AI-assisted content was reviewed by the authors, who take full responsibility for this paper.

REPRODUCIBILITY STATEMENT

We aim to make the experimental results in this work reproducible by providing detailed architecture, optimization, and training hyperparameters in Appendix A. The complete protocols for the conditioning analyses, including datasets, readout training, and metrics, are described in Appendix B. Appendix C specifies the evaluation datasets, decoding and verification procedures, sampling settings, and serving configuration used for the reported acceptance-length and throughput results. Unless otherwise stated, all baseline methods are trained and evaluated under the matched settings described in these appendices.

REFERENCES

Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. PARD: Accelerating LLM inference with low-cost PARallel draft model adaptation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=XbOyv7iVGL.

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for Medusa decoding. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=FbhjirzvJG.

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

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

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 Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR, 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. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026.

Keyu Chen, Zhifeng Shen, Daohai Yu, Haoqian Wu, Wei Wen, Jianfeng He, Ruizhi Qiao, and Xing Sun. ASPD: Unlocking adaptive serial-parallel decoding by exploring intrinsic parallelism in LLMs. arXiv preprint arXiv:2508.08895, 2025. URL https://arxiv.org/abs/2508.08895.

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, 2021.

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

Jacob K Christopher, Brian R. Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative diffusion decoding: Accelerating language generation through diffusion. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 12042–12059, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-long.601. URL https://aclanthology.org/2025.naacl-long.601/.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

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

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

Inco AI. DFlash 2: Keep drafting parallel. Inco AI Blog, August 2026. URL https://inco.ai/blog/dflash2/.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code, 2024.

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

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. DiffuSpec: Unlocking diffusion language models for speculative decoding. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 20896–20910, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1048. URL https://aclanthology.org/2026.findings-acl.1048/.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-Hard and BenchBuilder pipeline, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language*

Processing, pp. 7421–7432, Miami, Florida, USA, November 2024c. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/2024.emnlp-main.422/.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4562. URL https://papers.nips.cc/paper_files/paper/2025/hash/c7b5a35ea98b62512a869c19ea7b03cb-Abstract-Conference.html.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023.

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. URL https://arxiv.org/abs/2601.19278.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, pp. 932–949. ACM, 2024. doi: 10.1145/3620666.3651335.

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

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. arXiv preprint arXiv:2604.12989, 2026. URL https://arxiv.org/abs/2604.12989.

Matan Rusanovsky, Yoav Miron, Roy Uziel, Omer Belhasin, Hao Guo, Ran Zilberstein, Maor Ashkenazi, and Michael Elad. LiLiCorr: Lightweight likelihood correlation of parallel drafts for speculative decoding. arXiv preprint arXiv:2608.20530, 2026. URL https://arxiv.org/abs/2608.20530.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision, 2024.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Zheng Wang, Davis Wertheimer, Yu Chin Fabian Lim, Mudhakar Srivatsa, Raghu K. Ganti, Minjia Zhang, and Naigang Wang. xPress: Parallel refinement for diffusion drafters in speculative decoding. arXiv preprint arXiv:2608.02438, 2026. URL https://arxiv.org/abs/2608.02438.

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

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, 2025.

Jiebin Zhang, Zhenghan Yu, Song Liu, Eugene J. Yu, Zheng Li, Dawei Zhu, Jiangshan Duo, Weimin Xiong, Yifan Song, Guanghua Yu, Jianchen Zhu, and Sujian Li. DFlare: Scaling up draft capacity for block diffusion speculative decoding. arXiv preprint arXiv:2606.02091, 2026. URL https://arxiv.org/abs/2606.02091.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena, 2023a.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs, 2023b.

A Implementation Details of DSpine

Architecture. DSpine builds on the DFlash drafter for Qwen3-8B (Table 4): five Transformer layers with block-causal attention, target context features fused from five target layers and supplied as K/V to every layer, and the target's input embedding and LM head shared and frozen. Injection adds about 33M parameters, roughly 3% of the backbone.

Unified transfer space. Let $\bar{E}_v$ be the ℓ_2-normalized LM-head row of token v and μ its mean over the vocabulary, and let P_r and Λ_r hold the top r eigenvectors and eigenvalues of the centered Gram matrix $\sum_v (\bar{E}_v - \mu)(\bar{E}_v - \mu)^\top$. The table is $C_v = \mathrm{normalize}\big(\Lambda_r^{-1/2} P_r^\top (\bar{E}_v - \mu)\big)$, fixed and shared by all layers.

Initialization. Each readout matrix R_ℓ is initialized with $P_r^\top$; the write matrices W_ℓ and the predecessor gate matrix G_p are zero-initialized with $b = 0$, so every injection starts as the identity map.

Losses. Both the backbone output and the last-write refinement output use $\alpha = 0.1$ in Equation (7) and are summed with equal weight. The layer-wise supervision weight is $\lambda = \beta_t \, \mathrm{sg}(\mathrm{CE}_{\mathrm{bb}})$, where $\mathrm{CE}_{\mathrm{bb}}$ is the token cross-entropy of the backbone output and β_t rises linearly from 0 to 0.5 over the first 146 steps. For last-write refinement, the cross-entropy term is computed only where the correct token lies among the $K = 16$ candidates, and the ℓ_1 term only where the candidates hold more than 10^{-4} of the target probability, against the target distribution renormalized to the candidates.

Correct-predecessor curriculum. Each draft block is selected with probability p by one draw shared by layers 1–3, which then use $\sqrt{r}\, C_{y_{t-1}}$ as predecessor input. p is 0.5 for the first sixth of training and decays linearly to zero at one third.

Training setup. All drafters share the same training configuration: the same ShareGPT data with responses regenerated by the target model with thinking disabled, the same draft block of one anchor and 15 candidates, a global batch of 112 sequences, AdamW without weight decay at a peak learning rate of 6×10^{-4} with 4% linear warmup (146 steps) and cosine decay to zero, a maximum training sequence length of 3,072 tokens, and six epochs (3,655 steps). Table 4 lists the remaining DSpine hyperparameters.

Table 4: Remaining DSpine hyperparameters for Qwen3-8B.

Hyperparameter	Value
Draft layers L	5
Hidden size	4096
Target feature layers	$\{1, 9, 17, 25, 33\}$
Transfer-space dimension r	1024
Message dimension a	512
Candidates K	16
Anchors per sequence	512
Gradient-norm clip	1.0
Position decay γ	7
Loss mixing α	0.1

B Details of the Conditioning Experiments

B.1 Setup, Readouts, and Metrics

We use the GSM8K test set and HumanEval with the frozen official Qwen3-8B-DFlash-b16 drafter; greedy Qwen3-8B continuations (thinking disabled, at most 1,024 new tokens) serve as correct tokens. Each block contains one anchor and 15 candidates.

Table 5: Prefix-timing conditions. Schedules list layers 1–5; P/G denotes predicted/correct prefix features. Gains in K_8 are relative to the all-P baseline.

Schedule	GSM8K gain (%)	HumanEval gain (%)
P P P P P	+0.00	+0.00
P G G G G	+6.23	+7.12
P P G G G	+1.87	+3.93
P P P P G	−0.18	+0.26
P G P P G	+4.18	+4.50
P P G P G	+1.06	+3.09
P P P G G	+0.70	+1.73

Table 6: Accuracy gain (percentage points) from correcting only the predecessor. History refers to tokens before the predecessor; positions refer to the original block.

Dataset	History	Position 4	Position 8	Position 12
GSM8K	Predicted	14.88	19.63	23.38
GSM8K	Correct	17.62	31.87	42.50
HumanEval	Predicted	14.02	23.63	19.51
HumanEval	Correct	17.53	38.72	42.53

For layer ℓ, the readout is

$$q_\ell = \mathrm{softmax}(E[n(h_\ell) + U_\ell V_\ell n(h_\ell) + b_\ell]), \tag{9}$$

where E is the frozen target LM head, n is RMS normalization without a trainable scale, and $U_\ell V_\ell$ has rank 64. Each readout is fitted with cross-entropy on 500 GSM8K training questions and selected on another 100, both disjoint from the test data (AdamW, three epochs, batch size 256, learning rate 10^{-3}, weight decay 10^{-4}, 5% warmup, cosine decay).

For T evaluated positions, the number of consecutive correct tokens is

$$K_T = \sum_{j=1}^{T} \prod_{t=1}^{j} \mathbb{I}[\hat{y}_t = y_t], \tag{10}$$

which, unlike τ, compares predictions with fixed correct continuations. We average blocks within each task and then over tasks; panels (b) and (c) of Figure 1 average GSM8K and HumanEval with equal weight.

B.2 Prefix Content and Timing

We fix the correct token x_7 as a new anchor and fill $x_1, \ldots, x_6$ with either draft predictions (P) or correct tokens (G), keeping earlier context unchanged. The target model encodes the context up to x_6, and its fused features enter each draft layer as K/V; layer 1 always receives P features, while layers 2–5 follow the chosen schedule. We evaluate $x_8, \ldots, x_{15}$ and report the relative gain in K_8 over the all-P schedule. Table 5 lists all schedules, and Figure 1(b) shows the average of the two datasets.

B.3 Adjacent Predecessor versus Earlier History

For original-block positions 4, 8, and 12, we use the immediately preceding token as a new anchor; the earlier history and the anchor independently take draft predictions or correct tokens. The history is encoded by the target model and the predecessor enters the drafter through its input embedding, so each evaluated token is the first candidate of its new block. Table 6 reports the gain from correcting only the predecessor; the L1–L5 points in Figure 1(c) average it over the three positions. Correcting the predecessor helps in every setting; at position 12 with correct history, accuracy rises from 51.00% to 93.50% on GSM8K and from 48.78% to 91.31% on HumanEval.

Table 7: L2+L5 minus L4+L5 with correct earlier history. Accuracy differences are in percentage points; K_4 counts at most four consecutive correct candidates, excluding the anchor.

Dataset	Position	Accuracy difference	K_4 difference
GSM8K	4	7.38	0.226
GSM8K	8	17.00	0.565
GSM8K	12	18.63	0.755
HumanEval	4	11.28	0.280
HumanEval	8	15.85	0.694
HumanEval	12	18.60	0.745

B.4 Timing of Predecessor Features

We cache the predecessor's K/V at every layer in two forward passes, one with the predicted and one with the correct predecessor, and substitute the correct K/V only at selected layers. The comparison supplies correct features twice: at layer 5 and at one of layers 2–4. The all-predicted and all-correct schedules reproduce the two native forward passes, so the L1–L5 points in Figure 1(c) equal the content effect in Appendix B.3. Averaged over the three positions, L2+L5 improves accuracy over L4+L5 by 14.33 and 15.24 points on GSM8K and HumanEval with correct history (Table 7), and by 9.63 and 8.99 points with predicted history.

C Experimental Details

Evaluation. Each benchmark uses its complete test set (1,319 GSM8K, 500 MATH-500, 164 HumanEval, 500 MBPP, 1,055 LCB, 80 MT-Bench, and 500 Arena-Hard prompts; MT-Bench runs both turns). All methods receive identical prompts without a system message and generate at most 2,048 new tokens. Acceptance length in Table 1 is measured in SGLang with eight single-GPU instances, each serving up to 16 concurrent requests; at temperature one, tokens are sampled from the full target distribution. Every method proposes 15 draft tokens per round. τ is averaged over the responses of each benchmark, and Avg. is the unweighted mean over benchmarks.

Lossless verification. If the first rejection occurs at position j, we keep $\hat{x}_{1:j-1}$ and sample a replacement from

$$p_j^{\mathrm{corr}}(v) = \frac{[p_j(v) - q_{\mathrm{draft},j}(v)]_+}{\sum_{u\in\mathcal{V}}[p_j(u) - q_{\mathrm{draft},j}(u)]_+}, \qquad [a]_+ = \max(a, 0), \tag{11}$$

where $\mathcal{V}$ is the vocabulary; if all M candidates are accepted, one additional token is sampled from the target. Together with Equation (1), this preserves the target distribution.

Decoding at temperature one. All methods use SGLang's lossless verification. DSpark, DFlash2, and DSpine sample their proposals, the latter two from the softmax of their selector or transition-cache scores over the top-16 candidates of each position, and apply rejection sampling with these proposal probabilities; DFlash and Domino propose greedily and accept a candidate when it matches the token sampled from the target.

Serving. Each instance uses one GPU (tensor parallelism one, BF16, FlashAttention-3 (Shah et al., 2024), CUDA graphs) at temperature zero with at most 2,048 output tokens on the complete GSM8K, MATH-500, HumanEval, and MBPP test sets, with identical prompts in the same order. Throughput is the total number of output tokens divided by the wall-clock time of the whole workload, including prefill and scheduling. All methods use the same static budget of 15 draft tokens per round.

Supervision targets. The variants in Table 3 use the same training data, disable injection, and are trained for three epochs (1,827 steps); token CE decodes every layer through the shared final RMSNorm and the frozen target LM head. All variants are evaluated in SGLang at temperature zero.

Table 8: Throughput (tokens/s) on SGLang with a single GPU at temperature zero; full results for Figure 4. Baseline (AR) rows report autoregressive decoding; other entries report throughput with the speedup over the baseline in green. Bold marks the best result among speculative methods.

Task	Method	Qwen3-8B (concurrency)					Qwen3-4B (concurrency)				
		2	4	8	16	32	2	4	8	16	32
GSM8K	Baseline	381	747	1405	2729	4922	552	1072	2059	3794	6768
	DFlash	1163 3.05×	2191 2.93×	3799 2.70×	5349 1.96×	6327 1.29×	1615 2.92×	2991 2.79×	5250 2.55×	7873 2.08×	9534 1.41×
	Domino	1193 3.13×	2229 2.98×	3884 2.76×	5576 2.04×	6675 1.36×	1626 2.94×	3001 2.80×	5254 2.55×	7909 2.08×	10000 1.48×
	DSpark	1223 3.21×	2280 3.05×	3931 2.80×	5656 2.07×	6789 1.38×	1663 3.01×	3057 2.85×	5296 2.57×	7950 2.10×	10002 1.48×
	DFlash2	1327 3.49×	2473 3.31×	4234 3.01×	5899 2.16×	6975 1.42×	1876 3.40×	3460 3.23×	6015 2.92×	8807 2.32×	10683 1.58×
	DSpine	**1480 3.88×**	**2758 3.69×**	**4704 3.35×**	**6732 2.47×**	**7825 1.59×**	**2018 3.65×**	**3698 3.45×**	**6442 3.13×**	**9527 2.51×**	**11541 1.71×**
MATH-500	Baseline	380	743	1394	2629	4524	550	1061	2012	3577	6107
	DFlash	1218 3.21×	2329 3.13×	4017 2.88×	5581 2.12×	6589 1.46×	1673 3.04×	3144 2.96×	5605 2.79×	8192 2.29×	10115 1.66×
	Domino	1241 3.27×	2345 3.16×	4138 2.97×	5870 2.23×	6927 1.53×	1682 3.06×	3156 2.97×	5584 2.78×	8474 2.37×	10708 1.75×
	DSpark	1260 3.32×	2382 3.21×	4136 2.97×	5914 2.25×	7032 1.55×	1707 3.11×	3220 3.03×	5681 2.82×	8508 2.38×	10454 1.71×
	DFlash2	1403 3.69×	2647 3.56×	4542 3.26×	6322 2.40×	7384 1.63×	1991 3.62×	3691 3.48×	6452 3.21×	9638 2.69×	11453 1.88×
	DSpine	**1522 4.01×**	**2875 3.87×**	**4968 3.56×**	**7030 2.67×**	**8200 1.81×**	**2087 3.80×**	**3906 3.68×**	**6961 3.46×**	**10217 2.86×**	**12367 2.03×**
HumanEval	Baseline	379	735	1362	2541	4355	550	1060	2027	3654	5148
	DFlash	1107 2.92×	2077 2.83×	3595 2.64×	4925 1.94×	5595 1.28×	1512 2.75×	2788 2.63×	4809 2.37×	7326 2.00×	8617 1.67×
	Domino	1084 2.86×	2012 2.74×	3515 2.58×	5065 1.99×	5789 1.33×	1494 2.72×	2784 2.63×	4714 2.33×	7250 1.98×	8757 1.70×
	DSpark	1126 2.97×	2081 2.83×	3618 2.66×	5088 2.00×	5861 1.35×	1524 2.77×	2833 2.67×	4832 2.38×	7319 2.00×	8630 1.68×
	DFlash2	1225 3.23×	2329 3.17×	3876 2.85×	5433 2.14×	6141 1.41×	1755 3.19×	3347 3.16×	5488 2.71×	8334 2.28×	9515 1.85×
	DSpine	**1354 3.57×**	**2541 3.46×**	**4325 3.18×**	**6158 2.42×**	**6963 1.60×**	**1840 3.34×**	**3446 3.25×**	**5874 2.90×**	**8699 2.38×**	**10134 1.97×**
MBPP	Baseline	381	740	1393	2682	4705	549	1062	1996	3525	5828
	DFlash	1157 3.04×	2129 2.88×	3725 2.67×	5289 1.97×	6147 1.31×	1598 2.91×	2980 2.81×	5211 2.61×	7796 2.21×	9265 1.59×
	Domino	1139 2.99×	2117 2.86×	3677 2.64×	5338 1.99×	6302 1.34×	1569 2.86×	2932 2.76×	5063 2.54×	7775 2.21×	9448 1.62×
	DSpark	1176 3.09×	2174 2.94×	3801 2.73×	5501 2.05×	6440 1.37×	1624 2.96×	3023 2.85×	5274 2.64×	7803 2.21×	9515 1.63×
	DFlash2	1275 3.35×	2333 3.15×	4042 2.90×	5667 2.11×	6646 1.41×	1830 3.33×	3376 3.18×	6064 3.04×	8771 2.49×	10287 1.76×
	DSpine	**1399 3.68×**	**2586 3.50×**	**4465 3.21×**	**6411 2.39×**	**7321 1.56×**	**1948 3.55×**	**3673 3.46×**	**6346 3.18×**	**9369 2.66×**	**11072 1.90×**