D-LOOP: LOOPED DIFFUSION DRAFTING FOR SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Block diffusion accelerates speculative decoding by drafting multiple tokens in one forward pass. However, each position predicts a marginal distribution without observing earlier proposed tokens, limiting draft quality and acceptance length. We identify a concrete failure, the *repetition trap*, in which neighboring positions produce redundant copies of the same token. We explain this tendency theoretically and empirically examine its association with shorter accepted drafts. Recent methods refine marginal predictions with an additional causal head or a separately trained drafter, increasing parameter storage and introducing separate training objectives. We instead propose D-Loop, which introduces *intra-block causal conditioning* within the original diffusion drafter without additional model components. Inspired by semi-autoregressive generation and parameter sharing, D-Loop reuses the same backbone across looped passes. The first pass proposes a block, and the second conditions on a selected prefix to regenerate the suffix in parallel. A complementary prefix–suffix objective trains the shared drafter for both anchor-only prefix prediction and prefix-conditioned suffix prediction. Across eight math, code, and chat benchmarks, D-Loop can beat DFlash and DSpark on Qwen3-4B and Qwen3-8B with obvious gains.

1 INTRODUCTION

Large language models generate outputs through a sequence of autoregressive forward passes. Speculative decoding reduces this cost by using a lightweight draft model (*a.k.a.* drafter) to propose several tokens and a stronger target model to verify them in parallel (Leviathan et al., 2023; Chen et al., 2023). Its effectiveness depends on both the cost of producing a proposal and the length of the prefix that survives verification. Compared to a small autoregressive drafter, diffusion drafting offers an appealing way to reduce proposal latency. As a pioneer, DFlash (Chen et al., 2026), inspired by diffusion language models (DLMs), conditions a lightweight block diffusion model on target-model features and predicts a block of masked tokens in one forward pass. This parallel computation permits a deeper drafter than is typically affordable in token-by-token speculation. However, its central modeling limitation is the lack of *intra-block causal conditioning*, *i.e.,* each position predicts a marginal distribution without observing the tokens realized earlier in the same block, causing compromised proposal quality and limiting the accepted draft length.

In this paper, we identify a concrete failure of marginal modeling that we call the *repetition trap*, where neighboring positions produce redundant copies of the same token within a draft block. In Section 2, we provide a theoretical explanation of why marginal predictions can overproduce such repetitions and quantify their prevalence in actual draft outputs. These findings highlight the need to improve marginal draft predictions by accounting for dependencies among proposed tokens.

To this end, several recent methods improve marginal draft predictions by introducing additional refinement modules. For example, DSpark (Cheng et al., 2026) augments a parallel drafter with lightweight causal heads or refiners, while D^2SD (Zhang et al., 2026) introduces a separately trained diffusion drafter for prefix-conditioned redrafting. These approaches improve draft quality but require additional model parameters. Frameworks such as DSpark also combine multiple training objectives. This raises a simple question: *can we alleviate the limitations of marginal modeling within the original drafter, without introducing any additional components?* Such a design would be attractive for memory-constrained deployment by avoiding the storage of a separate refinement

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.

module or a second set of drafter weights. It also motivates a simpler training formulation that supports refinement without auxiliary head-specific objectives.

Notably, an well-established approach to strengthening generative quality of DLMs is semi-autoregressive generation, where each block conditions on previously generated blocks while retaining parallel prediction within a block (Arriola et al., 2025). Such an inter-block causal conditioning is devised without adding a separate model component. Drawing on this principle and parameter sharing in recurrent-depth models (Geiping et al., 2025), we propose D-Loop, a looped diffusion drafter that operates on discrete proposal states. The core idea of D-Loop is to introduce *intra-block causal conditioning* through looped shared-parameter forward passes. An earlier pass proposes a token block, and a later pass conditions on a selected prefix to regenerate the suffix in parallel. Both passes reuse the full drafter with the same parameters.

Our preliminary experiments found that the vanilla full-masking loss adopted by DFlash and DSpark is insufficient to enable looped forward passes. To make one drafter effective at both initial proposal and prefix-conditioned refinement, we develop a *complementary prefix–suffix training objective*. An anchor-only view supervises the prefix, and a revealed-prefix view supervises the suffix. Both views update shared parameters under one normalized objective. Despite an additional looped forward pass with a double drafting time (see Table 6), D-Loop still achieves better speedup performance (see Tables 3 and 4) due to its increased acceptance length compared to DFlash and DSpark.

Our contributions are threefold:

- We identify a *repetition trap* in block diffusion drafting for speculative decoding and provide a theoretical explanation of how marginal modeling can produce it. We quantify repetition in actual draft outputs and examine its association with shorter accepted drafts.
- Without adding a causal head or a second drafter, we propose D-Loop, which introduces *intra-block causal conditioning* within the diffusion drafter itself through shared-parameter looped passes. We train both passes with a *complementary prefix–suffix objective*.
- Experiments on math, code, and chat tasks with Qwen3-4B and Qwen3-8B demonstrate D-Loop's effectiveness across model scales and task domains.

2 Theoretical Analyses and Empirical Observations

By examining actual token proposals from marginal block diffusion drafting, we identify a systematic failure: the *repetition trap*, in which multiple positions emit the same token consecutively. We characterize this phenomenon through empirical observations and theoretical analysis.

Notation. Let C denote the verified context together with its available target features and a known anchor token x_0. A draft block contains $B = m + 1$ positions: the anchor and m proposed tokens $X_{1:m}$. Let $p(x_{1:m} \mid C)$ be the target continuation distribution.

Marginal block drafting. Consider a block diffusion drafter that predicts all masked proposal positions in parallel from an anchor-only state. With independent token sampling, its output distribution factorizes as

$$q_\theta^{(0)}(x_{1:m} \mid C) = \prod_{k=1}^{m} q_{\theta,k}(x_k \mid C, z^{(0)}), \qquad z^{(0)} = (x_0, [\textsc{Mask}], \ldots, [\textsc{Mask}]). \tag{1}$$

DFlash (Chen et al., 2026) instantiates this factorization by generating the entire candidate block in a single drafter forward pass before target verification. We refer to this setting as *one-pass block diffusion drafting*: one denoising pass fills all masked proposal positions, with no further refinement of the proposed tokens. DSpark (Cheng et al., 2026) also retains a single parallel backbone pass, but adds a sequential head whose token conditioning goes beyond Equation 1.

In Equation 1, each factor predicts a positional marginal without observing the tokens realized at other positions, even when the logits share attention and hidden representations. Prior work discusses this missing dependence and its consequences for draft acceptance. However, consecutive repetition within raw draft blocks has not been specifically characterized in these studies.

Table 1: DFlash repetition statistics and illustrative examples. Acc. Len. uses Qwen3-8B at $T = 1$ (Table 4). LCB denotes LiveCodeBench.

Dataset	Adjacent repeat	Mean longest run	Content-repeat blocks	Acc. Len.
GSM8K	17.4%	2.5	72.5%	5.28
LCB	29.4%	3.6	84.9%	3.26
MT-Bench	42.6%	4.7	94.5%	3.10

Type 1: Position confusion
Gold: `how many laps Beth`
Draft: `how many` **`laps laps`** `Beth`

Type 2: Endless
Draft: `subtracting` **`emails emails ...`**

An observed failure: the repetition trap. We observe that multiple positions in a single draft block can emit the same token consecutively, forming long repeated runs. We call this behavior the *repetition trap*. Table 1 quantifies this pattern in raw DFlash proposals. MT-Bench exhibits a higher adjacent-repeat rate than GSM8K (42.6% versus 17.4%) and longer repeated runs (4.7 versus 2.5 tokens), with LiveCodeBench between them on both metrics. Consecutive repeated content words occur in 72.5%–94.5% of blocks, showing that the phenomenon extends beyond whitespace repetition. The examples illustrate two forms: *position confusion* duplicates a word before continuing, while *endless* repetition persists through the shown suffix. This observation identifies a concrete failure associated with missing output dependencies. To our knowledge, this is the first report specifically documenting this phenomenon in block diffusion drafting for speculative decoding. Note that repetition has also been observed in non-autoregressive neural machine translation (Gu et al., 2018; Ran et al., 2020). We analyze its origin in marginal drafting and its cost under prefix verification.

Why marginal predictions can overproduce repetitions. For a fixed context c, write $p_c = p(X_{1:m} \mid c)$. Even exact marginals leave a product-approximation gap (Huang et al., 2022):

$$\min_{q=\prod_k q_k} D_{\mathrm{KL}}(p_c \| q) = \sum_k H(X_k \mid c) - H(X_{1:m} \mid c) = \sum_k I(X_k; X_{<k} \mid c). \tag{2}$$

The optimum is $q^\star = \prod_k p(X_k \mid c)$: each cross-entropy term is minimized by its target marginal, and the entropy chain rule gives the last equality. Each mutual-information term measures how much observing the realized prefix reduces uncertainty about the next token. Marginal drafting discards precisely this information, including *whether a word has already been emitted, leading to a potential repetition risk.* To make this comparison explicit, let $\pi_k(v) = p(X_k = v \mid c) > 0$. We compare the probability of the adjacent repetition $X_k = X_{k+1} = v$ under the independent-marginal distribution $q^\star$ with that under the target joint distribution p. Their difference is

$$\begin{aligned} \Delta_{k,v} &= q^\star(X_k = X_{k+1} = v \mid c) - p(X_k = X_{k+1} = v \mid c) \\ &= \underbrace{\pi_k(v)\pi_{k+1}(v)}_{\text{independent marginals}} - \underbrace{\pi_k(v)p(X_{k+1} = v \mid X_k = v, c)}_{\text{target joint distribution}} \\ &= \pi_k(v)\big[\pi_{k+1}(v) - p(X_{k+1} = v \mid X_k = v, c)\big]. \end{aligned} \tag{3}$$

The first product uses the next position's marginal probability, without observing whether the preceding position selected v. The second product is the target joint probability written using the chain rule: it conditions the next position on $X_k = v$, retaining the target's adjacent-token dependence. As we can see, whenever observing v lowers the probability of immediately emitting it again below $\pi_{k+1}(v)$, we have $\Delta_{k,v} > 0$. This reveals that independent marginal predictions assign a higher probability to the adjacent repetition (vv) at positions k and $k+1$ than the target joint distribution does, even when the marginals are exact. Thus, uncertainty about where a word belongs can become redundant copies when positions cannot condition on realized tokens.

Implications for causal drafting. Speculative verification accepts only a contiguous prefix. The repetition trap can therefore compound suffix decay in accepted drafts: an erroneous repeat can trigger early rejection and discard every subsequent proposal.

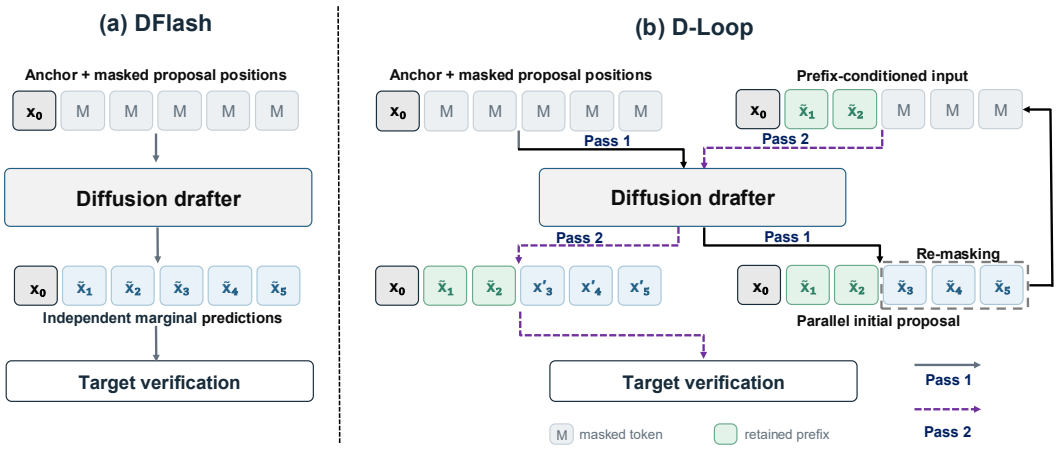


Figure 1: **Overview of DFlash and D-Loop.** (a) DFlash predicts all proposal tokens in one parallel pass. (b) D-Loop retains a selected prefix from the first pass, remasks the suffix, and invokes the same drafter to regenerate the suffix conditioned on the retained tokens. Both passes share all draft parameters and use target features from the verified context. Target verification follows the second pass. One selected prefix is shown for clarity.

3 D-Loop: Looped Diffusion Drafting

Overall. Instead of using any extra causal head, we propose D-Loop, which leverages intra-block causal conditioning within the diffusion drafter itself. An initial forward pass instantiates a provisional token prefix. Then, a second pass uses that prefix to regenerate the suffix through the same backbone. The loop operates on discrete proposal states and reuses DFlash's target-feature conditioning and token prediction interface. Figure 1 contrasts the two drafting procedures.

3.1 From One-Pass Marginals to a Prefix-Conditioned Loop

Block diffusion (Arriola et al., 2025) follows a semi-autoregressive factorization: it generates blocks sequentially, conditioning each block on previously generated blocks, while using diffusion to predict tokens in parallel within a block. Inspired by this block-causal structure, we introduce a similar dependence *inside a single speculative block*. An earlier pass proposes a prefix, and a later pass conditions on that prefix to regenerate the suffix. This yields *intra-block causal conditioning* while retaining parallel prediction within each stage.

Starting from $z^{(0)}$ in Equation 1, the two passes reuse the same drafter:

$$z^{(0)} \xrightarrow{q_\theta} \widetilde{x}_{1:m}, \qquad \underbrace{(x_0, \widetilde{x}_{1:\ell}, [\textsc{Mask}], \ldots, [\textsc{Mask}])}_{z^{(\ell)}(\widetilde{x}_{1:\ell})} \xrightarrow{q_\theta} y_{\ell+1:m}. \tag{4}$$

The first pass produces all m proposal tokens $\widetilde{x}_{1:m}$ in parallel. We select a length $\ell \in \{0, \ldots, m-1\}$ and retain the first ℓ tokens as $\widetilde{x}_{1:\ell}$. The second input $z^{(\ell)}(\widetilde{x}_{1:\ell})$ contains the known anchor x_0, this visible prefix, and $m - \ell$ mask tokens. The second pass predicts a new suffix $y_{\ell+1:m}$ in parallel, yielding the final proposal $y = (\widetilde{x}_{1:\ell}, y_{\ell+1:m})$. Both arrows denote a drafter forward pass followed by token selection, conditioned on the same verified context C. The superscript (ℓ) counts visible proposal tokens, excluding the anchor.

Selecting candidate prefixes. To select where to truncate $\widetilde{x}_{1:m}$, we adopt the confidence-guided top-K prefix selection (Zhang et al., 2026). Let $c_k = q_{\theta,k}(\widetilde{x}_k \mid C, z^{(0)})$ be the confidence assigned to the proposed token at position k. Each prefix length is scored by $r_\ell = \left(\prod_{k=1}^{\ell} c_k\right)(1 - c_{\ell+1}), \quad \ell \in \{0, \ldots, m-1\}$. This score favors a confident prefix followed by a low-confidence token, identifying a candidate boundary for suffix regeneration. We select the top-K lengths by r_ℓ.

For each selected ℓ, we retain exactly the first ℓ tokens of the original proposal, $\widetilde{x}_{1:\ell} = (\widetilde{x}_1, \ldots, \widetilde{x}_\ell)$, and mask positions $\ell + 1$ through m. The second pass then regenerates the suffix from $z^{(\ell)}(\widetilde{x}_{1:\ell})$.

Alleviating the repetition trap. With $K = 1$, the second pass reduces adjacent-repeat rates by 73.7%, 83.0%, and 79.8% on GSM8K, LCB, and MT-Bench (Table 2). In the observed runs, repeated tokens typically have lower confidence than the preceding token. Hence, repetition tends to leave them in the suffix for regeneration. The visible prefix lets the drafter account for tokens already emitted, reducing redundant copies. Correcting such repeats can avoid early rejection and preserve acceptance of later tokens. These results show that the loop mitigates the failure in Section 2, supporting repetition reduction as one mechanism behind longer accepted drafts.

Table 2: Adjacent-repeat rates.

Dataset	Pass 1	Pass 2
GSM8K	19.4%	**5.1%**
LCB	28.8%	**4.9%**
MT-Bench	44.5%	**9.0%**

For a fixed split selected independently of proposal tokens, the induced two-stage proposal distribution has the form

$$q_\theta^{(\ell)}(x_{1:m} \mid C) = \underbrace{\prod_{k=1}^{\ell} q_{\theta,k}(x_k \mid C, z^{(0)})}_{\text{parallel prefix}} \underbrace{\prod_{k=\ell+1}^{m} q_{\theta,k}(x_k \mid C, z^{(\ell)}(x_{1:\ell}))}_{\text{prefix-conditioned parallel suffix}}. \tag{5}$$

The first product is the probability of the retained prefix $x_{1:\ell}$ under the first-pass predictions. The second is the probability of the regenerated suffix $x_{\ell+1:m}$ given that same prefix as visible input. Thus, the final proposal combines the first-pass prefix with the second-pass suffix, replacing the original suffix. Each stage predicts its tokens in parallel, but the suffix now conditions on the prefix.

Intra-block causal conditioning. Equation 5 creates the directed dependence $X_{1:\ell} \to X_{\ell+1:m}$ through revealed tokens. It remains parallel within the prefix and within the regenerated suffix. The causal dependence comes from making proposed prefix tokens visible to the next pass through hard-token re-embedding. DFlash's bidirectional attention is retained over the visible prefix and masked suffix. Meanwhile, no additional token-level causal attention mask is required. Both stages belong to the same draft block, whose tokens remain provisional until target verification.

Parameter reuse versus a new sequential head. DSpark supplies realized-token dependencies through a separate Markov or RNN head (Cheng et al., 2026). D-Loop instead invokes its existing backbone again on a different visible state. This shares the motivation of recurrent-depth computation (Geiping et al., 2025), but does not recur over a continuous hidden state or assume arbitrary-depth extrapolation. Both passes can use one in-memory copy of the draft weights. A theoretical comparison with one-pass marginal drafting is provided in Appendix A.1.

3.2 Complementary Prefix–Suffix Training

Let $x_{1:m}$ be a training continuation generated by the target model. For each training block, we draw a prefix length $\ell \sim \pi_\beta$ to determine how many ground-truth proposal tokens to reveal in the suffix view. The sampling probability is

$$\pi_\beta(\ell) = \frac{(1-\beta)\beta^\ell}{1-\beta^m}, \qquad \ell \in \{0, \ldots, m-1\}, \quad 0 < \beta < 1. \tag{6}$$

Here $\pi_\beta(\ell)$ is the probability of selecting length ℓ, and β controls how quickly this probability decreases with length. Smaller β favors shorter prefixes more strongly. As β approaches one, the distribution becomes nearly uniform over the allowed lengths. Sampling from this distribution trains the drafter under varied amounts of visible prefix information. It is a training distribution, not an estimate of target acceptance or the inference-time selection rule.

Each sampled block yields two views. The *suffix view* exposes the ground-truth anchor and prefix, supervising only the masked suffix. The *prefix view* exposes only the anchor, supervising the complementary prefix positions. Write

$$\begin{aligned} h_{ik}^S &= -\log q_{\theta,k}(x_{ik} \mid C_i, z_i^{(\ell_i)}(x_{i,1:\ell_i})), & h_{ik}^P &= -\log q_{\theta,k}(x_{ik} \mid C_i, z_i^{(0)}), \\ w_{ik}^S &= v_{ik}\mathbf{1}\{k > \ell_i\} e^{-(k-\ell_i-1)/\tau_s}, & w_{ik}^P &= v_{ik}\mathbf{1}\{1 \le k \le \ell_i\} e^{-(k-1)/\gamma}, \end{aligned} \tag{7}$$

where i indexes sampled blocks and v_{ik} excludes invalid or padded positions. The positive scales τ_s and γ control loss decay within the suffix and prefix views, respectively. The anchor is never supervised. We optimize the minibatch objective

$$\mathcal{L}_{\text{comp}}(\theta) = \frac{\sum_i \sum_{k=1}^{m} \left(w_{ik}^P h_{ik}^P + w_{ik}^S h_{ik}^S \right)}{\sum_i \sum_{k=1}^{m} (w_{ik}^P + w_{ik}^S) + \epsilon}. \tag{8}$$

Here $\epsilon > 0$ is a small numerical constant that prevents division by zero when the total valid weight vanishes. This is one weighted cross-entropy objective over complementary prediction conditions. The prefix view does *not* reveal the suffix. For $\ell = 0$, it contributes no supervised positions and the suffix view reduces to standard anchor-only drafting. The short-prefix fallback used in our implementation is specified in Appendix A.3.

Why complementarity matters. In the basic paired construction, every valid position is supervised once per sampled block: either under the anchor-only condition or under the revealed-prefix condition. Without the prefix term, the training signal would emphasize how to *use* a prefix without directly covering how to *create* its tokens in that paired example. With uniform weights, the numerator for one example is exactly the negative log-likelihood of Equation 5, evaluated with a teacher-forced prefix. The actual position weights prioritize the start of each stage. As a result, Equation 8 is a weighted surrogate rather than an unweighted maximum-likelihood objective.

3.3 Inference and Verification

Algorithm 1 illustrates one greedy decoding cycle with $K = 1$. For $K > 1$, we conduct joint verification: one target forward processes all branches using the shared context KV cache and branch-specific causal attention. We commit the longest matching prefix and a target correction or bonus token, retaining the accepted KV entries. Due to the usage of shared context KV cache and branch-specific causal attention, the cost of joint verification is governable under an appropriate number of branches. For example, Table 6 reports 25.604 ms verification latency at $K = 4$, versus 25.363 ms for single-branch DFlash (only 0.95% higher).

Algorithm 1 D-Loop inference with greedy verification ($K = 1$, one decoding cycle)

Require: Context C, target T, shared drafter q_θ, proposal length m
Ensure: Verified token extension U

$Q^{(0)} \leftarrow q_\theta(\cdot \mid C, z^{(0)})$ ▷ anchor-only pass
$\widetilde{x}_{1:m} \leftarrow \text{Argmax}(Q^{(0)})$, $c_k \leftarrow Q_k^{(0)}(\widetilde{x}_k)$ ▷ parallel proposals
$\ell \leftarrow \arg\max_{0 \le j < m} r_j$ ▷ select one prefix us
$Q^{(\ell)} \leftarrow q_\theta(\cdot \mid C, z^{(\ell)}(\widetilde{x}_{1:\ell}))$ ▷ prefix-conditioned pass
$y \leftarrow (\widetilde{x}_{1:\ell}, \text{Argmax}(Q_{\ell+1:m}^{(\ell)}))$ ▷ retain prefix, replace suffix
$(a, b) \leftarrow \text{VerifyGreedy}(T, C, y)$ ▷ accepted length and target token
if $a < m$ **then**
    $U \leftarrow (y_{1:a}, b)$ ▷ target correction at rejection
else
    $U \leftarrow (y_{1:m}, b)$ ▷ target bonus after full acceptance
end if
return U ▷ truncate at EOS or the output-token limit

4 Experiments

Models and Evaluations. We conduct experiments with Qwen3-4B and Qwen3-8B (Yang et al., 2025) as target models. For each benchmark, we report average acceptance length (τ) and end-to-end decoding speedup over the vanilla autoregressive baseline. We count the target-generated correction or bonus token in τ, so it measures the average number of tokens committed per decoding cycle. The speedup is

$$\eta = \frac{L_{\text{target}}}{L} = \frac{\alpha\, L_{\text{target}}}{T_{\text{draft}} + T_{\text{verify}}}. \tag{9}$$

Here L_{target} and L are the average per-token decoding latencies of the autoregressive baseline and speculative decoding, respectively. The quantity $\alpha = \tau$ denotes the average tokens committed per

cycle. T_{draft} and T_{verify} are the corresponding average drafting and verification times per cycle. For D-Loop, costs include both draft passes, branch preparation, verification, and state updates.

Datasets. We train the drafter on PerfectBlend (Xu et al., 2024) prompts with responses regenerated by the target model to align the training continuations with its output distribution. We evaluate on eight benchmarks covering three domains. Mathematical reasoning includes GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2023), and AIME25 (OpenCompass, 2025). Code generation includes HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2024). Conversational tasks include MT-Bench (Zheng et al., 2023) and Alpaca (Taori et al., 2023).

Implementation. By following (Chen et al., 2026), we use a five-layer DFlash drafter with a block size of $B = 16$. For Qwen3-4B and Qwen3-8B, the drafters are initialized from Chen et al. (2026)'s pretrained checkpoints. To be fair, D-Loop and all baselines are re-trained with 5 epochs using an initial learning rate of 2e-4, and the evaluation is conducted using the final checkpoint with a batch size of 1. We set $\beta = 0.7$ and the training loss decay scales to $\tau_s = \gamma = 7$. At inference, both passes share the same drafter weights, and we select $K = 4$ candidate prefixes for suffix regeneration. More details can be found in Appendix A.3.

Baselines. We compare D-Loop with vanilla autoregressive decoding and three representative speculative decoding methods: EAGLE-3 (Li et al., 2025), DFlash (Chen et al., 2026), and DSpark (Cheng et al., 2026). EAGLE-3 uses autoregressive drafting with target-model features. DFlash generates a block in one parallel diffusion pass. DSpark adds a sequential head to a parallel diffusion drafter to introduce token dependencies. These baselines cover autoregressive, parallel, and semi-autoregressive drafting. We also report a separate comparison with published D^{2}SD results (Zhang et al., 2026). D^{2}SD uses DFlash for the initial proposal and a separately trained variable-prefix drafter for suffix regeneration.

Table 3: Average acceptance length (τ) and wall-clock speedup over autoregressive decoding on Qwen3-4B under greedy ($T = 0$) and sampling ($T = 1$) decoding. DFlash, DSpark, and D-Loop results are from our runs. EAGLE-3 results are quoted from Table 1 of Chen et al. (2026) with tree size 60. Bold indicates the best result among the methods measured in our runs.

	Greedy ($T = 0$)								**Sampling** ($T = 1$)							
	Acc. Len.				Speedup ($\times$)				Acc. Len.				Speedup ($\times$)			
Dataset	DFlash	EAGLE-3	DSpark	D-Loop	DFlash	EAGLE-3	DSpark	D-Loop	DFlash	EAGLE-3	DSpark	D-Loop	DFlash	EAGLE-3	DSpark	D-Loop
GSM8K	8.51	3.77	8.83	**10.20**	6.41	2.27	6.62	**6.77**	6.39	3.68	6.83	**8.96**	4.82	2.12	5.12	**5.94**
MATH-500	8.39	3.52	8.59	**9.84**	6.32	2.10	6.44	**6.53**	5.86	3.44	6.26	**7.86**	4.42	1.97	4.69	**5.21**
AIME25	7.93	3.51	7.63	**9.09**	5.98	2.13	5.72	**6.03**	3.87	3.20	4.24	**5.46**	2.92	1.83	3.18	**3.62**
HumanEval	6.60	3.47	6.88	**7.87**	4.97	2.12	5.16	**5.22**	5.05	3.39	5.33	**7.03**	3.81	1.94	3.99	**4.66**
MBPP	6.14	3.38	6.44	**7.43**	4.63	2.02	4.83	**4.93**	4.63	3.33	4.89	**6.75**	3.49	1.92	3.67	**4.48**
LiveCodeBench	6.26	3.22	6.40	**7.34**	4.72	1.90	4.80	**4.87**	3.75	3.14	4.01	**5.16**	2.83	1.82	3.01	**3.42**
MT-Bench	4.64	3.49	4.72	**5.47**	3.50	2.04	3.54	**3.63**	3.57	3.36	3.70	**4.82**	2.69	1.91	2.77	**3.20**
Alpaca	3.33	—	3.35	**3.76**	**2.51**	—	**2.51**	2.49	2.54	—	2.72	**3.52**	1.91	—	2.04	**2.33**
Average	6.48	3.48	6.61	**7.63**	4.88	2.08	4.95	**5.06**	4.46	3.36	4.75	**6.20**	3.36	1.93	3.56	**4.11**

4.1 Main Results

Tables 3 and 4 report Qwen3-4B and Qwen3-8B results under greedy ($T = 0$) and sampling ($T = 1$) decoding. We measure DFlash, DSpark, and D-Loop in our experiments. The EAGLE-3 results are quoted from DFlash (Chen et al., 2026) as a cross-study reference.

On Qwen3-4B, D-Loop achieves the longest acceptance length among the methods evaluated in our runs on all eight datasets at both temperatures, with average speedups of $5.06\times$ under greedy decoding and $4.11\times$ under sampling. Its relative improvement in average speedup over DSpark

Table 4: Average acceptance length (τ) and wall-clock speedup over autoregressive decoding on Qwen3-8B under greedy ($T = 0$) and sampling ($T = 1$) decoding. DFlash, DSpark, and D-Loop results are from our runs. EAGLE-3 results are quoted from Table 1 of Chen et al. (2026) with tree size 60. Bold indicates the best result among the methods measured in our runs.

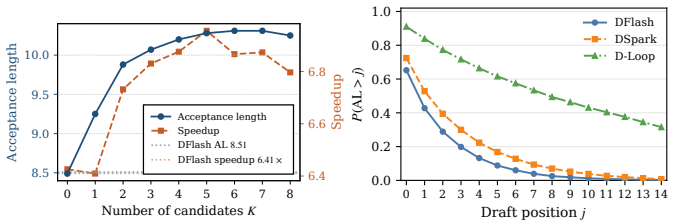
	Greedy ($T = 0$)								**Sampling** ($T = 1$)							
	Acc. Len.				Speedup (×)				Acc. Len.				Speedup (×)			
Dataset	DFlash	EAGLE-3	DSpark	D-Loop	DFlash	EAGLE-3	DSpark	D-Loop	DFlash	EAGLE-3	DSpark	D-Loop	DFlash	EAGLE-3	DSpark	D-Loop
GSM8K	7.35	3.71	7.42	**9.46**	5.58	2.23	5.63	**6.61**	5.28	3.59	5.74	**8.24**	4.01	2.07	4.36	**5.76**
MATH-500	7.62	3.49	8.04	**9.97**	5.79	2.05	6.10	**6.96**	5.21	3.38	5.71	**7.79**	3.96	1.94	4.34	**5.44**
AIME25	7.07	3.44	7.32	**8.69**	5.37	2.05	5.56	**6.07**	3.62	3.18	3.93	**5.41**	2.75	1.84	2.98	**3.78**
HumanEval	5.99	3.65	6.36	**7.41**	4.55	2.17	4.83	**5.18**	4.14	3.54	4.59	**6.36**	3.15	2.05	3.48	**4.44**
MBPP	5.19	3.25	5.77	**6.39**	3.94	1.93	4.38	**4.46**	3.87	3.16	4.28	**5.67**	2.94	1.85	3.25	**3.96**
LiveCodeBench	5.39	3.03	5.88	**6.75**	4.10	1.81	4.46	**4.71**	3.26	2.92	3.55	**4.73**	2.48	1.72	2.70	**3.30**
MT-Bench	4.34	3.26	4.31	**5.20**	3.30	1.90	3.27	**3.63**	3.10	3.05	3.31	**4.53**	2.36	1.70	2.51	**3.16**
Alpaca	3.21	—	3.16	**3.67**	2.44	—	2.40	**2.56**	2.49	—	2.50	**3.39**	1.89	—	1.90	**2.37**
Average	5.77	3.40	6.03	**7.19**	4.38	2.02	4.58	**5.02**	3.87	3.26	4.20	**5.77**	2.94	1.88	3.19	**4.03**

grows from 2.2% at $T = 0$ to 15.4% at $T = 1$, showing a larger benefit under sampling. Greedy Alpaca is the only speedup exception (2.49× versus DSpark's 2.51×), where longer acceptance does not fully offset the additional drafting cost.

On Qwen3-8B, D-Loop improves both acceptance length and speedup over DFlash and DSpark on every dataset at both temperatures. It achieves average acceptance lengths of 7.19 and 5.77, with average speedups of 5.02× and 4.03× under greedy and sampling decoding, respectively. The relative gain in average speedup over DSpark increases from 9.7% at $T = 0$ to 26.2% at $T = 1$, consistent with the larger sampling gains observed on Qwen3-4B. Longer accepted drafts thus offset the additional pass across math, code, and chat tasks.

4.2 Analysis of Drafting Efficiency

Figure 2(a,b) examines drafting efficiency on GSM8K using Qwen3-4B. (a) **A small candidate budget already improves performance.** Increasing K beyond the single-pass setting ($K = 0$) improves acceptance until it saturates, with speedup peaking at $K = 5$. Gains in both metrics are already evident at $K = 2$ and $K = 3$; (b) **D-Loop retains more accepted tokens toward the suffix.** It maintains a higher and more slowly decaying $P(\mathrm{AL} > j)$ than DFlash and DSpark. The separation persists toward the end of the block, indicating longer useful prefixes. This cumulative survival measure reflects both early rejections and later-token quality, rather than isolated per-position accuracy.



(a) Candidate budget. (b) Acceptance by position.

Figure 2: Qwen3-4B on GSM8K: (a) acceptance length and speedup versus K; (b) acceptance prob. over position j.

Table 5 compares D-Loop with published D^{2}SD results and ablates complementary prefix–suffix training on Qwen3-8B. Some obvervations can be found as follows: (a) **A single drafter with a looped structure is sufficient.** D-Loop uses one five-layer drafter, compared with two in D^{2}SD (which equals to 10-layer drafter). However, D-Loop can beat D^{2}SD in most cases, leading to higher average acceptance and speedup performance. (b) **The complementary prefix–suffix objective is effective.** With the same architecture, suffix-only training supervises prefix-conditioned prediction, while the full loss also trains anchor-only prefix prediction for the initial proposal. The full loss raises mean acceptance from 7.95 to 8.11 at $T = 0$ and from 6.26 to 6.37 at $T = 1$. Mean speedups

improve from $5.60\times$ to $5.67\times$ and from $4.40\times$ to $4.45\times$, respectively. Acceptance improves on five of six datasets at $T = 0$ and four at $T = 1$.

Table 5: Qwen3-8B comparison and loss ablation (acceptance length / speedup in $\times$). Bold marks metric maxima. D-Loop-suffix-loss uses suffix supervision only. D-Loop uses the full complementary prefix–suffix loss. D^{2}SD results are from Zhang et al. (2026), Table 3 ($\gamma = 16$, $K = 4$), using two five-layer drafters ($5 + 5 = 10$). Our two variants each share one five-layer drafter.

Decoding	Method	# Draft Layers	GSM8K	MATH	AIME25	HumanEval	MBPP	LCB	Average
Greedy ($T = 0$)	D^{2}SD	10	9.21 / 6.54	8.56 / 6.03	7.26 / 5.10	**7.47 / 5.37**	**6.96 / 5.03**	6.44 / 4.62	7.65 / 5.45
	D-Loop-suffix-loss	5	9.12 / 6.43	9.62 / 6.78	8.57 / 6.04	7.34 / 5.18	6.36 / 4.49	6.68 / 4.67	7.95 / 5.60
	D-Loop-full-loss	5	**9.46 / 6.61**	**9.97 / 6.96**	**8.69 / 6.07**	7.41 / 5.18	6.39 / 4.46	**6.75 / 4.71**	**8.11 / 5.67**
Sampling ($T = 1$)	D^{2}SD	10	7.38 / 5.32	6.81 / 4.90	**5.75 / 4.11**	5.50 / 4.05	4.88 / 3.24	**4.90 / 3.53**	5.87 / 4.19
	D-Loop-suffix-loss	5	8.23 / **5.80**	7.60 / 5.30	5.48 / 3.87	6.07 / 4.28	5.59 / 3.94	4.58 / 3.23	6.26 / 4.40
	D-Loop-full-loss	5	**8.24** / 5.76	**7.79 / 5.44**	5.41 / 3.78	**6.36 / 4.44**	**5.67 / 3.96**	4.73 / 3.30	**6.37 / 4.45**

Table 6 shows that **longer acceptance offsets the additional drafting cost.** At $K = 4$, drafting time rises from 4.208 to 9.428 ms per iteration, while verification stays near 25.5 ms. Longer acceptance spreads this verification cost over more committed tokens, yielding higher end-to-end speedup than DFlash and DSpark. Thus, the extra forward pass improves speedup when its acceptance gain offsets the additional drafting latency.

Table 6: Latency on GSM8K with Qwen3-8B. t_{ar}, t_{draft}, and t_{verify} mean the time consumption for autoregressive token-by-token decoding, drafting, and target verification.

Method	t_{ar} ms/tok	t_{draft} ms/iter	t_{verify} ms/iter	AL	Speedup
DFlash	22.29	4.208	25.363	5.28	3.98$\times$
DSpark	22.29	4.208	25.531	5.74	4.30$\times$
D-Loop	22.29	9.428	25.604	**8.24**	**5.24**$\times$

5 Related Work

Speculative drafting and refinement. Speculative decoding verifies inexpensive proposals with the target (Leviathan et al., 2023; Chen et al., 2023). EAGLE-3 (Li et al., 2025) uses target features for autoregressive drafting, while DFlash (Chen et al., 2026) predicts blocks in parallel. DSpark (Cheng et al., 2026) and Domino (Huang et al., 2026) add lightweight causal modules that condition corrections on preceding tokens. xPress (Wang et al., 2026) performs this refinement through parallel Jacobi updates. These methods assign causal refinement to an additional module. D^{2}SD (Zhang et al., 2026) selects candidate prefixes using token confidence and regenerates their suffixes with a separate variable-prefix drafter. It also evaluates direct DFlash reuse. D-Loop trains one shared backbone for both initial proposal and redrafting through complementary prefix–suffix supervision, exposing the proposed prefix to the full drafter during refinement.

Dependencies in parallel generation. Non-autoregressive translation studies repetition and multi-modality errors (Gu et al., 2018; Ran et al., 2020). We characterize repetition in speculative drafts, where early rejection also discards the suffix. Block diffusion (Arriola et al., 2025) conditions on preceding blocks while retaining parallel prediction within each block. D-Loop applies this principle within a speculative block. Recurrent-depth models (Geiping et al., 2025) share parameters across latent computation steps. Our loop conditions on discrete proposal tokens, adding computation without a second set of drafter weights.

6 Conclusion

We identify and analyze a repetition trap in marginal diffusion drafting. D-Loop introduces intra-block causal conditioning by reusing one drafter for initial proposal and prefix-conditioned suffix generation, supported by complementary prefix–suffix training. Experiments on Qwen3-4B and Qwen3-8B show longer accepted drafts and improved decoding speed across math, code, and chat tasks. These results support improving draft dependencies without a separate refinement model.

Limitations and future work. Our fixed looped schedule may be inefficient when acceptance is much short. On greedy Alpaca with Qwen3-4B, the acceptance gain does not offset the extra pass, slightly reducing speedup. Future work could adapt the number of passes to expected acceptance gains and latency, skipping refinement when its return is low and adding passes when worthwhile.

AI USE STATEMENT

In this work, we used generative AI tools for writing revision. We have not used generative AI tools for generating synthetic data sets, helping develop theoretical models or conceptual frameworks. Additionally, we used generative AI tools for checking the correctness of the theoretical formulation and polishing the paper. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models, 2025. URL https://arxiv.org/abs/2503.09573v3. arXiv:2503.09573v3.

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

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding, 2026. URL https://arxiv.org/abs/2602.06036v2. arXiv:2602.06036v2.

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. URL https://arxiv.org/abs/2107.03374.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, Qinyu Chen, Chengqi Deng, Jiping Yu, Damai Dai, Zhengyan Zhang, Yixuan Wei, Yixuan Tan, Wenkai Yang, Runxin Xu, Yu Wu, Zhean Xu, Xuanyu Wang, Muyang Chen, Rui Tian, Xiao Bi, Zhewen Hao, Shaoyuan Chen, Huanqi Cao, Wentao Zhang, Anyi Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. DSpark: Confidence-Scheduled Speculative Decoding with Semi-Autoregressive Generation, 2026. URL https://arxiv.org/abs/2607.05147v1. arXiv:2607.05147v1.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach, 2025. URL https://arxiv.org/abs/2502.05171v2. arXiv:2502.05171v2.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. Non-Autoregressive Neural Machine Translation, 2018. URL https://arxiv.org/abs/1711.02281. arXiv:1711.02281.

Fei Huang, Tianhua Tao, Hao Zhou, Lei Li, and Minlie Huang. On the Learning of Non-Autoregressive Transformers. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9356–9376. PMLR, 2022. URL https://proceedings.mlr.press/v162/huang22k.html.

Jianuo Huang, Yaojie Zhang, Qituan Zhang, Hao Lin, Hanlin Xu, and Linfeng Zhang. Domino: Decoupling Causal Modeling from Autoregressive Drafting in Speculative Decoding, 2026. URL https://arxiv.org/abs/2605.29707v1. arXiv:2605.29707v1.

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. URL https://arxiv.org/abs/2403.07974.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding, 2023. URL https://arxiv.org/abs/2211.17192. arXiv:2211.17192.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test, 2025. URL https://arxiv.org/abs/2503.01840.

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. URL https://arxiv.org/abs/2305.20050.

OpenCompass. AIME 2025 Dataset. Hugging Face, 2025. URL https://huggingface.co/datasets/opencompass/AIME2025. Includes AIME 2025-I and AIME 2025-II. Accessed: 2026-09-26.

Qiu Ran, Yankai Lin, Peng Li, and Jie Zhou. Learning to Recover from Multi-Modality Errors for Non-Autoregressive Neural Machine Translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 3059–3069. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.277. URL https://aclanthology.org/2020.acl-main.277/.

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. GitHub repository, 2023. URL https://github.com/tatsu-lab/stanford_alpaca.

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, 2026. URL https://arxiv.org/abs/2608.02438v1. arXiv:2608.02438v1.

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, 2024. URL https://arxiv.org/abs/2409.20370.

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

Liyuan Zhang, Jiarui Zhang, Jinwei Yao, Ran Yan, Yuchen Yang, Jiahao Zhang, Tongkai Yang, Yi Wu, and Binhang Yuan. D^2SD: Accelerating Speculative Decoding with Dual Diffusion Draft Models, 2026. URL https://arxiv.org/abs/2606.04446.

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, 2023. URL https://arxiv.org/abs/2306.05685.

A Derivations and Implementation Clarifications

A.1 Why Intra-Block Causal Conditioning Helps

In Equation 1, every position predicts from the same verified context without observing the tokens selected at earlier positions. Each marginal therefore averages over possible continuations, even after the first pass has produced a particular prefix. D-Loop exposes that prefix to the backbone, allowing suffix predictions to adapt to the continuation already proposed. Bidirectional attention over masked positions alone cannot provide this realized-token information.

To isolate this modeling advantage, fix a split ℓ independently of the proposal tokens. Let $q^\star_{\mathrm{one}}$ be Equation 1 with exact target marginals, and let $q^\star_{\mathrm{loop}}$ be Equation 5 with the same prefix marginals and exact prefix-conditioned suffix probabilities. Their expected log-likelihood difference, evaluated on target continuations and their contexts, is

$$\mathbb{E}_p\left[\log \frac{q^\star_{\mathrm{loop}}(X_{1:m} \mid C)}{q^\star_{\mathrm{one}}(X_{1:m} \mid C)}\right] = \sum_{k=\ell+1}^{m} I(X_k; X_{1:\ell} \mid C) \geq 0. \tag{10}$$

Each term measures information that the prefix provides about one suffix token beyond the verified context. Equivalently, it is the reduction in that token's uncertainty from $H(X_k \mid C)$ to $H(X_k \mid X_{1:\ell}, C)$. The prefix factors are identical and cancel. Therefore, the gain comes entirely from replacing context-only suffix marginals with prefix-conditioned predictions (Appendix A.2). It is strictly positive whenever at least one suffix token depends on the prefix given C.

Thus, intra-block causal conditioning can recover dependencies that Equation 1 discards, while keeping suffix prediction parallel. This comparison describes the advantage with exact predictions. In practice, a trained shared drafter must learn both prediction conditions to realize it. Dependencies within each parallel stage remain unresolved, and a likelihood gain alone does not guarantee longer acceptance. The complementary objective in Section 3.2 trains the same backbone for both passes.

A.2 Likelihood Advantage of Prefix Conditioning

For a fixed split ℓ, define the ideal distributions by

$$\begin{aligned} q^\star_{\mathrm{one}}(x_{1:m} \mid C) &= \prod_{k=1}^{m} p(x_k \mid C), \\ q^\star_{\mathrm{loop}}(x_{1:m} \mid C) &= \prod_{k=1}^{\ell} p(x_k \mid C) \prod_{k=\ell+1}^{m} p(x_k \mid x_{1:\ell}, C). \end{aligned} \tag{11}$$

These minimize expected negative log-likelihood within the respective factorizations when each component distribution is unrestricted. The common prefix factors cancel in their ratio. Taking expectations under the target joint distribution gives

$$\begin{aligned} \mathbb{E}_p\left[\log \frac{q^\star_{\mathrm{loop}}(X_{1:m} \mid C)}{q^\star_{\mathrm{one}}(X_{1:m} \mid C)}\right] &= \sum_{k=\ell+1}^{m} \mathbb{E}_p\left[\log \frac{p(X_k \mid X_{1:\ell}, C)}{p(X_k \mid C)}\right] \\ &= \sum_{k=\ell+1}^{m} I(X_k; X_{1:\ell} \mid C). \end{aligned} \tag{12}$$

Nonnegativity follows from that of conditional mutual information. The same quantity equals $\mathbb{E}_C[D_{\mathrm{KL}}(p \| q^\star_{\mathrm{one}}) - D_{\mathrm{KL}}(p \| q^\star_{\mathrm{loop}})]$, because the target entropy cancels. The identity concerns likelihood under the target distribution, not the distribution of generated prefixes or token-dependent split selection. It does not assert that a finite shared network attains both ideal distributions, nor that a likelihood improvement necessarily improves greedy acceptance.

A.3 Training Implementation and Short-Prefix Fallback

Complementary views. For a sampled prefix length $\ell_i \geq 2$, both training views use the same anchor a_i. The prefix view reveals only the anchor and supervises positions $1, \ldots, \ell_i$. The suffix view additionally reveals the ground-truth prefix and supervises positions $\ell_i + 1, \ldots, m$. This is the paired construction in Section 3.2.

Fallback for $\ell_i < 2$. An anchor-only prefix view would receive no supervised token when $\ell_i = 0$ and only one when $\ell_i = 1$. To make better use of this forward pass, we replace that view with a full anchor-only training block at a neighboring sampled anchor in the same sequence. The suffix view remains at a_i with its original sampled prefix length. The neighboring anchor is obtained by cyclically shifting the sampled anchor array by one position, as in `torch.roll(anchor_positions, 1, dims=1)`. For anchors $(a_1, \ldots, a_N)$, this gives $(a_N, a_1, \ldots, a_{N-1})$. Thus, "neighboring" refers to the previous entry in the sampled anchor array, with wraparound, rather than necessarily the preceding token.

Let $\pi(i)$ denote this cyclic predecessor. The selected prefix-view anchor and its loss weights, for $k = 1, \ldots, m$, are

$$a_i^P = \begin{cases} a_i, & \ell_i \geq 2, \\ a_{\pi(i)}, & \ell_i < 2, \end{cases} \qquad \widetilde{w}_{ik}^P = v_{ik}^P e^{-(k-1)/\gamma} \begin{cases} \mathbf{1}\{k \leq \ell_i\}, & \ell_i \geq 2, \\ 1, & \ell_i < 2. \end{cases} \tag{13}$$

Here v_{ik}^P excludes invalid or padded prediction positions, and γ is the anchor-relative loss decay scale. The context and ground-truth labels are taken from the selected anchor a_i^P. Its input contains that anchor followed by m mask tokens. In the fallback case, all valid prediction positions receive supervision with the usual DFlash decay. For example, with $B = 16$ and $\ell_i = 1$, the suffix view supervises positions $2, \ldots, 15$ at the original anchor. The replacement prefix view supervises all 15 proposal positions at the neighboring anchor. The anchor token itself is never supervised.

Loss accumulation and view isolation. The implementation processes the two views in separate forward passes with shared drafter parameters. Each view is divided into anchor chunks with gradient checkpointing to reduce peak activation memory. Weighted cross-entropy sums and weight sums are accumulated across both views before applying the normalization in Equation 8, using $\epsilon = 10^{-6}$. For a fallback block, the prefix terms use a_i^P and $\widetilde{w}_{ik}^P$ from Equation 13. The anchor-only view cannot access the revealed prefix of its companion view. Target features are restricted to the context available at the anchor of each view.

In the basic paired construction, each prediction position is supervised in exactly one view before weighting. The fallback instead mixes prefix-conditioned suffix training with full anchor-only training on another sampled block. Therefore, this exact partition no longer holds for short-prefix blocks. This training fallback adds no model parameters or inference passes.

A.4 Correctness of Greedy and Stochastic Verification

Deterministic drafting at both temperatures. Both draft passes use position-wise argmax at $T = 0$ and $T = 1$. Prefix selection also remains deterministic. We compute first-pass token confidences from the unscaled draft softmax and select the top-K prefix lengths using the score in Section 3.1. Here top-K ranks prefix lengths, not vocabulary tokens. For the same context and model state, the initial proposal, selected prefixes, and regenerated suffixes are therefore identical at both temperatures. During a drafting–verification cycle, temperature affects only the target's token selection.

Greedy decoding. Let $g(C)$ be the target's next-token argmax with a deterministic tie-breaking rule. A branch $y_{1:m}$ is verified by comparing y_k to $g(C, y_{<k})$. Before its first disagreement, induction gives that every y_k equals the corresponding token of the target's greedy continuation. Therefore all branches' accepted prefixes are prefixes of the same target continuation, and choosing the longest one cannot change that continuation. At its next position, a target correction or bonus token also equals the next target token. Repeating the argument cycle by cycle proves equality of the complete greedy output, subject to correct branch-specific causal attention, cache rollback, and numerical evaluation of the target.

Target token selection. Let $a_v(h)$ be the target logit for token v given output history h, and let $g(h)$ be its argmax with fixed tie-breaking. The implementation selects a token according to

$$p_T(v \mid h) = \begin{cases} \mathbf{1}\{v = g(h)\}, & T < 10^{-5}, \\ \dfrac{\exp(a_v(h)/T)}{\sum_{u \in \mathcal{V}} \exp(a_u(h)/T)}, & T \geq 10^{-5}, \end{cases} \tag{14}$$

where $\mathcal{V}$ is the vocabulary. The first case is implemented by argmax. Otherwise, we draw one categorical sample from the temperature-scaled softmax, without top-k or nucleus (top-p) filtering. The initial target-generated token and correction or bonus tokens use this same rule.

One shared draw across surviving branches. Joint verification computes target logits for each branch under its own causal history. At position k, the surviving branches all match the previously sampled tokens $y_{<k}$ and thus share the target distribution $p_T(\cdot \mid C, y_{<k})$. We use the logits of the first surviving branch to draw a single token y_k. Every surviving branch is compared against this same draw, and those with a different proposal at position k are discarded. If none remains, the already sampled y_k becomes the correction token without resampling. The next cycle starts from the resulting output history. If a complete draft survives, the target draws a bonus token using Equation 14.

Preservation of the target distribution. At each step, the emitted token follows $p_T(\cdot \mid h)$ for the actual output history h. By induction, the generated sequence has the same distribution as autoregressive target sampling at the same temperature without vocabulary filtering. Deterministic proposal and prefix selection affect how far cached verification can continue, but do not change this conditional distribution. This argument requires causal branch isolation, consistent position indices, correct KV-cache updates, and one shared target draw per position across surviving branches.

Effect on acceptance length. For a fixed history h and proposal token v, stochastic matching occurs with probability $p_T(v \mid h)$. With multiple surviving branches, the probability of continuing verification is the total target probability mass of their distinct next-token proposals. A token accepted under greedy verification can therefore be rejected under sampling, while a non-argmax proposal can be accepted. Once a sampled token differs from the greedy continuation, subsequent cycles also receive different contexts and may produce different proposals. The observed reduction in acceptance length at $T = 1$ thus reflects stochastic target matching and diverging generation histories, rather than randomness in the drafter. It is not a universal ordering of acceptance lengths across temperatures.