DEDIT: ITERATIVE DRAFT EDITING FOR SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates autoregressive LLMs by having a lightweight drafter propose tokens that the target model verifies in parallel. Diffusion-based drafters further reduce drafting latency by proposing all tokens at once. However, these tokens are predicted independently, so a single early error causes prefix verification to discard the rest of the draft, even when it contains useful downstream predictions. We introduce DEdit, a diffusion-based drafter that can not only draft by conventional parallel unmasking but also iteratively edit its draft through token-to-token predictions. Through editing, later predictions can serve as bidirectional context for repairing earlier errors and extending the accepted prefix. To teach the model to repair errors while preserving correct predictions, we propose PROPOSALMIX, a training scheme that mixes draft predictions with ground-truth tokens based on first-pass confidence during training. Across seven benchmarks, DEdit achieves the highest macro-average token acceptance among the evaluated drafters for both Qwen3-4B and Qwen3-8B under greedy and stochastic decoding, and its acceptance increases with more editing iterations and wider drafting windows. In our execution setting, speedup follows the same trends, and DEdit reaches the highest macro-average speedups of $5.72\times$ and $5.97\times$ over autoregressive decoding under greedy decoding.

1 INTRODUCTION

Autoregressive large language models (LLMs) generate text one token at a time, creating a sequential bottleneck for inference speed. Speculative decoding mitigates this bottleneck by using a lightweight drafter to propose multiple tokens that the target model verifies in a single forward pass (Leviathan et al., 2023; Chen et al., 2023). Its efficiency depends on both the number of accepted tokens and the cost of producing each proposal. However, autoregressive drafters themselves generate proposals sequentially, retaining the overhead of repeated forward passes. To reduce this overhead, parallel drafting, particularly diffusion-based drafting, has been widely adopted in recent work on speculative decoding (Chen et al., 2026b; Huang et al., 2026; Cheng et al., 2026; Wang et al., 2026).

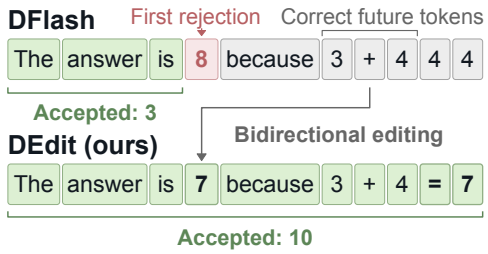

Figure 1: Bidirectional editing before verification extends the accepted prefix.

This shift to parallel drafting introduces a different challenge. Because multiple draft positions are predicted without conditioning on one another, proposal quality tends to degrade at later positions, causing acceptance rates to decay. Existing methods address this limitation by reintroducing causal dependencies among draft tokens to improve proposal quality (Huang et al., 2026; Cheng et al., 2026; Wang et al., 2026). We take a different view. The lack of causal dependence also means that an error at an early position does not necessarily invalidate later predictions. In fact, later candidates can still be correct even beyond the first rejected token. However, standard prefix verification discards this informative suffix entirely. We therefore ask whether these otherwise wasted parallel predictions can instead be used to revise earlier errors and extend the accepted prefix before target verification.

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.

We introduce DEdit, a diffusion-based drafter that generates an initial proposal from masked inputs and then iteratively refines all positions in parallel using bidirectional context. This allows later predictions to help repair earlier errors before verification, after which the target model verifies only the final proposal using the standard procedure. Figure 1 illustrates how editing can correct both an early mismatch and the downstream continuation, extending the accepted prefix.

Training DEdit requires inputs that resemble imperfect first-pass drafts while retaining useful predictions that should not be overwritten. We therefore introduce PROPOSALMIX, which constructs training proposals by mixing first-pass draft predictions with ground-truth tokens according to confidence. This creates partially correct proposals with both reliable context and realistic errors, teaching the editor when to preserve a prediction and when to revise it.

Our analyses show that the drafter improves token acceptance through inference-time edits beyond the single supervised editing step. Editing also makes wider proposal windows more useful: extending the window from 16 to 32 tokens raises acceptance by 10.3% after editing, compared with 2.1% for DFlash (Chen et al., 2026b). PROPOSALMIX teaches the editor to avoid harmful edits, halving the rounds in which editing shortens the accepted prefix. Controlled comparisons indicate that editing gains rely on future proposal context, which causal correctors such as Domino (Huang et al., 2026), restricted to prefix context, cannot use: hiding later tokens from the editor reduces acceptance on every task we evaluate. This benefit is largest on predictable outputs: on reasoning, summary, and synthetic copy-heavy tasks, DEdit's acceptance advantage over Domino exceeds that on the standard seven-benchmark suite and is largest on the copy-heavy task.

Across seven benchmarks under greedy and stochastic decoding, DEdit achieves the highest macro-average token acceptance among the evaluated drafters on both Qwen3-4B and Qwen3-8B. Editing trades additional drafter computation for longer accepted prefixes, so its speed benefit depends on how much drafting cost the execution backend exposes. With CUDA Graph drafting and eager target verification, where the CPU dispatches target operations while the drafter runs, DEdit also achieves the highest macro-average speedup, reaching $5.72\times$ on Qwen3-4B and $5.97\times$ on Qwen3-8B under greedy decoding.

Our contributions are:

- We introduce DEdit, a bidirectional drafter that generates and iteratively edits complete proposals before verification. Its acceptance improves with additional inference-time edits and continues to increase as proposal windows expand from 16 to 32 tokens.
- We develop joint proposal-and-edit training with PROPOSALMIX, which teaches the drafter to preserve correct tokens and revise incorrect ones in a way that aligns editing with the prefix-acceptance objective of speculative decoding.
- We provide controlled evidence that access to future proposal context is a key source of the gains from bidirectional editing, revealing a capability unavailable to causal correction methods restricted to prefix context.

2 PRELIMINARIES

2.1 SPECULATIVE DECODING

Speculative decoding accelerates generation from a frozen autoregressive target model p using an efficient drafter q (Leviathan et al., 2023; Chen et al., 2023). Each decoding round uses a proposal window of width W, comprising one committed anchor token y_0 and $W-1$ unverified candidate tokens. The anchor is the last token of the current output prefix and remains fixed during drafting and editing. The target evaluates the window in a single forward pass and accepts the longest candidate prefix consistent with its own predictions; the first mismatch terminates acceptance, and the remaining candidates are discarded regardless of matches at later positions. If a candidates are accepted, verification appends them and one additional target token, which becomes the anchor for the next round, advancing generation by $a+1$ tokens. We denote the expected token acceptance per round by $\tau = \mathbb{E}[a+1]$. Because DEdit changes only proposal generation and retains this verification protocol, its improvements appear directly as a higher τ.

2.2 PARALLEL DIFFUSION DRAFTING

Parallel diffusion drafters predict candidate blocks from masked inputs. We consider the target-conditioned, single-pass formulation exemplified by DFlash (Chen et al., 2026b). Given decoded context $\boldsymbol{x}$ and target features $\boldsymbol{h}(\boldsymbol{x})$, the drafter predicts all masked positions in $\boldsymbol{z}^{(0)} = [y_0, \text{MASK}, \ldots, \text{MASK}]$ simultaneously:

$$q_{\theta,i}^{(1)}(v) := q_\theta\Big(Y_i = v \mid \boldsymbol{x}, \boldsymbol{h}(\boldsymbol{x}), \boldsymbol{z}^{(0)}\Big), \qquad \hat{y}_i^{(1)} = \arg\max_{v \in \mathcal{V}} q_{\theta,i}^{(1)}(v), \quad i = 1, \ldots, W-1.$$

The predictions form the initial proposal $\boldsymbol{z}^{(1)} = [y_0, \hat{y}_1^{(1)}, \ldots, \hat{y}_{W-1}^{(1)}]$. DEdit uses this proposal as the starting point for bidirectional editing before target verification.

3 DEDIT

3.1 OVERVIEW

DEdit uses a diffusion-based drafter to not only generate, but also *iteratively edit proposals* before target verification. We first describe the iterative editing procedure, explaining how each pass uses the preceding proposal as bidirectional context to update candidate tokens (Section 3.2). We then introduce joint proposal-and-edit training and PROPOSALMIX, which constructs editing inputs that teach the drafter to revise imperfect proposals while preserving correct predictions (Section 3.3). Finally, we describe how the trained editor is adapted to larger proposal windows through continued training (Section 3.4). Figure 2 summarizes the inference and training pipelines.

3.2 ITERATIVE PROPOSAL EDITING

As illustrated in Figure 2(a), DEdit applies $K-1$ editing passes to the initial proposal $\boldsymbol{z}^{(1)}$ using the same drafter, where K denotes the total number of drafting passes, including initial proposal generation. Throughout editing, the decoded context $\boldsymbol{x}$, target features $\boldsymbol{h}(\boldsymbol{x})$, and anchored token y_0 remain fixed. At each pass $k \in \{2, \ldots, K\}$, the drafter takes the complete preceding proposal $\boldsymbol{z}^{(k-1)}$ as input and predicts:

$$q_{\theta,i}^{(k)}(v) := q_\theta\Big(Y_i = v \mid \boldsymbol{x}, \boldsymbol{h}(\boldsymbol{x}), \boldsymbol{z}^{(k-1)}\Big), \qquad \hat{y}_i^{(k)} = \arg\max_{v \in \mathcal{V}} q_{\theta,i}^{(k)}(v), \quad i = 1, \ldots, W-1.$$

These predictions form the updated proposal $\boldsymbol{z}^{(k)} = [y_0, \hat{y}_1^{(k)}, \ldots, \hat{y}_{W-1}^{(k)}]$.

All candidate positions are updated simultaneously from the same preceding proposal. With bidirectional attention, position i can condition on both earlier and later candidate tokens, including those at positions $j > i$. Later predictions can therefore provide context for correcting earlier errors and extending the accepted prefix. Every candidate position is eligible for revision at each pass, without confidence gating.

After K passes, only the final proposal $\boldsymbol{z}^{(K)}$ is submitted to the target verifier. All passes share the same parameters, so each additional edit requires one drafter forward pass but introduces neither additional model parameters nor an intermediate target query.

3.3 JOINT TRAINING WITH PROPOSALMIX

We train a single network for both a proposal pass (P1) and an editing pass (P2), as illustrated in Figure 2(b). The proposal objective trains generation from masked inputs, while the editing objective teaches the network to preserve correct tokens and revise errors. To support editing supervision, PROPOSALMIX supplies ground-truth tokens at positions selected using first-pass confidence and retains draft predictions elsewhere. As the drafter's predictions change during training, confidence-based selection adapts the editing inputs to its evolving error patterns (Appendix B.3).

For each position $i \in \{1, \ldots, W-1\}$, we define:

$$c_i = \max_{v \in \mathcal{V}} q_{\theta,i}^{(1)}(v), \qquad m_i = \mathbb{I}[c_i > \eta].$$

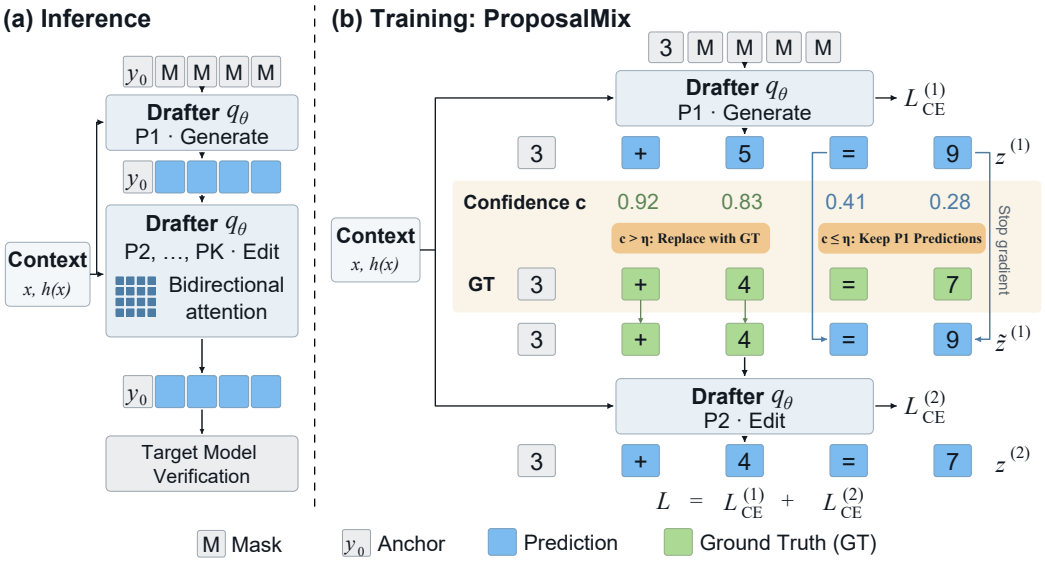

Figure 2: **DEdit inference and training. (a)** One shared drafter produces P1, edits through PK, and sends only the final proposal to the frozen target. **(b)** PROPOSALMIX constructs P2 inputs for joint proposal-and-edit training.

Given the ground-truth target token $y_i^\star$, the mixed input is

$$\tilde{z}_i^{(1)} = \begin{cases} y_i^\star, & m_i = 1, \\ \hat{y}_i^{(1)}, & m_i = 0. \end{cases}$$

The anchored token y_0 remains fixed, and we set $\eta = 0.5$ across all experiments. At high-confidence positions ($m_i = 1$), the ground-truth token provides correct context for editing. This leaves correct first-pass predictions unchanged and replaces high-confidence errors. At the remaining positions ($m_i = 0$), the original predictions are retained, exposing the editor to model-generated errors as well as correct predictions that it should preserve.

Ground-truth tokens are used as proposal inputs only during training. The first-pass argmax predictions and confidence masks used to construct $\tilde{\boldsymbol{z}}^{(1)}$ are detached from the computational graph. During training, the second-pass distribution is

$$q_{\theta,i}^{(2)}(v) := q_\theta\Big(Y_i = v \mid \boldsymbol{x}, \boldsymbol{h}(\boldsymbol{x}), \mathrm{detach}(\tilde{\boldsymbol{z}}^{(1)})\Big), \qquad i = 1, \ldots, W-1.$$

The editing loss therefore does not backpropagate through the first-pass predictions or confidence decisions used to construct its input.

Both passes are supervised across all valid future token positions $\mathcal{T} \subseteq \{1, \ldots, W-1\}$. We use position-decay weights $w_i = \exp(-(i-1)/\gamma)$, where $\gamma > 0$ controls the decay. These weights emphasize earlier positions because an early mismatch prevents later candidates from being accepted, while retaining supervision for every valid position:

$$\mathcal{L}_{\mathrm{CE}}^{(k)} = -\frac{\sum_{i \in \mathcal{T}} w_i \log q_{\theta,i}^{(k)}(y_i^\star)}{\sum_{i \in \mathcal{T}} w_i}, \qquad k \in \{1, 2\}.$$

We write $\mathcal{L}_{\mathrm{unmask}} = \mathcal{L}_{\mathrm{CE}}^{(1)}$ for the proposal pass and $\mathcal{L}_{\mathrm{edit}} = \mathcal{L}_{\mathrm{CE}}^{(2)}$ for the editing pass, and train the shared parameters with the sum of these separately normalized losses:

$$\mathcal{L} = \mathcal{L}_{\mathrm{unmask}} + \mathcal{L}_{\mathrm{edit}}.$$

We refer to minimizing $\mathcal{L}$ over shared parameters θ as *joint training*. It contrasts with *two-stage training*, in which a separate drafter is first trained with $\mathcal{L}_{\mathrm{unmask}}$ and frozen, and an editor is then

trained with $\mathcal{L}_{\text{edit}}$ on the drafter's proposals. Joint training avoids a second model at inference, and it may expose the editing pass to a wider range of errors, since the errors of the proposal pass change as it improves during training. PROPOSALMIX changes only the input to P2; both passes use the same target labels, valid positions, and loss weights. Editing supervision covers both positions supplied with ground-truth tokens and those retaining draft predictions. At inference time, every editing pass consumes the preceding model-generated proposal, and passes $k \geq 3$ reuse the learned editing operation without additional training. Appendix B.2 provides the training hyperparameters.

3.4 WINDOW EXPANSION

Bidirectional editing allows earlier positions to use later proposal tokens as context. Expanding the proposal window therefore provides both more candidates for verification and additional future context for editing. However, parallel diffusion drafters can exhibit lower conditional token acceptance rates at later positions (Sandler et al., 2026; Inco AI, 2026). Prior evaluations of DFlash further show that increasing the draft budget can reduce both acceptance length and end-to-end speedup (Hu et al., 2026). To benefit from a larger window, the drafter must produce useful predictions at the additional positions and use them when revising earlier ones. A drafter trained at $W = 16$, however, has never been supervised at these positions, so we continue training it at larger windows to adapt both proposal generation and editing to the expanded context.

We initialize independent training runs at $W = 24$ and $W = 32$ from the same converged checkpoint trained at $W = 16$, and continue each for 3,000 optimization steps. We increase the position-decay parameter from $\gamma = 7$ at $W = 16$ to $\gamma = 11$ at $W = 24$ and $\gamma = 15$ at $W = 32$, slowing the decay of loss weights at more distant positions.

4 EXPERIMENTS

4.1 SETUP

Models and Benchmarks. We evaluate DEdit on Qwen3-4B and Qwen3-8B (Yang et al., 2025), the targets used by DFlash and DSpark, across seven benchmarks: GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021; Lightman et al., 2023), and AIME25 (Mathematical Association of America, 2025) (MATH); HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2024) (CODE); and MT-Bench (Zheng et al., 2023) (CHAT). Main results (§4.2) use an approximately 800K target-aligned corpus of Nemotron and CodeAlpaca prompts (Nathawani et al., 2025; Chaudhary, 2023); training-design ablations (§4.3) use a separately prepared 100K corpus. Appendix B.1 reports their sources and exact statistics.

Baselines and Drafter Configurations. We compare against DFlash (Chen et al., 2026b), a single-pass block-diffusion drafter; Domino (Huang et al., 2026), which adds lightweight causal correction; and DSpark (Cheng et al., 2026), which adds a sequential module and confidence-scheduled verification. At $W = 16$, all methods are trained on the same corpus for the same number of epochs, use a five-layer drafter conditioned on the same target layers, and share the evaluation protocol. DFlash and DEdit have identical architectures; Domino adds a lightweight GRU corrector, and DSpark adds a low-rank Markov head and a confidence head, which it uses to verify only a confidence-selected prefix of its draft. Following their original configurations, Domino and DSpark draft W candidates per window rather than $W - 1$. Main results compare DEdit at $W = 32$ with two editing passes ($K = 3$, P3) against baselines at their native $W = 16$. Section 4.3 reports window-matched comparisons, and Appendix C.1 the trade-off across editing passes.

Evaluation Protocol. We evaluate greedy ($T_p = 0$) and stochastic ($T_p = 1$) decoding with a maximum of 2,048 generated tokens, following the verification protocol in Section 2. Under greedy decoding, verification accepts candidates matching the target's top-1 predictions; under stochastic decoding, we follow the DFlash implementation (Chen et al., 2026b), in which the drafter proposes argmax tokens and verification accepts candidates matching tokens sampled from the target. Thinking mode is disabled for both target models, both when generating training responses and during evaluation. We report the mean token acceptance per round τ and the end-to-end speedup over autoregressive decoding, measured on a single NVIDIA H100 GPU at batch size 1. Drafting uses CUDA Graphs and target verification runs in eager mode for all methods, including the autoregressive

baseline; Appendix C.2 details this execution setting and its effect on measured latency. Full training configurations are provided in Appendix B.2.

4.2 Main Results

Table 1 reports τ and end-to-end speedup for both target models under greedy and stochastic decoding. We draw three conclusions.

Table 1: End-to-end speedup and mean token acceptance (τ) under the evaluation protocol in Section 4.1; Avg. is the seven-suite macro average. Baselines operate at $W = 16$ (Domino and DSpark draft 16 candidates; DFlash drafts 15), while DEdit uses the $W = 32$ P3 model after a 3K-step continuation from $W = 16$. Bold marks the best value in each model/temperature block.

Model	Method	Math						Code						Chat		Overall	
		GSM8K		MATH-500		AIME25		HumanEval		MBPP		LCB		MT-Bench		Avg.	
Temperature: 0		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Qwen3-4B	DFlash	4.99×	6.04	6.52×	7.76	5.82×	7.07	4.45×	5.32	4.08×	5.06	4.81×	5.83	2.98×	4.17	4.81×	5.89
	DSpark	5.37×	7.13	6.52×	9.02	6.06×	8.23	4.61×	6.23	4.37×	6.11	5.32×	7.14	2.67×	4.53	4.99×	6.91
	Domino	5.53×	6.76	7.21×	8.87	6.65×	8.12	4.76×	5.84	4.56×	5.70	5.30×	6.39	3.13×	4.53	5.30×	6.60
	DEdit P3	**6.51**×	**7.75**	**8.18**×	**10.67**	**6.84**×	**9.28**	**4.95**×	**6.48**	**4.71**×	**6.13**	**5.71**×	**7.61**	**3.17**×	**5.11**	**5.72**×	**7.58**
Qwen3-8B	DFlash	5.05×	6.09	6.39×	7.91	6.20×	7.39	4.49×	5.30	4.48×	5.04	5.06×	6.17	2.81×	4.05	4.93×	5.99
	DSpark	5.60×	7.40	6.78×	9.42	6.29×	8.72	4.73×	6.44	4.76×	**6.23**	5.38×	7.60	2.57×	4.57	5.16×	7.20
	Domino	5.85×	6.86	7.66×	9.06	7.27×	8.72	4.85×	5.97	**4.90**×	5.71	5.40×	6.73	3.00×	4.48	5.56×	6.79
	DEdit P3	**6.46**×	**7.87**	**8.68**×	**10.94**	**7.48**×	**9.98**	**5.08**×	**6.48**	4.87×	6.09	**6.04**×	**8.00**	**3.16**×	**4.89**	**5.97**×	**7.75**
Temperature: 1		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Qwen3-4B	DFlash	4.55×	5.62	5.13×	6.38	4.09×	4.86	4.17×	4.82	3.95×	4.76	4.36×	5.35	2.86×	3.85	4.16×	5.09
	DSpark	4.87×	6.52	5.31×	7.28	3.66×	5.19	4.27×	5.72	4.15×	**5.62**	4.89×	6.58	2.50×	4.10	4.24×	5.86
	Domino	5.28×	6.21	5.76×	7.07	4.28×	5.38	4.46×	5.40	4.36×	5.24	4.90×	5.89	2.91×	4.11	4.56×	5.61
	DEdit P3	**5.61**×	**6.96**	**6.37**×	**8.26**	**4.33**×	**5.62**	**4.94**×	**5.86**	**4.67**×	5.56	**5.22**×	**6.71**	**3.01**×	**4.58**	**4.88**×	**6.22**
Qwen3-8B	DFlash	4.48×	5.54	5.34×	6.27	3.74×	4.69	3.94×	4.59	4.03×	4.52	4.73×	5.75	2.53×	3.57	4.11×	4.99
	DSpark	4.86×	6.61	5.08×	7.18	3.54×	5.13	4.05×	**5.45**	4.14×	**5.37**	5.03×	7.07	2.45×	4.02	4.16×	5.83
	Domino	5.22×	6.18	5.72×	7.01	4.21×	5.25	4.16×	4.99	4.39×	5.01	5.10×	6.22	2.72×	3.93	4.50×	5.51
	DEdit P3	**5.69**×	**6.94**	**6.47**×	**8.02**	**4.50**×	**5.59**	**4.53**×	5.32	**4.55**×	5.31	**5.64**×	**7.18**	**2.89**×	**4.27**	**4.90**×	**6.09**

DEdit achieves the highest macro-average τ in all four target and decoding settings. Under greedy decoding, it reaches $\tau = 7.58$ on Qwen3-4B and 7.75 on Qwen3-8B, 9.7% and 7.6% higher than DSpark, the baseline with the highest τ; under stochastic decoding, the advantage narrows to 6.1% and 4.5%. This gain comes from editing rather than from a stronger initial proposal: DEdit shares DFlash's architecture, and its P1 acceptance is on par with DFlash's (Table 2(a)).

The acceptance gain varies across benchmarks. On MATH-500, DEdit improves τ over the strongest baseline by 12–18% across the four settings, and on GSM8K and MT-Bench by 5–13%. On HumanEval and MBPP, DEdit and DSpark are within 4% of each other, and DSpark leads in four of the eight model–decoding combinations. The benefit of editing is therefore task-dependent; Section 5.2 examines how it varies with the future context available in a proposal.

In our execution setting (Section 4.1), the acceptance advantage also yields the highest macro-average speedup, but by a smaller margin. DEdit reaches $5.72\times$ and $5.97\times$ under greedy decoding and $4.88\times$ and $4.90\times$ under stochastic decoding, 7.0–8.9% faster than Domino, the fastest baseline. On Qwen3-4B under greedy decoding, DEdit's τ is 14.8% higher than Domino's, whereas its speedup is 7.9% higher, because each editing pass adds a drafter forward. Editing thus trades per-round drafting cost for longer accepted prefixes, and whether this trade pays off depends on how much drafting cost the execution backend exposes (Appendix C.2).

4.3 Ablation Studies

DEdit combines three design choices: iterative editing (Section 3.2), joint training with ProposalMix (Section 3.3), and window expansion (Section 3.4). We ablate them on Qwen3-4B under greedy decoding and report macro-average τ over the seven benchmarks in Table 2. Table 2(a) varies the window width and the number of editing passes to test whether wider windows help and whether this depends on editing; Table 2(b) removes ProposalMix, joint training, or both to measure how each contributes to editing quality.

Table 2: Window scaling and training ablations on Qwen3-4B. Both panels report macro-average τ across seven benchmarks. **(a)** Proposal width and editing passes at $T_p = 0$; Domino drafts W candidates and the other methods $W - 1$. **(b)** Training components in a controlled 100K setting: a frozen DFlash drafter supplies P1, and a separately trained editor is evaluated at P2 with $W = 16$; $\Delta\tau$ is relative to Full.

(a) Window scaling

Method	Pass	$W = 16$	$W = 24$	$W = 32$
DFlash	P1	5.89	6.01	6.01
Domino	P1	6.60	6.91	6.93
DEdit	P1	5.82	5.93	5.93
	P2	6.63	7.02	7.12
	P3	6.87	7.39	**7.58**

(b) Training ablations

Variant	Joint	ProposalMix	τ	$\Delta\tau$
Full	✓	✓	**5.499**	—
w/o Joint	–	✓	5.414	–0.085
w/o ProposalMix	✓	–	5.355	–0.144
w/o Both	–	–	5.242	–0.257

Wider windows raise acceptance mainly through editing. Without editing, extending $W = 16$ to $W = 32$ barely changes DEdit's τ (5.82→5.93), and DFlash and Domino gain only 2.1% and 5.0%. With two editing passes, the same extension raises τ by 10.3% (6.87→7.58) and, in a separate timing run, speedup from $5.55\times$ to $5.80\times$. Each editing pass also contributes more at wider windows: the second editing pass (P3) adds 0.24 at $W = 16$ but 0.46 at $W = 32$, although it is never supervised during training. The additional positions of a wider window are therefore more valuable as context for editing than as extra candidates for verification.

PROPOSALMIX and joint training both improve editing, with PROPOSALMIX contributing more. To isolate the training objective, Table 2(b) holds the proposal fixed: a frozen DFlash drafter supplies P1 for all variants, and a separate editor is trained for each. With joint training, this editor is trained with both $\mathcal{L}_{\text{unmask}}$ and $\mathcal{L}_{\text{edit}}$; without it, the editor is trained with $\mathcal{L}_{\text{edit}}$ only, as in two-stage training (Section 3.3). Without PROPOSALMIX, the editor receives unmodified DFlash proposals. Removing PROPOSALMIX lowers τ by 0.144 and removing joint training by 0.085, while removing both lowers it by 0.257. Joint training helps even though the editor never generates P1 at inference, suggesting that learning to draft from masked inputs also improves editing.

5 ANALYSIS

Section 4 shows that DEdit's acceptance gains come from editing, and that they depend on the training objective and grow with window width. These aggregate results do not show how the editor achieves them, which raises two questions. First, PROPOSALMIX raises τ (Table 2(b)), but what behavior does it actually teach the editor (Section 5.1)? Second, wider windows help mainly through editing (Table 2(a)), suggesting that the editor uses later proposal tokens as context; does it, and when does this help most (Section 5.2)?

Metric. We report A, the number of accepted draft tokens per round, excluding the additional target token. To compare drafters with different window widths, all methods in each analysis are scored on the same number of leading positions.

5.1 WHAT DOES PROPOSALMIX TEACH THE EDITOR?

PROPOSALMIX is designed to teach the editor when to preserve a prediction and when to revise it (Section 3.3). To examine what the editor actually learns, Figure 3(a) compares the Full and w/o PROPOSALMIX editors of Table 2(b), which edit proposals from the same frozen DFlash drafter, and measures the change in A from P1 to P2.

The editor trained with PROPOSALMIX preserves more and revises better. It shortens the accepted prefix in half as many rounds (9.9% to 4.7%). The share of rounds in which it extends the prefix stays nearly the same (30.1% to 28.8%), but more of these are large repairs: rounds in which editing extends the prefix by five or more tokens rise from 5.4% to 6.7%. Together, these raise the mean edit gain from 0.687 to 0.797. PROPOSALMIX therefore improves acceptance by avoiding harmful edits

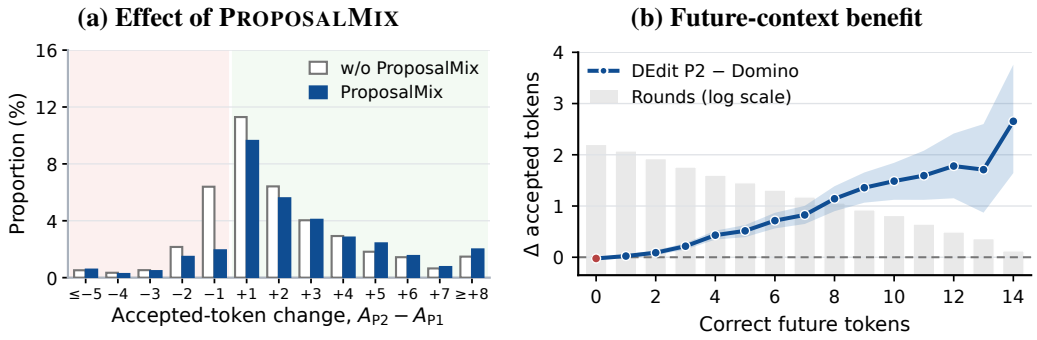

Figure 3: **Editing outcomes and future context on Qwen3-4B. (a)** Edit-gain distribution $A_{P2} - A_{P1}$ for the Full and w/o PROPOSALMIX editors of Table 2(b). Percentages use all rounds; zero-gain rounds are omitted (60.0% without PROPOSALMIX, 66.5% with it). **(b)** $W = 16$ DEdit P2-minus-Domino gain versus correct future-token count. Points and bands show response-cluster means and 95% cluster-bootstrap CIs; gray bars show the number of rounds per bin on a log scale, from 56k to 127. Fully accepted rounds (7.7%) are excluded.

rather than by editing more often. This matters under prefix verification, where a single harmful edit can invalidate an otherwise accepted prefix.

5.2 HOW DOES EDITING USE FUTURE CONTEXT?

Prefix verification discards correct predictions after the first mismatch, which we call *correct future tokens* (the trailing C's in `[C,C,W,C,C]`, with C correct and W wrong). A bidirectional editor can attend to them, whereas a causal corrector cannot. We ask whether editing uses these tokens, and on which tasks this helps most.

Does editing use future context? DEdit's advantage over a causal corrector grows with the number of correct future tokens. We apply Domino's causal corrector (Huang et al., 2026) and DEdit's P2 editor to identical $W = 16$ proposals from Domino's backbone; the DEdit editor comes from the main experiments and has never seen Domino's proposals. Figure 3(b) shows that the advantage is negligible at zero correct future tokens and clearly positive from two onward. After stratifying by accepted-prefix length, each additional correct future token is associated with a 0.140-token increase in the advantage (95% CI $[0.130, 0.150]$). Since both correct the same proposals, this growth points to future context as the source of the advantage.

Hiding future tokens from the same editor reduces acceptance. Keeping DEdit's weights and P1 proposals fixed, we replace bidirectional attention within the proposal by a lower-triangular mask. On General, Table 3 shows that bidirectional attention adds 0.222 accepted tokens at P2 and 0.290 at P3. It both repairs the first rejected token more often and corrupts the accepted prefix less often (Appendix A.4), so future context supports repair and preservation alike. Domino, a trained causal corrector, provides a reference: at P2, causal DEdit falls below Domino on General, Reasoning, and Summary, whereas bidirectional DEdit exceeds it on every slice. Because the editor was trained with bidirectional attention, part of this gap may reflect the change in attention pattern at inference.

When does future context help most? The benefit of future context is consistent across tasks and largest when outputs are highly predictable. Besides General, we evaluate Reasoning and Summary slices of a visible solution and its summary, and a synthetic Copy-heavy task that copies an input with one specified change (Appendix A.2). Bidirectional attention improves acceptance on every slice at both P2 and P3. On Reasoning and Summary, it adds about 0.20 tokens at P2 and 0.25 at P3, comparable to General. On Copy-heavy, it adds 0.724 and 0.897, about three times as much. When most of the output is determined by the input, proposals likely contain more correct future tokens for editing to use.

Table 3: Fixed-weight visibility intervention. Values are A over the first 16 candidate positions, averaged over rounds within each slice; General averages the seven suite-level means. Causal and bidirectional DEdit share a $W = 32$ checkpoint and P1 proposal. Domino ($W = 16$) takes the first token of this proposal and drafts the remaining 15 with its own backbone and corrector. Δ: bidirectional minus causal; underlined: below Domino. Details in Appendix A.3.

		DEdit P2			DEdit P3		
Task slice	Domino	Causal	Bidir.	Δ	Causal	Bidir.	Δ
General (7-suite avg.)	5.168	<u>4.997</u>	5.219	**+0.222**	5.192	5.482	**+0.290**
Reasoning	3.260	<u>3.224</u>	3.422	**+0.199**	3.371	3.624	**+0.253**
Summary	3.756	<u>3.718</u>	3.913	**+0.195**	3.881	4.141	**+0.260**
Copy-heavy	8.560	9.374	10.098	**+0.724**	9.819	10.717	**+0.897**

6 Related Work

Drafters for speculative decoding. Speculative decoding uses a lightweight drafter to propose tokens that the target model verifies in parallel (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023). Drafters include multi-head predictors (Cai et al., 2024; Ankner et al., 2024), feature-conditioned autoregressive drafters such as EAGLE (Li et al., 2024a;b; 2025b), and diffusion drafters that denoise all positions in parallel (Christopher et al., 2025; Sandler et al., 2026; Li et al., 2025a; Cheng et al., 2025; Chen et al., 2026b; Zhang et al., 2026). Diffusion drafters are fast because they predict a whole block in one forward pass, but each position is predicted without conditioning on the others. Multi-token prediction (Gloeckle et al., 2024; DeepSeek-AI, 2024) instead trains the target itself to predict several future tokens, and its prediction heads can be reused as drafters. These methods improve how the initial proposal is generated; DEdit instead edits the realized proposal before verification.

Dependency recovery for parallel drafts. Predicting draft positions independently causes acceptance to degrade at later positions. Recent methods restore dependencies with lightweight causal correction (Huang et al., 2026; Cheng et al., 2026; Zheng & Li, 2026; Wang et al., 2026), extend such correction to draft trees (Li et al., 2026), or select a coherent path among candidates (Rusanovsky et al., 2026). In causal correction, each position conditions only on earlier tokens, so correct predictions later in the draft cannot inform earlier ones. DEdit instead revises each position using proposal tokens on both sides.

Iterative refinement and training for editing. Iterative refinement is well established in non-autoregressive generation, from Mask-Predict (Ghazvininejad et al., 2019) to diffusion language models that revise generated tokens (Bie et al., 2026; Chen et al., 2026c), and Speculative Correction (Chen et al., 2026a) applies bidirectional refinement to complete drafts. Using refinement for speculative drafting further requires training, since edits must extend the accepted prefix rather than merely improve token-level accuracy. PARD-2 (An et al., 2026) and VAT (Gu et al., 2026) align drafter training with target acceptance. DEdit brings iterative refinement to speculative drafting, and ProposalMix trains the editor on partially correct proposals to preserve reliable predictions under prefix verification.

7 Conclusion

We propose DEdit, a diffusion-based drafter that generates proposal tokens and iteratively edits them before the target model verifies them. To train this drafter, we design ProposalMix, a training scheme that teaches the model to repair errors in its proposals while keeping the correct ones. Across seven benchmarks under greedy and stochastic decoding, DEdit achieves the highest macro-average token acceptance among the evaluated drafters and, in our execution setting, the highest macro-average speedup. Our analyses show that editing uses correct future tokens as context, so the additional positions of a wider window are more valuable for editing than as extra candidates.

AI USE STATEMENT

In this work, generative AI tools assisted with manuscript drafting and language editing and with implementing parts of the experimental code. The training corpora include target-model-generated responses, as described in Appendix B.1. The authors take full responsibility for the final manuscript, experimental code, and reported results.

REPRODUCIBILITY STATEMENT

All model architectures, training hyperparameters, loss formulations, and evaluation protocols are detailed in Section 4 and Appendix B. The evaluation datasets used across all seven benchmarks are publicly accessible. Code, checkpoints, and evaluation scripts will be made publicly available to facilitate reproducibility.

REFERENCES

Zihao An, Taichi Liu, Ziqiong Liu, Dong Li, Ruofeng Liu, and Emad Barsoum. PARD-2: Target-aligned parallel draft model for dual-mode speculative decoding. arXiv preprint arXiv:2605.08632, 2026. URL https://arxiv.org/abs/2605.08632.

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for Medusa decoding. In *First Conference on Language Modeling*, 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. *arXiv preprint arXiv:2108.07732*, 2021.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Xingyu Mou, Zhenxuan Pan, Kaida Qiu, Yuji Ren, Jianfeng Tan, Yiding Tian, Zian Wang, Lanning Wei, Tao Wu, Yipeng Xing, Wentao Ye, Liangyu Zha, Tianze Zhang, Xiaolu Zhang, Junbo Zhao, Da Zheng, Hao Zhong, Wanli Zhong, Jun Zhou, Junlin Zhou, Liwang Zhu, Muzhi Zhu, and Yihong Zhuang. LLaDA2.1: Speeding up text diffusion via token editing. arXiv preprint arXiv:2602.08676, 2026. URL https://arxiv.org/abs/2602.08676.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR, 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

Sahil Chaudhary. Code Alpaca: An instruction-following LLaMA model for code generation. GitHub repository, 2023. URL https://github.com/sahil280114/codealpaca.

Brian K. Chen, Chong Wu, and Kenji Kawaguchi. Speculative correction: Draft-then-refine decoding for diffusion language models. arXiv preprint arXiv:2608.02625, 2026a. URL https://arxiv.org/abs/2608.02625.

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026b. arXiv:2602.06036.

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

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. DMax: Aggressive parallel decoding for dLLMs. arXiv preprint arXiv:2604.08302, 2026c. URL https://arxiv.org/abs/2604.08302.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, et al. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026.

Zicong Cheng, Guo-Wei Yang, Jia Li, Zhijie Deng, Meng-Hao Guo, and Shi-Min Hu. DEER: Draft with diffusion, verify with autoregressive models. *arXiv preprint arXiv:2512.15176*, 2025.

Jacob K. Christopher, Brian R. Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative diffusion decoding: Accelerating language generation through diffusion. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 12042–12059, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-Predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 6112–6121, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1633. URL https://aclanthology.org/D19-1633/.

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

Geonmo Gu, Byeongho Heo, HeeJae Jun, Yoohoon Kang, Sangmin Lee, Sangdoo Yun, and Dongyoon Han. Verification-aware training for speculative decoding. arXiv preprint arXiv:2608.30135, 2026. URL https://arxiv.org/abs/2608.30135.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 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.

Inco AI. DFlash 2: Keep Drafting Parallel, 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. *arXiv preprint arXiv:2403.07974*, 2024.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 19274–19286, 2023.

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. DiffuSpec: Unlocking diffusion language models for speculative decoding. *arXiv preprint arXiv:2510.02358*, 2025a.

Tianyi Li, Yaxin Luo, Xinyi Shang, and Zhiqiang Shen. DARTree: Speculative diffusion decoding with autoregressive draft trees. arXiv preprint arXiv:2608.13524, 2026. URL https://arxiv.org/abs/2608.13524.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR, 2024a. URL https://proceedings.mlr.press/v235/li24bt.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. doi: 10.52202/085713-4562. URL https://proceedings.neurips.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. *arXiv preprint arXiv:2305.20050*, 2023.

Mathematical Association of America. American invitational mathematics examination, 2025. URL https://maa.org/maa-invitational-competitions/.

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

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.

Jameson Sandler, Jacob K. Christopher, Thomas Hartvigsen, and Ferdinando Fioretto. SpecDiff-2: Scaling diffusion drafter alignment for faster speculative decoding. In *Proceedings of Machine Learning and Systems*, volume 8, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/041dad5ed2191b44ba3ed0e00cdc3187-Abstract-Conference.html.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31, 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.

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

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.

Hong-Kai Zheng and Piji Li. DeLS-Spec: Decoupled long-short contexts for parallel speculative drafting. arXiv preprint arXiv:2607.07409, 2026. URL https://arxiv.org/abs/2607.07409.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023.

A Mechanism and Task Analysis

A.1 Qualitative Editing Trace

The trace in Figure 4 uses a fixed-width proposal, so "insertion" and "deletion" patterns refer to repairing token shifts under sequence alignment; the tensor length stays fixed.

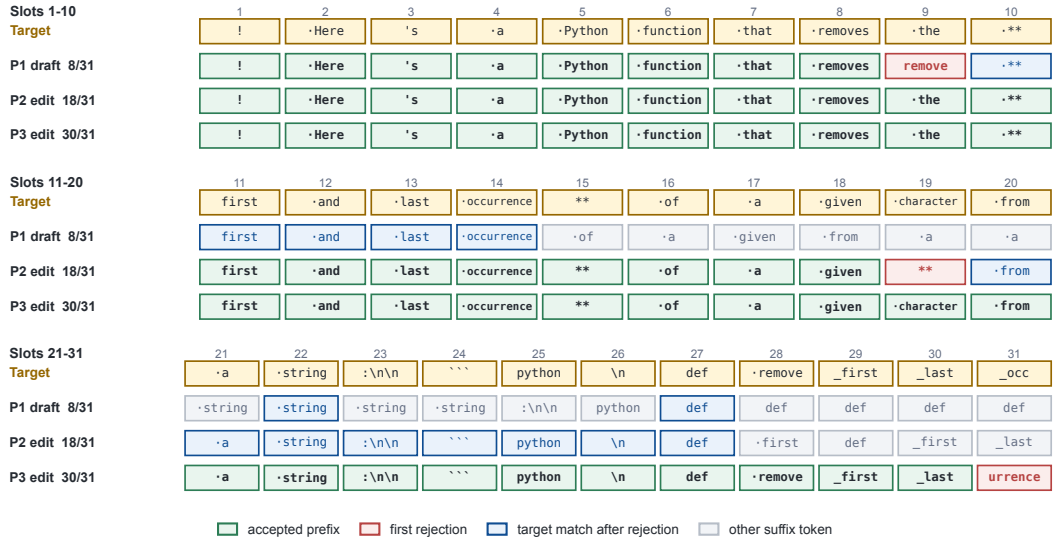

Figure 4: **A full $W = 32$ editing trace.** On an MBPP prompt, successive passes extend the accepted prefix from 8 to 18 to 30 of 31 proposal tokens. Blue cells already match the target but remain blocked by an earlier error.

Figure 4 shows the complete first verification round for the $W = 32$ DEdit checkpoint used in the main results. The P1 proposal first fails at slot 9 even though slots 10–14 already match the target. P2 repairs the prefix through slot 18 and leaves another correct run at slots 20–27 after its first rejection. P3 repairs the prefix through slot 30 but proposes `urrence` rather than the target token `_occ` at slot 31. The verifier therefore accepts 30 proposal tokens and emits `_occ` as the next-round anchor. The reported counts are contiguous accepted-prefix lengths, not token-level accuracies.

Observational audit details. The $W = 16$ comparison in Section 5.2 contains 176,823 decoding rounds from 2,830 response clusters. DEdit outperforms Domino on 17.91% of rounds, matches on 69.03%, and loses on 13.06%. For example, `[C,C,W,C,C]` has two correct future tokens after the first rejection. At a future-token count of 14, the observed gain reaches $+2.654$, although such high-count rounds are rare.

A.2 Controlled Task Definitions

We use three reasoning–summary constructions and one copy-heavy task to control how much useful future information is already visible. These are mechanism diagnostics, not new task benchmarks. All reasoning prompts are derived from frozen GSM8K examples, use visible model output with `enable_thinking=false`, and are selected without using drafter performance. The single-response construction permits separate Reasoning and Summary measurements; the two-turn and frozen-prefix forms isolate the same summary-continuation effect under alternative boundaries.

Shared reasoning example. The three reasoning–summary formats below use this representative question:

> Amber, Micah, and Ahito ran 52 miles in total. Amber ran 8 miles. Micah ran 3.5 times what Amber ran. How many miles did Ahito run?

Two-turn visible reasoning to summary. The first turn requests an explicit solution:

> Write a detailed, checkable solution. Show the essential equations and intermediate calculations. End with a separate line in exactly this format: `Reasoning conclusion: <answer>`.

The complete target response is then retained in the conversation, and the second turn asks:

> Using the solution above, write a concise self-contained solution in 3 to 5 sentences. Keep only the essential calculation steps. End with a separate line in exactly this format: `Final answer: <answer>`. Do not mention the previous response, a reasoning trace, or these instructions.

Thus the Summary continuation can condition on the exact visible Reasoning generated in the preceding turn.

Single-response reasoning and summary. This construction asks for both stages in one assistant response:

> Solve the problem and return exactly two plain-text sections in this order. Do not write any text before the first section label.
>
> **Reasoning:**
> Write a detailed, checkable solution with the essential equations and intermediate calculations.
>
> **Summary:**
> Write a concise self-contained solution in 3 to 5 sentences. Keep only the essential calculation steps and end with a separate line in exactly this format: `Final answer: <answer>`.

A representative frozen target response is:

> **Reasoning:**
> The total distance is 52 miles. Amber ran 8 miles, and Micah ran $3.5 \times 8 = 28$ miles. Together they ran $8 + 28 = 36$ miles, so Ahito ran $52 - 36 = 16$ miles.
>
> **Summary:**
> Amber ran 8 miles, and Micah ran $3.5 \times 8 = 28$ miles. Together they ran 36 miles. Ahito therefore ran $52 - 36 = 16$ miles.
> Final answer: 16

The stage boundary is the literal `Summary:` delimiter, so proposal statistics can be accumulated separately before and after it.

Frozen-reasoning summary continuation. To isolate Summary generation from variation in the preceding Reasoning, we freeze the target-produced prefix and begin evaluation after the delimiter:

> **Reasoning:**
> The total distance run by Amber, Micah, and Ahito is 52 miles. Amber ran 8 miles. Micah ran $3.5 \times 8 = 28$ miles. Amber and Micah therefore ran $8 + 28 = 36$ miles, so Ahito ran $52 - 36 = 16$ miles.
>
> **Summary:**

The expected continuation is:

> Amber ran 8 miles, and Micah ran $3.5 \times 8 = 28$ miles. Together, Amber and Micah ran 36 miles. Ahito ran $52 - 36 = 16$ miles.
> Final answer: 16

Only the Summary suffix is drafted and measured in this construction.

Copy-heavy continuation. The CodeXGLUE-derived task exposes the exact old-to-new replacement and asks for the complete normalized method:

> Apply the requested bug fix to the normalized Java method.
> Replace the exact token sequence `<OLD>VAR_1</OLD>` with `<NEW>VAR_3</NEW>`.

> Change nothing else. Return the complete corrected method as a single line with one space between tokens. Return only the method, with no Markdown, code fences, labels, or explanation.
>
> ```
> private boolean METHOD_1 ( java.lang.String VAR_1 ) { boolean
> VAR_2 = false ; try { java.lang.Boolean . METHOD_2 ( VAR_3 )
> ; VAR_2 = true ; } catch ( TYPE_1 error ) { VAR_4 . METHOD_3
> ( STRING_1 ) ; } return VAR_2 ; }
> ```

The expected response changes only the supplied token:

```
private boolean METHOD_1 ( java.lang.String VAR_3 ) { boolean
VAR_2 = false ; try { java.lang.Boolean . METHOD_2 ( VAR_3 )
; VAR_2 = true ; } catch ( TYPE_1 error ) { VAR_4 . METHOD_3
( STRING_1 ) ; } return VAR_2 ; }
```

Because the gold replacement is present in the prompt, this experiment measures lexical preservation and local correction, not standard CodeXGLUE program-repair quality.

A.3 Task-Slice Evaluation Details

Table 3 evaluates the visibility intervention on the seven-suite general workload and the three controlled slices defined above: a visible reasoning section, its conditioned summary, and a copy-heavy continuation. All three configurations receive the same target context. The two DEdit configurations use the same $W = 32$ checkpoint, P1 proposal, and attention implementation, and edit all 31 candidate positions. Domino drafts 16 candidates at $W = 16$ (Section 4.1); to align its input with DEdit, we fix its first candidate to the first token of the DEdit P1 proposal, and its backbone and corrector generate the remaining 15. Domino is a reference rather than part of the fixed-weight intervention.

For all configurations, we score the accepted prefix, after P2 and P3 for DEdit, over the same first 16 candidate positions and exclude rounds in which fewer than 16 tokens of the target continuation remain. Reasoning and Summary use a fixed 1,000-prompt GSM8K-derived set; 943 responses satisfy the required section markers and contribute 26,572 Reasoning and 8,919 Summary rounds. Copy-heavy uses a fixed 1,000-prompt set and contributes 5,874 rounds. General is the macro average over seven suite-level round means.

A.4 Fixed-Weight Future-Visibility Counterfactual

Intervention. We directly remove future proposal visibility from the Qwen3-4B $W = 32$ DEdit editor used in the main results while holding its checkpoint, target hidden states, P1 proposal, verifier, and target-reference trajectory fixed. Both configurations pass an explicit attention mask over the complete $W = 32$ proposal. The bidirectional configuration exposes all proposal positions, while the causal configuration replaces only intra-proposal visibility with a lower-triangular mask; all target-prefix hidden states remain visible. At P2, both configurations therefore receive exactly the same P1 input. P3 iteratively edits the proposal produced by its own P2 configuration. Following the task-slice protocol, reported acceptance statistics evaluate the first 16 proposal tokens and exclude rounds in which fewer than 16 tokens of the target continuation remain.

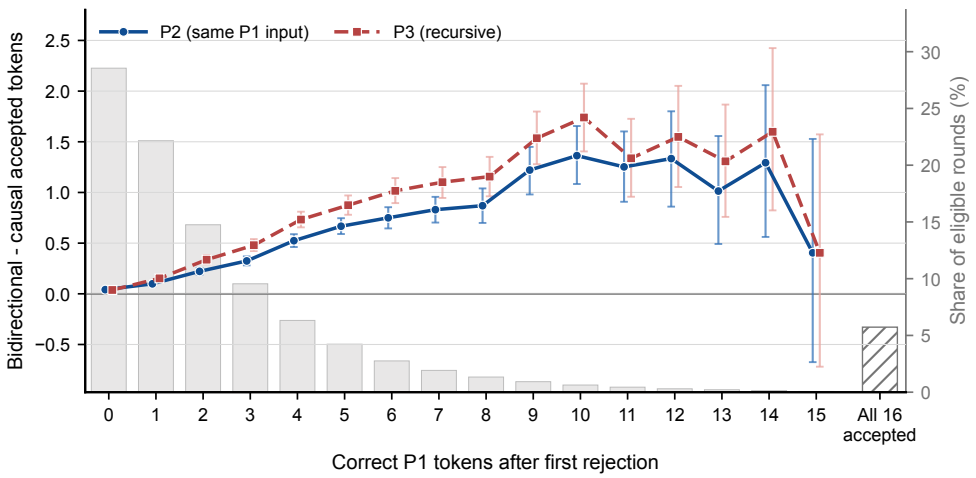

Figure 5: Fixed-weight intervention on future proposal visibility. Lines report response-cluster mean bidirectional-minus-causal accepted tokens with 95% response-bootstrap intervals; gray bars report each bin's share of 181,205 eligible rounds. P2 compares both masks on the same P1 proposal, while P3 applies the next edit to each configuration. The hatched bar denotes the 10,382 rounds in which all 16 shared proposal tokens are accepted.

Effect by available future information. Averaged over response clusters rather than rounds as in Table 3, bidirectional visibility adds 0.268 accepted tokens at P2, with a 95% interval of $[0.253, 0.283]$, and 0.361 at P3, with an interval of $[0.343, 0.380]$. The effect is positive even when no correct token follows the first rejection: $+0.041$ at P2 and $+0.038$ at P3. It then generally increases with available correct future tokens, reaching $+0.750$ and $+1.016$ at a count of six.

Task-level results after one and two editing passes are reported in Table 3.

Repair and preservation. On General, bidirectional and causal visibility repair the first rejected token in 27.85% and 27.11% of P2 rounds, respectively, while corrupting the original accepted prefix in 2.82% and 4.75%. At P3, the repair rates are 29.68% and 27.10%, and the prefix-corruption rates are 3.14% and 4.69%. Thus the editor's bidirectional advantage is visible in both more frequent repair and better preservation, and its gains are largest when more correct future tokens follow the first rejection (Figure 5).

B Training Details

B.1 Training Data

Table 4 reports the source composition of the target-aligned corpora. Each record pairs a user prompt with a response generated by the corresponding Qwen3 target with thinking disabled. Counts refer to released records before training-time tokenization and sequence-length handling; percentages use each corpus's actual row count.

Main training corpora (800K). The main experiments use prompts from the chat, code, and math partitions of Nemotron Post-Training Dataset v2 (Nathawani et al., 2025) and CodeAlpaca-20k (Chaudhary, 2023), paired with target-specific responses. The Qwen3-4B and Qwen3-8B corpora contain 799,864 and 800,000 records, respectively. Their source counts differ by 136 chat records; code, math, and CodeAlpaca counts are identical.

Controlled training corpus (100K). The Qwen3-4B training-design ablations use a 99,987-record corpus. Its recorded source families are `nemotron`, `openr1_math`, `opencodeinstruct`, and `evol_codealpaca`. The `nemotron` records span chat, code, math, and STEM partitions. Its sources and proportions differ from those of the 800K corpora.

Table 4: Training-corpus composition: records (percentage of each corpus). Counts are computed from the released `source` field and, for 100K, the joint `source`/`split` fields. Nemotron rows denote v2 partitions for 800K and the mirror's `nemotron` labels for 100K. A dash denotes no records with that source label.

Source / partition	800K (Qwen3-4B)	800K (Qwen3-8B)	100K (Qwen3-4B)
Nemotron / chat	365,375 (45.68%)	365,511 (45.69%)	28,791 (28.79%)
Nemotron / code	175,000 (21.88%)	175,000 (21.88%)	2,195 (2.20%)
Nemotron / math	239,467 (29.94%)	239,467 (29.93%)	6,698 (6.70%)
Nemotron / STEM	–	–	23,197 (23.20%)
CodeAlpaca-20k	20,022 (2.50%)	20,022 (2.50%)	–
OpenR1 Math	–	–	12,429 (12.43%)
OpenCodeInstruct	–	–	19,364 (19.37%)
Evol-CodeAlpaca	–	–	7,313 (7.31%)
Total	799,864 (100%)	800,000 (100%)	99,987 (100%)

Window-expansion data and settings. The $W = 24$ and $W = 32$ continuations use a fixed 200,000-record sample from the corresponding target's 800K corpus, selected uniformly without replacement with seed 42 and retained in source order. This continuation sample is separate from the 100K training-design corpus. Each continuation runs 3,000 steps with learning rate 10^{-4}, block size W, decay $\gamma = 11$ and 256 anchors at $W = 24$, and $\gamma = 15$ and 192 anchors at $W = 32$; other settings follow Table 5. DFlash, Domino, and DEdit use the same continuation settings.

B.2 Training Implementation

Our implementation adopts the DFlash (Chen et al., 2026b) backbone and target-feature conditioning scheme. The target model remains frozen throughout drafter training. Attention over the decoded context is causal, while attention across candidate positions is bidirectional. The selected target layers and other training settings are listed in Table 5.

Within each scale, DFlash, Domino (Huang et al., 2026), DSpark (Cheng et al., 2026), and DEdit are trained on the same corpus for the same number of epochs. Each baseline keeps its original training objective; DSpark, for example, combines CE and L1 losses with weights 0.1 and 0.9. DEdit applies position-weighted token CE to every valid position in both trained passes, in the main and the 100K settings. Training supervises P1/P2; later passes iteratively reuse the editor at inference without additional training.

Table 5: Training configurations for the controlled ablations and main results. Shared entries span both columns.

Setting	Controlled	Main
Role	Training-design ablations	Main system comparison
Target(s)	Qwen3-4B	Qwen3-4B and Qwen3-8B
Corpus	100K target-aligned	800K Nemotron-based mixture
Global batch	8	16
Epochs	6	
Optimizer	AdamW, LR 6×10^{-4}, warmup 0.04, clip 1.0	
Sequence	Length 2,048, block 16, 256 anchors	
Target layers / decay	1/9/17/25/33 / 7	
DEdit objective	Position-weighted P1+P2 CE	
Precision / attention	BF16 / SDPA	
Seed	42	

B.3 ProposalMix Input Statistics

Table 6 shows how ProposalMix inputs change as the proposal pass improves. We replay intermediate checkpoints of the Full model in Table 2(b) on a fixed probe set, using each checkpoint as

both P1 and P2, with $\eta = 0.5$. Ground-truth selection includes positions whose first-pass prediction already matches the target; overwrite counts only positions whose token changes.

Table 6: PROPOSALMIX input rates (%; $\eta = 0.5$) when each checkpoint generates its own P1 on a fixed probe set. GT selection marks positions chosen for ground-truth conditioning; overwrite counts those whose tokens change.

Step	P1 accuracy	GT selection	Overwrite
12,221	33.19	28.10	5.79
36,663	42.80	37.35	5.84
73,326	49.87	44.91	5.84

B.4 POSITION-WEIGHTED TWO-PASS CE

Let $\mathcal{T}$ contain every valid non-anchor token in the sampled training blocks. If $r_i \in \{1, \dots, W-1\}$ is token i's proposal offset, its weight is

$$w_i = \exp\left(-\frac{r_i - 1}{\gamma}\right), \qquad Z = \sum_{i \in \mathcal{T}} w_i.$$

For each trained pass $k \in \{1, 2\}$, the implementation computes a separately normalized cross-entropy,

$$\mathcal{L}_{\text{CE}}^{(k)} = -\frac{1}{Z} \sum_{i \in \mathcal{T}} w_i \log q_{\theta,i}^{(k)}(y_i^\star), \qquad \mathcal{L} = \mathcal{L}_{\text{unmask}} + \mathcal{L}_{\text{edit}},$$

where $\mathcal{L}_{\text{unmask}} = \mathcal{L}_{\text{CE}}^{(1)}$ and $\mathcal{L}_{\text{edit}} = \mathcal{L}_{\text{CE}}^{(2)}$. $W = 16$, $W = 24$, and $W = 32$ use $\gamma = 7, 11, 15$, respectively. The valid-token mask and positional weights are the same for both passes; only the P2 input differs. In particular, PROPOSALMIX does not narrow P2 supervision to uncertain positions.

The positional decay reflects prefix verification: errors near the beginning of a proposal prevent later tokens from being accepted in the same round. Supervising the full block still teaches later tokens that can become useful after an earlier edit. Appendix C.1 reports how acceptance and speedup vary with the number of editing passes.

C INFERENCE AND SYSTEMS ANALYSIS

C.1 EDITING-PASS TRADE-OFF

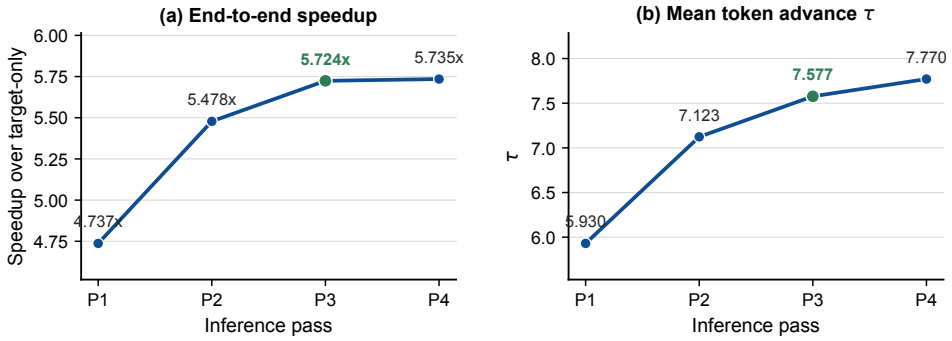

Figure 6: Trade-off across editing passes for the same $W = 32$ checkpoint at $T_p = 0$. Each configuration is evaluated twice with a shared autoregressive baseline. Drafting uses CUDA Graphs and target verification remains eager. Training supervises P1 and P2; P3 and P4 reuse the trained editor at inference.

As Figure 6 shows, adding a second editing pass (from P2 to P3) raises τ from 7.123 to 7.577 and speedup from $5.478\times$ to $5.724\times$. P4 further increases τ to 7.770, while speedup rises by only 0.19% to $5.735\times$. Although P4 is the fastest measured configuration, we use P3 as the default because it retains 99.8% of the observed P4 speedup with one fewer drafter forward per round.

C.2 Execution Setting

We evaluate single-request decoding with CUDA Graph-optimized drafting and eager target verification. CUDA Graph replay reduces host submission overhead by launching a captured sequence of GPU operations through one submission. We preallocate fixed-shape input, position, attention, and cache buffers for each drafter.

DFlash (Chen et al., 2026b) captures its single parallel draft forward. Domino (Huang et al., 2026) captures its backbone, output head, and complete GRU correction loop. DEdit captures all K passes and proposal construction between passes. DSpark (Cheng et al., 2026) captures its fixed-shape backbone, Markov correction, and confidence kernels; its native confidence readback and dynamic width decision remain in the measured host path. Target verification remains eager for every method, as does the autoregressive baseline.

Draft and target GPU operations execute sequentially on the same CUDA stream. During draft graph execution, however, the CPU can enqueue the subsequent eager target operations. This enqueue-ahead window can reduce gaps between target operations, partially hiding the additional drafting overhead. Consequently, independently measured draft and target latencies need not sum to the observed round latency. Additional passes still incur GPU computation, and their runtime benefit depends on the execution backend and workload.

C.2.1 Enqueue-Ahead Diagnostic

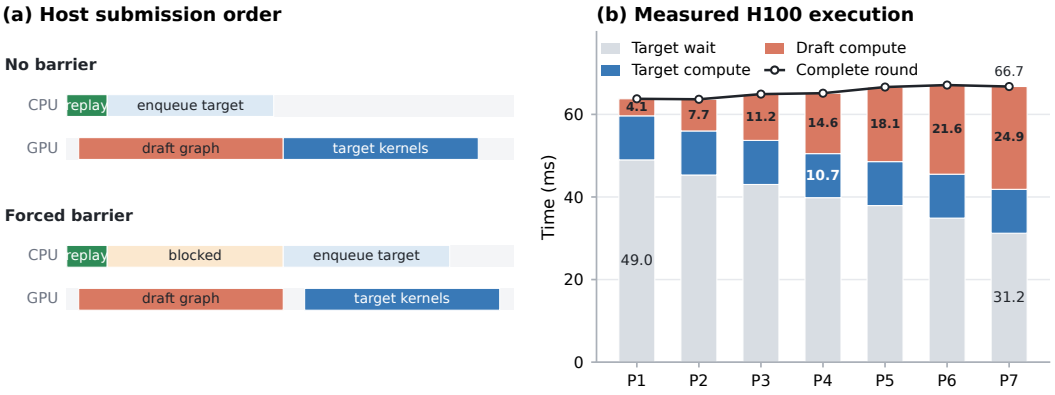

Figure 7: **Host enqueue-ahead during CUDA Graph replay.** (a) The CPU submits eager target operations while the GPU executes the captured drafter; draft and target GPU operations remain sequential. A forced host barrier delays target submission. (b) Nsight traces decompose each round into the pre-target draft interval, target compute, and gaps between target operations.

In Figure 7, all seven profiled settings execute the same 2,294 target GPU operations. Target compute remains nearly constant, while longer draft execution is accompanied by shorter inter-operation gaps. The forced-barrier control delays target dispatch without changing the draft graph or arithmetic, supporting the role of host enqueue-ahead. Because Nsight instrumentation perturbs latency, these traces illustrate the mechanism rather than quantify its contribution to the main-table speedups.

D Limitations and Future Work

Limitations. Our wall-clock gains are measured in a single-request setting with CUDA Graph-optimized drafting and eager target verification. CPU enqueue-ahead partially hides drafting overhead in this setup, so these results do not directly establish speedups in production serving with continuous batching. Additional editing passes still incur computation, and their exposed cost can outweigh the benefit of longer accepted prefixes. We evaluate targets of up to 8B parameters. Because every drafter is trained from scratch for each target under a matched budget, extending this controlled comparison to larger or newer targets is left to future work.

Future Work. Realizing these acceptance gains in production requires joint algorithm and scheduling design. A direct opportunity is to skip editing passes that do not help: about two-thirds of editing

passes leave the accepted prefix unchanged (Figure 3(a)), yet each costs a drafter forward. Predicting such rounds, for example from proposal confidence or from whether predictions stop changing across passes, could reduce drafting cost with little loss in acceptance. The same signals could inform how to adapt proposal width and verification length, and drafting and verification could be interleaved across requests. Evaluating whether these signals support effective compute allocation under realistic serving workloads remains future work.