xPRESS: PARALLEL REFINEMENT FOR DIFFUSION DRAFTERS IN SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Block-diffusion drafters like dFlash generate an entire block of draft tokens in a single forward pass, drastically reducing the overhead of multiple-token drafting in speculative decoding. The crucial final step of the single-pass discrete denoising process involves using the logit distribution at each position to sample conditionally independent tokens. The resulting draft is thus a set of per-position marginals, instead of a joint distribution: no draft token is guaranteed to depend on its predecessors. Such independently sampled marginals tend to produce sequences with tokens that are individually likely, but jointly improbable under the target model's distribution, which verifies each token conditionally. This can cause early rejection and limits acceptance length. To address this, we propose **xPRESS**, a lightweight causal refiner that restores the missing causality in diffusion drafters. xPRESS refines the entire diffusion block in parallel, propagating causal dependencies across the draft without a token-by-token loop. On Qwen3-8B, across seven math, code, and chat benchmarks, xPRESS raises **acceptance length by $\sim 35\%$ on average (up to $+58\%$)** and its **end-to-end decoding throughput by $\sim 1.3\times$ on average (up to $1.5\times$)** compared to the original dFlash diffusion drafter.

1 INTRODUCTION

Speculative Decoding (SD) (Chen et al., 2023; Leviathan et al., 2023) accelerates autoregressive generation by using a lightweight draft model to propose future tokens, which the larger target model then verifies in its next forward pass. This multi-token verification pass costs about the same as a standard single-token target-model decoding step in low concurrency scenarios, due to parallelism, so every draft token that matches target model outputs is another token produced at marginal target cost. Consequently, the speedup of SD is primarily determined by two factors: the number of drafted tokens accepted per verification step (the *acceptance length*), and the cost of producing those drafts. Ideally, a drafter should maximize the expected acceptance length without proportionally increasing drafting overhead. AutoRegressive (AR) drafting, exemplified by the EAGLE series (Li et al., 2024a;b; 2025b), achieves high-quality draft tokens with a remarkably lightweight drafter. However, producing n draft tokens requires n sequential drafter forward passes. As the draft length grows, this sequential cost increases proportionally, while the realized acceptance length does not necessarily grow at the same rate. dFlash (Chen et al., 2026) addresses this bottleneck with a block-diffusion drafter that proposes an entire block of draft tokens in a single forward pass, replacing n serial drafting steps with one parallel step, and decoupling draft length from drafting overhead.

While this allows parallel diffusion drafters to produce long drafts at near-constant overhead, it also carries an inherent accuracy limitation. Unlike an AR drafter, where each position is conditioned on the preceding tokens, positions in a diffusion drafter are decoded from their marginal distributions, so the block is a set of individually plausible tokens with no guarantee of causality. This can violate natural linguistic dependencies across positions, even when every token is locally high-probability (Wu et al., 2025; Liu et al., 2025; Wang et al., 2026b). At verification, these locally reasonable but jointly incoherent samples are rejected early by the left-to-right target model, limiting the achievable acceptance length.

One line of prior work tries to address this limitation by constructing a draft token tree. Tree-based drafting proposes a tree of candidate continuations and verifies the whole tree in a single target pass, so that the longest accepted path through the tree can be selected (Wang et al., 2026b; Ringel & Romano, 2026; Hu et al., 2026). Nevertheless, tree drafting has real limitations. It does not improve the drafter's top-1 accuracy, and its sparse, irregular attention makes tree drafting costly and complex

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

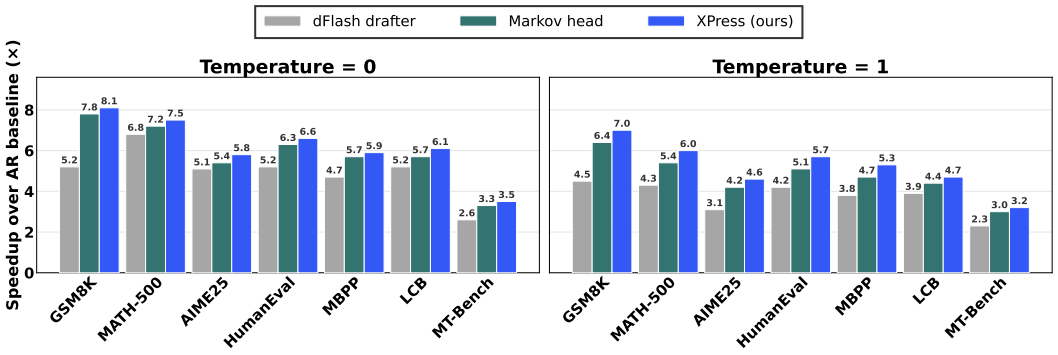


Figure 1: **End-to-end decoding throughput speedup over the autoregressive baseline** on Qwen3-8B with a block-16 dFlash drafter under greedy decoding (T=0, left) and lossless sampling (T=1, right). The throughput speedup values are measured on H200. XPRESS consistently outperforms the Markov head and dFlash diffusion drafter.

to serve (Miao et al., 2024). More importantly, since a tree spans tens to hundreds of tokens, its gains fade at large batch sizes, where the target pass is already compute-bound and extra tokens are no longer free. Therefore, we ask the following research question:

Can we re-inject causal information into the diffusion drafter, so as to increase acceptance length and raise end-to-end decoding throughput, without sacrificing the near-constant overhead?

Introducing causality naively reinforces the very sequential dependency that diffusion was designed to avoid. However, we can leverage the fact that the correct token is often already among the drafter's top-k candidates at each position (Wang et al., 2026b). The drafter's block-level marginals already narrow each position to a small candidate set, it just lacks the causal information needed to identify the right individual token. The problem of correction therefore reduces to picking the right token from a narrow preexisting set. Based on this insight, we formalize causality injection as a logits refinement problem and introduce XPRESS, a lightweight causal refiner that still runs in parallel. Specifically, XPRESS keeps the diffusion drafter's parallel block proposal intact and reranks it in place: a small head reads the drafter's hidden states together with the preceding sampled tokens, and emits a per-position additive causal bias on the diffusion drafter's own logits.

A straightforward implementation of causal refinement is left-to-right, with each position refined only after the positions before it are frozen. In practice, such strict sequential refinement is largely unnecessary. The diffusion drafter already produces high-quality drafts (Chen et al., 2026), and most positions in the initial draft are already correct. XPRESS exploits this strong initialization with Jacobi decoding (Song et al., 2021; Santilli et al., 2023), updating all positions in parallel from earlier positions in the previous refinement step. Since the initial draft is already mostly correct, only a few parallel iterations are needed for convergence. XPRESS therefore introduces:

- **Causal dependency injection at near-zero cost.** XPRESS injects causal information with a deliberately minimal mechanism: token embeddings and drafter hidden states are projected into low-rank r-space, where a single light-weight *lower-triangular* linear mixer lets every position aggregate information from its predecessors. The resulting logit bias is then applied to the diffusion drafter's own logits as a causal refinement.
- **Fast Parallel refinement.** XPRESS resolves the refiner's causal dependency with K Jacobi iterations over the whole draft, instead of a conventional left-to-right AR decoding process. Empirically, these iterations converge in far fewer steps than the diffusion block length ($K \ll B$), so a handful of batched forwards reproduce B serial autoregressive steps while retaining parallelism.
- **Consistent end-to-end gains.** On Qwen3-4B/8B, XPRESS achieves the highest speedup and acceptance length on all benchmarks under both greedy decoding and T=1 lossless sampling. Relative to dFlash, it improves the average acceptance length by 24–28% at T=0 and by 36–41% at T=1. These gains translate into end-to-end throughput: XPRESS is 1.23–1.26× faster than dFlash on average at T=0 and 1.32–1.40× at T=1.

2 Related Works

2.1 Diffusion drafters

Recent SD methods combine parallel diffusion drafting with AR verification. TiDAR (Liu et al., 2025) uses a hybrid model for quadratic self-speculation, while DiffuSpec (Li et al., 2025a) and SpecDiff-2 (Sandler et al., 2025) employ large pretrained dLLMs with inference-time search or train–test alignment to improve acceptance. In contrast, dFlash (Chen et al., 2026) uses a lightweight block-diffusion drafter to generate an entire block in one forward pass, and P-EAGLE (Hui et al., 2026) similarly converts the autoregressive EAGLE drafter into a parallel drafter that predicts all draft tokens in one forward pass. Nemotron-Labs-Diffusion (Fu et al., 2026) further enables linear self-speculation where the same model serves as both diffusion drafter and AR target. Another line better exploits the multiple candidates available at each diffusion position: PRESTO (Wang et al., 2026b), DDTree (Ringel & Romano, 2026), and JetSpec (Hu et al., 2026) construct draft trees and jointly verify multiple continuations, significantly improving acceptance. However, tree drafting hedges around the non-causality of the diffusion drafter rather than resolving it. By exploring multiple paths, tree drafting increases the chance that the correct token is verified, but it does not improve the drafter's top-1 accuracy. xPress is complementary to both directions: it refines the drafter's marginal predictions by restoring intra-block causal conditioning, and can therefore be combined with different block drafters and tree drafting methods.

2.2 Causal information injection for diffusion drafters

Closest to us, concurrent works Domino (Huang et al., 2026) and DSpark (Cheng et al., 2026) share our goal of restoring causality to diffusion drafters through lightweight correction heads. Domino processes the block left-to-right with a recurrent GRU, while DSpark uses a lighter Markov head conditioned on the previously sampled token. Both restore causal structure, but retain two limitations. First, they are inherently serial: Domino propagates recurrent state across positions, while DSpark conditions token k on the sampled identity of token $k{-}1$, requiring $B{-}1$ ordered steps for a block of B tokens. Second, their conditioning is relatively limited. DSpark only observes the previous token id and does not exploit the drafter's hidden states, while Domino summarizes preceding context through a recurrent state without accessing the entire block jointly. In contrast, xPress mixes over positions explicitly and, despite its causal output structure, reads the entire block's drafter hidden states and resolves its causal dependency with K parallel Jacobi passes.

3 Background

3.1 Speculative decoding

Speculative decoding (SD) uses a lightweight drafter to propose multiple future tokens that are jointly verified by the target model, which accepts the longest acceptable prefix plus one bonus token of its own. Under greedy decoding the accepted tokens are identical to standard decoding. For probabilistic sampling, rejection sampling against the draft distribution preserves the target distribution exactly. Let τ denote the expected acceptance length; the average per-token latency is then

$$\mathcal{L} \;=\; \frac{T_{\text{draft}} + T_{\text{verify}}}{\tau}, \tag{1}$$

so speedup over the target's autoregressive latency mainly comes from two levers: raising the acceptance length τ, or lowering the drafting cost T_{draft} (Chen et al., 2026).

3.2 Block-diffusion drafters

Block-diffusion drafters decouple drafting cost from draft length by predicting a block of B tokens in one parallel forward pass. Specifically, the drafter produces logits $s_k \in \mathbb{R}^{|V|}$ and marginals $q_k(x_k) = \mathrm{softmax}(s_k)$ for all positions k. Unlike an AR drafter, these predictions do not condition on the realized intra-block prefix $x_{<k}$, yielding the factorized proposal $\prod_{k=1}^{B} q_k(x_k)$, whereas the target verifies against the path-conditioned distribution $\prod_{k=1}^{B} p(x_k|x_{<k})$. This mismatch between parallel marginal drafting and causal verification limits acceptance length in diffusion-based SD (Wang et al., 2026b).

3.3 Jacobi decoding

Autoregressive decoding generates B tokens sequentially as $y_k = \arg\max_y p(y \mid c, y_{<k})$. Jacobi decoding (Song et al., 2021; Santilli et al., 2023) instead updates all positions in parallel from an

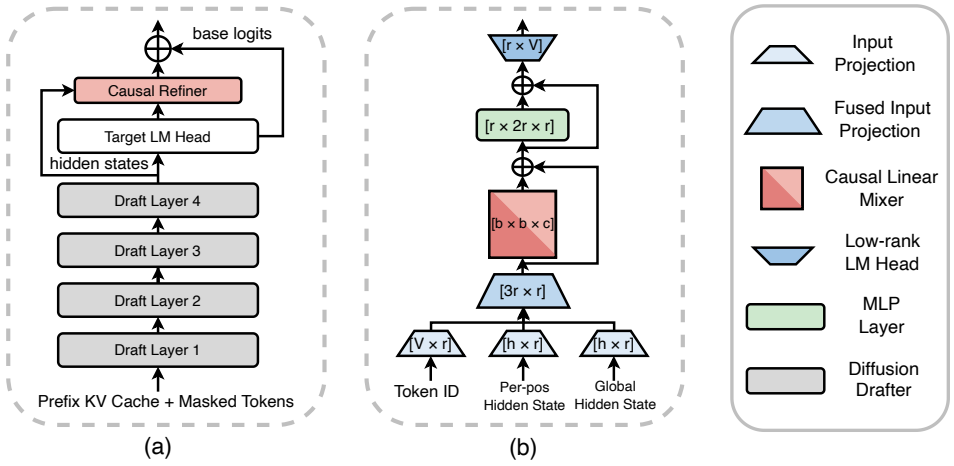


Figure 2: **(a)** The full pipeline. The block-diffusion drafter produces hidden states, the target LM head reads out the base logits, and the refiner adds a learned logits bias to form the final logits. **(b)** Inside the refiner. The three inputs (token id, global hidden state, and per-position hidden state) are down-projected into r-space, fused, mixed causally across block positions, passed through the r-space MLP, and read back out to vocabulary by the shared low-rank head.

initial guess $y^{(0)}$:

$$y_k^{(i+1)} \;=\; \arg\max_{y}\, p\Big(y \mid c,\, y_{<k}^{(i)}\Big), \qquad k = 1, \ldots, B, \tag{2}$$

Each iteration requires one parallel forward pass and converges to the greedy autoregressive output in at most B iterations (Song et al., 2021). In practice, cold-start Jacobi decoding often converges too slowly to provide substantial speedup (Santilli et al., 2023), motivating methods such as consistency LLMs (CLLMs) (Kou et al., 2024), which train intermediate Jacobi states to predict the final fixed point and thereby reduce the number of iterations.

4 xPress: Parallel Refinement for Diffusion Drafters

In this section, we first introduce the design of the lightweight causal refiner, using intra-block causal dependencies to shift the drafter's per-position marginals (Section 4.1). We then show how to resolve the refiner's causal dependency in parallel via Jacobi decoding, so a block converges in a few iterations instead of a serial left-to-right pass (Section 4.2). Finally, we describe the training of xPress, which is co-trained with the drafter under an acceptance-oriented objective, plus a consistency term that aligns training with the self-conditioned inputs seen during the refinement process (Section 4.3).

4.1 A lightweight causal refiner

Four properties govern our causal refinement mechanism. First, it should be **lightweight**, preserving the efficiency of the drafter's parallel forward pass. Second, it should be **drafter-grounded**, leveraging existing drafter representations such as hidden states, which encode richer information than token identities alone (Wang et al., 2026a). Third, it should be **causal**, conditioning each token on its predecessors rather than smoothing local marginals. Finally, it should incur **low overhead**, maintaining low latency and avoiding serial left-to-right passes that scale with draft length.

Our lightweight causal refiner instantiates all four properties. As shown in Figure 2(a), the diffusion drafter proposes the initial block in one pass, the target LM head reads out the base logits s_k, and the refiner adds a small per-position logit bias δ_k that re-ranks a narrow set of candidates the drafter already favours. Formally, the refiner is a map $\mathcal{R}_\theta : \big(h_{1:B},\, x_{0:B-1}\big) \longmapsto (\delta_1, \ldots, \delta_B) \in \mathbb{R}^{B\times V}$, with B, V the block length and vocab size, $h_{1:B}$ the drafter's hidden states, and $x_{0:B-1}$ the current draft tokens (x_0 a fixed anchor). Composed with the drafter's logits, the refiner induces an updated per-position distribution:

$$p_k^r(\cdot \mid x_{<k}, h_{1:B}) \;=\; \mathrm{softmax}\big(s_k + \delta_k\big), \qquad \delta_k = \mathcal{R}_\theta(h_{1:B}, x_{0:B-1})_k. \tag{3}$$

The design of $\mathcal{R}_\theta$ is shown in Figure 2(b). It consists of three lightweight stages: $\mathcal{R}_\theta = \mathrm{Fuse} \circ \mathrm{Mix} \circ \mathrm{Transform}$. Given V the vocabulary size, H the drafter's hidden dimension, B the block length, and r the low-rank dimension, we describe each stage below.

Drafter-grounded feature fusion. At each position k, the refiner combines three inputs: the drafter's per-position hidden state $h_k \in \mathbb{R}^H$, the previously sampled token x_{k-1}, and a block-global summary $g = \frac{1}{B}\sum_{j=1}^{B} h_j \in \mathbb{R}^H$. They are projected and fused in the low-dimensional r-space:

$$a_k = W_{\mathrm{in}}\big[\, W_h h_k \,\|\, W_g g \,\|\, W_e[x_{k-1}] \,\big] \in \mathbb{R}^r. \tag{4}$$

$W_e \in \mathbb{R}^{r\times V}$ is the token embedding, $W_h, W_g \in \mathbb{R}^{r\times H}$ are the down-projections for h_k and g, and $W_{\mathrm{in}} \in \mathbb{R}^{r\times 3r}$ is the fusion projection.

Causal position mixing. The fused feature a_k sees only token x_{k-1}, not the rest of the sampled prefix. We therefore let each position read its prefix directly, with a causal mixer:

$$c_k = a_k + \sum_{j\le k} w_{k,j} \odot a_j, \qquad w_{k,j} \in \mathbb{R}^r, \tag{5}$$

where $\odot$ denotes element-wise multiplication. Each of the r channels is mixed along the block independently. Writing $a^{(d)} \in \mathbb{R}^B$ for channel d across the B positions, the map is $c^{(d)} = (I_B + L^{(d)})\, a^{(d)}$ with $L^{(d)}_{k,j} = w_{k,j}[d]$ lower-triangular, so each position sees only itself and earlier latent embeddings. I_B leaves an identity path and we initialize $L^{(d)} = 0$, making the mixer a zero-initialized residual branch (Bachlechner et al., 2020). The overall refiner therefore begins as a bigram model (from the fused-feature input), and widens its receptive field during training. This mixer design - a fixed lower-triangular matrix per channel - is far cheaper than self-attention over the block, which also imposes a retrieval framing that is redundant, given the short sequence length.

Proposition 1 (Causal in the sampled tokens)**.** *For every position k, the correction δ_k depends on the sampled tokens only through the prefix $x_{<k}$. Consequently, the refined block factorizes autoregressively in the token argument:*

$$p^r(x_{1:B} \mid h_{1:B}) = \prod_{k=1}^{B} p^r_k\big(x_k \mid x_{<k},\, h_{1:B}\big), \tag{6}$$

recovering the path-conditioned structure that the drafter's marginals $\prod_k q_k(x_k)$ lack.

Transformation and output projection. Each position is then transformed independently by a gated MLP (Shazeer, 2020) with a residual connection,

$$z_k = c_k + W_{\mathrm{down}}\big(\mathrm{SiLU}(W_{\mathrm{gate}} c_k) \odot W_{\mathrm{up}} c_k\big), \qquad W_{\mathrm{gate}}, W_{\mathrm{up}} \in \mathbb{R}^{2r\times r}, \quad W_{\mathrm{down}} \in \mathbb{R}^{r\times 2r}, \tag{7}$$

where $\odot$ is again element-wise multiplication and all projections are bias-free. The refiner finally reads out through $W_r \in \mathbb{R}^{V\times r}$, giving the correction $\delta_k = W_r z_k$ and the refined logits $\ell_k = s_k + \delta_k$.

4.2 Parallel refinement via Jacobi decoding

The causal refiner lets each draft token depend on all its preceding tokens (Proposition 1), so the block can be decoded exactly in an autoregressive manner. The diffusion drafter alone, however, already produces high-quality drafts (Chen et al., 2026), so a left-to-right refinement would spend most of its serial steps re-deriving tokens the drafter already got right. xPress instead updates all positions in parallel with Jacobi iteration, avoiding serial refinement entirely.

Formally, sequential greedy decoding computes $y_k = \arg\max_v p_k(v \mid y_{<k}, h, g)$ in order, whereas Jacobi decoding solves the same equations from a seed by updating every position at once, based on prior predictions in prefix positions:

$$y_k^{(j+1)} = \arg\max_v \; p_k\big(v \mid y^{(j)}_{<k},\, h, g\big), \qquad k = 1, \ldots, B. \tag{8}$$

We seed with the drafter's own per-position argmax, $y_k^{(0)} = \arg\max_v q_k(v)$. The detailed refinement algorithm is shown in Appendix A. Note that the features h and g are computed once and stay fixed, so the tokens are the only thing that change between iterations, and the convergence above still holds.

Proposition 2 (Finite convergence to the sequential decode)**.** *Assume the per-position* $\arg\max$ *in equation 8 is unique (ties broken by a fixed rule) and that* h, g *are constant across iterations. Let* $Y^\star = (y_1^\star, \ldots, y_B^\star)$ *be the sequential greedy decode,* $y_k^\star = \arg\max_v p_k(v \mid y_{<k}^\star, h, g)$*. Then from* any *seed* $Y^{(0)}$*, the Jacobi iteration equation 8 satisfies*

$$y_k^{(j)} = y_k^\star \quad \textit{for all } j \geq k, \tag{9}$$

so it reaches $Y^\star$ *in at most* B *iterations. The complete proof is in Appendix B.*

Once a prefix of length n is correct, it never changes, and it makes position $n{+}1$ correct on the next pass at minimum, so stability spreads rightward from the anchor token. Proposition 2 is thus the worst case. In practice the drafter's seed is already a good guess, so many positions settle at once and K iterations lock in far more than K tokens.

Probabilistic sampling. For temperature $T > 0$, re-drawing the noise each iteration destroys the fixed point, so we draw it once per block and hold it fixed. Making the prefix dependence explicit as $\ell_k(v \mid y_{<k})$ and drawing an i.i.d. standard Gumbel field $G = (G_{k,v})$ once per block, the iteration becomes

$$y_k^{(j+1)} = \arg\max_v \left\{ \ell_k(v \mid y_{<k}^{(j)})/T + G_{k,v} \right\}, \qquad k = 1, \ldots, B. \tag{10}$$

Corollary 3. *Let* $G = (G_{k,v})$ *have i.i.d. standard Gumbel entries, drawn once per block independently of* h, g *and held fixed across Jacobi iterations. The iteration equation 10 reaches a unique fixed point* $Y^\star$ *within* B *iterations, and* $\Pr\left[Y^\star = y\right] = \prod_{k=1}^{B} \mathrm{softmax}\left(\ell_k(\cdot \mid y_{<k})/T\right)_{y_k}$*, i.e.* $Y^\star$ *is an exact temperature-*T *sample from the refined drafter. The complete proof is in Appendix C*

If we redraw G at every iteration, the sampled prefix itself might change, and so the iteration would not converge. Fixing G thus makes prefix locking safe, and once a prefix stops changing, the scores at the positions it feeds stop changing too (Proposition 1). With the same scores and the same noise, those positions pick the same tokens. Freezing the noise therefore gives exactly the result the full iteration would have reached.

Theorem (Losslessness at $T > 0$)**.** *Suppose the refiner reads only the prefix (Proposition 1), and* G *has i.i.d. standard Gumbel entries, drawn once per block and shared by the seed generation and all Jacobi iterations. Assume the seed is generated position-wise, so that* $y_k^{(0)}$ *depends on the Gumbel field only through* $G_{k,\cdot}$*. If the verifier receives* $Y^{(K)}$ *with proposals*$q_k = \mathrm{softmax}\big(\ell_k(\cdot \mid y_{<k}^{(K-1)})/T\big)$*, then for every* $K \geq 1$ *and* $T > 0$*, modified rejection sampling (Leviathan et al., 2023) emits tokens distributed exactly as standard autoregressive sampling from the target at* T*.*

The complete proof is in Appendix D. Because the drafter reads only the prefix, the noise row $G_{k,\cdot}$ driving $y_k^{(K)}$ is independent of everything used to form q_k. Each draft token is thus an exact sample from the proposal q_i reported to the target for verification. Rejection sampling therefore operates independently of Jacobi iteration, and K and the seed affect only the acceptance rate.

4.3 Training xPress

The causal refiner is co-trained with the drafter on ground-truth sequences and the predictions of the frozen target model, which contribute two complementary per-position terms. The first is a teacher-forced cross-entropy on the ground-truth token x_k^*, the standard next-token objective. The second targets acceptance directly: under speculative sampling a token drawn from p is accepted against the target p^t with probability $\sum_x \min\big(p(x), p^t(x)\big) = 1 - \mathrm{TV}(p, p^t)$ (Leviathan et al., 2023), so $\|p_k - p_k^t\|_1 = 2\big(1 - \text{accept rate}_k\big)$ and minimizing this ℓ_1 distance is exactly maximizing acceptance. Cross-entropy points the refiner at the right token, while the ℓ_1 term shapes the distribution the way acceptance is actually scored. We weight positions by $w_k = \exp(-(k{-}1)/\gamma)$ (Chen et al., 2026):

$$\mathcal{L}(p) = \sum_k w_k \Big[\alpha_{\mathrm{ce}}\big(-\log p_k(x_k^*)\big) + \alpha_{\ell_1} \|p_k - p_k^t\|_1\Big]. \tag{11}$$

A forward pass yields the drafter's base distribution $p_k^b = \mathrm{softmax}(s_k)$ and the refined p^r, and applying $\mathcal{L}$ to p^r alone is insufficient for two reasons. First, the Jacobi iteration is seeded from the drafter's own predictions, so a drafter that co-adapts to the refiner may drift and supply a worse seed. As the intervening token sampling is not differentiable, we instead anchor p^b with the same loss. Second, teacher forcing trains the refiner to correct gold prefixes, whereas at inference it sees

self-conditioned ones. Since refiner passes are cheap, we run a second pass whose token inputs are $\arg\max(p^r)$ and apply $\mathcal{L}$ to its output $\hat{p}^r$. The full objective is

$$\mathcal{L}_{\text{total}} = \underbrace{\mathcal{L}(p^r)}_{\text{refiner}} + \lambda \underbrace{\mathcal{L}(p^b)}_{\text{drafter anchor}} + \beta \underbrace{\mathcal{L}(\hat{p}^r)}_{\text{consistency}} . \tag{12}$$

5 EXPERIMENTS

5.1 EVALUATION SETTINGS

Models and Datasets. We evaluate XPRESS on Qwen3-4B/8B (Yang et al., 2025) as the target model with non-thinking mode, with the released block-16 dFlash drafter (Chen et al., 2026) as the base diffusion drafter. For training dataset, we use Open-PerfectBlend (Xu et al., 2024). Following the dFlash recipe, assistant responses are regenerated by the target model under its chat template in non-thinking mode, so the refiner is trained on the distribution it must correct at inference. We evaluate on math benchmarks including GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2023), and AIME25 (MAA); coding benchmarks including HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2024); and chat tasks including MT-Bench (Zheng et al., 2023). None of the evaluation tasks overlap the training corpus.

Baselines and Training Settings. We compare against three baselines: the plain dFlash drafter (Chen et al., 2026); Domino's RNN correction head, evaluated with its officially released checkpoints (Huang et al., 2026); and DSpark's sequential Markov head (Cheng et al., 2026), retrained under our recipe: same data, loss schedule and step budget, so that it differs from XPRESS only in the correction architecture. We report the *acceptance length* τ and the *end-to-end decoding throughput speedup*, the ratio of speculative decoding throughput to that of autoregressive decoding with the same target. For XPRESS evaluation, we use $K = 7$ for best efficiency and accuracy trade-off. All heads are co-trained with the drafter against the frozen target using AdamW at learning rate 6×10^{-4}, with diffusion block size 16, and $r = 256$. The consistency loss on the drafter-seeded second pass is weighted by $\beta = 0.3$, and the anchor term holding the base drafter to the target is annealed from $\lambda = 0.6$ to a floor of 0.2, shifting capacity toward the refiner as the drafter adapts.

Implementation Details. For the single-batch evaluation, we use HuggingFace-based harness with `torch.compile` on all modules and CUDA-graph capture of the correction rollout. For serving-engine evaluation, we integrate XPRESS into vLLM (Kwon et al., 2023) to evaluate XPRESS performance on large batch sizes. Because GPU clocks on a shared cluster drift by several percent, every throughput number is the mean of five measured rounds. For evaluation, we use a single Nvidia H200 GPU. For the training process, we use 32 Nvidia H100 GPUs to train each setting.

5.2 RESULTS

Single Batch. Table 1 reports single-batch decoding speedup and acceptance length with block size 16. XPRESS is best in all configurations, outperforming both baselines in all seven benchmarks. At T=0, XPRESS achieves an average speedup of $6.9\times$ (τ=7.66) on Qwen3-4B and $6.2\times$ (τ=8.02) on Qwen3-8B. These correspond to $1.23\times$ and $1.26\times$ higher throughput than plain dFlash, with the largest improvement on GSM8K. Against the stronger Markov-head refiner, XPRESS still adds $+0.24$ and $+0.38$ accepted tokens per step on average (up to $+0.55$ on LCB). At T=1, dFlash degrades sharply, whereas XPRESS sustains $5.9\times$ and $5.2\times$, a $1.32\times$ and $1.40\times$ gain over dFlash and $1.09\times$ and $1.11\times$ over the Markov head. These results confirm the benefit of conditioning each drafted token on the full preceding context, rather than predicting positions independently as in the diffusion drafter or conditioning on a single previous token as in the Markov head.

Large Batches on vLLM. Tables 2 and 3 evaluate XPRESS inside vLLM's speculative-decoding stack at batch sizes 16 and 32 with greedy decoding. Specifically, XPRESS reaches the highest throughput in every cell of both tables, for example 13594 tok/s on GSM8K with Qwen3-8B at bs=32, against 9702 for dFlash and 4731 for the AR baseline. Speedups over AR shrink as the batch grows, since batching raises AR throughput by roughly 70% while the speculative methods are already compute-bound. At bs=32 XPRESS still delivers 2.5–$3.1\times$ on math and code for Qwen3-8B and 3.6–$4.3\times$ for Qwen3-4B, and it is 1.11–$1.40\times$ and 1.02–$1.33\times$ faster than dFlash at both batch sizes, so the refiner remains worthwhile under large batch sizes.

Table 1: Decoding speedup over the autoregressive baseline (Sp.) and average acceptance length (τ) on Qwen3-4B and Qwen3-8B, single-sequence decoding, at most 2048 generated tokens. At T=1, τ is averaged over 5 seeds. $\times$ **Gain** is xPRESS's throughput over the plain dFlash diffusion drafter.

Model	Method	MATH						CODE						CHAT		*Avg.*	
		GSM8K		MATH-500		AIME25		HumanEval		MBPP		LCB		MT-Bench		*Avg.*	
Temperature = 0		Sp.	τ	Sp.	τ	Sp.	τ	Sp.	τ	Sp.	τ	Sp.	τ	Sp.	τ	Sp.	τ
Q3-4B	dFlash	5.8×	6.32	6.5×	7.43	5.9×	6.96	6.0×	6.45	5.5×	5.75	6.2×	7.11	3.0×	3.40	5.6×	6.20
	Domino	8.3×	9.55	6.9×	8.31	6.3×	7.31	6.5×	6.96	5.8×	6.61	5.9×	7.12	3.3×	3.78	6.1×	7.09
	Markov head	8.6×	9.55	7.6×	8.84	6.4×	7.53	6.9×	7.40	6.5×	6.84	6.5×	7.53	3.8×	4.28	6.6×	7.42
	xPRESS (ours)	**8.9×**	**9.83**	**8.0×**	**9.28**	**6.8×**	**7.93**	**7.2×**	**7.62**	**6.6×**	**6.94**	**6.6×**	**7.63**	**3.9×**	**4.40**	**6.9×**	**7.66**
	$\times$ **Gain**	**1.53×**		**1.23×**		**1.15×**		**1.20×**		**1.20×**		**1.06×**		**1.30×**		**1.23×**	
Q3-8B	dFlash	5.2×	6.48	6.8×	7.71	5.1×	7.10	5.2×	6.44	4.7×	5.75	5.2×	7.11	2.6×	3.18	5.0×	6.25
	Domino	7.0×	9.52	6.9×	8.63	5.2×	7.37	5.8×	7.25	5.5×	6.74	5.2×	7.22	3.0×	3.88	5.5×	7.23
	Markov head	7.8×	9.67	7.2×	9.24	5.4×	7.95	6.3×	7.76	5.7×	6.90	5.7×	7.85	3.3×	4.13	5.9×	7.64
	xPRESS (ours)	**8.1×**	**10.11**	**7.5×**	**9.62**	**5.8×**	**8.35**	**6.6×**	**8.15**	**5.9×**	**7.11**	**6.1×**	**8.40**	**3.5×**	**4.38**	**6.2×**	**8.02**
	$\times$ **Gain**	**1.55×**		**1.10×**		**1.14×**		**1.27×**		**1.26×**		**1.17×**		**1.35×**		**1.26×**	
Temperature = 1		Sp.	τ	Sp.	τ	Sp.	τ	Sp.	τ	Sp.	τ	Sp.	τ	Sp.	τ	Sp.	τ
Q3-4B	dFlash	5.5×	5.96	5.2×	5.89	4.0×	4.66	5.2×	5.51	4.5×	4.86	4.3×	4.92	2.7×	3.00	4.5×	4.97
	Domino	7.0×	7.81	5.4×	6.00	3.9×	4.50	5.5×	5.97	4.9×	5.57	4.2×	5.08	3.0×	3.23	4.8×	5.45
	Markov head	7.4×	8.43	6.3×	7.53	4.6×	5.98	5.9×	6.83	5.3×	6.19	4.7×	6.10	3.3×	4.07	5.4×	6.45
	xPRESS (ours)	**7.9×**	**8.84**	**7.1×**	**8.00**	**5.3×**	**6.44**	**6.5×**	**7.22**	**5.8×**	**6.50**	**5.1×**	**6.19**	**3.6×**	**4.19**	**5.9×**	**6.77**
	$\times$ **Gain**	**1.44×**		**1.37×**		**1.32×**		**1.25×**		**1.29×**		**1.19×**		**1.33×**		**1.32×**	
Q3-8B	dFlash	4.5×	5.83	4.3×	5.71	3.1×	4.35	4.2×	5.40	3.8×	4.83	3.9×	5.58	2.3×	2.90	3.7×	4.94
	Domino	5.8×	7.70	4.5×	6.05	3.2×	4.59	4.5×	5.90	4.2×	5.60	4.1×	6.08	2.5×	3.20	4.1×	5.59
	Markov head	6.4×	8.76	5.4×	7.60	4.2×	6.22	5.1×	6.99	4.7×	6.39	4.4×	6.43	3.0×	3.97	4.7×	6.62
	xPRESS (ours)	**7.0×**	**9.20**	**6.0×**	**8.12**	**4.6×**	**6.68**	**5.7×**	**7.36**	**5.3×**	**6.68**	**4.7×**	**6.65**	**3.2×**	**4.13**	**5.2×**	**6.97**
	$\times$ **Gain**	**1.56×**		**1.40×**		**1.48×**		**1.36×**		**1.39×**		**1.21×**		**1.39×**		**1.40×**	

Table 2: Throughput (tok/s), speedup over baseline, and average acceptance length τ on vLLM (FA3 backend) for Qwen3-4B.

Method	GSM8K	MATH	HEval	MBPP	MT-B
Acceptance length τ					
dFlash	6.33	7.61	6.46	5.89	3.28
xPRESS	**9.86**	**9.01**	**7.62**	**7.00**	**4.25**
Throughput (tok/s), bs = 16					
AR	2428	1602	2253	1767	1604
dFlash	9549	8296	9190	7203	4283
	3.9×	5.2×	4.1×	4.1×	2.7×
xPRESS	**12528**	**8749**	**10055**	**7563**	**4813**
	5.2×	**5.5×**	**4.5×**	**4.3×**	**3.0×**
Throughput (tok/s), bs = 32					
AR	4749	2985	3570	2628	2701
dFlash	13413	12342	12442	9125	5272
	2.8×	4.1×	3.5×	3.5×	2.0×
xPRESS	**17794**	**12900**	**12709**	**10147**	**6054**
	3.7×	**4.3×**	**3.6×**	**3.9×**	**2.2×**

Table 3: Throughput (tok/s), speedup over baseline, and average acceptance length τ on vLLM (FA3 backend) for Qwen3-8B.

Method	GSM8K	MATH	HEval	MBPP	MT-B
Acceptance length τ					
dFlash	6.41	7.62	6.40	5.75	3.20
xPRESS	**10.13**	**9.67**	**8.05**	**7.18**	**4.31**
Throughput (tok/s), bs = 16					
AR	2794	2665	2722	2432	2415
dFlash	8812	9756	8504	7706	4304
	3.2×	3.7×	3.1×	3.2×	1.8×
xPRESS	**12049**	**11530**	**9516**	**8770**	**5337**
	4.3×	**4.3×**	**3.5×**	**3.6×**	**2.2×**
Throughput (tok/s), bs = 32					
AR	4731	4455	4553	3832	3789
dFlash	9702	11798	9858	8856	4890
	2.1×	2.6×	2.2×	2.3×	1.3×
xPRESS	**13594**	**13667**	**11166**	**9859**	**5982**
	2.9×	**3.1×**	**2.5×**	**2.6×**	**1.6×**

5.3 ABLATIONS AND ANALYSIS

How many iterations are needed? More Jacobi iterations lock in more of the prefix, but τ need not rise monotonically with K: once the accepted prefix passes the settled region, an extra iteration can overwrite an unconverged tail token that happened to match the target. In practice (Figure 3), τ rises quickly, passes the Markov baseline within a few iterations, and plateaus by $K \approx 7/8$. This fast convergence comes from the seed as the diffusion drafter's draft already places some positions near the fixed point. After the plateau, more iterations only add refinement latency, so throughput drops as shown in Figure 4a (xPRESS still achieves better throughput than the Markov head overall).

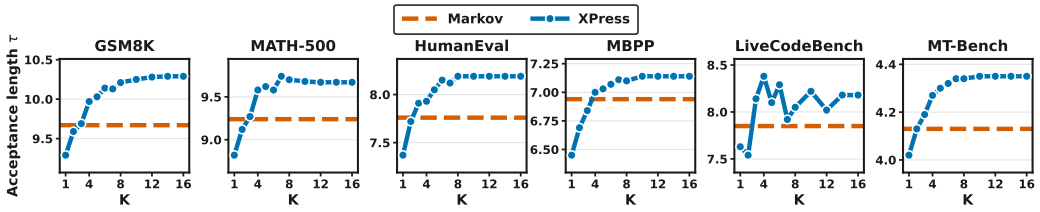
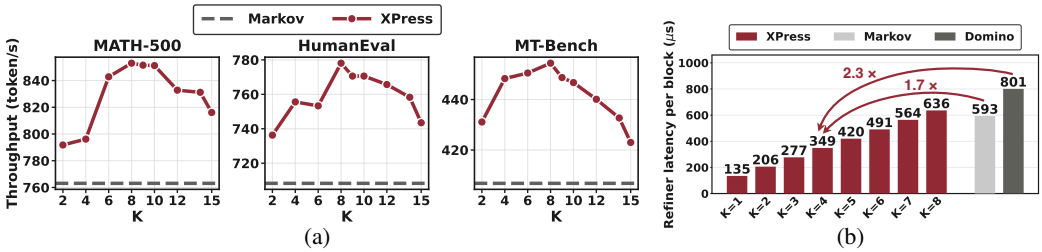


Figure 3: Acceptance length τ versus the number of K for Qwen3-8B.



(a) (b)

Figure 4: Effect of K on XPress: (a) end-to-end decode throughput and (b) refinement latency.

Drafting-time latency. As shown in Figure 4b, XPRESS's refiner cost grows linearly with K, at about 71 µs per Jacobi iteration. Both baselines run a fixed 15-step serial decode regardless of K, costing 593 µs for the Markov head and 801 µs for Domino with more computation. XPRESS crosses the Markov head only between $K{=}7$ and $K{=}8$, but it never needs to go that far. By $K{=}4$ it already matches or beats the Markov head's acceptance on every benchmark (Figure 3), and there it costs just 349 µs, which is $1.7\times$ faster than Markov and $2.3\times$ faster than Domino.

Table 4: Ablation on training objectives.

Objective	Math	Code	Chat
$\mathcal{L}(p^r) + \lambda\mathcal{L}(p^b) + \beta\mathcal{L}(\hat{p}^r)$	9.36	7.89	4.38
$\mathcal{L}(p^r) + \lambda\mathcal{L}(p^b)$	9.13	7.72	4.15
$\mathcal{L}(p^r) + \beta\mathcal{L}(\hat{p}^r)$ (co-train)	9.17	7.81	4.30
$\mathcal{L}(p^r) + \beta\mathcal{L}(\hat{p}^r)$ (frozen)	8.71	7.43	3.41

Table 5: Ablation on refiner architecture.

Design	Math	Code	Chat
w/o per-pos hidden state	9.05	7.72	4.19
w/o global hidden state	9.15	7.83	4.21
w/o MLP	9.09	7.74	4.17
w/o Mixer	8.98	7.70	4.10

Training Objective. We ablate each term of the training objective in Table 4. Removing the consistency loss $\beta\mathcal{L}(\hat{p}^r)$ lowers τ by up to 0.23, as the refiner no longer sees the self-conditioned inputs it receives during refinement. Freezing the drafter costs up to 0.89, since co-training lets the two modules adapt to each other. Even so, the refiner on a frozen drafter still lifts τ over dFlash from $7.10/6.43/3.18$ to $8.71/7.43/3.41$ on Math/Code/Chat, so most of the gain comes from the refiner itself. The drafter loss $\lambda\mathcal{L}(p^b)$ adds up to 0.19 more by providing better seeds for Jacobi refinement.

Refiner Architecture. Each variant in Table 5 is trained from scratch under the identical recipe. Removing the causal mixer costs the most, indicating that letting a position condition on its predecessors is crucial for accuracy. Removing the global hidden state costs the least. This means the per-position hidden states already carry some global information due to the bidirectional diffusion mechanism. Removing the per-position hidden state also costs a lot, confirming that hidden states can provide more information than the pure token id. Finally, removing the MLP costs about as much, as its non-linearity is also necessary to make the refiner more expressive.

6 CONCLUSION

XPRESS is a lightweight causal refiner for block-diffusion drafters. It reads the drafter's own hidden states and reconciles the whole block through a few parallel Jacobi iterations, restoring the token-to-token dependencies that a parallel drafter drops, without falling back to a serial, left-to-right decode. Across seven math, code, and chat benchmarks on Qwen3-8B, XPRESS raises the dFlash drafter's acceptance length by about 35% in average and its end-to-end throughput by about $1.3\times$ in average. Moreover, by running four Jacobi iterations, XPRESS already matches or beats the Markov head's acceptance length while running about $1.7\times$ faster in terms of the refinement latency.

AI USE STATEMENT

In this work, we used generative AI tools to help formulate mathematical claims and to assist in the writing of proofs. We have not used generative AI tools to generate synthetic data sets, develop theoretical models or conceptual frameworks, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation, clean or reformat datasets, support qualitative and thematic data analysis, or interpret results. Additionally, we used generative AI tools to draft parts of the paper and to edit the paper to improve readability.

We have reviewed all AI-assisted work. AI tools were used to help state mathematical claims precisely and to write out proofs; the key ideas underlying the proofs were developed by the authors. Every AI-assisted statement and proof was checked line by line for correctness by two authors, and we verified that the assumptions and conclusions of each claim are accurately stated. All AI-drafted or AI-edited text was reviewed and revised by the authors to ensure technical accuracy and to avoid introducing unsupported claims. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

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.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W. Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth, 2020. URL https://arxiv.org/abs/2003.04887.

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.

J. Chen, Y. Liang, and Z. Liu. Dflash: Block diffusion for flash speculative decoding. 2026. 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, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. 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/2607.05147.

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.

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, Chenhan Yu, Ye Yu,

Weijia Chen, Sajad Norouzi, Jingyu Liu, Shiyi Lan, Ligeng Zhu, Jin Wang, Jindong Jiang, Morteza Mardani, Mehran Maghoumi, Song Han, Ante Jukić, Nima Tajbakhsh, Jan Kautz, and Pavlo Molchanov. Nemotron-labs-diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding, 2026. URL https://arxiv.org/abs/2607.05722.

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, 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, 2026. URL https://arxiv.org/abs/2605.29707.

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

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.

S. Kou, L. Hu, Z. He, Z. Deng, and H. Zhang. Cllms: Consistency large language models. *ICML*, 2024. arXiv:2403.00835.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. *ICML*, 2023. arXiv:2211.17192.

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. Diffuspec: Unlocking diffusion language models for speculative decoding, 2025a. URL https://arxiv.org/abs/2510.02358.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *ICML*, 2024a. arXiv:2401.15077.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees, 2024b. URL https://arxiv.org/abs/2406.16858.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test, 2025b. 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.

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. Tidar: Think in diffusion, talk in autoregression, 2025. URL https://arxiv.org/abs/2511.08923.

MAA. American invitational mathematics examination (aime). https://maa.org/math-competitions/aime. Mathematics Competition Series; n.d.a.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating large language model serving

with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, pp. 932–949. ACM, April 2024. doi: 10.1145/3620666.3651335. URL http://dx.doi.org/10.1145/3620666.3651335.

L. Ringel and Y. Romano. Accelerating speculative decoding with block diffusion draft trees (ddtree). 2026. arXiv:2604.12989.

Jameson Sandler, Jacob K. Christopher, Thomas Hartvigsen, and Ferdinando Fioretto. Specdiff-2: Scaling diffusion drafter alignment for faster speculative decoding, 2025. URL https://arxiv.org/abs/2511.00606.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodola. Accelerating transformer inference for translation via parallel decoding. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12336–12355. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.689. URL http://dx.doi.org/10.18653/v1/2023.acl-long.689.

Noam Shazeer. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Yang Song, Chenlin Meng, Renjie Liao, and Stefano Ermon. Accelerating feedforward computation via parallel nonlinear equation solving. *ICML*, 2021. arXiv:2002.03629.

S. L. Wang, P. Isola, and B. Cheung. The truth lies somewhere in the middle (of the generated tokens). 2026a. arXiv:2605.09969.

Z. Wang, Z. Ye, Y. Fu, Q. Cheng, Z. Wang, F. Zhu, H. Zhao, H. Shi, P. Molchanov, and M. Zhang. Presto: Prefix-aligned tree drafting for diffusion speculative decoding. In *ICML Workshop (AdaptFM)*, 2026b. OpenReview:VKD70LJaDy.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm, 2025. URL https://arxiv.org/abs/2509.26328.

Tengyu Xu, Eryk Helenowski, Karthik Abinav Sankararaman, Di Jin, Kaiyan Peng, Eric Han, Shaoliang Nie, Chen Zhu, Hejia Zhang, Wenxuan Zhou, Zhouhao Zeng, Yun He, Karishma Mandyam, Arya Talabzadeh, Madian Khabsa, Gabriel Cohen, Yuandong Tian, Hao Ma, Sinong Wang, and Han Fang. 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.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

A JACOBI DECODING ITERATIONS FOR XPRESS REFINEMENT PROCESS

Algorithm 1 Parallel refinement via Jacobi iteration

Require: drafter hidden states $\{h_k\}_{k=1}^{B}$, block summary g, base logits $\{s_k\}_{k=1}^{B}$, anchor token y_0, iterations K
Ensure: refined block $Y^{(K)}$
1: $y_k^{(0)} \leftarrow \arg\max_v s_{k,v}$ **for** $k = 1, \ldots, B$ ▷ seed from the drafter's one-shot predictions
2: **for** $j = 0, 1, \ldots, K-1$ **do**
3: $(\delta_1, \ldots, \delta_B) \leftarrow \textsc{Refiner}\big(Y^{(j)},\, y_0,\, \{h_k\},\, g\big)$ ▷ one parallel forward; position k reads only $y_{<k}$
4: **for** $k = 1, \ldots, B$ **in parallel do**
5: $y_k^{(j+1)} \leftarrow \arg\max_v \big(s_k + \delta_k\big)_v$
6: **end for**
7: **end for**
8: **return** $Y^{(K)}$

Greedy decoding ($T = 0$). The seed is the drafter's own per-position argmax, read off logits the drafter has already computed. Inside the loop h, g and s never change, so every iteration applies the same update rule and the refiner's hidden-dependent half is computed once and reused. Since position k reads only $y_{<k}$, and y_0 is given, position 1 is final after the first iteration, position 2 after the second, and so on. Each iteration settles one more position, and after at most B iterations the block is exactly what sequential greedy decoding would produce. Stopping at $K < B$ therefore leaves the first K positions exact and can differ from sequential decoding only on the tail. In practice the drafter's seed is already a good guess, so many positions settle at once and K iterations lock in far more than K tokens.

Probabilistic Sampling ($T > 0$). Each $\arg\max$ becomes a Gumbel-max draw, $y_k \leftarrow \arg\max_v \big((s_k + \delta_k)_v/T + \varepsilon_{k,v}\big)$ with $\varepsilon_{k,v} \sim \mathrm{Gumbel}(0,1)$, which samples from $\mathrm{softmax}((s_k + \delta_k)/T)$. The only change to Algorithm 1 is that ε is drawn once per block, before the loop, and shared by the seed and all K iterations. Freezing it is what keeps the analysis intact: with ε fixed the update is again a deterministic, causal rule, so the argument above applies unchanged, with the fixed point now the block that sequential sampling would have produced under the same ε. Redrawing ε every iteration would apply a different rule each time, leaving no fixed point to converge to. It also keeps the loop free of per-iteration random-number generation, so the sampling path is captured in the same CUDA graph as the greedy one.

B PROOF OF FINITE CONVERGENCE OF THE REFINEMENT PROCESS

Proposition (Finite convergence to the sequential decode)**.** *Assume the per-position* $\arg\max$ *in equation 8 is unique (ties broken by a fixed rule), that h, g are constant across iterations, and that the anchor y_0 is given and never updated. Let $Y^\star = (y_1^\star, \ldots, y_B^\star)$ be the sequential greedy decode, $y_k^\star = \arg\max_v p_k(v \mid y_{<k}^\star, h, g)$. Then from* any *seed $Y^{(0)}$, the Jacobi iteration equation 8 satisfies*

$$y_k^{(j)} = y_k^\star \quad \textit{for all } j \geq k, \tag{13}$$

so it reaches $Y^\star$ in at most B iterations, and $Y^\star$ is its only fixed point.

Proof. By the causal structure of the refiner (Proposition 1), each $p_k(\cdot \mid y_{<k}, h, g)$ depends on the token guesses only through the prefix $y_{<k}$. Combined with h, g being fixed, the update at position k is therefore a function of $y_{<k}$ alone. In particular, for $k = 1$, the prefix contains only the fixed anchor y_0, so the update at position 1 is constant across iterations.

We prove the invariant

$$y_k^{(j)} = y_k^\star \qquad \text{for all } k \leq j$$

by induction on j. The base case $j = 0$ is vacuous. Assume the invariant holds after iteration j, and fix any $k \leq j + 1$. Then every non-anchor token in the prefix $y_{<k}^{(j)}$ has index at most $k - 1 \leq j$, and hence agrees with its sequential counterpart by the induction hypothesis. Since the anchor y_0 is fixed as well, we have

$$y_{<k}^{(j)} = y_{<k}^\star.$$

Therefore,

$$\begin{aligned} y_k^{(j+1)} &= \arg\max_v p_k\big(v \mid y_{<k}^{(j)}, h, g\big) \\ &= \arg\max_v p_k\big(v \mid y_{<k}^{\star}, h, g\big) \\ &= y_k^{\star}. \end{aligned}$$

Thus the invariant holds after iteration $j+1$. By induction, $y_k^{(j)} = y_k^{\star}$ for every $j \geq k$, proving equation 13. In particular, taking $j = B$ gives

$$Y^{(B)} = Y^{\star},$$

so the Jacobi iteration reaches the sequential greedy decode in at most B iterations. Since the argument does not depend on the initialization $Y^{(0)}$, the conclusion holds from any seed.

For uniqueness, let $Y = (y_1, \ldots, y_B)$ be any fixed point. Then

$$y_k = \arg\max_v p_k(v \mid y_{<k}, h, g)$$

for every k. At position 1, the update depends only on the fixed anchor y_0 and the fixed quantities h, g, so necessarily $y_1 = y_1^{\star}$. Assuming inductively that $y_i = y_i^{\star}$ for all $i < k$, we have $y_{<k} = y_{<k}^{\star}$, and hence

$$y_k = \arg\max_v p_k(v \mid y_{<k}^{\star}, h, g) = y_k^{\star}.$$

Induction on k therefore forces $Y = Y^{\star}$, proving that $Y^{\star}$ is the unique fixed point.

The argument above uses only that, conditioned on all quantities other than the prefix, the update at position k is a deterministic function of $y_{<k}$. It therefore extends directly to $T > 0$ under a fixed Gumbel coupling. Specifically, for each position k and vocabulary item v, draw

$$G_{k,v} \sim \mathrm{Gumbel}(0, 1)$$

once per block and reuse the same realization across all Jacobi iterations. Conditioned on these noises, the update becomes

$$y_k^{(j+1)} = \arg\max_v \left\{ \log p_k\big(v \mid y_{<k}^{(j)}, h, g\big) + G_{k,v} \right\},$$

which is again a deterministic function of the prefix $y_{<k}^{(j)}$. Define $Y^{\star}$ to be the sequential Gumbel-max decode obtained using the same fixed noise realization:

$$y_k^{\star} = \arg\max_v \left\{ \log p_k\big(v \mid y_{<k}^{\star}, h, g\big) + G_{k,v} \right\}.$$

The same induction then applies unchanged, so for every fixed Gumbel realization,

$$y_k^{(j)} = y_k^{\star} \qquad \text{for all } j \geq k,$$

and the Jacobi iteration reaches exactly the block produced by sequential Gumbel-max sampling under that same realization in at most B iterations. Marginalizing over the Gumbel noises recovers the usual sequential sampling distribution. □

C Exact sampling with fixed Gumbel noise

Corollary. *Let $G = (G_{k,v})$ be as above, drawn independently of h, g and held fixed across Jacobi iterations, with ties in equation 10 broken by the fixed rule of Proposition 2. Then the iteration equation 10 reaches a unique fixed point $Y^{\star}$ within B iterations from any seed, and*

$$\Pr\left[Y^{\star} = u \mid h, g\right] \;=\; \prod_{k=1}^{B} \mathrm{softmax}\left(\ell_k(\cdot \mid u_{<k})/T\right)_{u_k} \qquad \textit{for every } u = (u_1, \ldots, u_B) \in \mathcal{V}^B,$$

i.e. $Y^{\star}$ is an exact temperature-T sample from the refined drafter.

Proof. Throughout, condition on h, g; since G is independent of h, g, its entries remain i.i.d. standard Gumbel under this conditioning.

Convergence. Fix a realization of G. The update equation 10 is then a deterministic map whose k-th coordinate depends only on the prefix $y^{(j)}_{<k}$. With the fixed tie-breaking rule, each coordinate update is therefore a deterministic function of its prefix. Hence the proof of Proposition 2 applies unchanged: from any seed $Y^{(0)}$, the iteration reaches a unique fixed point $Y^\star$ within B iterations.

Characterizing the fixed point. At the fixed point,

$$y^\star_k \;=\; \arg\max_v \big\{\ell_k(v \mid y^\star_{<k})/T + G_{k,v}\big\}, \qquad k = 1, \dots, B.$$

Thus $Y^\star$ is exactly the block obtained by sequential Gumbel-max sampling using the same noise realization G. Under the conditioning on h, g and the fixed anchor, $y^\star_1$ is a function of $G_{1,\cdot}$ alone, while $y^\star_k$ is a function of $y^\star_{<k}$ and $G_{k,\cdot}$. By induction on k, $y^\star_{<k}$ is therefore a function only of

$$G_{<k,\cdot} = (G_{1,\cdot}, \dots, G_{k-1,\cdot}).$$

Conditional law. Since the rows of G are independent, $G_{k,\cdot}$ is independent of $G_{<k,\cdot}$ and hence of $y^\star_{<k}$. Therefore, conditional on $y^\star_{<k} = u_{<k}$, the row $G_{k,\cdot}$ remains i.i.d. standard Gumbel, and

$$y^\star_k = \arg\max_v \big\{\ell_k(v \mid u_{<k})/T + G_{k,v}\big\}.$$

The Gumbel–max identity then gives

$$\Pr\left[y^\star_k = u_k \,\middle|\, y^\star_{<k} = u_{<k},\, h, g\right] = \mathrm{softmax}\left(\ell_k(\cdot \mid u_{<k})/T\right)_{u_k}.$$

Ties have probability zero, so the tie-breaking rule does not affect this probability.

Joint law. Applying the chain rule,

$$\begin{aligned}\Pr\left[Y^\star = u \mid h, g\right] &= \prod_{k=1}^{B} \Pr\left[y^\star_k = u_k \,\middle|\, y^\star_{<k} = u_{<k},\, h, g\right] \\ &= \prod_{k=1}^{B} \mathrm{softmax}\left(\ell_k(\cdot \mid u_{<k})/T\right)_{u_k},\end{aligned}$$

which is exactly the joint law of sequential temperature-T sampling from the refined drafter. □

D Proof of lossless probabilistic sampling

Theorem (Losslessness at $T > 0$)**.** *Suppose the refiner reads only the prefix (Proposition 1), and G has i.i.d. standard Gumbel entries, drawn once per block and shared by the seed generation and all Jacobi iterations. Assume the seed is generated position-wise, so that $y^{(0)}_k$ depends on the Gumbel field only through $G_{k,\cdot}$. If the verifier receives $Y^{(K)}$ with proposals$q_k = \mathrm{softmax}\big(\ell_k(\cdot \mid y^{(K-1)}_{<k})/T\big)$, then for every $K \geq 1$ and $T > 0$, modified rejection sampling (Leviathan et al., 2023) emits tokens distributed exactly as standard autoregressive sampling from the target at T.*

Proof. Throughout the proof, condition on the fixed model states h, g and the preceding context. Let the verifier randomness be independent of the Gumbel field G.

Modified rejection sampling is lossless provided that, whenever a draft position is reached, its draft token is an exact sample from the proposal distribution reported to the verifier, conditional on the history available before that position is verified (Leviathan et al., 2023). It therefore suffices to show that, for every position k,

$$\Pr\Big[y^{(K)}_k = v \mid \mathcal{H}_k\Big] = q_k(v), \tag{14}$$

where

$$\mathcal{H}_k = \sigma\Big(Y^{(0)}_{<k}, G_{<k,\cdot}, R_{<k}\Big),$$

and $R_{<k}$ denotes all verifier randomness used at positions strictly before k.

We first establish the prefix-local dependency structure of the Jacobi iterates. We claim that, for every iteration $j \geq 0$ and every position i,

$$y_i^{(j)} \quad \text{depends on the Gumbel field only through} \quad G_{\leq i,\cdot} \tag{15}$$

For $j = 0$, this follows from the position-wise seed construction. In particular, when the same Gumbel field is used to sample the initial diffusion draft,

$$y_i^{(0)} = \arg\max_v \{s_i(v)/T + G_{i,v}\},$$

so $y_i^{(0)}$ depends on G only through the row $G_{i,\cdot}$.

Now suppose equation 15 holds at iteration j. By the causal structure of the refiner (Proposition 1),

$$y_i^{(j+1)} = \arg\max_v \left\{ \ell_i\left(v \mid y_{<i}^{(j)}\right)/T + G_{i,v} \right\}.$$

By the induction hypothesis, each token in $y_{<i}^{(j)}$ depends only on Gumbel rows with indices strictly smaller than i. Hence $y_{<i}^{(j)}$ depends only on $G_{<i,\cdot}$, and the update above depends on the Gumbel field only through

$$\left(G_{<i,\cdot}, G_{i,\cdot}\right) = G_{\leq i,\cdot}.$$

This proves equation 15 for all j by induction.

Consequently, for every k,

$$y_{<k}^{(K-1)} \quad \text{and} \quad y_{<k}^{(K)}$$

depend on the Gumbel field only through $G_{<k,\cdot}$. In particular, the proposal distribution supplied to the verifier,

$$q_k = \mathrm{softmax}\left(\ell_k(\cdot \mid y_{<k}^{(K-1)})/T\right),$$

is $\mathcal{H}_k$-measurable.

The verification history before position k is also $\mathcal{H}_k$-measurable. Indeed, every earlier draft token $y_i^{(K)}$, $i < k$, and its corresponding proposal q_i depend only on Gumbel rows $G_{\leq i,\cdot} \subseteq G_{<k,\cdot}$. The target distribution at each such position is determined by the previously emitted prefix, and the accept/reject decision additionally uses only verifier randomness from $R_{<k}$. Thus, recursively, all verifier decisions before position k, including the event that position k is reached, are measurable with respect to $\mathcal{H}_k$.

Crucially, although the same Gumbel field is also used to generate the initial seed, $\mathcal{H}_k$ contains only the seed prefix $Y_{<k}^{(0)}$. By the position-wise seed construction, $Y_{<k}^{(0)}$ depends only on $G_{<k,\cdot}$ and contains no information about the current row $G_{k,\cdot}$. Since the rows of G are independent and the verifier randomness is independent of G, we therefore have

$$G_{k,\cdot} \perp \mathcal{H}_k. \tag{16}$$

For $K \geq 1$, the token at position k is determined by the final Jacobi update as

$$y_k^{(K)} = \arg\max_v \left\{ \ell_k\left(v \mid y_{<k}^{(K-1)}\right)/T + G_{k,v} \right\}.$$

Notice that after at least one Jacobi iteration, $y_k^{(K)}$ no longer depends on its own seed value $y_k^{(0)}$; only the seed and iterates at earlier positions can affect its prefix. Conditioned on $\mathcal{H}_k$, the logits

$$\ell_k\left(\cdot \mid y_{<k}^{(K-1)}\right)/T$$

are therefore fixed, while $G_{k,\cdot}$ remains an i.i.d. standard Gumbel vector by equation 16. The Gumbel–max identity then gives

$$\begin{aligned}\Pr\left[y_k^{(K)} = v \mid \mathcal{H}_k\right] &= \mathrm{softmax}\left(\ell_k(\cdot \mid y_{<k}^{(K-1)})/T\right)_v \\ &= q_k(v),\end{aligned}$$

which proves equation 14. Ties occur with probability zero and hence do not affect the distribution.

Since the event that position k is reached is itself $\mathcal{H}_k$-measurable, the same statement holds conditional on that event. Thus, whenever position k is processed by the verifier, $y_k^{(K)}$ is an exact conditional sample from the proposal q_k reported for that position.

Importantly, q_k need not be conditioned on the same prefix as the target distribution. Modified rejection sampling requires only that the proposed token have conditional law q_k; its acceptance rule and residual distribution correct this proposal to the target distribution. Therefore, conditioned on the verification history up to position k, modified rejection sampling emits the next token according to

$$p_k^{\mathrm{tar}}(\cdot \mid z_{<k}),$$

where p_k^{tar} is the target model's temperature-T distribution and $z_{<k}$ is the already emitted prefix.

Applying this argument successively to every emitted position gives

$$\Pr[z_{1:n} = u_{1:n}] = \prod_{k=1}^{n} p_k^{\mathrm{tar}}\big(u_k \mid u_{<k}\big),$$

which is exactly the ancestral autoregressive sampling law of the target model at temperature T.

The proof never requires the Jacobi iterates to have converged. The only requirement is $K \geq 1$, so that the token at position k has been updated from its prefix and no longer depends on its own seed value. Hence the result holds for every finite $K \geq 1$. □