FORK AND REGENERATE: MTP-GUIDED EARLY FORKING AND SUFFIX REGENERATION FOR BLOCK DIFFUSION SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Block diffusion accelerates speculative decoding by predicting multiple draft tokens in parallel. However, verification accepts only a matching prefix, so an early mismatch prevents the remaining draft tokens from being accepted. Moreover, later predictions cannot use the tokens selected at earlier masked positions in the same pass. To address these issues, we introduce MTP-guided early **fo**rking and suffix **re**generation (**FoRE**). Since earlier predictions determine how much of a draft can be accepted, we investigate whether the target's pretrained multi-token prediction (MTP) module can improve candidate selection at the first drafted position. Our analysis shows that MTP provides higher target-token coverage than DFlash at this position, motivating us to retain multiple MTP candidates and supply each to the block diffusion drafter before generating the remaining tokens. To further improve acceptance at later positions, draft refinement selects alternative tokens and regenerates the remaining tokens from each modified prefix. Training with visible prefixes of varying lengths enables the same block diffusion drafter to perform both stages, without an added correction head or a separately trained second drafter. The resulting drafts are verified together in one batched target-model forward pass, so the output follows the target's greedy decoding. Across seven benchmarks on Qwen3.5-9B and Qwen3.5-4B, our method achieves 1.53–$1.55\times$ the average acceptance length of DFlash and 1.28–$1.32\times$ its average decoding speedup, reaching 3.82–$3.93\times$ over autoregressive decoding. When applied to DFlash 2, our method also improves acceptance length and speedup.

1 INTRODUCTION

Autoregressive language models generate tokens one at a time, requiring a separate forward pass for each token. Speculative decoding reduces this cost by proposing several candidate tokens, which the target model verifies together in a single forward pass (Leviathan et al., 2023; Cai et al., 2024; Li et al., 2024b; 2025). By accepting multiple draft tokens at once, the target can advance generation with fewer target forward passes. Recently, DFlash further reduced drafting cost by introducing a block diffusion drafter that uses target-model hidden states to predict multiple tokens in parallel (Chen et al., 2026).

While block diffusion drafters predict multiple tokens in parallel, verification accepts only the longest draft prefix that matches the target. This gives rise to two challenges. First, an early mismatch prevents more tokens from being accepted than a later mismatch, making accurate predictions at earlier positions particularly important. Draft-tree methods provide alternative tokens at selected positions, increasing the chance of finding a longer matching sequence (Ringel & Romano, 2026; Oh et al., 2026; Zhang et al., 2026b; Wang et al., 2026b). Second, the drafter predicts each masked position independently given the target-model features and visible prefix, so later predictions do not condition on the tokens selected at earlier masked positions in the same pass (Qiang et al., 2026). Selecting an alternative token therefore leaves the subsequent predictions unchanged. To account for these token choices, existing work uses causal correction heads (Huang et al., 2026) or autoregressive adapters (Oda et al., 2026) to adjust later predictions, or a separately trained diffusion drafter to generate the remaining tokens again from selected prefixes (Zhang et al., 2026a). CURE replaces uncertain draft tokens and resamples their suffixes without retraining the drafter (Liu et al., 2026).

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.

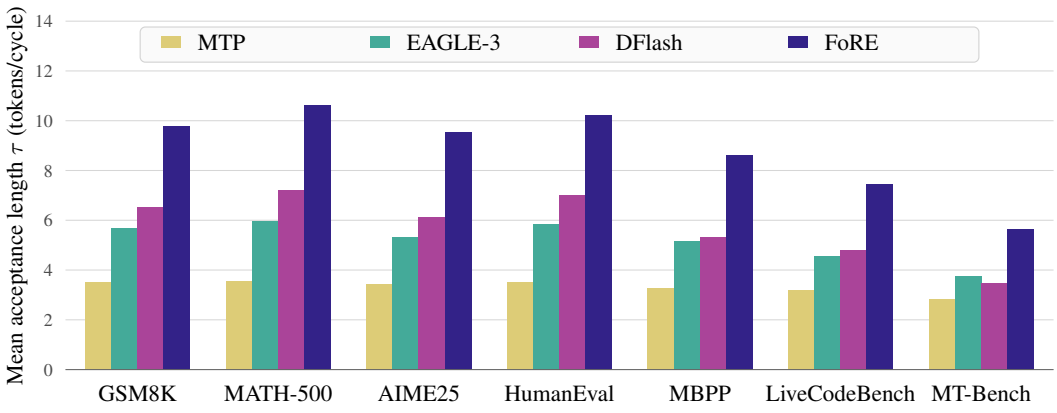


Figure 1: Mean acceptance length by benchmark on Qwen3.5-9B under greedy decoding. EAGLE-3 was evaluated in SGLang; the other methods were evaluated in our Transformers harness.

In this work, we jointly address these challenges through two techniques. First, to improve candidate selection at the first drafted position, we investigate the pretrained multi-token prediction (MTP) modules (Gloeckle et al., 2024; DeepSeek-AI, 2024) available in recent models, including Qwen3.5 and MiMo (Qwen Team, 2026a; Xiaomi LLM-Core Team, 2025). Our empirical comparison on Qwen3.5-9B shows that MTP achieves higher target-token coverage than DFlash at this position, but this advantage does not extend to later positions (Figure 3). Retaining multiple MTP candidates further increases coverage. These observations motivate **MTP-guided early forking**: we retain the top-ranked MTP candidates at the first drafted position and supply each candidate to the block diffusion drafter, which completes all candidates in one batched forward pass.

Second, because later mismatches can still stop acceptance, we propose **draft refinement via prefix-conditioned suffix regeneration**. We consider alternative tokens at later positions in the initial drafts. Since an alternative can extend acceptance only when the preceding tokens also match the target, we select replacements based on the probabilities assigned to both the preceding tokens and the alternative itself, following factorized prefix scoring in DDTree and BASTION (Ringel & Romano, 2026; Oh et al., 2026). We then supply each modified prefix to the same block diffusion drafter and regenerate the remaining tokens, allowing the new predictions to account for the replacement. The original and refined drafts are verified together in a single batched target-model forward pass. To support both passes, we train the same block diffusion drafter with visible prefixes of varying lengths, extending the variable-prefix training D^2SD uses for a separate second drafter (Zhang et al., 2026a), so one drafter completes MTP-selected candidates and regenerates tokens after later replacements without an added correction head or second drafter.

We evaluate FoRE using Qwen3.5-9B and Qwen3.5-4B as target models, both equipped with pretrained MTP modules, across seven benchmarks spanning mathematics, code generation, and conversation. FoRE achieves $1.53\times$ and $1.55\times$ the average acceptance length of DFlash (Chen et al., 2026), respectively, with the Qwen3.5-9B comparison shown in Figure 1. These gains translate into average decoding speedups of $3.93\times$ and $3.82\times$ over autoregressive decoding, corresponding to $1.28\times$ and $1.32\times$ the average speedup achieved by DFlash. We further apply our method to the newly released DFlash 2 (Inco AI, 2026) on Qwen3.8-27B (Qwen Team, 2026b), achieving $1.22\times$ its average acceptance length.

2 Related Work

Speculative Decoding. Speculative decoding can speed up autoregressive language model outputs by having a drafter propose multiple tokens that the target model verifies in a single forward pass (Leviathan et al., 2023). Early methods use a separate, smaller autoregressive model to generate these proposals one token at a time (Leviathan et al., 2023). Medusa adds multiple decoding heads to the target model to predict future tokens in parallel and uses tree attention (Miao et al., 2024) to verify candidate continuations (Cai et al., 2024). The EAGLE series instead uses hidden features

from the target model to guide drafting: EAGLE performs drafting in feature space, while EAGLE-3 combines features from multiple target layers to predict draft tokens directly (Li et al., 2024a; 2025). Beyond adding components specifically for speculative decoding, some models natively include modules trained to predict future tokens. For example, Qwen3.5 includes a pretrained multi-token prediction (MTP) module that can be applied repeatedly to draft tokens (Qwen Team, 2026a).

Block-Diffusion Speculative Decoding. Inspired by block diffusion models, which predict masked tokens within a block in parallel (Arriola et al., 2025), DFlash introduces a small diffusion drafter that takes target-model hidden states as input and generates draft tokens in a single forward pass (Chen et al., 2026). Subsequent work further improves parallel block drafting in two ways: by accounting for dependencies among draft tokens and by constructing multiple candidate sequences for verification. In the first direction, Domino adds a lightweight causal head that refines parallel predictions using preceding tokens (Huang et al., 2026), while DFlash 2 introduces local convolutions and a selector that scores adjacent token pairs (Inco AI, 2026), and LiLiCorr scores adjacent candidate pairs to select a coherent block (Rusanovsky et al., 2026). In the second direction, DDTree adapts OPT-Tree (Wang et al., 2025) to build a draft tree from the drafter's per-position marginals in one forward pass (Ringel & Romano, 2026), while BASTION and CaDDTree account for verification cost when selecting the tree size (Oh et al., 2026; Zhang et al., 2026b). PRESTO further incorporates a prefix-dependent score into candidate ranking to guide tree construction (Wang et al., 2026b). Moreover, several works jointly consider both directions. Weaver constructs multiple candidate sequences through an autoregressive adapter that conditions later predictions on earlier token choices (Oda et al., 2026). D^{2}SD retains selected prefixes from an initial draft and uses a separately trained diffusion drafter to predict the remaining tokens (Zhang et al., 2026a). CURE describes how it selects uncertain positions using confidence margins, substitutes alternative tokens, and resamples the remaining suffixes without retraining the drafter (Liu et al., 2026). FoRE also jointly considers both directions. Different from existing works, we use the target's pretrained MTP module to select candidates at the first drafted position of a block-diffusion drafter and train the drafter to complete their suffixes. Because the drafter is trained on supplied prefixes, it also regenerates suffixes after selected replacements at later positions, without an added correction head or a separate second drafter.

3 METHOD

3.1 PRELIMINARIES

Speculative decoding. Let x denote the context already processed by the target model. At the start of each decoding cycle, a target-produced token t_1 is available, and a lightweight drafter generates tokens $\hat{t}_2, \ldots, \hat{t}_B$ to form a block of length B, including t_1. The target verifies the block in one forward pass and, under greedy decoding, accepts the longest prefix matching its predictions (Leviathan et al., 2023). Its prediction at the first mismatch, or after the final position if all draft tokens are accepted, supplies the initial token for the next cycle. Let τ denote the average acceptance length, defined as the mean number of accepted draft tokens plus one target-produced token per decoding cycle, and let T_{draft} and T_{verify} denote the average drafting and verification times per cycle. Excluding unrelated overhead, the average time per output token is approximately $\frac{T_{\text{draft}}+T_{\text{verify}}}{\tau}$. Increasing acceptance length therefore improves decoding speed only when the gain outweighs any additional drafting and verification cost (Chen et al., 2026).

Block-diffusion drafting. Block diffusion models generate blocks sequentially while predicting masked tokens within each block in parallel (Arriola et al., 2025). DFlash drafts with a small block-diffusion model that reads hidden features of the target model computed from x (Chen et al., 2026). Its input consists of the target-produced token t_1 followed by $B-1$ masked positions:

$$[t_1, \texttt{[MASK]}, \ldots, \texttt{[MASK]}].$$

In one forward pass, the drafter outputs a token distribution q_j for each position $j = 2, \ldots, B$ and takes its argmax.

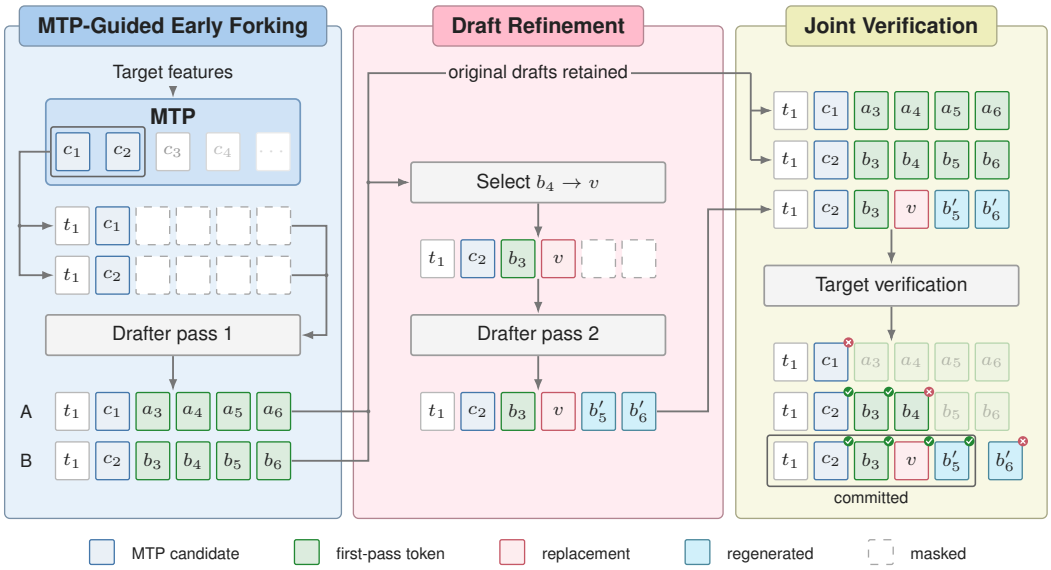


Figure 2: Overview of FoRE. The target's MTP module supplies top-K alternatives at t_2. The conditionally trained drafter completes the K forks in one batched pass. First-pass scores select up to R later replacement candidates, and the same drafter regenerates their suffixes in a second batched pass. The target verifies all $K + R$ complete branches together and commits the longest matching prefix. Two original branches and one regenerated fork are drawn; ticks and crosses mark accepted and first rejected tokens, and the frame marks the committed prefix.

3.2 MTP-Guided Early Forking

Let A_k be the event that at least k drafted tokens are accepted in a decoding cycle. Here, $\Pr(A_k)$ is the fraction of decoding cycles in which this occurs. A cycle with m accepted draft tokens satisfies $A_1, \ldots, A_m$. Because t_1 is supplied by the target model in every cycle:

$$\tau = 1 + \sum_{k=1}^{B-1} \Pr(A_k)$$

Every A_k requires a matching candidate at t_2, so we prioritize picking the right token at this position before completing the rest of the block. To illustrate this, note how on the baseline DFlash drafter an equal small improvement in conditional match probability at t_2 has approximately the same first-order effect on τ as that improvement at each of $t_6, \ldots, t_{16}$ combined, because every cycle reaches t_2 and only one in five reaches t_6 (see Appendix C).

Meanwhile, recent model families, including Qwen3.5 and MiMo (Qwen Team, 2026a; Xiaomi LLM-Core Team, 2025), increasingly support pretrained multi-token prediction (MTP) components that predict future tokens from target-model features. These readily available predictors motivate us to investigate whether MTP can help a block-diffusion drafter produce more accurate draft sequences. Specifically, we conduct an empirical study to compare MTP and DFlash across draft positions using Qwen3.5-9B as the target model. We measure how often the target-selected token appears among their top-K predictions for $K = 1$ and $K = 3$. This comparison allows us to identify where MTP provides more accurate predictions and how often retaining multiple candidates recovers the target token when the highest-probability prediction is incorrect.

Observation. As shown in Figure 3, we observe that: *1) MTP is more accurate at the first drafted position.* MTP achieves higher target-token coverage than DFlash at t_2 for both $K = 1$ and $K = 3$, but this advantage does not extend to later positions. *2) Multiple candidates improve coverage.* Retaining three MTP candidates increases coverage at t_2 over retaining only the top candidate.

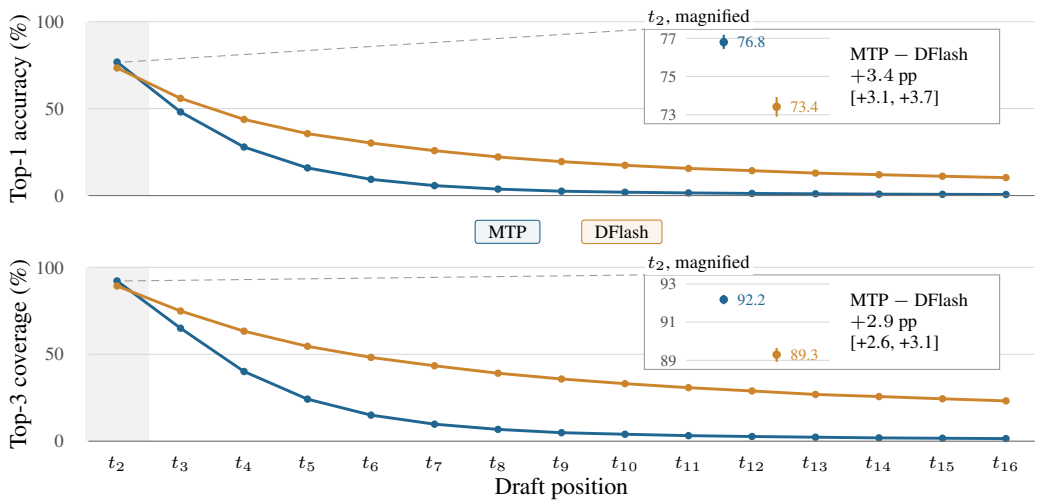


Figure 3: Accuracy of predicting the target's token at each draft position on Qwen3.5-9B. MTP is the target's own MTP module run recursively, each step getting its previous prediction. The DFlash drafter proposes the whole block in one pass. MTP leads only at t_2; insets magnify t_2.

These findings motivate us to propose **MTP-guided early forking**, which uses the target's pretrained MTP module to select multiple candidates at the first drafted position and conditions the diffusion drafter on each candidate to generate the remaining tokens. We implement it in three steps:

1. **Select candidates.** Retain the top-K tokens predicted by the target's pretrained MTP module at t_2, denoted $c_1, \ldots, c_K$.
2. **Construct input blocks.** For each candidate c_k, form a block of length B:
$$[t_1, c_k, \texttt{[MASK]}, \ldots, \texttt{[MASK]}].$$
3. **Generate the remaining tokens.** Process all K blocks in one batched forward pass through the diffusion drafter. For each block, the drafter predicts positions $t_3, \ldots, t_B$ conditioned on the target-model features and the supplied prefix $[t_1, c_k]$.

3.3 Draft Refinement via Prefix-Conditioned Regeneration

MTP-guided early forking provides multiple candidates at the first drafted position, but mismatches at later positions can still stop acceptance. Moreover, the diffusion drafter predicts the remaining positions in parallel, so later predictions do not observe the tokens selected at earlier masked positions in the same pass. To address these limitations, we propose **draft refinement via prefix-conditioned regeneration**, which selects alternative tokens using the probabilities computed during MTP-guided early forking and regenerates the remaining tokens with each modified prefix supplied as input. Prior work adds a correction head (Huang et al., 2026), trains a separate second drafter (Zhang et al., 2026a), or resamples suffixes with an unmodified drafter (Liu et al., 2026); here, the same drafter, trained on supplied prefixes, produces these additional drafts.

Draft token replacement. We consider replacing tokens at positions after t_2 in the K draft sequences produced by MTP-guided early forking. A replacement can extend the accepted prefix only if the preceding tokens also match the target. Following the prefix scoring used in DDTree and BASTION (Ringel & Romano, 2026; Oh et al., 2026), we therefore rank each replacement candidate using the probabilities assigned to both its retained prefix and alternative token. Let $d_{k,j}$ denote the token at position j in draft k, and let $q_{k,j}$ denote the corresponding token distribution computed during MTP-guided early forking. For a candidate that replaces $d_{k,m}$ with $v \neq d_{k,m}$ at position $3 \leq m \leq B$, we define

$$s(k, m, v) = q_{\text{MTP}}(c_k) \left(\prod_{j=3}^{m-1} q_{k,j}(d_{k,j}) \right) q_{k,m}(v), \tag{1}$$

where $q_{\mathrm{MTP}}(c_k)$ is the MTP probability of candidate c_k, and the product is 1 when $m = 3$. The score combines the MTP probability at t_2 with the diffusion drafter's probabilities for the retained tokens and the replacement token. We use this score to prioritize replacement candidates and retain up to R as additional drafts, where R is the replacement budget. Since the scores reuse token distributions computed during MTP-guided early forking, this selection requires no additional drafter forward pass. Each selected replacement defines a modified prefix for subsequent suffix regeneration.

Prefix-conditioned regeneration. For each selected replacement (k, m, v), we retain the tokens before position m, insert v, and mask the remaining positions:

$$\big[\underbrace{t_1, c_k, d_{k,3}, \ldots, d_{k,m-1}, v}_{\text{modified prefix}}, \underbrace{\texttt{[MASK]}, \ldots, \texttt{[MASK]}}_{B-m \text{ positions}}\big].$$

Here, the retained-token segment after c_k is empty when $m = 3$. Simply keeping the original suffix after this replacement can already recover some accepted tokens, as the matched analysis in Appendix D shows. However, that suffix was generated without the replacement token as input. We therefore process the modified blocks using the same trained diffusion drafter in a second batched forward pass, generating the remaining tokens conditioned on each modified prefix and the target-model features. Replacements at $m = B$ require no suffix regeneration.

Joint verification. We combine the K drafts from MTP-guided early forking with the R additional drafts and verify all $K+R$ sequences as separate rows in a single batched target-model forward pass. Retaining the original drafts, as in D^{2}SD and CURE (Zhang et al., 2026a; Liu et al., 2026), allows verification to use them when refinement does not produce a longer matching prefix. We select the longest prefix matching the target's greedy predictions and append the next target-produced token.

3.4 Training with Variable-Length Prefixes

Both MTP-guided early forking and draft refinement require the block diffusion drafter to predict the remaining tokens from a supplied prefix. In MTP-guided early forking, this prefix contains the target-produced token t_1 and an MTP-selected candidate at t_2; in draft refinement, it also contains retained draft tokens and a replacement at a later position. We therefore fine-tune the block diffusion drafter on blocks with visible prefixes of varying lengths, allowing the same model to complete MTP-selected candidates and regenerate suffixes after later replacements. D^{2}SD trains a separate second drafter on ground-truth prefixes of sampled length (Zhang et al., 2026a); we apply this training to the same drafter used for MTP-guided early forking.

We use teacher forcing to train the block diffusion drafter to predict masked tokens from ground-truth prefixes of varying lengths. Specifically, in each diffusion draft block, we sample the number L of ground-truth tokens to reveal after t_1, keep these tokens visible, and mask the remaining positions. The resulting input is $[t_1, t_2, \ldots, t_{L+1}, \texttt{[MASK]}, \ldots, \texttt{[MASK]}]$, with $B - L - 1$ masked positions. This construction covers three cases, allowing the drafter to learn ordinary block drafting as well as prediction from the supplied prefixes used in our two inference stages. When $L = 0$, only t_1 is visible, and the drafter predicts all remaining positions, retaining the ordinary block-drafting task. When $L = 1$, the ground-truth token at t_2 is also visible, teaching the drafter to predict the remaining tokens from a two-token prefix, as required by MTP-guided early forking. When $L \geq 2$, the drafter receives a longer ground-truth prefix, matching the input formats used in draft refinement. To include all three cases during training, we allocate half of the blocks to $L = 0$ and half to $L > 0$. Within the latter half, 35% use $L = 1$, matching the prefix format of MTP-guided early forking, and the remaining 65% use $L \geq 2$, covering the longer prefixes used in draft refinement. We sample these longer prefixes according to

$$P(L = \ell \mid L \geq 2) \propto 0.8^{\ell-2}, \qquad 2 \leq \ell \leq B - 1.$$

This distribution favors shorter prefixes while still including later refinement positions. When $L = B - 1$, no masked positions remain and the block contributes no token loss. For blocks with masked positions, we compute token-level cross-entropy only at masked positions and apply D-PACE (Wu et al., 2026) to adaptively weight the loss terms using the drafter's predicted probabilities. Visible prefix tokens serve as conditioning inputs and do not contribute to the loss.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models and evaluations. We use Qwen3.5-9B and Qwen3.5-4B as our primary and secondary target models respectively, both of which include pretrained multi-token prediction (MTP) modules (Qwen Team, 2026a). Following DFlash (Chen et al., 2026), we evaluate on seven benchmarks spanning mathematics, code generation, and conversation. The mathematics benchmarks include GSM8K, MATH-500, and AIME25 (Cobbe et al., 2021; Hendrycks et al., 2021; Lightman et al., 2024; Balunovic et al., 2025). The code generation benchmarks include HumanEval, MBPP, and LiveCodeBench (LCB) (Chen et al., 2021; Austin et al., 2021; Jain et al., 2025), while MT-Bench evaluates conversational workloads (Zheng et al., 2023). We report average acceptance length τ, including one target-produced token per decoding cycle, and decoding speedup over autoregressive decoding with the same target model. We use 128 prompts per benchmark, except AIME25 and MT-Bench, which use 30 and 80 prompts, respectively. All methods receive the same prompts under evaluation seed 0 and use greedy decoding with at most 2,048 generated tokens per response.

Training data. Following DFlash (Chen et al., 2026), we construct our training data from prompts in the NVIDIA Nemotron-Post-Training-Dataset-v2 (Nathawani et al., 2025) and CodeAlpaca-20k (Chaudhary, 2023), with responses generated by the target model. After removing a held-out split and examples that overlap five of the evaluation benchmarks, the Qwen3.5-9B training set contains 97,896 examples (details in Appendix A.1). Matched training controls use the same initialization, data, loss, optimization schedule, training duration, and seed, changing only the evaluated factor.

Implementation. We initialize our five-layer block diffusion drafter from the pretrained DFlash drafter (Chen et al., 2026) and fine-tune it for one epoch. Unless otherwise specified, we use a block size of $B = 16$, $K = 8$ MTP candidates, and $R = 8$ replacements. All methods run in the same Transformers (Wolf et al., 2020) evaluation harness on a single NVIDIA H200 NVL GPU with FlashAttention-2 (Dao, 2024). We exclude target prefill and the initial drafter prefill from decoding time and average up to three timing repetitions for each method and benchmark. Complete training and timing settings are provided in Appendices A.1 and A.2.

Baselines. We compare with three speculative decoding baselines: **MTP**, which uses the target's pretrained MTP module for drafting (Qwen Team, 2026a); **EAGLE-3**, an autoregressive drafting method using target-model features (Li et al., 2025); and **DFlash**, which generates draft tokens in parallel using a block diffusion drafter (Chen et al., 2026). Autoregressive decoding with the same target model serves as the reference for computing speedup.

4.2 MAIN RESULTS

Table 1 compares decoding speedup and average acceptance length on Qwen3.5-9B and Qwen3.5-4B across seven benchmarks. FoRE outperforms DFlash and MTP on both metrics and has a higher acceptance length than EAGLE-3 across all benchmarks. On Qwen3.5-9B, average speedup over autoregressive decoding increases from $3.07\times$ with DFlash to $3.93\times$ with our method, accompanied by an increase in average acceptance length from 5.78 to 8.84. The same pattern holds on Qwen3.5-4B: average speedup increases from $2.90\times$ to $3.82\times$, while average acceptance length rises from 5.56 to 8.62. These gains span mathematics, code generation, and conversation, showing that the additional accepted tokens translate into faster decoding across different workloads. EAGLE-3 results are included for reference, with speedups measured against SGLang's own autoregressive baseline (Zheng et al., 2024) rather than the Transformers baseline used by the other methods.

The two configurations of FoRE further show the benefit of draft refinement. MTP-guided early forking with forks alone ($K = 8,\ R = 0$) improves both metrics over DFlash on every benchmark, achieving average speedups of $3.50\times$ on Qwen3.5-9B and $3.33\times$ on Qwen3.5-4B. Adding refinement ($K = 8,\ R = 8$) increases average acceptance length from 6.87 to 8.84 and from 6.68 to 8.62, respectively. Although refinement requires a second drafter pass and expands verification from 8 to 16 sequences, it also raises the average speedup factor by approximately 12% on 9B and 15% on 4B. Thus, in these settings, the extra accepted tokens outweigh the added cost.

Table 1: Main results on Qwen3.5-9B and Qwen3.5-4B. Each cell: acceptance length τ (speedup vs. autoregressive decoding). $R \geq 1$ adds refined drafts (16 total verified rows instead of 8 at $R = 8$). MTP: the target's own MTP module drafting three tokens recursively. †Run in SGLang; speedup is relative to SGLang's own autoregressive baseline.

				FoRE	
Benchmark	MTP	EAGLE-3†	DFlash	K=8, R=0	K=8, R=8
Qwen3.5-9B					
GSM8K	3.50 (1.80×)	5.70 (2.81×)	6.54 (3.44×)	7.56 (3.91×)	**9.77 (4.44×)**
MATH-500	3.56 (1.81×)	5.97 (3.00×)	7.23 (3.77×)	8.31 (4.16×)	**10.64 (4.65×)**
AIME25	3.42 (1.74×)	5.31 (2.65×)	6.13 (3.13×)	7.30 (3.60×)	**9.53 (4.03×)**
HumanEval	3.53 (1.80×)	5.83 (2.82×)	7.03 (4.11×)	8.10 (4.38×)	**10.24 (4.86×)**
MBPP	3.26 (1.65×)	5.17 (2.57×)	5.31 (2.73×)	6.63 (3.35×)	**8.61 (3.88×)**
LiveCodeBench	3.18 (1.62×)	4.58 (2.29×)	4.79 (2.50×)	5.74 (2.89×)	**7.44 (3.20×)**
MT-Bench	2.85 (1.45×)	3.75 (1.86×)	3.46 (1.80×)	4.45 (2.22×)	**5.65 (2.46×)**
Mean	3.33 (1.70×)	5.19 (2.57×)	5.78 (3.07×)	6.87 (3.50×)	**8.84 (3.93×)**
Qwen3.5-4B					
GSM8K	3.50 (1.81×)	5.83 (2.50×)	6.38 (3.26×)	7.54 (3.69×)	**9.62 (4.28×)**
MATH-500	3.50 (1.79×)	5.81 (2.47×)	6.89 (3.52×)	7.97 (3.92×)	**10.30 (4.51×)**
AIME25	3.37 (1.70×)	5.32 (2.21×)	6.07 (3.04×)	7.20 (3.42×)	**9.24 (3.89×)**
HumanEval	3.41 (1.77×)	5.67 (2.36×)	6.75 (3.82×)	7.75 (4.15×)	**10.00 (4.64×)**
MBPP	3.20 (1.63×)	5.20 (2.25×)	5.10 (2.56×)	6.49 (3.18×)	**8.47 (3.76×)**
LiveCodeBench	3.11 (1.60×)	4.41 (1.77×)	4.33 (2.35×)	5.44 (2.80×)	**7.10 (3.23×)**
MT-Bench	2.78 (1.40×)	3.69 (1.52×)	3.43 (1.75×)	4.39 (2.13×)	**5.62 (2.39×)**
Mean	3.27 (1.67×)	5.13 (2.16×)	5.56 (2.90×)	6.68 (3.33×)	**8.62 (3.82×)**

Comparison with DFlash 2. We further evaluate FoRE on the recently released DFlash 2, which extends DFlash with local convolutions and a selector that scores adjacent token pairs to account for dependencies among draft tokens (Inco AI, 2026). We retain these components and fine-tune its Qwen3.8-27B (Qwen Team, 2026b) block diffusion drafter ($B = 8$) with visible prefixes of varying lengths. At inference, MTP-guided early forking generates four initial drafts, and draft refinement produces eight additional drafts; all twelve sequences are verified in a single target-model batch. Table 2 shows that FoRE increases average acceptance length from 5.69 to 6.93, achieving $1.22\times$ the DFlash 2 baseline. Average decoding speedup over autoregressive decoding also increases from $2.97\times$ to $3.12\times$, with improvements over DFlash 2 on six of the seven benchmarks.

Table 2: Qwen3.8-27B acceptance length τ and speedup over KV-cached autoregressive decoding. FoRE (DFlash 2) applies FoRE with K=4 MTP-guided drafts and R=8 refined drafts to a fine-tuned DFlash 2 drafter, keeping DFlash 2's pair selector.

Method		GSM8K	MATH-500	AIME25	HumanEval	MBPP	LCB	MT-Bench	Mean
DFlash 2	τ	6.17	6.30	5.94	6.75	6.13	4.59	3.96	5.69
	Speedup (×)	3.22	3.24	3.01	**3.62**	3.19	2.41	2.13	2.97
FoRE (DFlash 2)	τ	**7.36**	**7.45**	**7.22**	**7.55**	**7.22**	**6.25**	**5.46**	**6.93**
	Speedup (×)	**3.38**	**3.38**	**3.07**	3.46	**3.27**	**2.82**	**2.45**	**3.12**

4.3 Ablation study and analysis

Effect of visible-prefix training. Table 3 compares FoRE without draft refinement with single-draft DFlash and a matched $L = 0$ training control. The control and FoRE both use MTP-guided early forking with $K = 8$; only FoRE is trained with visible draft prefixes. FoRE improves acceptance length by 7.9–11.2% per benchmark over the control and achieves the highest value on every benchmark. This isolates the benefit of training on supplied prefixes.

Table 3: Average acceptance length τ on Qwen3.5-9B. All $K = 8$ configurations use MTP-guided early forking and no draft refinement; the control is trained without visible prefixes ($L = 0$).

Configuration	GSM8K	MATH-500	AIME25	HumanEval	MBPP	LCB	MT-Bench
DFlash ($K = 1$)	6.54	7.23	6.13	7.03	5.31	4.79	3.46
Control ($K = 8, R = 0$)	6.89	7.69	6.74	7.46	5.96	5.32	4.10
FoRE ($K = 8, R = 0$)	**7.56**	**8.31**	**7.30**	**8.10**	**6.63**	**5.74**	**4.45**

Table 4: Drafting budget on Qwen3.5-9B, averaged over seven benchmarks. Left: varying the number of MTP-guided drafts K without (R=0) and with (R=8) additional drafts. Right: varying the number of additional drafts R at K=8. $K+R$ rows are verified per step.

	$R = 0$		$R = 8$	
K	τ	Speedup	τ	Speedup
2	6.73	3.38×	8.69	3.90×
4	6.77	3.41×	8.87	**3.97**×
8	6.87	3.50×	8.84	3.93×
16	6.92	3.41×	8.91	3.18×

	$K = 8$	
R	τ	Speedup
0	6.87	3.50×
4	8.47	3.76×
8	8.84	3.93×
16	**9.25**	3.34×

Effect of the drafting budgets. Table 4 examines how the number of initial drafts K and the replacement budget R affect acceptance length and decoding speed. Without refinement, increasing K from 2 to 8 provides modest gains, while increasing it to 16 raises acceptance length only slightly and reduces speedup. Allocating part of the verification budget to refinement is more beneficial: with 16 verified sequences, $K = 8, R = 8$ achieves an average acceptance length of 8.84 and a $3.93\times$ speedup, compared with 6.92 and $3.41\times$ for $K = 16, R = 0$. However, refinement also exhibits diminishing returns. At $K = 8$, increasing R from 8 to 16 raises acceptance length from 8.84 to 9.25 but lowers speedup from $3.93\times$ to $3.34\times$. The highest measured speedup is $3.97\times$ at $K = 4, R = 8$, slightly above our default setting. These results show that both the total candidate budget and its allocation between initial drafting and refinement matter: accepting more tokens does not necessarily compensate for the additional drafting and verification cost.

5 Conclusion

We presented FoRE: MTP-guided early forking and suffix regeneration for block diffusion speculative decoding. MTP-guided early forking uses the target's pretrained MTP module to select multiple candidates at the first drafted position, where a mismatch would prevent all later draft tokens from being accepted. The block diffusion drafter generates the remaining tokens from each candidate, then refines the resulting drafts by regenerating tokens after selected replacements at later positions. Training with visible prefixes of varying lengths enables the same drafter to support both stages without an added correction head or a separately trained second drafter. Experiments on Qwen3.5-9B and Qwen3.5-4B show improvements in both average acceptance length and decoding speed over DFlash, while applying the method to DFlash 2 also improves acceptance length. These results support combining candidate selection with training the drafter to condition on those candidates.

Our evaluation focuses on greedy decoding with targets equipped with pretrained MTP modules. Future work includes evaluating other MTP designs (Gloeckle et al., 2024) and extending the method to sampling. Our implementation also repeats shared prefixes across verification rows, which tree verification for hybrid-attention targets (Wang et al., 2026a) could avoid. Serving-platform integration is left to future work.

AI use statement

In accordance with the ICLR 2027 policy on large language model (LLM) usage, we disclose that LLM-based tools (e.g., ChatGPT) were used in drafting and assisting in polishing during the writing process. In addition, they were used in writing, editing, and running code used to implement our methods, prepare training data, conduct experiments, and analyze results, and in interpreting the

results. They were also used for brainstorming, for feedback on research methodology, experiments, and hypotheses, and for identifying, summarizing, and verifying related work and references. The ideas explored in this work originated with the authors. The remaining tasks requiring disclosure, such as developing theoretical models, formulating mathematical claims, writing proofs, translation, and qualitative data analysis, are not applicable to this work. We reviewed all AI-assisted work. All scientific content, results, and conclusions are the responsibility of the authors.

Ethics statement

We train on publicly available datasets (Nemotron-Post-Training-Dataset-v2 and CodeAlpaca-20k) and openly released models (Qwen3.5 and Qwen3.8). FoRE is designed to speed up greedy decoding while preserving the target model's selected tokens through verification. It does not address risks such as bias or misuse associated with the target model, and faster generation may make both beneficial and harmful uses less costly.

Reproducibility statement

We provide FoRE's Qwen3.5 training and inference code as supplementary material, including the scripts that build the prompt pool, regenerate responses, and construct the training and held-out splits, together with the prompt ids of the held-out sets and the decontamination list. The package pins dependencies for its Qwen3.5 training and inference code and specifies the separate response-generation environment. Section 4 reports prompt counts, the evaluation seed, and hardware. Appendix A.1 gives the Qwen3.5-9B data construction and training configuration, and Appendix A.2 describes wall-clock timing aggregation. With fixed weights, prompts, decoding settings, and numerical environment, acceptance length is repeatable; for methods timed in our harness, wall-clock speedups are averaged over up to three clean repetitions. Responses for training are sampled at temperature 0.7, so regenerating them can yield different responses.

References

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2503.09573.

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

Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. MathArena: Evaluating LLMs on uncontaminated math competitions. In *Advances in Neural Information Processing Systems*, volume 38, pp. 22851–22888, 2025. doi: 10.52202/085713-0679. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/1d27c01ebd3e3aebe226b44fc970d803-Paper-Datasets_and_Benchmarks_Track.pdf.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235, 2024.

Sahil Chaudhary. Code Alpaca: An instruction-following LLaMA model for code generation. https://huggingface.co/datasets/sahil2801/CodeAlpaca-20k, 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 (ICML)*, 2026. URL https://arxiv.org/abs/2602.06036.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

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.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, pp. 35549–35562, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/98ed250b203d1ac6b24bbcf263e3d4a7-Abstract-Conference.html.

DeepSeek-AI. DeepSeek-V3 technical report, 2024. URL https://arxiv.org/abs/2412.19437.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex-Attention: A programming model for generating fused attention variants. In *Proceedings of Machine Learning and Systems*, volume 7, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/61a9278dfef5f871b5e472389f8d6fa1-Abstract-Conference.html.

Sneha Murthy Ghantasala. TreeWY: Speculative verification for Gated DeltaNet hybrids, 2026. URL https://arxiv.org/abs/2608.20961.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15706–15734, 2024.

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. URL https://openreview.net/forum?id=7Bywt2mQsCe.

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.

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. In *International Conference on Learning Representations*, pp. 58791–58831, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/94074dd5a072d28ff75a76dabed43767-Paper-Conference.pdf.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7421–7432, 2024b. doi: 10.18653/v1/2024.emnlp-main.422.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38, pp. 136737–136756, 2025. doi: 10.52202/085713-4562. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/c7b5a35ea98b62512a869c19ea7b03cb-Paper-Conference.pdf.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

Aofan Liu, Jingxiang Meng, Fangxin Liu, and Yongbiao Chen. CURE: Local uncertainty repair for block-parallel speculative decoding, 2026. URL https://arxiv.org/abs/2608.00531.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1608.03983.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101.

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)*, pp. 932–949, 2024. doi: 10.1145/3620666.3651335.

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

NVIDIA. NVIDIA Nemotron Nano 2: An accurate and efficient hybrid Mamba-Transformer reasoning model, 2025. URL https://arxiv.org/abs/2508.14444.

Yuma Oda, Ryan Mathieu, Roman Knyazhitskiy, and Artur Chakhvadze. Trees from marginals: Autoregressive drafting with factorized priors, 2026. URL https://arxiv.org/abs/2607.06763.

Soowon Oh, Nam Cao, Yujin Kim, Hojung Jung, Huzama Ahmad, Sangmin Bae, and Se-Young Yun. BASTION: Budget-aware speculative decoding with tree-structured block diffusion drafting. In *Advances in Neural Information Processing Systems*, 2026. URL https://arxiv.org/abs/2605.29727.

Xinwei Qiang, Xiang Fang, Chang Chen, Zaifeng Pan, Yue Guan, and Yufei Ding. Beyond parallel blindness: Information floors and model gaps in block drafting, 2026. URL https://arxiv.org/abs/2608.27339.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026a. URL https://qwen.ai/blog?id=qwen3.5.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL https://qwen.ai/blog?id=qwen3.8.

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. In *Conference on Language Modeling (COLM)*, 2026. URL https://openreview.net/forum?id=2ivmZcTz07.

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, 2026. URL https://arxiv.org/abs/2608.20530.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. OPT-Tree: Speculative decoding with adaptive draft tree structure. *Transactions of the Association for Computational Linguistics*, 13:188–199, 2025. doi: 10.1162/tacl_a_00735.

Li Wang, Yi Su, Xiabao Wu, Chiran You, Yongchao Liu, Zhan Qiu, Juelu Zhang, Jiajun Zheng, Fangxin Liu, Jie Zhang, Chen Tian, and Chengying Huan. Bole: Efficient tree speculation for hybrid-attention language models, 2026a. URL https://arxiv.org/abs/2608.01651.

Zheng Wang, Zhifan Ye, Qi Cheng, Yonggan Fu, Ziyan Wang, Feng Zhu, Haozhe Zhao, Jan Kautz, Pavlo Molchanov, Humphrey Shi, and Minjia Zhang. PRESTO: Prefix-aligned tree drafting for diffusion speculative decoding, 2026b. URL https://arxiv.org/abs/2607.22634.

Zhibin Wang, Xuying Han, Zhaohua Yang, Fuliang Liu, Xue Li, Rong Gu, Sheng Zhong, and Chen Tian. SpecLA: Efficient speculative decoding for linear-attention models, 2026c. URL https://arxiv.org/abs/2607.16673.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45. Association for Computational Linguistics, October 2020. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6/.

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

Yangchao Wu, Zongyue Qin, Alex Wong, and Stefano Soatto. STree: Speculative tree decoding for hybrid state space models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2505.14969.

Xiaomi LLM-Core Team. MiMo: Unlocking the reasoning potential of language model – from pretraining to posttraining, 2025. URL https://arxiv.org/abs/2505.07608.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=r8H7xhYPwz.

Liyuan Zhang, Jiarui Zhang, Jinwei Yao, Ran Yan, Yuchen Yang, Jiahao Zhang, Tongkai Yang, Yi Wu, and Binhang Yuan. D^2SD: Accelerating speculative decoding with dual diffusion draft models, 2026a. URL https://arxiv.org/abs/2606.04446.

Shuai Zhang, Huachuan Qiu, Hongliang He, and Yong Dai. Cost-aware diffusion draft trees for speculative decoding, 2026b. URL https://arxiv.org/abs/2606.01813.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *Proceedings of the VLDB Endowment*, 16(12): 3848–3860, 2023. doi: 10.14778/3611540.3611569.

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. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024. doi: 10.52202/079017-2000. URL https://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

A Experimental Details

A.1 Training Configurations and Matched Controls

Data. We take 100,000 prompts from Nemotron-Post-Training-Dataset-v2 (revision 5c89e01) chat, math, and code splits (Nathawani et al., 2025; NVIDIA, 2025) and `sahil2801/CodeAlpaca-20k` (Chaudhary, 2023) at revision 152bb5e. For Qwen3.5-9B, we regenerate responses with SGLang (Zheng et al., 2024) at temperature 0.7, top-p 0.95, thinking disabled, and at most 3,072 tokens. We reserve 2,000 prompts as a holdout with seed 7 and remove examples sharing at least eight distinct-word 8-grams with, or at least 30% of the word 8-grams in, a GSM8K, MATH-500, HumanEval, MBPP, or MT-Bench test item. This leaves 97,896 training examples and 1,997 held-out prompts. The training mix is 65.0% chat, 13.0% math, 12.2% Nemotron code, and 9.8% CodeAlpaca.

Optimization and compute. We initialize the drafter from `z-lab/Qwen3.5-9B-DFlash` at revision `492f4b5` (Chen et al., 2026). We train for one epoch with AdamW (Loshchilov & Hutter, 2019) at learning rate 10^{-4}, with zero weight decay, gradient clipping at 1.0, and cosine decay (Loshchilov & Hutter, 2017) after a linear warmup over the first 4% of steps. We use D-PACE with $\alpha = 0.5$, which weights block positions by their contribution to a surrogate of acceptance length (Wu et al., 2026). The effective batch is four sequences (one per GPU, without gradient accumulation), with a maximum sequence length of 3,072 and at most 512 anchor blocks per sequence. The optimizer, gradient clipping, warmup and cosine schedule, maximum sequence length, and anchor count follow DFlash (Chen et al., 2026). The 9B run uses seed 42, bf16 computation, fp32 master weights, FSDP `SHARD_GRAD_OP` (Zhao et al., 2023), and `flex_attention` (Dong et al., 2025) on four H200 GPUs in one NVLink group.

Other targets. We use the same prompt pool for Qwen3.5-4B and Qwen3.8-27B, regenerating responses with each target under the sampling settings above. For each target, a seed-7 holdout of 2,000 prompts leaves 98,000 training examples; the benchmark-overlap exclusions described above apply to the 9B data. The 4B fine-tune starts from `z-lab/Qwen3.5-4B-DFlash` at revision `96899cc` and otherwise uses the 9B training settings on four A6000 GPUs. The 27B fine-tune starts from the released `z-lab/Qwen3.8-27B-DFlash2` checkpoint at revision `50307d4c4c` and retains its convolution and selector components. It uses $B = 8$, a selector loss coefficient of 1.0, and `sdpa` on four H200 GPUs; the remaining optimization settings match the 9B fine-tune.

Matched control. The main prefix-length distribution and loss masking are defined in Section 3.4. Let $p = \Pr(L > 0)$. The $p = 0$ control (the $L = 0$ control of Section 4.3) always uses an empty within-block prefix and otherwise retains the same initialization, data, loss objective, optimization schedule, training length, and seed as FoRE.

A.2 Evaluation and Timing Protocol

For methods measured in our Transformers harness, each wall-clock measurement runs KV-cached autoregressive decoding and the speculative configuration on the same prompts. We measure decode time with CUDA-synchronized wall clocks using the prefill exclusions stated in Section 4. In FoRE's timed implementation, we cache the drafter's K-row context and the MTP module's KV across steps to avoid rebuilding context for its candidate forks. For each response, we divide decode time by its output-token count. We average this time per output token across responses separately for autoregressive and speculative decoding, then divide the two means to obtain speedup. We compute this ratio in each timing repetition and report its mean across repetitions. Because prompts, weights,

Table 5: Acceptance length τ for FoRE trained with the original DFlash loss or D-PACE. Each pair uses the same inference configuration and the Table 1 evaluation protocol. Values are rounded; relative means in the text use unrounded results.

	$K = 8, R = 0$		$K = 8, R = 8$	
Benchmark	DFlash loss	D-PACE	DFlash loss	D-PACE
GSM8K	7.33	7.56	9.52	9.77
MATH-500	8.07	8.31	10.37	10.64
AIME25	6.92	7.30	9.29	9.53
HumanEval	8.00	8.10	10.23	10.24
MBPP	6.37	6.63	8.29	8.61
LiveCodeBench	5.49	5.74	7.35	7.44
MT-Bench	4.28	4.45	5.45	5.65

and greedy decoding are fixed, the repetitions measure timing variability rather than additional samples of τ. EAGLE-3 speedups are measured separately in SGLang relative to its own autoregressive baseline.

A.3 Verification on Hybrid Targets

An ancestor-only attention mask (Miao et al., 2024) lets each draft token read its own prefix in a packed tree (Ringel & Romano, 2026), but it does not route the branch-specific recurrent state of Qwen3.5's Gated DeltaNet layers (Qwen Team, 2026a; Yang et al., 2025). STree accumulates diagonal state-space transitions along tree paths (Wu et al., 2025); Weaver (Oda et al., 2026), SpecLA (Wang et al., 2026c), Bole (Wang et al., 2026a), and TreeWY (Ghantasala, 2026) develop tree verification for Gated DeltaNet or related linear-attention recurrences. We instead verify each complete candidate sequence as a separate row in one target batch. Each row starts from the committed prefix state and advances through its own proposed tokens; after selecting the longest matching prefix, we reconstruct the target state for that accepted prefix.

B Additional Training Ablations

B.1 Loss-objective control

We compare two one-epoch variable-prefix fine-tunes initialized from the same released drafter, with matched data, learning rate, seed, and visible-prefix training. They differ only in the training objective: D-PACE with $\alpha = 0.5$ or the original DFlash loss with decay parameter $\gamma = 7$ (Wu et al., 2026; Chen et al., 2026). We evaluate both under the same inference configurations and the Table 1 protocol; Table 5 reports the results.

D-PACE yields a higher acceptance length on every benchmark in both configurations. Averaging the per-benchmark relative changes, τ increases by 3.65% at $K = 8, R = 0$ and by 2.38% at $K = 8, R = 8$. This loss-only comparison shows an additional accepted-length gain under fixed conditioning; it does not establish that D-PACE is required for conditioning to help, and it does not measure a wall-clock speedup difference.

B.2 Fine-Tuning Control

We compare our training with a matched $p = 0$ fine-tune that never receives a visible within-block prefix during training. Each pair shares the DFlash initialization, data, loss, schedule, training duration, and seed, and is evaluated under the same inference configuration. On Qwen3.5-9B, our training raises mean speedup from $3.24\times$ to $3.50\times$ at $K = 8, R = 0$ (8.1%) and from $3.42\times$ to $3.93\times$ at $K = 8, R = 8$ (15.0%). On Qwen3.5-4B, mean speedup at $K = 8, R = 8$ rises from $3.32\times$ to $3.82\times$ (15.1%). Mean acceptance length also rises in all three comparisons. Because visible-prefix training is the only changed part of the training recipe within each pair, these gains cannot be explained by an additional fine-tuning epoch or a wider inference procedure alone.

C EARLY POSITION LEVERAGE

We measure how sensitive τ is to the conditional match probability at each draft position, using the released Qwen3.5-9B DFlash drafter with $B = 16$ over 148,816 cycles from 1,997 held-out prompts.

Because the target supplies t_1, drafted token j occupies slot t_{j+1}. Let S_k be the probability that the first k drafted tokens are accepted, with $S_0 = 1$, and let $\alpha_j = S_j/S_{j-1}$ be the conditional match probability of drafted token j given that all earlier drafted tokens match. Then

$$\tau = 1 + \sum_{k=1}^{B-1} S_k, \qquad D_j \equiv \frac{\partial \tau}{\partial \alpha_j} = \frac{\sum_{k=j}^{B-1} S_k}{\alpha_j} = S_{j-1}\left(1 + \sum_{k=j+1}^{B-1} \prod_{i=j+1}^{k} \alpha_i\right).$$

Here S_{j-1} is the reach of slot t_{j+1}, the chance that the chain gets that far, and the parenthesized follow-on term is the expected number of tokens accepted from that slot onward once it matches. This is the decomposition D-PACE uses for its training weights, with measured conditional match probabilities in place of draft confidences (Wu et al., 2026). Reach falls from 100% at t_2 to 5.3% at t_{11}, while the follow-on term stays between 3.2 and 4.3 tokens over the same slots. Table 6 reports $0.01D_j$, the first-order τ gain from one percentage point of conditional match probability, and each slot's share $D_j / \sum_i D_i$.

Table 6: Position-wise acceptance and marginal sensitivity of the released DFlash drafter. Reach is the fraction of valid cycles whose earlier draft positions match.

Slot	α_j	Reach (%)	$0.01D_j$	Share (%)
t_2	0.734	100.0	0.0322	26.8
t_3	0.651	73.4	0.0250	20.9
t_4	0.643	47.8	0.0179	14.9
t_5	0.679	30.7	0.0125	10.4
t_6	0.727	20.8	0.0088	7.3
t_7	0.755	15.2	0.0064	5.4
t_8	0.760	11.4	0.0049	4.1
t_9	0.775	8.7	0.0037	3.1
t_{10}	0.793	6.7	0.0027	2.3
t_{11}	0.797	5.3	0.0020	1.7
t_{12}	0.817	4.3	0.0015	1.2
t_{13}	0.818	3.5	0.0010	0.9
t_{14}	0.822	2.8	0.0007	0.6
t_{15}	0.804	2.3	0.0004	0.3
t_{16}	0.792	1.9	0.0002	0.2

The t_2 share is 26.8%, nearly equal to the 26.9% for t_6–t_{16} combined (computed before rounding). Per percentage point of conditional match probability, t_2 has 3.68× the effect of t_6 and 11.8× that of t_{10}. These are single-chain sensitivities, not the measured gain of our method; Tables 1 and 3 report the decoding results.

D CONTRIBUTION OF SUFFIX REGENERATION

We test whether regenerating the suffix adds accepted tokens beyond the selected replacement. We replay Qwen3.5-9B steps recorded on the fixed $K = 8$ trajectory using the Table 1 prompt sets and response cap. We hold the original full forks and selected replacements fixed. We compare those forks alone with three proposal sets that add rows for the same selected replacements: short rows ending at the replacement token (leaf-only), rows retaining the source fork's original suffix, and rows whose suffix is masked and regenerated in a second drafter pass. The added rows are scored against each recorded target continuation but never committed into later steps. All acceptance lengths include the token supplied by the target at the first mismatch or after full acceptance.

Table 7 shows that leaf-only rows already raise τ over the initial forks by 0.43–0.53 tokens per step (ranges here are computed before rounding), and keeping the original suffix raises τ over leaf-

Table 7: Acceptance length on matched steps. Every column retains the same full $K = 8$ initial branches. The three columns on the right add rows for the same selected later replacements: a short leaf ending at the replacement, a row with the source fork's original suffix, or a row with its suffix regenerated from the modified prefix.

		Same selected later replacements added		
Set	$K = 8$ branches only	Leaf rows	Original suffix	Regenerated suffix
GSM8K	7.59	8.11	8.67	9.67
MATH-500	8.31	8.83	9.32	10.38
AIME25	7.36	7.88	8.32	9.28
HumanEval	8.08	8.54	9.06	9.75
MBPP	6.63	7.15	7.72	8.57
LiveCodeBench	5.74	6.26	6.67	7.46
MT-Bench	4.44	4.87	5.16	5.61

only rows by 0.29–0.58 tokens per step. Thus, a selected replacement by itself can be a useful point fix where that fork would otherwise be rejected, while the original suffix can still contribute accepted tokens. Regenerating the suffix then adds a further 0.44–1.07 tokens per step over keeping the original suffix. The original-suffix and regenerated-suffix rows use the same replacements and have equal lengths, so this last gain comes from regenerating the suffix rather than a longer proposal. After a selected replacement matches the target and leaves another draft token to check, regenerating the suffix raises the next-token match rate over retaining the original suffix by 31.5–43.7 percentage points across benchmarks.

E Training Seeds

Table 8: Seed robustness of the Qwen3.5-9B fine-tune: means over seven benchmarks. Seed-43 speedups are from one run per benchmark; Seed-42 speedups average up to three.

	τ		Speedup	
Configuration	Seed 42	Seed 43	Seed 42	Seed 43
$K = 8, R = 0$	6.87	6.89	3.50×	3.46×
$K = 8, R = 8$	8.84	8.92	3.93×	3.93×

We repeat the Qwen3.5-9B fine-tune with training seed 43, holding the other training settings fixed. As Table 8 shows, mean τ across seven benchmarks changes from 6.87 to 6.89 at $K = 8, R = 0$, and from 8.84 to 8.92 with $R = 8$ rows. The corresponding speedups are also similar. The accepted-length and speedup gains are therefore consistent across the two training seeds.