ONE CALL AHEAD: PACKED SELF-SPECULATIVE DECODING WITH DIFFUSION DRAFTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive (AR) language models generate tokens sequentially, so generating long outputs is costly. Speculative decoding mitigates this by using a lightweight drafter to propose multiple tokens for the target AR model to verify in parallel, while a diffusion-based drafter can produce an entire draft block in one forward pass. In models with AR and diffusion modes, one network both drafts and verifies, but drafting still waits for verification, so the standard schedule needs two sequential model calls per iteration. Existing one-call schedules avoid the wait but incur input size quadratic in the speculative width. Motivated by this efficiency bottleneck, we introduce Packed-SS, a self-speculative decoding schedule that verifies and drafts in a single model call with input size linear in the speculative width, while preserving the target model's distribution. The key idea is to let the diffusion drafter run *one call ahead*: while verifying the current draft, the model simultaneously constructs the next draft from an older verified prefix and keeps its unresolved suffix for the next iteration. Under a mild monotonicity assumption on the drafter, we further show that Packed-SS requires no more model calls than the standard two-call decoding in the greedy setting. Experiments on Nemotron 3B/8B and SetDLM 1.7B show higher throughput and lower mean wall time than both baselines under greedy decoding. Code is available here.

1 INTRODUCTION

Modern language models (LMs) increasingly generate long sequences of intermediate reasoning before producing a final answer, making inference latency a central bottleneck. Standard autoregressive (AR) models (Brown et al., 2020) generate one token at a time from left to right. Producing an L-length sequence therefore requires at least L sequential model calls, where the computation for each new token depends on all previously generated tokens.

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Xia et al., 2023) accelerates AR inference by using a lightweight *draft model* that proposes multiple tokens for the full-size *target model* to verify *in parallel*. A rejection-sampling correction step ensures that the resulting sequence follows exactly the target model's distribution. The target model can thus output several tokens from a single verification call. However, when the drafter is AR, creating a draft block remains sequential. Therefore, speculative decoding shortens the expensive serial chain, but does not eliminate it.

Masked diffusion language models (MDLMs) provide a natural way to parallelize drafting (Sahoo et al., 2024; Shi et al., 2024; Gong et al., 2025; Nie et al., 2025; Ye et al., 2025). By learning to reverse a corruption process that transforms data into noise, an MDLM can predict multiple tokens simultaneously. Therefore, when used as a speculative drafter, it can draft an entire block in one forward pass (Christopher et al., 2024; Li et al., 2025a; Chen et al., 2026a; Zhang et al., 2026). Some LMs support both causal AR prediction and parallel masked diffusion prediction in the same network (Liu et al., 2025; Fu et al., 2026). They provide a natural form of self-speculative decoding, with the drafter and verifier sharing the same model weights.

However, shared weights do not eliminate the sequential dependency between drafting and verification. The standard self-speculative schedule, referred to as *Linear-SS*, uses two sequential calls: one AR call verifies the current draft and one diffusion call constructs the next draft from the newly verified prefix. Drafting cannot begin until verification finishes to determine where the verified prefix ends.

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.

Quadratic-SS (Liu et al., 2025) eliminates this dependency by preparing a continuation for every possible verification outcome. With speculative width W, it pre-drafts separately for the cases in which $0, 1, \ldots, W$ draft tokens are accepted. Whichever outcome occurs, the next draft is already available. This reduces each iteration to one call, but the number of evaluated positions grows quadratically with W. Thus, Linear-SS evaluates $O(W)$ positions in two calls, whereas Quadratic-SS evaluates $O(W^2)$ in one.

At small batch sizes, a substantial fraction of a call's latency can come from loading model weights from memory, making it attractive to evaluate more positions per call (Cai et al., 2024; Fu et al., 2024). To illustrate this tradeoff, suppose a call evaluating n positions costs $\tau + \delta n$, where τ is a fixed overhead, mostly weight loading, and δ is the marginal cost per position. An iteration costs approximately $2\tau+2\delta W$ for Linear-SS and $\tau+\delta W^2$ for Quadratic-SS. Verifying and drafting in one call with approximately $2W$ evaluated positions would cost only $\tau + 2\delta W$ per iteration, avoiding both the second-call overhead and the quadratic branching cost.

This raises the central question of this work:

Can we verify and draft in one call, while only evaluating positions linear in the speculative width?

Contributions. We answer affirmatively with *Packed-SS*, a plug-and-play self-speculative decoding schedule for such models. The idea is simple: the next draft does not need to wait for the newest verified prefix. Instead, we let the diffusion drafter run *one call ahead*. At iteration t, one forward pass simultaneously (i) verifies the draft, and (ii) proposes a new draft block using the prefix available at the beginning of iteration t. If verification advances by A positions, the first A positions of this new draft have already been resolved and are discarded. The remaining suffix becomes the draft for the next iteration. Drafting therefore no longer waits for verification. Figure 1 illustrates the difference between Packed-SS and existing schedules.

The two key observations behind this design are as follows. First, a diffusion drafter predicts tokens independently given its conditioning prefix, so a rejection at one position does not invalidate later draft tokens. Second, draft freshness affects speed rather than correctness: standard acceptance and residual correction remain exact when they use the proposal distribution that generated each draft token. Thus, an older draft may be accepted less often, but does not bias the output.

These observations remove the synchronization between drafting and verification. With an appropriate attention mask, they can be packed into one forward pass. Each iteration thus requires only one model call, while the number of evaluated positions remains linear in W.

Our contributions are summarized as follows:

1. We introduce *Packed-SS*, which simultaneously verifies the previous draft and constructs the next draft in a single model call, with input size linear in the speculative width, while preserving the target model's distribution.
2. We theoretically prove that, assuming that correct draft predictions remain correct as verified context grows, Packed-SS requires no more model calls than Linear-SS to generate any prescribed output length.
3. We evaluate Packed-SS on Nemotron 3B/8B (Fu et al., 2026) and SetDLM 1.7B (Arriola & Kuleshov, 2026). Across the primary greedy evaluations, Packed-SS achieves higher throughput than Linear-SS and Quadratic-SS. Under temperature-1 sampling, it consistently outperforms Linear-SS while remaining competitive with Quadratic-SS.

2 Background

2.1 Notation

We write $\mathcal{V}$ for the vocabulary and θ for model parameters. For a sequence x we write $x^{(<i)} = (x^{(1)}, \ldots, x^{(i-1)})$, and (x, y) denotes the sequence x followed by item y. We use $[u]_+ = \max\{u, 0\}$ to denote the positive part of u. A complete list of variables can be found in Appendix A.

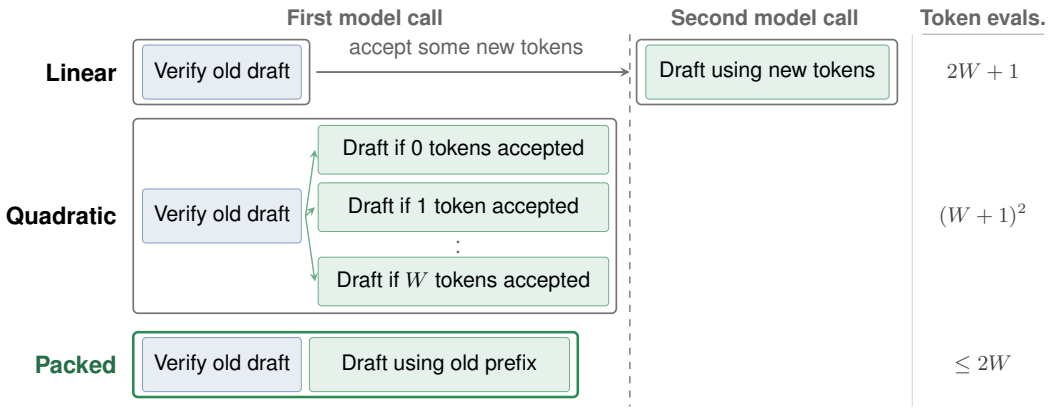


Figure 1: **Where drafting happens in one call.** Outlined boxes are model calls. Linear-SS must finish verifying before it can draft, so an iteration costs two calls. Quadratic-SS fits in one call by drafting a continuation for every place verification might stop, so its input grows quadratically with the speculative width W. Packed-SS also uses one call, building a single draft from the previously verified prefix alone. The right column gives total draft and verifier prediction counts per iteration, including predictions that may later be discarded.

2.2 PRELIMINARIES

Speculative decoding. Given prefix P, the drafter proposes W draft tokens $D = (d^{(1)}, \ldots, d^{(W)})$, drawing each $d^{(i)}$ from a drafter distribution $q^{(i)}$, where i counts positions after P. In one model call, the target model p_θ uses P and D to return target distributions $p^{(i)} = p_\theta(\cdot \mid P, d^{(<i)})$ for every $i \leq W + 1$. The draft tokens are then verified from left to right, where each draft token $d^{(i)}$ is accepted with probability $\min\{1, p^{(i)}(d^{(i)})/q^{(i)}(d^{(i)})\}$. At the first rejection, the decoder replaces that draft token with a correction token from the residual distribution $\propto [p^{(i)} - q^{(i)}]_+$. If all tokens in a draft are accepted, the target model yields one extra bonus token drawn from $p^{(W+1)} = p_\theta(\cdot \mid P, D)$. The rejection-sampling correction ensures that the generated text follows the target distribution exactly, so a low-quality drafter costs speed but never quality.

In practice, we use the target model to sample at a certain temperature, with temperature 1 being the model's original distribution and lower temperature yielding a sharper distribution. At temperature 0, the output distribution mass is on its $\arg\max$ token. This is also called *greedy decoding*.

Masked diffusion language models. A masked diffusion language model (MDLM) generates text by repeatedly filling missing token positions, represented by a special mask token MASK (Sahoo et al., 2024; Shi et al., 2024). Following the denoising framework of discrete diffusion (Hoogeboom et al., 2021; Austin et al., 2021a; Campbell et al., 2022; Lou et al., 2024), the model is trained to predict original tokens from sequences in which randomly selected tokens have been replaced by MASK. During inference, let C denote the visible tokens and their positions, and U the remaining masked positions. One forward pass predicts a distribution $q_\theta^{(i)}(\cdot \mid C)$ for each position $i \in U$. Sampling independently from these predictions gives the parallel proposal distribution $q_\theta(z_U \mid C) = \prod_{i \in U} q_\theta^{(i)}(z_i \mid C)$. The decoder fills selected positions, often those with the smallest entropy of predicted distributions, and predicts the remaining positions again using the updated context. Block diffusion applies this procedure within blocks generated from left to right, completing each block before starting the next (Arriola et al., 2025). A block size of one recovers autoregressive generation, while a single block covering the full sequence recovers full-sequence masked diffusion.

Shared-weight drafting and verification. A model with shared-weight AR and diffusion modes can perform both roles in speculative decoding: its masked mode serves as the *drafter* q_θ, proposing multiple tokens after the verified prefix in parallel, while its causal mode serves as the *verifier* p_θ. When these modes differ only in their attention masks, Packed-SS can combine them in one

forward pass using a custom attention mask (Miao et al., 2024; Spector & Ré, 2023; Fu et al., 2024). Nemotron-Labs-Diffusion supports both modes through joint training on autoregressive and diffusion objectives (Fu et al., 2026). SetDLM also supports this construction through its any-order formulation: single-position conditionals in left-to-right order provide AR verification, while parallel predictions of unresolved positions provide diffusion drafting (Arriola & Kuleshov, 2026). See Appendix D for model-specific input layouts and attention-mask constructions.

2.3 Related work

Sequential draft–verify decoding. Most speculative decoding methods improve drafting within a sequential draft–verify schedule (Leviathan et al., 2023; Chen et al., 2023). Self-speculative methods reuse the target network: Draft & Verify (Zhang et al., 2024) skips selected layers without retraining, whereas LayerSkip (Elhoushi et al., 2024) trains intermediate layers for early-exit drafting. EAGLE (Li et al., 2024a; 2025b) variants instead train auxiliary drafters using target features. Parallel drafting reduces sequential work, from blockwise parallel decoding (Stern et al., 2018) and SpecDec (Xia et al., 2023) to diffusion-based drafters such as SpecDiff (Christopher et al., 2024) and DiffuSpec (Li et al., 2025a). DEER (Cheng et al., 2025) aligns prefix-conditioned drafts with an AR teacher, while DFlash (Chen et al., 2026a) uses a shallow block-diffusion drafter conditioned on target features. DSpark (Cheng et al., 2026) adds confidence-based verification-length scheduling, while DFlow (Zhang et al., 2026) reuses verifier features from rejected suffixes to improve subsequent drafts.

Other speculative decoding schedules. SpecInfer (Miao et al., 2024) and Medusa (Cai et al., 2024) use tree attention to verify multiple candidate continuations, while EAGLE-2 (Li et al., 2024b) adaptively constructs draft trees using confidence estimates. Tree-structured verification covers more candidate continuations, but does not by itself remove the dependency between drafting and verification. Jacobi decoding (Santilli et al., 2023) takes a different approach, updating multiple tentative positions in parallel through fixed-point iteration. Lookahead Decoding (Fu et al., 2024) turns these trajectories into cached n-gram candidates and combines a lookahead branch with a verification branch in one forward pass. For diffusion decoding, SSD (Gao et al., 2025) and FreeDave (Wu & Zhang, 2025) batch multiple speculative decoding states, with FreeDave also reusing accepted-state predictions for subsequent drafting. Within shared-weight models, the quadratic schedules in Samragh et al. (2025), TiDAR (Liu et al., 2025), and Nemotron (Fu et al., 2026) prepare continuations at every possible acceptance boundary, combining drafting and verification at the cost of $O(W^2)$ evaluated positions for speculative width W. Apple's linear decoding (Samragh et al., 2025) and Introspective DLMs (Yu et al., 2026) discard continuations conditioned on tentative tokens after rejection, which frequently fails to simultaneously yield a draft. Speculative Jacobi Decoding (Teng et al., 2024) reuses predictions after rejection, but its proposals also condition on tentative tokens.

3 Our Method

In this section, we first present the high-level description of Packed-SS. We then describe the procedure under greedy decoding in Section 3.1, where the main idea is clearest. Section 3.2 describes the general procedure.

At a high level, iteration t of Packed-SS reads the verified prefix P_{t-1} and the current draft D_{t-1}, and performs verification and drafting simultaneously and independently in one model call. The verifier accepts A_t tokens from the standard speculative decoding process (including one correction or bonus token). These A_t tokens are extended to P_{t-1} to form the next prefix P_t. For speculative width W, the drafter reads only P_{t-1} and generates a new proposal $Z_t = (z_t^{(i)})_{i=1}^{W}$ for the next W positions. The first A_t of those positions have just been filled by the verifier, so their proposal tokens are discarded, and the rest become the draft D_t that iteration $t+1$ verifies. See Figure 2 for an illustration of one iteration under greedy decoding.

3.1 Packed self-speculative procedure under greedy decoding

We now state the procedure under greedy decoding, and summarize it in Algorithm 1.

Initialization. Given a prompt P, start with the prefix $P_0 = P$ and draft $D_0 = \varnothing$.

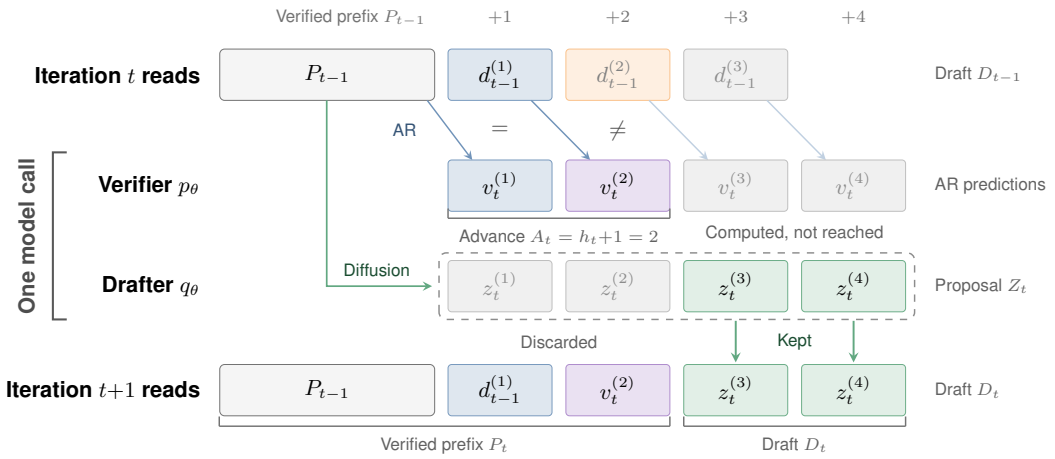


Figure 2: **One iteration of greedy Packed-SS.** Columns are token positions. The old draft token $d_{t-1}^{(i)}$, the verifier prediction $v_t^{(i)}$, and the new proposal token $z_t^{(i)}$ are in the same position. A single forward pass takes the verified prefix P_{t-1} and draft D_{t-1} as input and produces the two middle rows: the AR verifier's next-token predictions $v_t^{(i)}$, conditioned on P_{t-1} and $d_{t-1}^{(1)}, \ldots, d_{t-1}^{(i-1)}$, and the diffusion drafter's proposal $Z_t = (z_t^{(i)})_{i=1}^{W}$, conditioned on P_{t-1} alone. Here, one matching draft token ($h_t = 1$) and the correction token $v_t^{(2)}$ yield $A_t = 2$ output tokens. Proposals at these two positions are discarded, leaving $D_t = (z_t^{(3)}, z_t^{(4)})$ as the next draft.

Iteration $t \geq 1$. Given the prefix P_{t-1} and draft $D_{t-1} = (d_{t-1}^{(1)}, d_{t-1}^{(2)}, \ldots)$ following P_{t-1}, iteration t evaluates the verifier and drafter in one forward pass.

For verification, the AR mode predicts the token at each position of the previous draft plus one position after it, conditioned on the prefix P_{t-1} and the draft tokens before that position:

$$v_t^{(i)} = \arg\max_{x \in \mathcal{V}} p_\theta(x \mid P_{t-1}, d_{t-1}^{(<i)}), \qquad 1 \leq i \leq |D_{t-1}| + 1. \tag{1}$$

Let h_t be the number of leading draft tokens that match the verifier's predictions:

$$h_t = \max\{0 \leq h \leq |D_{t-1}| : d_{t-1}^{(i)} = v_t^{(i)} \text{ for every } i \leq h\}. \tag{2}$$

Verification yields the accepted draft prefix followed by $v_t^{(h_t+1)}$, which corrects the first mismatch or supplies a bonus after full acceptance. Thus, a total of $A_t = h_t + 1$ tokens are accepted.

In the same call, the diffusion mode drafts the W positions that follow the previous prefix P_{t-1}:

$$Z_t = (z_t^{(1)}, \ldots, z_t^{(W)}), \qquad z_t^{(i)} = \arg\max_{x \in \mathcal{V}} q_\theta^{(i)}(x \mid P_{t-1}). \tag{3}$$

It is important to note that the draft tokens Z_t are generated conditioned on the previous prefix P_{t-1}, rather than in Linear-SS the new prefix P_t. This distinction allows drafting and verification to run in parallel while Linear-SS needs to be evaluated sequentially. On nonterminal iterations, the suffix of Z_t after its first A_t tokens becomes the new draft:

$$D_t = (z_t^{(A_t+1)}, \ldots, z_t^{(W)}). \tag{4}$$

Termination. Stop at the first accepted end-of-sequence (EOS) token or after L new tokens.

Comparison with Linear-SS and Quadratic-SS. Linear-SS alternates between drafting-only and verification-only calls, whereas Packed-SS performs both in each call. By retaining the unresolved suffix of Z_t as the next draft, Packed-SS avoids a separate drafting call. This overlap comes at the cost of a shorter draft conditioned on less verified context: on nonterminal iterations of Algorithm 1,

Algorithm 1 Packed-SS with LTR greedy decoding

Require: prompt P, speculative width W, output length limit L, verifier p_θ, drafter q_θ

1: $P_0 \leftarrow P,\ D_0 \leftarrow \varnothing,\ t \leftarrow 1$
2: **while** $|P_{t-1}| - |P| < L$ and ($t = 1$ or the last token of P_{t-1} is not EOS) **do**
  In one model call:
3: $v_t^{(i)} \leftarrow \arg\max_x p_\theta(x \mid P_{t-1}, d_{t-1}^{(<i)})$ for $i = 1, \ldots, |D_{t-1}| + 1$ ▷ Verifier predictions
4: $z_t^{(i)} \leftarrow \arg\max_x q_\theta^{(i)}(x \mid P_{t-1})$ for $i = 1, \ldots, W$ ▷ Drafter proposals
  end
5: $h_t \leftarrow \max\{0 \le h \le |D_{t-1}| : d_{t-1}^{(i)} = v_t^{(i)} \text{ for all } i \le h\}$ ▷ Count matches
6: $C_t \leftarrow (d_{t-1}^{(1)}, \ldots, d_{t-1}^{(h_t)}, v_t^{(h_t+1)}),\ A_t \leftarrow h_t + 1$ ▷ Add verifier token
7: $a \leftarrow \min\big(\{A_t, L + |P| - |P_{t-1}|\} \cup \{1 \le i \le A_t : C_t^{(i)} = \text{EOS}\}\big)$ ▷ Ending condition
8: $C_t \leftarrow (C_t^{(1)}, \ldots, C_t^{(a)})$ ▷ Prune by stopping index
9: $P_t \leftarrow (P_{t-1}, C_t)$ ▷ Prepare next prefix, append accepted tokens
10: $D_t \leftarrow (z_t^{(a+1)}, \ldots, z_t^{(W)})$ ▷ Prepare next draft, $z_t^{(i)}$ is generated in line 4
11: $t \leftarrow t + 1$
12: **end while**
13: **Output:** P_{t-1}

Packed-SS retains $|D_t| = W - A_t \le W - 1$ tokens by equation 4, whereas Linear-SS drafts W tokens conditioned on A_t more verified tokens. In addition, both Packed-SS and Quadratic-SS draft and verify simultaneously in one model call. However, Packed-SS evaluates $O(W)$ positions per iteration rather than $O(W^2)$. The trade-off is conditioning context: Quadratic-SS's selected proposal excludes only the final verifier token from P_t, whereas Packed-SS conditions on P_{t-1}, excluding all A_t newly appended tokens. Conceptually, Packed-SS can be viewed as Quadratic-SS with only the branch for zero accepted draft tokens.

Theoretical guarantees. Under greedy decoding and the monotonicity assumption below, Packed-SS requires no more forward passes than Linear-SS to generate a prescribed number of tokens from the same prompt.

Fix a prompt and speculative width W, and let $x_1, x_2, \ldots$ denote the target model's greedy continuation. Let $\widehat{x}_k(c)$ denote the deterministic drafter's prediction at position k, conditioned on the prompt and $x_1, \ldots, x_c$.

Assumption 3.1 (Monotonicity of draft correctness)**.** For each output position k, there exists a threshold $c_k^\star \le k$ such that $\mathbb{1}\{\widehat{x}_k(c) = x_k\} = \mathbb{1}\{c \ge c_k^\star\}$ for every $c < k$ evaluated by either schedule.

Here, $c_k^\star = k$ indicates that position k is never correctly predicted before it is verified. Thus, additional verified context cannot turn a correct prediction into an incorrect one. The thresholds need not increase with k: a later position may become correctly predictable with less context than an earlier one.

Theorem 3.2 (Pathwise dominance)**.** *Suppose Packed-SS and Linear-SS use the same target model, drafter, and speculative width $W \ge 1$, and start from a common verified prefix. Under Assumption 3.1, Packed-SS's verified prefix after $2t$ forward passes is at least as long as Linear-SS's after t iterations, for every $t \ge 0$.*

Let $\mathrm{NFE}_S(L)$ denote the number of forward passes schedule S requires to generate L tokens after P. Since each Linear-SS iteration uses two forward passes, for every $L \ge 0$, one has

$$\mathrm{NFE}_{\textit{Packed-SS}}(L) \le \mathrm{NFE}_{\textit{Linear-SS}}(L). \tag{5}$$

Since both schedules follow the same greedy continuation, the bound also holds when stopping at the first emitted EOS token. See Appendix B.1 for the proof.

3.2 General packed self-speculative sampling

We now present Packed-SS for general sampling. Compared with the greedy procedure in Algorithm 1 from Section 3.1, the general sampling requires two minor modifications.

Consider iteration t. First, we store beyond each draft token $d_t^{(i)}$ the distribution from which it was sampled, $Q_t^{(i)} = q_t^{(a+i)} = q_\theta^{(a+i)}(\cdot \mid P_{t-1})$. At temperature 0, $Q_{t-1}^{(i)}$ collapses to a point mass on $d_{t-1}^{(i)}$. So storing the draft token alone is sufficient as in Algorithm 1.

Second, when verifying each draft token $d_{t-1}^{(i)}$, we replace the deterministic match test $d_{t-1}^{(i)} = v_t^{(i)}$ in Equation (2) with the standard rejection-sampling acceptance rule. With $p_t^{(i)} = p_\theta(\cdot \mid P_{t-1}, d_{t-1}^{(<i)})$ the target conditional at that position and $Q_{t-1}^{(i)}$ its stored drafter distribution, we accept the draft token $d_{t-1}^{(i)}$ with probability $\min\{1, p_t^{(i)}(d_{t-1}^{(i)})/Q_{t-1}^{(i)}(d_{t-1}^{(i)})\}$. Crucially, the denominator is the stored drafter distribution $Q_{t-1}^{(i)}$ from which $d_{t-1}^{(i)}$ was originally sampled, even though it may have been computed from an older prefix. At the first rejection, we draw a correction from the normalized residual distribution $\propto [p_t^{(i)} - Q_{t-1}^{(i)}]_+$. If the entire draft is accepted, we draw a bonus token from $p_t^{(|D_{t-1}|+1)}$.

We summarize the resulting procedure in Algorithm 2 in Appendix B.2. The following theorem, proved in Appendix B, states that Packed-SS preserves the distribution of the AR mode. Since greedy decoding is recovered at temperature 0, it also implies the correctness of Algorithm 1.

Theorem 3.3 (Distributional correctness)**.** *The continuation generated by Packed-SS after the prompt P has the same distribution as sampling from the AR model p_θ, stopped at the first EOS or at the output length limit L.*

Finally, storing the drafter distributions $Q_{t-1}^{(i)}$ need not require a probability for every vocabulary token at each draft position. With top-k (Fan et al., 2018) or top-p truncation (Holtzman et al., 2020), we need to store only the nonzero entries of $Q_{t-1}^{(i)}$. For instance, Qwen3 recommends top-$k = 20$ for a vocabulary of 151,936 tokens (Qwen Team, 2025), and the reference implementation of Llama 3 uses top-$p = 0.9$ by default (Grattafiori et al., 2024).

4 Experiments

Models and datasets. We evaluate Packed-SS on Nemotron-Labs-Diffusion (Fu et al., 2026) and SetDLM (Arriola & Kuleshov, 2026). Nemotron 3B and 8B are evaluated on five datasets: GSM8K (Cobbe et al., 2021) and MATH-500 (Hendrycks et al., 2021) for mathematical reasoning, HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021b) for code generation, and WritingBench (Wu et al., 2025) for generation speed on writing prompts. For SetDLM, we evaluate two separately fine-tuned checkpoints on GSM8K released by (Arriola & Kuleshov, 2026).

Experimental setup. All experiments use batch size one on a single A100 80GB GPU, with Nemotron in BF16 and SetDLM in FP32. Generations are capped at 512 tokens for code and writing and 1,024 tokens for mathematics, to avoid repetitive responses dominating the comparison. Because output lengths can differ across schedules, our primary efficiency metric is aggregate token throughput, which decomposes as

$$\text{Throughput} = \frac{\text{total generated tokens}}{\text{total decoder wall time}} = \frac{\text{Tokens/NFE}}{\text{Time/NFE}}.$$

Tokens per NFE (Tokens/NFE) and decoder time per NFE (Time/NFE) are recorded to separate token yield from cost per NFE. Detailed setup and results are provided in Appendices E and F.

4.1 Greedy decoding

Nemotron. Linear-SS and Packed-SS use diffusion block size $B = 32$, while Quadratic-SS uses $B = 8$. Each queried block includes the final token of the verified prefix, so the effective speculative width is $W = B - 1$. This gives Packed-SS and Quadratic-SS comparable maximum input sizes of 64 and 72 token rows, respectively. These block sizes also give the highest throughput among the settings tested in the small-scale experiment in Appendix F.2.

Results are summarized in Figure 3, with numerical results in Table 6. Packed-SS achieves the highest measured throughput in all ten Nemotron settings. Compared with Linear-SS, throughput

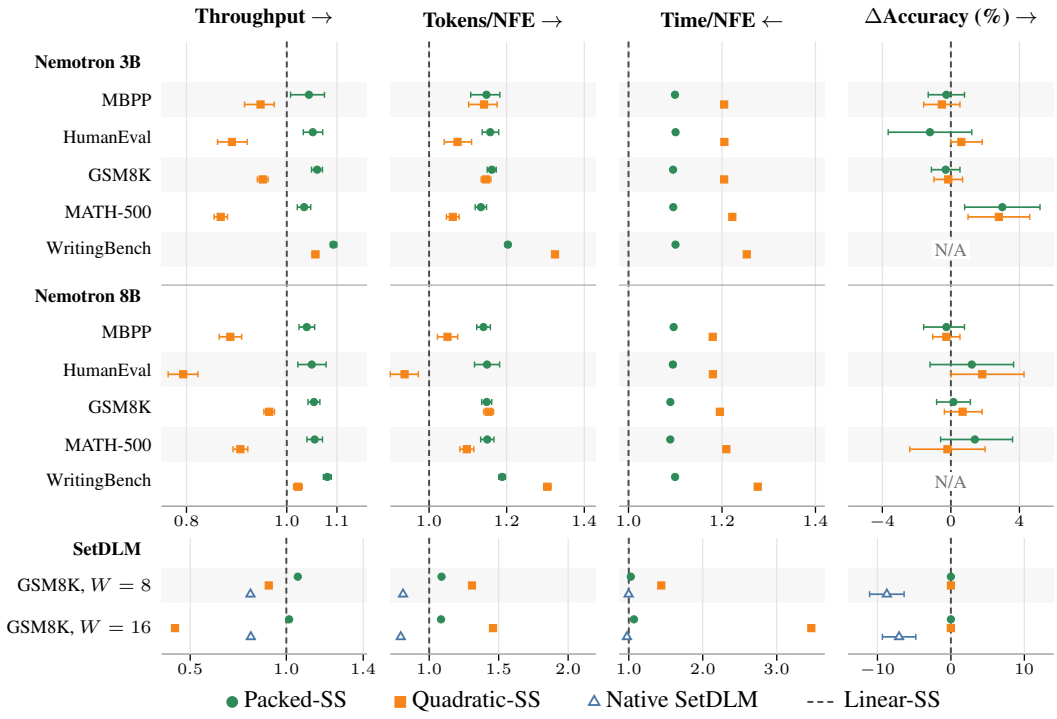


Figure 3: **Greedy decoding efficiency and accuracy.** Efficiency metrics are ratios to Linear-SS; accuracy differences are in percentage points. Error bars show 95% paired bootstrap confidence intervals. Native is the released non-speculative SetDLM diffusion decoder, shown for reference. WritingBench measures speed only.

improves by 3.5–9.3%, with all ten 95% confidence intervals for the throughput ratio above one. The improvement comes from a 13.4–20.3% increase in tokens per NFE, which more than offsets the 8.9–10.1% increase in decoder time per NFE. Compared with Quadratic-SS, throughput improves by 3.4–32.3% at the evaluated block sizes. Accuracy differences between schedules are observed, but they do not provide a direct comparison of decoding quality because they arise from finite-precision effects.

SetDLM. The checkpoints with training horizons $S_{\max} = 16$ and $S_{\max} = 32$ are evaluated at speculative widths $W = 8$ and $W = 16$, respectively. All three speculative schedules use the same width within each checkpoint.

Packed-SS improves throughput over Linear-SS by 6.0% at $W = 8$ and 1.5% at $W = 16$, as in Figure 3 and Table 8. The additional token yield outweighs the higher cost per NFE: tokens per NFE increase by 8.5–8.9%, while decoder time per NFE increases by 2.7–7.0%. Packed-SS also has higher throughput than Quadratic-SS in both settings. The improvements extend to confidence-based verification order; details are provided in Appendix F.3.

We also test the native SetDLM decoding schedule, which is diffusion decoding instead of speculative decoding. At $W = 8$ and $W = 16$, Packed-SS achieves higher throughput than Native, with accuracy higher by 8.72 and 7.05 percentage points, respectively.

4.2 Decoding at temperature 1

We further evaluate the sampling procedure of Section 3.2 at temperature 1, where draft tokens are verified probabilistically. We use the same five datasets and both Nemotron scales as in greedy decoding, with block size $B = 32$ for Linear-SS and Packed-SS and $B = 8$ for Quadratic-SS.

Scope and baselines. Linear-SS with speculative sampling is the standard stochastic setting (Leviathan et al., 2023; Chen et al., 2023). However, prior empirical evaluations of Quadratic-SS are reported under greedy decoding (Liu et al., 2025; Samragh et al., 2025; Fu et al., 2026). We therefore use Linear-SS as the primary baseline at temperature 1. For completeness, we extend Quadratic-SS to rejection sampling by applying the same probabilistic verification to its selected branch as a secondary comparison. More details are in Appendix E.4.

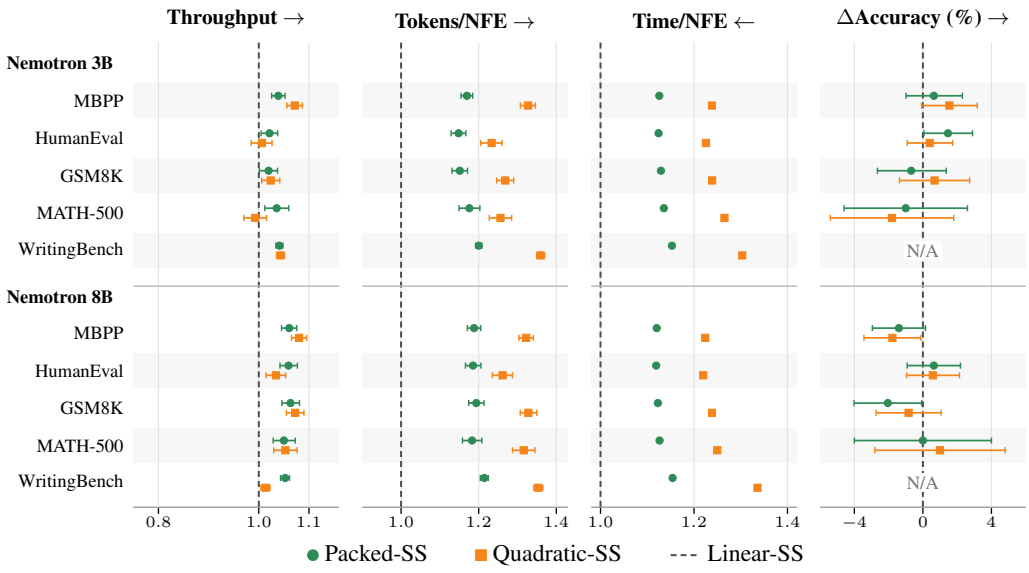


Figure 4: **Efficiency and accuracy at temperature 1.** Efficiency metrics are ratios to Linear-SS, accuracy differences are in percentage points. Error bars show 95% paired bootstrap confidence intervals. WritingBench measures speed only.

Results. Results are presented in Figure 4, with numerical results in Table 7. Compared with Linear-SS, Packed-SS achieves higher throughput in all ten settings, improving throughput by 2.0–6.3%, with all ten 95% confidence intervals for the throughput ratio above one. The improvement again comes from increased token yield: Packed-SS generates 14.8–21.5% more tokens per NFE, which outweighs the 11.9–15.4% increase in decoder time per NFE.

Compared with Quadratic-SS, Packed-SS shows comparable performance in terms of throughput. Its throughput point estimate is higher in four settings and lower in six. The paired 95% confidence intervals lie above one in three settings, below one in two, and include one in the remaining five.

5 Discussion and Limitations

We presented Packed-SS, which overlaps drafting and verification in one forward pass per iteration while evaluating only $O(W)$ positions. Under greedy decoding and a monotonicity assumption, we showed that Packed-SS requires no more forward passes than Linear-SS to generate a prescribed number of tokens from the same prompt at the same width. Experiments across multiple tasks and model families demonstrate that Packed-SS can improve throughput over existing self-speculative decoding schedules.

Several limitations and directions for future work remain. Our single-call implementation requires shared-weight models supporting parallel drafting and autoregressive verification. Combining this schedule with drafters based on intermediate-layer representations remains unexplored. The current efficiency analysis relies on an idealized assumption; extending it to weaker conditions and stochastic sampling would broaden its applicability. Finally, preparing only a subset of the continuation branches used by Quadratic-SS could provide intermediate designs between Packed-SS and Quadratic-SS, potentially improving upon the current algorithm.

AI USE STATEMENT

In this work, we used generative AI tools for assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, interpreting results. We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, and generating synthetic data sets, assisting with translation, cleaning and reformatting dataset, supporting qualitative and thematic data analysis are not applicable to this work. We have reviewed all AI-assisted work. Complete proofs were verified by the authors, LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Marianne Arriola and Volodymyr Kuleshov. Set diffusion: Interpolating token orderings between autoregression and diffusion for fast and flexible decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2607.01775.

Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/7ede97c3e082c6df10a8d6103a2eebd2-Abstract-Conference.html.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, 2021a. URL https://papers.neurips.cc/paper/2021/hash/958c530554f78bcd8e97125b70e6973d-Abstract.html.

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

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

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

Andrew Campbell, Joe Benton, Valentin De Bortoli, Tom Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. In *Advances in Neural Information Processing Systems*, 2022.

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026a. URL https://arxiv.org/abs/2602.06036.

Kecheng Chen, Ziru Liu, Xijia Tao, Hui Liu, Xinyu Fu, Suiyun Zhang, Dandan Tu, Lingpeng Kong, Rui Liu, and Haoliang Li. Beyond confidence: Adaptive and coherent decoding for diffusion language models. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026b. URL https://arxiv.org/abs/2512.02044.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. 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, et al. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026. URL https://arxiv.org/abs/2607.05147.

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

Jacob K. Christopher, Brian R. Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative diffusion decoding: Accelerating language generation through diffusion. *arXiv preprint arXiv:2408.05636*, 2024. URL https://arxiv.org/abs/2408.05636.

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. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681/.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 889–898, 2018. URL https://aclanthology.org/P18-1082/.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 14060–14079. PMLR, 2024. URL https://proceedings.mlr.press/v235/fu24a.html.

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. *arXiv preprint arXiv:2607.05722*, 2026. URL https://arxiv.org/abs/2607.05722.

Yifeng Gao, Ziang Ji, Yuxuan Wang, Biqing Qi, Hanlin Xu, and Linfeng Zhang. Self speculative decoding for diffusion large language models. *arXiv preprint arXiv:2510.04147*, 2025. URL https://arxiv.org/abs/2510.04147.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, Hao Peng, and Lingpeng Kong. Scaling diffusion language models via adaptation from autoregressive models. In *International Conference on Learning Representations*, 2025.

Aaron Grattafiori et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Ligong Han, Hao Wang, Han Gao, Kai Xu, and Akash Srivastava. S2D2: Fast decoding for diffusion LLMs via training-free self-speculation. *arXiv preprint arXiv:2603.25702*, 2026. URL https://arxiv.org/abs/2603.25702.

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://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1904.09751.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://arxiv.org/abs/2102.05379.

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.

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

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 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. *arXiv preprint arXiv:2503.01840*, 2025b. URL https://arxiv.org/abs/2503.01840.

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. *arXiv preprint arXiv:2511.08923*, 2025. URL https://arxiv.org/abs/2511.08923.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning*, 2024.

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*, 2024. doi: 10.1145/3620666.3651335. URL https://arxiv.org/abs/2305.09781.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/eb0b13cc515724ab8015bc978fdde0ad-Abstract-Conference.html.

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your LLM knows the future: Uncovering its multi-token prediction potential. *arXiv preprint arXiv:2507.11851*, 2025. URL https://arxiv.org/abs/2507.11851.

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 https://aclanthology.org/2023.acl-long.689/.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and generalized masked diffusion for discrete data. In *Advances in Neural Information Processing Systems*, 2024.

Benjamin Spector and Chris Ré. Accelerating LLM inference with staged speculative decoding. *arXiv preprint arXiv:2308.04623*, 2023. URL https://arxiv.org/abs/2308.04623.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/c4127b9194fe8562c64dc0f5bf2c93bc-Abstract.html.

Yao Teng, Han Shi, Xian Liu, Xuefei Ning, Guohao Dai, Yu Wang, Zhenguo Li, and Xihui Liu. Accelerating auto-regressive text-to-image generation with training-free speculative jacobi decoding. *arXiv preprint arXiv:2410.01699*, 2024. URL https://arxiv.org/abs/2410.01699.

Shutong Wu and Jiawei Zhang. Free draft-and-verification: Toward lossless parallel decoding for diffusion large language models. *arXiv preprint arXiv:2510.00294*, 2025. URL https://arxiv.org/abs/2510.00294.

Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, and Fei Huang. Writingbench: A comprehensive benchmark for generative writing. *arXiv preprint arXiv:2503.05244*, 2025. URL https://arxiv.org/abs/2503.05244.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Yifan Yu, Yuqing Jian, Junxiong Wang, Zhongzhu Zhou, Donglin Zhuang, Xinyu Fang, Sri Yanamandra, Qingyang Wu, Tri Dao, Xiaoxia Wu, Shuaiwen Leon Song, Ben Athiwaratkun, James Zou, Fan Lai, and Chenfeng Xu. Introspective diffusion language models. *arXiv preprint arXiv:2604.11035*, 2026. URL https://arxiv.org/abs/2604.11035.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. doi: 10.18653/v1/2024.acl-long.607. URL https://aclanthology.org/2024.acl-long.607/.

Yaojie Zhang, Linfeng Zhang, Bin Cui, and Xupeng Miao. DFlow: Enabling verifier information flow in block diffusion speculative decoding. *arXiv preprint arXiv:2609.06498*, 2026. URL https://arxiv.org/abs/2609.06498.

A NOTATION

Table 1 collects the symbols used in Section 3. In LTR, $d_t^{(i)}$ is indexed after P_t, while $z_t^{(i)}$ and $v_t^{(i)}$ are indexed after P_{t-1}. In the any-order procedure, position indices are fixed output positions after the prompt.

Table 1: **Symbols.** Sizes and indices, decoder state, and distributions.

Sizes and indices	
t	decoding iteration, starting at 1
i, j	token position: relative in LTR, fixed in any-order decoding
r, s	rank in the verification order (any-order setting)
W	speculative width: the fixed number of future positions proposed by the LTR drafter, $W \geq 1$, and the size of the active window in the any-order setting. The schedule and Theorem 3.2 are stated in W alone
B	Nemotron's root-inclusive block size, $B = W + 1$, including the known root token
u_t	number of unresolved positions in the active window, $u_t = \lvert U_t \rvert \leq W$ (any-order setting)
κ_t	available draft-prefix length in verification order π_t, with $0 \leq \kappa_t \leq u_t$ (any-order setting)
h_t	number of consecutive accepted draft tokens in verification order
A_t	advancement before applying the stopping rule; $A_t = h_t + 1$ for the next-token verifier and $A_t = \min(h_t + 1, u_t)$ for the any-order same-position verifier (Appendix C)
a	emitted token count after applying the stopping rule, $a \leq A_t$ (LTR)
L	maximum number of generated tokens, excluding the prompt
E	current output endpoint in any-order decoding; initially L, reduced when an EOS is committed
$n_t = \lvert P_t \rvert$	LTR prefix length, including the prompt
Decoder state	
P, P_t	prompt; verified prefix (LTR) or committed history (any-order), with $P_0 = P$
D_t	draft available for verification at iteration $t + 1$, with $D_0 = \varnothing$; indexed after P_t in LTR, with $\lvert D_t \rvert \leq W$. In any-order decoding, a position–token map on J_t
Z_t	new proposal: W positions after the input prefix in LTR, or positions in U_t in any-order decoding
$v_t^{(i)}$	verifier token at position i
U_t, J_t	active unresolved positions; positions retained for the next draft, $J_t \subseteq U_t$ (any-order)
c_t, $c_t^{(i)}$	stored draft-confidence map and its value at position $i \in J_t$ (any-order)
π_t	verification order used by iteration t
σ	generation order of an any-order model
C_t	verified outputs of iteration t, pruned at the stopping boundary; tokens in LTR or ordered position–token pairs in any-order decoding
Distributions	
$\mathcal{V}$, θ	vocabulary; shared weights
p_θ	verifier conditional, read from causally masked rows
$q_\theta^{(i)}$	drafter conditional at position i: relative to its conditioning prefix in LTR, fixed in any-order decoding
$q_t^{(j)}$	fresh drafter conditional $q_\theta^{(j)}(\cdot \mid P_{t-1})$, before prefix advancement (LTR)
$Q_t^{(i)}$	distribution stored with $d_t^{(i)}$; $Q_t^{(i)} = q_t^{(a+i)}$ after retaining the LTR suffix
$p_t^{(i)}$	verifier conditional at position i after P_{t-1} (LTR)
Analysis (Theorem 3.2)	
$c_k^\star$	correctness threshold for output position k; $c_k^\star = k$ denotes no correct draft prediction before verification (Assumption 3.1)
x_k, $\widehat{x}_k(c)$	target greedy token at output position k; drafter's prediction from the first c verified output tokens
ℓ_t, c_t	verified output lengths after t Linear iterations and t Packed forward passes, respectively (proof-local counters)
$F(c)$	endpoint of fresh drafting and verification from c verified output tokens, before stopping (Lemma B.1)
$\mathrm{NFE}_S(L)$	total model calls used by schedule S to generate L tokens after the prompt

B PROOFS

B.1 PATHWISE DOMINANCE

The endpoint comparisons below are for the schedules before applying EOS or length truncation; stopping is handled at the end of the theorem's proof.

Lemma B.1 (Monotone fresh endpoint)**.** *Count verified output tokens after the prompt by c, and define*

$$F(c) = \min\{c + W + 1,\ \min\{k > c : c_k^\star > c\}\}, \tag{6}$$

with the minimum of an empty set read as $+\infty$. Then F is nondecreasing. Under Assumption 3.1, it is the endpoint of fresh drafting and verification at any context c evaluated by either schedule.

Proof. The first position with $c_k^\star > c$ is the first incorrect proposal; verification commits the earlier matches and the corrected token there, and when all W draft tokens match the bonus token sits at $c + W + 1$. For $c' \geq c$: if $c' \geq F(c)$ then $F(c') \geq c' + 1 > F(c)$. Otherwise every k with $c' < k < F(c)$ has $c_k^\star \leq c \leq c'$, and $c' + W + 1 \geq c + W + 1 \geq F(c)$; hence $F(c') \geq F(c)$ by definition. □

Lemma B.2 (Two forward passes reach the fresh endpoint)**.** *If a Packed-SS iteration begins with c verified output tokens and its successor ends with c'', then $c'' \geq F(c)$.*

Proof. The first iteration proposes positions $c + 1, \ldots, c + W$ from c while verifying its current, possibly empty draft, advancing to some $c' > c$. If $c' \geq F(c)$ the claim holds. Otherwise the successor retains positions $c' + 1, \ldots, F(c) - 1$, each correct under the original context c by the definition of $F(c)$ and each within the original proposal range since $F(c) - 1 \leq c + W$. It therefore accepts them all and emits a correction or bonus at a position at least $F(c)$. □

Proof of Theorem 3.2. Consider the schedules before EOS or length truncation. Let ℓ_t and c_t count verified output tokens after the prompt for Linear-SS after t iterations and Packed-SS after t forward passes, with $c_0 = \ell_0 = 0$. Lemmas B.1 and B.2 and induction give $c_{2t+2} \geq F(c_{2t}) \geq F(\ell_t) = \ell_{t+1}$. For $L > 0$, let t be Linear-SS's first iteration with $\ell_t \geq L$; then $c_{2t} \geq L$, so Packed-SS finishes within $2t$ forward passes, exactly Linear-SS's charge. For $L = 0$, neither schedule makes a call. The first Packed-SS call has an empty draft and advances by one, so Lemma B.2 applies from $c_0 = 0$. Both schedules follow the same reference continuation, so an emitted EOS is a common deterministic horizon, and truncating the final output does not remove a forward pass already performed. □

The comparison counts all calls of the stated schedules. The wider first proposal used by Nemotron (Appendix E.2) also satisfies Lemma B.2 provided its predictions on the first W positions obey the same threshold assumption. If Linear-SS instead emits one direct target token before alternating drafting and verification, both schedules share the same one-token prefix after one call. The same induction gives dominance at $1 + 2t$ calls, so the total-call bound still holds.

B.2 Exact sampling

Lemma B.3 (One-token correction under the actual proposal law)**.** *Let $\mathcal{G}$ contain the emitted history H and the normalized vectors P, Q, but not the draft token's draw, and suppose $\Pr(D = x \mid \mathcal{G}) = Q(x)$ with $P(x) = p(x \mid H)$. Accept D with probability $\min\{1, P(D)/Q(D)\}$, and otherwise emit an independent draw from the normalized positive part of $P - Q$. Then the emitted token has conditional law P given $\mathcal{G}$.*

Proof. Write $\beta = \sum_x \min\{P(x), Q(x)\}$. Given $\mathcal{G}$, acceptance contributes $\min\{P(x), Q(x)\}$ to the probability of emitting x; rejection has probability $1 - \beta = \sum_y [P(y) - Q(y)]_+$ and contributes $[P(x) - Q(x)]_+$. Their sum is $P(x)$. A sampled draft token has $Q(D) > 0$ almost surely so the ratio is defined, and if $\beta = 1$ the residual branch is never taken. No full-support assumption is needed on either distribution. □

This is the standard speculative-sampling identity (Leviathan et al., 2023; Chen et al., 2023), stated conditionally so that a stale proposal is admissible. When applying the lemma, the conditioning excludes draft tokens not yet tested, even if they have already been computed; an earlier acceptance decision must not depend on them.

The state the decoder keeps. Algorithm 2 states the LTR procedure and returns the completed sequence, including the prompt. The limit L counts only newly generated tokens. Two pieces of state distinguish it from Algorithm 1: each retained draft token carries the normalized law $Q^{(i)}_{t-1}$ it was drawn from, and those draws are taken only after the advance has fixed which positions survive. The loop starts from an empty draft and follows the greedy procedure, with matching replaced by Lemma B.3.

Algorithm 2 Packed-SS with LTR rejection sampling

Require: prompt P, speculative width W, output length limit L, verifier p_θ, drafter q_θ

$P_0 \leftarrow P,\ D_0, Q_0 \leftarrow \varnothing,\ t \leftarrow 1$

while $|P_{t-1}| - |P| < L$ and ($t = 1$ or the last token of P_{t-1} is not EOS) **do**

In one model call:

$p^{(i)}_t \leftarrow p_\theta(\cdot \mid P_{t-1}, d^{(<i)}_{t-1})$ for $i = 1, \dots, |D_{t-1}| + 1$ ▷ Verifier distributions

$q^{(j)}_t \leftarrow q^{(j)}_\theta(\cdot \mid P_{t-1})$ for $j = 1, \dots, W$ ▷ Drafter proposals

end

$u_i \sim \mathrm{Unif}(0,1),\ \alpha_i \leftarrow \min\{1,\ p^{(i)}_t(d^{(i)}_{t-1}) / Q^{(i)}_{t-1}(d^{(i)}_{t-1})\}$ for $i = 1, \dots, |D_{t-1}|$ ▷ Acceptance tests

$h_t \leftarrow \max\{0 \le h \le |D_{t-1}| : u_i < \alpha_i \text{ for all } i \le h\}$ ▷ Count accepted tokens

$c \sim R, \quad R \propto \begin{cases} [p^{(h_t+1)}_t - Q^{(h_t+1)}_{t-1}]_+, & h_t < |D_{t-1}|, \\ p^{(|D_{t-1}|+1)}_t, & h_t = |D_{t-1}|. \end{cases}$ ▷ Correction or bonus

$C_t \leftarrow (d^{(1)}_{t-1}, \dots, d^{(h_t)}_{t-1}, c),\ A_t \leftarrow h_t + 1$ ▷ Add verifier token

$a \leftarrow \min\big(\{A_t, L + |P| - |P_{t-1}|\} \cup \{1 \le i \le A_t : C^{(i)}_t = \mathrm{EOS}\}\big)$ ▷ Ending condition

$C_t \leftarrow (C^{(1)}_t, \dots, C^{(a)}_t)$ ▷ Prune by stopping index

$P_t \leftarrow (P_{t-1}, C_t)$ ▷ Append to prefix

$Q^{(j)}_t \leftarrow q^{(a+j)}_t,\ d^{(j)}_t \sim Q^{(j)}_t$ for $j = 1, \dots, \max(0, W - a)$ ▷ Sample retained draft

$D_t \leftarrow (d^{(1)}_t, \dots, d^{(\max(0,W-a))}_t)$ ▷ Prepare next draft

$t \leftarrow t + 1$

end while

Output: P_{t-1}

The indented block is the one packed model call per iteration; the draws, acceptance tests, residual construction, and state updates add none. Prompt rows are cached on the first call. Subsequent cache promotion follows Appendix D: only the old root and the reached accepted draft rows are retained, and $C^{(a)}_t$ becomes the next uncached root.

Lemma B.4 (The retained suffix keeps its cached law)**.** *Let $\mathcal{G}_t$ contain the verified prefix before call t, its old proposal state, the current verification randomness and outcome, and the drafter vectors, but not the fresh draws. Then the updated prefix and retained indices are $\mathcal{G}_t$-measurable, and for any $\mathbf{d} = (d^{(i)})_{i=1}^{[W-a]_+}$,*

$$\Pr\big(D_t = \mathbf{d} \mid \mathcal{G}_t\big) = \prod_{i=1}^{[W-a]_+} Q^{(i)}_t(d^{(i)}). \tag{7}$$

Proof. The advance a is determined by the verification outcome, which is in $\mathcal{G}_t$, so the retained index set is too, and the draws are conditionally independent given $\mathcal{G}_t$, with the stated marginals. Conditioning on earlier accepted draft tokens during the next verification does not change a later draft token's law, since those decisions involve only earlier draws and independent verifier randomness. Drawing all coordinates before a is known gives the same conditional product law provided the suffix choice cannot inspect their values; deferring the draws is a transparent way to enforce that. □

Proof of Theorem 3.3. With $D_0 = \varnothing$, the first emitted token is a direct target draw. At each output position, either no draft token remains, in which case the decoder samples from the target conditional, or the next draft token has the law of Equation (7) and Lemma B.3 makes its emission exactly the target conditional. Earlier accepted draft tokens equal the emitted history at that point, so their teacher-forced verifier conditionals are the serial ones; target logits past a rejection may have read a rejected token but are never used. After an accepted draft token its real-token representation may stay in the cache, its attention dependencies agreeing with the serial history, while a residual

correction is processed as the next uncached root and neither draft-lane nor masked query rows are promoted. These cache invariants are implemented by the updates in Appendix D, so the verifier still returns the target conditionals at the next call. Induction over output positions and the chain rule give the stated joint law, and since only emitted EOS values are tested for termination, stopping the target process at its first EOS or the output cap gives the same stopped distribution. Every iteration uses one fused forward pass; the draws, acceptance tests, and residual construction add no additional forward passes, though they do add memory traffic. □

Greedy specialization. For greedy decoding, take the target and drafter conditionals to be point masses at their deterministically chosen argmax tokens. Then Theorem 3.3 gives the greedy correctness of Algorithm 1, including EOS and length stopping.

Remark B.5 (Any-order scope). For a permutation fixed independently of the draft token values, relabelling the induction by rank gives the corresponding ordered model law, and for SetDLM-style position queries full acceptance of the queried set needs no outside-set bonus. Two restrictions apply before extending this to an adaptive order. Sorting by the probabilities of *sampled* draft token values can change their conditional laws; an order fixed before the draws, for instance from entropy or maximum probability, avoids this. Separately, an any-order model may define different joint distributions for different generation orders. A stochastic guarantee must therefore specify a fixed order or a rule that chooses each next position using only information already revealed, without inspecting future token draws. Conditioning afterward on a token-dependent order does not give this guarantee. If all position conditionals come from the same joint distribution, choosing positions by such a rule preserves that distribution; this requires an additional assumption beyond the packed attention mask.

C ANY-ORDER EXTENSION

Section 3 resolves positions left to right, but a call needs only that each draft token be scored on committed context and on the draft tokens visited before it, in whatever order that is. An any-order AR model serves as such a verifier, conditioning on previously generated positions in a permutation σ:

$$p_\theta^\sigma(x_{1:L} \mid P) = \prod_{r=1}^{L} p_\theta(x_{\sigma(r)} \mid P, x_{\sigma(<r)}; \sigma).$$

The order specifies the conditionals, so different orders can produce different greedy outputs. Carrying Packed-SS and Quadratic-SS onto such a verifier needs three changes from LTR.

In the any-order setting, the speculative width W specifies the size of the active window. The drafter predicts only the unresolved positions within this window, whose number may be smaller than W. First, the state carries positions rather than an ordered suffix: P_{t-1} becomes a committed history of position–token pairs, D_{t-1} maps positions to draft tokens with stored confidences $c_{t-1}^{(i)}$, and the active set U_t holds the $u_t = |U_t|$ unresolved positions in a window of size W, starting at the leftmost gap. Some positions in U_t may have no stored draft token, including all positions in iteration 1. Second, the verify step acquires an order: each call fixes π_t—increasing position for LTR, or decreasing stored confidence—and draft tokens stay attached to their own positions even when visited out of order. Write κ_t for the available draft-prefix length in this order. The prefix ends at the first missing entry, so $\kappa_t \le |\mathrm{dom}(D_{t-1}) \cap U_t| \le u_t \le W$. Here, $\pi_t(r)$ is the fixed output position at verification rank r, with output positions numbered from 1 after the prompt. Position superscripts refer to these fixed positions. The verifier uses earlier-ranked draft tokens to predict

$$v_t^{(\pi_t(r))} = \arg\max_x p_{\theta,\pi_t(r)}\big(x \mid P_{t-1}, (\pi_t(s), d_{t-1}^{(\pi_t(s))})_{s<r}\big), \tag{8}$$

and only ranks up to the first unavailable draft token are needed. Third, the draft and retain steps follow the active set: the new proposal predicts all of U_t from P_{t-1}, and the next draft keeps its uncommitted positions and their scores. The accept step changes only if the verifier is *same-position*, reading the token at each queried position rather than the one after it. If all u_t unresolved positions have draft tokens and all match, the iteration commits u_t tokens; otherwise, it commits the matching draft prefix followed by one verifier token. Thus, the advancement before applying the stopping rule is $\min(h_t + 1, u_t)$, with no extra token outside the active window.

Algorithm 3 states the procedure; an out-of-order EOS sets an endpoint, and generation fills all earlier gaps before returning. Because different schedules store different confidences they can realize different orders, so greedy decoding is preserved along the order actually taken, not between independently chosen trajectories. Appendix E.6 gives the ranking and tie-breaking rules; Appendix D gives the attention masks and cache updates. Confidence-order experiments are in Appendix F.3.

In Algorithm 3, $\mathrm{dom}(P_t)$ denotes the committed output positions in the history, excluding the prompt, and $P_t(j)$ is the token stored at position j. The restriction $\cdot|_{\le E}$ removes output pairs beyond E, preserving the prompt and the order of the remaining history; restricting a history also evicts the corresponding cached state as in Appendix D.2. The completed sequence is returned with the prompt included and its continuation restored to textual order.

Algorithm 3 Packed-SS with any-order greedy decoding

Require: prompt P, speculative width $W \geq 1$, output length limit $L > 0$, verifier p_θ, drafter q_θ, order policy

$P_0 \leftarrow P,\ D_0, c_0 \leftarrow \varnothing,\ E \leftarrow L,\ t \leftarrow 1$
while $\{1, \ldots, E\} \not\subseteq \mathrm{dom}(P_{t-1})$ **do**
  $\ell \leftarrow \min\big(\{1, \ldots, E\} \setminus \mathrm{dom}(P_{t-1})\big)$ ▷ Leftmost unresolved position
  $U_t \leftarrow \{\ell, \ldots, \min(E, \ell + W - 1)\} \setminus \mathrm{dom}(P_{t-1}),\ u_t \leftarrow |U_t|$ ▷ Active positions
  Rank U_t by the order policy to obtain π_t ▷ Verification order
  $\kappa_t \leftarrow \max\{0 \leq k \leq u_t : \pi_t(r) \in \mathrm{dom}(D_{t-1}) \text{ for all } r \leq k\}$ ▷ Available draft prefix
  In one model call:
    Compute $v_t^{(\pi_t(r))}$ by Equation (8) for $r = 1, \ldots, \min(\kappa_t + 1, u_t)$ ▷ Verifier predictions
    $z_t^{(j)} \leftarrow \arg\max_x q_\theta^{(j)}(x \mid P_{t-1})$ for $j \in U_t$ ▷ Drafter proposals
  end
  $h_t \leftarrow \max\{0 \leq h \leq \kappa_t : d_{t-1}^{(\pi_t(r))} = v_t^{(\pi_t(r))} \text{ for all } r \leq h\}$ ▷ Count matches
  $A_t \leftarrow \min(h_t + 1, u_t),\ C_t \leftarrow \big((\pi_t(r), v_t^{(\pi_t(r))})\big)_{r=1}^{A_t}$ ▷ Commit verified tokens
  $E \leftarrow \min\big(\{E\} \cup \{j : (j, \mathrm{EOS}) \in C_t\}\big)$ ▷ Ending condition
  $C_t \leftarrow C_t|_{\leq E},\ P_t \leftarrow P_{t-1}|_{\leq E}$ ▷ Prune by endpoint
  $P_t \leftarrow (P_t, C_t)$ ▷ Append to history
  $J_t \leftarrow \{j \in U_t \setminus \mathrm{dom}(P_t) : j \leq E\}$ ▷ Retained draft positions
  $D_t \leftarrow \{j \mapsto z_t^{(j)} : j \in J_t\}$ ▷ Prepare next draft
  $c_t \leftarrow \{j \mapsto q_\theta^{(j)}(z_t^{(j)} \mid P_{t-1}) : j \in J_t\}$ ▷ Store draft confidence
  $t \leftarrow t + 1$
end while
Output: $(P, P_{t-1}(1), \ldots, P_{t-1}(E))$ ▷ Textual order

Stopping and the serial reference. An any-order decoder may have committed positions beyond a newly discovered endpoint. It removes those positions from the cache without recomputing the remaining cached states, which may still reflect attention to the removed positions. The serial reference in Proposition C.1 follows the same verification orders and committed history. It applies the same endpoint and cache updates after the same groups of tokens at the end of each packed call. The equivalence uses this retained cache state; recomputing the shortened sequence or choosing a new confidence order defines a different reference. In Equation (8), P_{t-1} denotes this ordered committed context together with its retained KV state.

Proposition C.1 (Greedy correctness under a realized order)**.** *Assume the verifier exposes only committed context and the draft tokens earlier in the verification order, and that conditionals, tie-breaking, and stopping match the serial reference. Then Algorithm 3 returns the serial greedy output along its realized verification order, restored to textual order.*

Proof of Proposition C.1. Fix the permutation π_t each call actually used rather than letting the reference recompute a ranking. Induct on committed positions: by Equation (8), each verifier conditional agrees with the serial reference, so consecutive matching draft tokens and any correction are the same greedy choices. If every queried position matches, no outside-window prediction is needed. Matching real-token rows have exactly the accepted earlier-rank dependencies and can be cached; a correction predicted at a mask has no reusable real-token row and is evaluated as a pending committed token in the next call, and neither masked prediction rows nor rejected draft rows enter the cache. Applying endpoint detection and cache eviction at the same call boundaries in both procedures, as specified above, covers an EOS discovered out of textual order, and induction to the endpoint followed by the same textual-order return proves the claim. □

D ATTENTION MASK LAYOUTS

The algorithms in the main text describe token values and their dependencies. This appendix specifies the packed inputs, attention masks, and persistent KV state for each model family. Logical draft length and physical input length are distinguished throughout: a missing draft entry is not an additional speculative token, even when storage for that entry is evaluated. Input-size comparisons are in Appendix F.1.

Verifier conventions. The two families realize the packed call differently. Nemotron uses a next-token verifier and concatenates the drafting and verification regions along the sequence axis (Fu et al., 2026), whereas SetDLM uses a same-position verifier and duplicates draft tokens and masked queries, so that each query sees the draft tokens before it without reading its own (Arriola & Kuleshov, 2026; Han et al., 2026). Which verifier a family uses changes only what a fully matched draft advances, not the schedule (Appendix C).

D.1 NEMOTRON: SEQUENCE PACKING AND PADDING

The recurrent input begins with the known, already-emitted root $P_{t-1}^{(n_{t-1})}$, followed by $B-1$ draft slots. The first-call construction is given in Appendix E.2.

Physical layout. Let $\bar{D}_{t-1}$ be D_{t-1} followed by enough masks to occupy $B-1$ slots. The measured Packed call has input

$$[P_{t-1}^{(n_{t-1})}, \bar{D}_{t-1} \mid P_{t-1}^{(n_{t-1})}, \textsc{Mask}^{B-1}],$$

concatenated along the sequence axis. Its shape is $[1, 2B]$, or $[1, 64]$ at $B = 32$; this is a single sequence, not a batch of two requests or a batch-axis stack of branches.

Let T_B be a lower-triangular Boolean matrix including its diagonal, and J_B an all-ones matrix. Uncached-row visibility is

$$\begin{pmatrix} T_B & 0 \\ 0 & J_B \end{pmatrix},$$

and every row additionally sees the cached prefix $P_{t-1}^{(<n_{t-1})}$. The causal region yields the shifted next-token verifier predictions. The proposal region holds a duplicate of the known root, whose output is unused, and $B-1$ mask queries, whose outputs are the entries of Z_t. Padded verifier rows are evaluated but cannot extend acceptance, since verification stops at the first absent draft token, represented physically by a mask.

Cache update and finite overlap. After an advance of a, the decoder retains the first a causal input rows, namely the old root and $a-1$ matching draft tokens, and the last verifier output becomes the next uncached root; all proposal-region KV entries and unaccepted verifier inputs are discarded. Among the proposal outputs, only zero-based physical indices $a+1, \ldots, W$ remain usable, giving the suffix of Equation (4) with length $|D_t| = \max(0, W-a)$. Full acceptance of the W-token initial draft can advance $W+1$ tokens and leaves an empty draft. The code pads the remainder of the next recurrent input with masks, including the all-accept case where only the new root is known. The reported implementation uses the full root-inclusive block size B.

D.2 SETDLM: POSITION-ALIGNED QUERIES AND ORDERED CONTEXT

Window and stored draft tokens. Let U_t be the unresolved positions within the active window of size W, and write $u_t = |U_t|$. Previously committed positions inside the window need no queries, so u_t can be smaller than W, especially under confidence ordering, and each draft token remains attached to its textual position. Unlike Nemotron, SetDLM predicts a token at a masked query's own position, so placing the draft token at that query would expose the answer. The verifier follows the chosen generation order and uses separate real-token draft rows and masked prediction rows, following SetDLM's clean/noisy construction and the position-aligned verification discussed by S2D2 (Arriola & Kuleshov, 2026; Han et al., 2026).

Packed input and attention. The uncached input is

$$[N_t \mid O_t \mid M_t \mid R_t], \qquad |N_t| = p \in \{0, 1\}, \quad |O_t| = |M_t| = |R_t| = u_t.$$

Here N_t is an optional committed correction whose real-token KV row has not yet been computed; O_t contains the previous draft D_{t-1}, with masks for missing entries; M_t contains verifier mask queries; and R_t contains new proposal mask queries. All regions read the persistent cache of committed tokens. The draft token and query regions additionally read N_t, if present, and the pending row sees itself.

Write $\mathcal{K}_r$ for earlier ranks $s < r$ in the call's order π_t that have a draft token. Remaining within-call visibility is

$$O_r \leftarrow \{O_s : s \in \mathcal{K}_r\} \cup \{O_r\}, \qquad M_r \leftarrow \{O_s : s \in \mathcal{K}_r\} \cup \{M_r\}, \qquad R_r \leftarrow \{R_r\}.$$

The three rows at rank r share the textual position ID $\pi_t(r)$, so new proposals see committed context but no old speculative draft token. The physical call evaluates all u_t verifier queries, while the logical algorithm needs predictions only through the first missing draft token or disagreement.

Cache promotion. Only matching O_t rows and the pending real-token row enter the persistent cache. When M_t predicts a correction, the model has processed a mask at that position, so its keys and values do not represent the predicted token. The correction is processed as a token in the next call through N_{t+1}. Masked verifier and draft rows are never promoted, even when their output token is committed. Under full acceptance there is no additional outside-window token or pending correction. When the output endpoint shrinks, cached rows beyond it are evicted without recomputing the retained rows (Appendix C).

E EXPERIMENTAL DETAILS

E.1 HARDWARE, MODELS, AND TASKS

All runs use batch size one on a single A100 80GB GPU. Nemotron is evaluated in bfloat16 (BF16), and SetDLM in float32 (FP32). Nemotron uses left-to-right AR verification in every main setting. Both scales run the same four tasks—HumanEval, MBPP, MATH-500, and GSM8K (Chen et al., 2021; Austin et al., 2021b; Hendrycks et al., 2021; Cobbe et al., 2021)—under greedy decoding and under rejection sampling. Each cell draws on a fixed prompt set: 164 HumanEval problems, 378 MBPP problems, 500 MATH-500 problems, and all 1,319 GSM8K test questions, and these counts are the number of examples behind every mean and interval reported for that cell. Both scales additionally run all 555 English WritingBench prompts (Wu et al., 2025) at both temperatures. The SetDLM matrix uses a 1,024-token target canvas and active windows 8 and 16; because its two windows come from separately adapted checkpoints, comparing them is not a same-weights window-size ablation. At both temperatures, Linear-SS and Packed-SS use Nemotron block size $B = 32$, while Quadratic-SS uses $B = 8$, with speculative width $W = B - 1$.

Training horizon versus inference window. The SetDLM training construction uses a desired prediction budget $S \in \{4, 8, 16\}$ and permits schedules reaching $2S$ future positions (Arriola & Kuleshov, 2026). Checkpoints are named by their maximum horizon. Hence smax32 is the $S = 16$, maximum-horizon-32 model, and its matched active evaluation window is $W = 16$ rather than 32.

E.2 INITIALIZATION AND STOPPING

Nemotron Packed-SS under greedy decoding. The initial call evaluates the causal prompt and a separate block of B mask rows. The last prompt row supplies the first AR token; the mask rows predict their own output positions. The decoder replaces the first draft entry with the AR token, appends that token to the output, and caches only the prompt, so the recurrent input begins with a known, already-emitted *root* $P_{t-1}^{(n_{t-1})}$ followed by $B - 1$ draft slots. The first proposal covers $B = W + 1$ future positions and leaves W draft tokens after the first output, rather than the $W - 1$ of Algorithm 1. Later proposals contain the known root and $B - 1 = W$ future positions; every iteration uses one call.

SetDLM Packed-SS under greedy decoding. The SetDLM implementation builds the prompt cache in one counted call, then follows the empty-draft loop of Algorithm 3.

Rejection sampling. The measured sampling runs begin with a causal prefill and a root-conditioned proposal call, both counted in NFE, then follow the verification-and-drafting loop. This supplies a full draft before the first verification, whereas Algorithm 2 starts from an empty draft. Both procedures store the distribution used to sample each draft token and use it in that token's acceptance test (Theorem 3.3).

Output caps. All three schedules use the same output cap within each cell to limit the influence of repetitive or non-terminating responses on timing comparisons. The two code tasks use a 512-token output cap at both temperatures. In the main experiments, the two mathematics tasks use a 1,024-token output cap at both temperatures. Responses that reach the cap are retained in all measurements. WritingBench keeps a 512-token cap at both temperatures and is a capped generation-speed study, with no writing-quality grader or reported score. On the code and mathematics tasks, fewer responses reach the cap and output lengths differ across methods, so we compare token throughput.

Terminators. Nemotron truncates output at the first EOS token or at the generation limit, and an EOS inside an unaccepted proposal does not stop the decoder. Empty drafts still permit a correction, and full acceptance adds at most B output tokens.

For SetDLM, the measured harness supports configured single- and multi-token terminators, recognizing a terminator only once all of its positions have been committed. The output endpoint is the earliest end position among completed terminators and moves earlier if an earlier terminator is committed later. Positions beyond the endpoint are removed from the output, draft, and cache.

Decoding continues until every position up to the endpoint is verified, filling any remaining gaps before returning. Algorithm 3 presents the single-token EOS case, and the implementation applies the same update at the end of a committed stop sequence.

E.3 Prompts and scoring

Nemotron uses zero-shot prompts: function completion for HumanEval, visible original test cases for MBPP, and boxed final answers for the mathematics tasks. MBPP is the 378-problem EvalPlus subset scored on its original base tests rather than the expanded MBPP+ tests, and MATH-500 uses a lightweight boxed-answer exact match rather than a canonical symbolic-equivalence grader. The temperature-1 code runs use the same prompts and the same grading harness as the temperature-0 runs, so the two modes are scored identically. These scores therefore document task outcomes under our shared harness and are not directly comparable with leaderboard values obtained under different prompting or grading. SetDLM uses the same GSM8K questions for every method and compares normalized final answers. Scored-task tables report accuracy or pass@1 at temperature 0, and mean per-response correctness at temperature 1; the WritingBench study measures speed only and carries no judge-based score.

E.4 Sampling and repetitions

The code and mathematics cells reuse the temperature-0 prompt construction, graders, and caps of Appendix E, drawing 20 responses per HumanEval prompt, 7 per MBPP prompt, and one per MATH-500 or GSM8K prompt. The WritingBench cells use all 555 English prompts, with one response per prompt and method. Each complete WritingBench query is placed in the checkpoint's user-message chat template with thinking disabled, and prompts are not truncated. All cells at both scales use left-to-right order, batch one, bfloat16 model execution with float32 probability normalization, temperature 1, and no adapter. Sampling uses the full vocabulary without top-k or top-p filtering and stops at EOS or the output cap: 512 new tokens for code and WritingBench, and 1,024 for mathematics. The WritingBench cells remain a capped generation-speed study rather than the benchmark's full-length, judge-scored evaluation.

Sampling uses the acceptance and residual correction of Appendix B.2. Packed-SS retains the aligned unresolved suffix of the freshly computed proposal distributions while Quadratic-SS selects the branch matching the accepted draft prefix, and both draw their next draft tokens only *after* that selection, keeping the actual q for the subsequent acceptance test and residual correction. We store full distributions for residual sampling.

E.5 Timing, metrics, and confidence intervals

Inclusive decoder wall time is CUDA-synchronized and includes prefill and all decoding work, excluding model loading, prompt preparation, tokenization, output-text decoding, and scoring. Each model forward pass, including prompt processing, increments NFE by one, regardless of the number of uncached rows evaluated. The row counts in Table 2 describe iterations with cached prompt context.

Emitted length is the number of tokens actually committed, excluding the prompt, padding, and uncommitted draft tokens, but including emitted stop and special tokens and any generated reasoning text. For a method m with emitted length L_{mi} and inclusive wall time t_{mi} on example i, writing P for Packed-SS and b for a baseline, we report

$$T_m = \frac{\sum_i L_{mi}}{\sum_i t_{mi}}, \qquad R_{\mathrm{P}/b} = \frac{T_{\mathrm{P}}}{T_b}, \qquad \Delta t = \frac{1}{n}\sum_i \big(t_{\mathrm{P}i} - t_{bi}\big).$$

Throughput is therefore a ratio of totals, not an average of per-example rates. With N_{mi} denoting total NFE, the other per-NFE metrics are

$$U_m = \frac{\sum_i L_{mi}}{\sum_i N_{mi}}, \qquad C_m = \frac{\sum_i t_{mi}}{\sum_i N_{mi}}, \qquad T_m = \frac{U_m}{C_m}.$$

Here U_m is tokens/NFE and C_m is inclusive decoder seconds/NFE, including work outside the model forward pass. The first three columns of Figures 3 and 4 divide throughput, tokens/NFE, and

time/NFE by Linear-SS's corresponding value in the same setting. The throughput ratio also equals the tokens/NFE ratio divided by the time/NFE ratio. The fourth column reports each method's accuracy minus Linear-SS's accuracy in percentage points, with reference zero. All intervals are 95% percentile intervals from 20,000 nonparametric bootstrap resamples of examples within each checkpoint, task, decoding mode, and decoding order. For every comparison, each draw resamples the same prompt indices for all methods and recomputes the statistics above, preserving the dependence between length, latency, NFE, and method. On the repeated-sample code tasks the resampling unit is the prompt cluster rather than the response: drawing a prompt contributes all of its repeats under all three methods, so the 20 HumanEval and 7 MBPP samples per prompt are not treated as independent examples. Paired confidence intervals accompany the numerical comparisons in Appendices F.7 and F.8. These intervals quantify variation across examples rather than across repeated executions, and they are individual intervals without multiple-comparison correction.

E.6 SETDLM DECODING POLICIES

Verification order. LTR visits unresolved positions in increasing order. Confidence order sorts stored, finite token probabilities decreasingly, breaks ties by textual position, and places unscored positions last in LTR order; the order is fixed before the call. The schedules use different stored probabilities for this ranking. Packed uses its preceding proposal, computed from committed context. Quadratic uses the proposal selected after its preceding verification. Linear uses the preceding verifier's stored predictions and probabilities, then generates a fresh draft in a separate call under the chosen order. The schedules can therefore choose different verification orders and produce different continuations.

Native baseline. Native SetDLM uses the released sliding-window decoder with argmax token selection and confidence threshold 0.9, and caches the keys and values of committed tokens. It commits confident parallel predictions without a speculative acceptance test and therefore defines a different decoding policy from the three verified schedules. Both runs cover all 1,319 GSM8K questions at batch one under the same prompt construction, canvas, and terminator-aware stopping convention as the speculative schedules, at $W = 8$ for smax16 and $W = 16$ for smax32.

Because it commits by threshold rather than verifying, native has no verification order, so it appears once in Table 8 as a reference point rather than as an LTR or confidence-order variant.

F Additional Experiments and Tables

This appendix collects input-size accounting, additional experiments, absolute measurements, and numerical values supporting the main figures. Measurement, scoring, and confidence-interval protocols are in Appendix E.

F.1 Logical queries and evaluated rows

For SetDLM, a recurrent update has u_t new proposal distributions and the logical verifier needs at most u_t same-position predictions. Packed needs only the first $\min(\kappa_t + 1, u_t)$ predictions in verification order. The kernels nevertheless evaluate the regions in Appendix D, so row accounting includes masked padding and repeated positions.

Table 2: **Execution cost per iteration with cached prompt context.** Evaluated uncached rows count token representations processed by the transformer, including duplicates and evaluated padding, summed across calls. Nemotron uses root-inclusive block size B. In SetDLM, $u_t \leq W$ positions remain unresolved in an active window of size W, and $p \in \{0, 1\}$ is a pending correction row; maximum row counts take $u_t = W$ and $p = 1$. Prompt rows are processed once and depend on prompt length.

		Evaluated uncached rows per iteration			
			SetDLM maximum		
Method	NFE/iteration	Nemotron	$W = 8$	$W = 16$	SetDLM general
Linear-SS	2	$2B$	25	49	$p + 3u_t$
Quadratic-SS	1	$B(B + 1)$	81	289	$p + 2u_t + u_t^2$
Packed-SS (Ours)	1	$2B$	25	49	$p + 3u_t$

Linear's draft call contains $p + u_t$ rows and its verification call $2u_t$, with the pending row cached between them, giving a peak of $2u_t$ and a total of $p + 3u_t$; Packed combines that same total into one forward pass. Quadratic contains $p + 2u_t$ pending, draft token, and verifier rows plus u_t proposal regions of u_t rows each, where each proposal conditions on one possible matching prefix rather than on the correction predicted simultaneously at a mask query. If h draft tokens match, the selected region is the one for h, capped at $u_t - 1$, and no retained draft is needed when all u_t positions are accepted; the selected proposal need not equal a fresh proposal conditioned on the entire updated context.

Nemotron uses next-token prediction, so its causal root also supplies verifier work; its physical peaks are B, $2B$, and $B(B+1)$ for Linear, Packed, and Quadratic, and Linear's two-call total is $2B$. These row counts describe input size. FLOPs and latency also depend on attention, context length, caching, and implementation.

F.2 Nemotron block size and execution cost

Table 3 compares complete decoder time on the first ten GSM8K test questions for Nemotron 8B. All runs use a total output limit of 512 new tokens, including thinking and answer tokens. Quadratic's mean time rises from 2.232 to 12.398 seconds as the root-inclusive block size B increases from 8 to 32, whereas Packed takes 2.798 to 2.333 seconds. The selected block sizes, $B = 32$ for Packed and $B = 8$ for Quadratic, evaluate similar-sized calls: 64 and 72 rows. Linear uses $B = 32$ to match Packed's speculative width $W = 31$. The probe is descriptive: numerical trajectory differences produce different output lengths and task scores.

Table 3: **Nemotron 8B GSM8K block-size probe.** Each row uses the same ten questions. Wall is mean total decoder time per response, including prompt processing; total NFE and output-token counts are also per-response means. EM retains the task outcome. Rows/call is the recurrent maximum. Tokens/s is calculated from the displayed means.

Method	B	Rows/call	Wall (s)	Total NFE	Tokens	Tokens/s	EM (%)
Linear-SS	8	8	2.824	98.2	267.9	94.9	100
Linear-SS	16	16	2.448	84.6	277.5	113.4	100
Linear-SS	32	32	2.237	77.4	272.8	121.9	100
Packed-SS	8	16	2.798	89.0	265.2	94.8	100
Packed-SS	16	32	2.315	73.3	285.9	123.5	100
Packed-SS	32	64	2.333	75.4	299.7	128.5	90
Quadratic-SS	8	72	2.232	64.9	272.9	122.3	100
Quadratic-SS	16	272	3.246	57.6	274.7	84.6	100
Quadratic-SS	32	1056	12.398	64.1	314.6	25.4	80

F.3 Confidence-ordered verification on SetDLM

We let verification follow confidence order, each iteration ranking unresolved positions by stored token confidence and verifying that order causally. The ranking rule is shared, but the schedules store different predictions and can therefore realize different orders (Appendix E.6). Packed-SS remains fastest at both windows (Table 8), reducing Linear-SS's wall time by 9.5% at $W = 8$ and 9.7% at $W = 16$ and its total NFE by 12.4% and 16.5%. Its throughput gains over Linear-SS are 10.2% and 12.8%, and at $W = 16$ it reaches $2.20\times$ Quadratic-SS's throughput; all eight SetDLM paired intervals favour Packed-SS (Table 10). Two effects should be kept apart, because only one of them is a cost. The three LTR schedules have identical per-example correctness labels within each checkpoint, so Packed-SS's LTR speedup incurs no observed accuracy cost. Changing the order can affect accuracy. Confidence ordering lowers Packed-SS's latency by a further 4.1% at $W = 8$ and 13.1% at $W = 16$ relative to LTR, but accuracy falls from 68.39% to 65.35% and from 66.57% to 59.06%; the second drop is large enough to give up the advantage over SetDLM's own threshold decoder, which scores 59.51% there. Adaptive position choices alter the greedy trajectory even in exact arithmetic, and single-step confidence is in any case a weak criterion for choosing what to commit (Chen et al., 2026b). We therefore report the ordering as a speed–accuracy trade-off of the decoding policy rather than a property of the schedule: Packed-SS gives the lowest latency in each regime, and LTR is the configuration we would run.

F.4 Proposal-only LoRA accounting

We apply the proposal-only low-rank adapter to Nemotron 8B on all GSM8K test questions, changing diffusion drafting while preserving the AR verifier. The base and LoRA measurements in this study use a total output cap of 512 new tokens, including thinking and answer tokens. This cap differs from the 1,024-token GSM8K setting in Table 6, so absolute timings are not directly comparable with Table 6. Packed-SS reduces NFE by 12.0% relative to Linear-SS but takes 1.749 rather than 1.698 seconds per example, a 3.0% slowdown (Table 4).

Table 4: **Nemotron 8B GSM8K with proposal-only LoRA (n = 1,319).** Means; model execution time and non-model decoder time sum to inclusive decoder wall time. Emitted length and throughput accompany latency, and accuracy is reported for comparison.

Method	Tokens	Total NFE	Model (s)	Non-model (s)	Wall (s)	Tokens/s	EM (%)
Linear-SS	272.2	55.04	1.561	0.137	**1.698**	**160.32**	85.90
Quadratic-SS	273.3	54.44	1.781	0.343	2.123	128.72	86.35
Packed-SS	274.2	**48.44**	1.570	0.179	1.749	156.78	85.82

In calls with the adapter enabled, the implementation computes an update for every input row but applies it only to proposal rows. Packed therefore computes 64 updates but applies only 32, whereas Linear places its 32 proposal rows in a separate call and disables the adapter for its 32-row verifier. Quadratic applies updates to 64 proposal rows but computes them for all 72 input rows.

Table 5: **Nemotron 8B GSM8K model execution time per iteration.** Linear's iteration contains both calls. "Selected" counts rows receiving adapter updates; "evaluated" counts rows for which those updates are computed.

Method	Query rows	Selected	Evaluated	Base ms	LoRA ms	Increase
Linear-SS	$32 + 32$	32	32	54.41	56.77	4.3%
Quadratic-SS	72	64	72	28.90	32.69	13.1%
Packed-SS	64	32	64	28.58	32.37	13.3%

With LoRA, mean model execution time is nearly equal for Packed-SS and Linear-SS (1.570 versus 1.561 seconds). After the first call, Packed-SS averages 47.44 iterations versus Linear-SS's 27.02, but its model time per iteration is 32.37 versus 56.77 milliseconds, so these differences almost cancel. Non-model decoder time adds 0.179 seconds for Packed-SS against 0.137 for Linear-SS, and the Packed-SS-minus-Linear-SS wall-time difference is 0.0513 seconds with paired 95% interval $[0.0243, 0.0788]$. Token throughput is also lower: Packed-SS emits 274.2 tokens on average against Linear-SS's 272.2, and its throughput ratio is 0.978 with paired interval $[0.969, 0.987]$. The base and LoRA results follow different model trajectories, so their percentage changes are diagnostic rather than a controlled ablation. Shared densely fine-tuned weights would remove this execution path, but whether they improve Packed-SS's latency also depends on the resulting acceptance profile and call cost, which we have not measured.

F.5 Model execution and non-model decoder time

To separate model evaluation from decoder overhead, we normalize each task's component times by its Linear-SS total and then average tasks equally within each model scale, so that long-output tasks do not dominate the decomposition. The decomposition covers GSM8K and MATH-500 only.

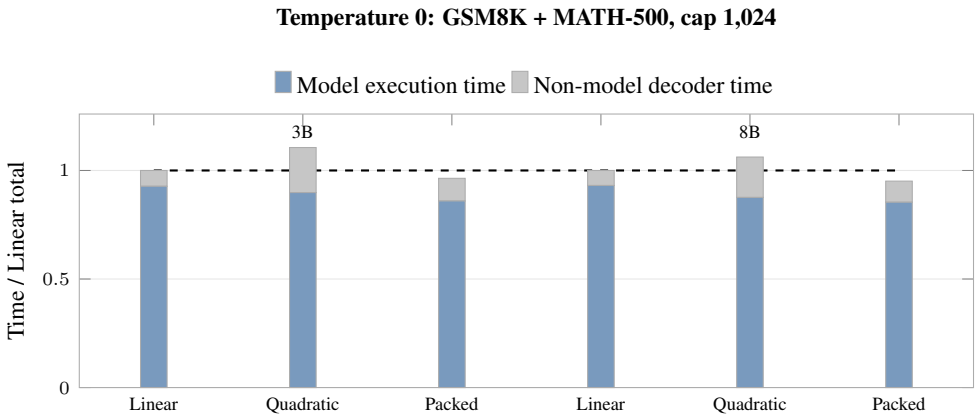


Figure 5: **Normalized Nemotron timing decomposition on GSM8K and MATH-500.** Greedy decoding with a 1,024-token cap. Each task's component times are divided by that task's Linear-SS total, then the two tasks are averaged with equal weight within each model scale. The dashed line marks Linear-SS's total. Non-model decoder time covers draft token construction, selection, and state updates. This decomposition is restricted to the two mathematics tasks, for which complete component timings are available at the selected cap.

Packed-SS cuts model execution time to 0.860 and 0.854 of Linear-SS's total at 3B and 8B while raising non-model decoder time by about 40%, for totals of 0.964 and 0.951. Quadratic-SS cuts model execution time comparably, to 0.898 and 0.876, but its non-model decoder time is 2.9 and 2.7 times Linear-SS's, so its totals exceed one at 1.106 and 1.062. The gap between call saving and wall-time saving follows: a packed call is larger than one Linear-SS call, and the extra state-update work it requires returns part of the saving.

F.6 ABSOLUTE MEASUREMENTS

Tables 6 to 8 report all 68 method–setting combinations used in the main efficiency figures. Tokens, wall time, and total NFE are means per response; tokens/s, tokens/NFE, and time/NFE use aggregate totals. All time measurements include decoder overhead. The first four metric columns follow the order in the figures, with absolute values reported in the tables. WritingBench has no accuracy score. Table 8 also includes the confidence-order SetDLM measurements.

Table 6: **Nemotron absolute measurements at temperature 0.** Bold marks the highest tokens/s within each model and task.

Task	Method	Tokens/s	Tokens/ NFE	Time/NFE (ms)	Accuracy (%)	Tokens	Wall (s)	NFE
Nemotron 3B								
MBPP	Linear-SS	173.46	3.757	21.658	74.87	65.8	0.379	17.50
	Quadratic-SS	164.38	4.288	26.089	74.34	65.9	0.401	15.36
	Packed-SS	**181.10**	4.312	23.812	74.60	67.0	0.370	15.53
HumanEval	Linear-SS	237.45	5.158	21.722	79.27	90.5	0.381	17.55
	Quadratic-SS	211.48	5.536	26.177	79.88	91.1	0.431	16.46
	Packed-SS	**249.75**	5.971	23.908	78.05	90.0	0.361	15.08
GSM8K	Linear-SS	190.05	4.142	21.795	88.40	177.1	0.932	42.76
	Quadratic-SS	180.93	4.750	26.253	88.25	178.9	0.989	37.66
	Packed-SS	**201.55**	4.812	23.873	88.10	179.1	0.888	37.22
MATH-500	Linear-SS	222.40	4.928	22.160	55.40	489.0	2.199	99.23
	Quadratic-SS	193.10	5.229	27.081	58.20	488.5	2.530	93.41
	Packed-SS	**230.08**	5.586	24.280	58.40	492.9	2.142	88.24
WritingBench	Linear-SS	84.56	1.864	22.038	—	502.1	5.938	269.43
	Quadratic-SS	89.39	2.468	27.613	—	502.0	5.616	203.39
	Packed-SS	**92.44**	2.242	24.253	—	502.9	5.441	224.34
Nemotron 8B								
MBPP	Linear-SS	151.96	4.421	29.095	85.45	56.9	0.375	12.87
	Quadratic-SS	134.86	4.631	34.335	85.19	59.3	0.440	12.81
	Packed-SS	**158.00**	5.041	31.902	85.19	58.3	0.369	11.56
HumanEval	Linear-SS	195.71	5.695	29.099	80.49	94.2	0.481	16.54
	Quadratic-SS	155.30	5.335	34.353	82.32	91.5	0.589	17.15
	Packed-SS	**205.46**	6.546	31.859	81.71	92.8	0.452	14.18
GSM8K	Linear-SS	127.39	3.744	29.389	90.52	313.2	2.459	83.66
	Quadratic-SS	122.93	4.319	35.134	91.21	313.9	2.553	72.68
	Packed-SS	**134.29**	4.300	32.021	90.67	318.0	2.368	73.94
MATH-500	Linear-SS	147.42	4.281	29.037	60.40	640.4	4.344	149.61
	Quadratic-SS	133.78	4.698	35.116	60.20	630.7	4.715	134.26
	Packed-SS	**155.61**	4.922	31.632	61.80	634.7	4.079	128.94
WritingBench	Linear-SS	66.12	1.898	28.701	—	501.7	7.587	264.35
	Quadratic-SS	67.59	2.476	36.633	—	501.5	7.420	202.55
	Packed-SS	**71.46**	2.255	31.559	—	502.6	7.033	222.87

Table 7: **Nemotron absolute measurements at temperature 1.** Bold marks the highest tokens/s within each model and task.

Task	Method	Tokens/s	Tokens/ NFE	Time/NFE (ms)	Accuracy (%)	Tokens	Wall (s)	NFE
Nemotron 3B								
MBPP	Linear-SS	120.38	2.508	20.834	68.14	102.7	0.853	40.95
	Quadratic-SS	**129.01**	3.328	25.800	69.69	108.7	0.843	32.67
	Packed-SS	125.06	2.933	23.454	68.78	101.8	0.814	34.72
HumanEval	Linear-SS	169.58	3.526	20.792	66.89	96.9	0.571	27.49
	Quadratic-SS	170.63	4.348	25.484	67.29	95.6	0.561	22.00
	Packed-SS	**173.20**	4.048	23.374	68.35	97.7	0.564	24.12
GSM8K	Linear-SS	137.38	2.977	21.670	84.15	197.7	1.439	66.41
	Quadratic-SS	**140.65**	3.776	26.843	84.84	199.2	1.416	52.76
	Packed-SS	140.07	3.428	24.471	83.47	198.4	1.416	57.88
MATH-500	Linear-SS	150.13	3.266	21.753	53.80	518.2	3.451	158.66
	Quadratic-SS	149.05	4.102	27.521	52.00	515.6	3.459	125.68
	Packed-SS	**155.49**	3.841	24.701	52.80	523.9	3.369	136.41
WritingBench	Linear-SS	74.96	1.575	21.012	—	498.8	6.654	316.68
	Quadratic-SS	**78.20**	2.141	27.382	—	497.0	6.355	232.09
	Packed-SS	78.04	1.890	24.222	—	500.1	6.409	264.58
Nemotron 8B								
MBPP	Linear-SS	84.45	2.316	27.426	76.76	129.9	1.538	56.08
	Quadratic-SS	**91.22**	3.062	33.572	74.98	130.4	1.429	42.58
	Packed-SS	89.56	2.752	30.722	75.36	133.6	1.491	48.54
HumanEval	Linear-SS	107.78	2.944	27.316	70.70	120.3	1.116	40.87
	Quadratic-SS	111.48	3.714	33.317	71.28	120.6	1.082	32.48
	Packed-SS	**114.17**	3.490	30.567	71.34	118.7	1.040	34.01
GSM8K	Linear-SS	85.66	2.341	27.324	87.34	351.0	4.098	149.97
	Quadratic-SS	**91.86**	3.109	33.840	86.50	348.1	3.790	111.99
	Packed-SS	91.08	2.794	30.678	85.29	345.0	3.787	123.46
MATH-500	Linear-SS	90.91	2.576	28.338	54.20	657.3	7.230	255.14
	Quadratic-SS	**95.73**	3.391	35.417	55.20	658.5	6.879	194.22
	Packed-SS	95.47	3.047	31.919	54.20	664.6	6.962	218.10
WritingBench	Linear-SS	52.63	1.433	27.225	—	500.9	9.518	349.59
	Quadratic-SS	53.34	1.940	36.370	—	496.5	9.308	255.94
	Packed-SS	**55.39**	1.741	31.424	—	498.8	9.006	286.58

Table 8: **SetDLM measurements on GSM8K.** All 1,319 test questions. Top: LTR verification and native decoding; bottom: confidence-ordered verification. Tokens, inclusive wall time, and total NFE are means per response; rates use aggregate totals. Bold marks the highest tokens/s within each window and order.

Window	Method	Tokens/s	Tokens/ NFE	Time/NFE (ms)	Accuracy (%)	Tokens	Wall (s)	NFE
LTR verification and native decoding								
$W = 8$	Linear-SS	94.93	2.610	27.496	68.39	257.2	2.709	98.53
	Quadratic-SS	86.25	3.414	39.583	68.39	257.1	2.981	75.31
	Packed-SS	**100.60**	2.842	28.247	68.39	257.2	2.556	90.49
	Native	77.21	2.119	27.442	59.67	252.4	3.269	119.11
$W = 16$	Linear-SS	98.85	2.792	28.243	66.57	259.9	2.629	93.09
	Quadratic-SS	41.60	4.074	97.934	66.57	260.4	6.260	63.92
	Packed-SS	**100.30**	3.030	30.209	66.57	259.9	2.591	85.77
	Native	80.59	2.220	27.552	59.51	253.6	3.147	114.21
Confidence-ordered verification								
$W = 8$	Linear-SS	93.93	2.586	27.535	65.05	254.5	2.709	98.39
	Quadratic-SS	90.83	3.470	38.200	65.58	254.6	2.803	73.38
	Packed-SS	**103.52**	2.943	28.431	65.35	253.8	2.451	86.21
$W = 16$	Linear-SS	99.06	2.794	28.206	53.07	247.0	2.493	88.40
	Quadratic-SS	50.90	4.462	87.664	60.80	252.1	4.954	56.51
	Packed-SS	**111.76**	3.408	30.491	59.06	251.5	2.251	73.81

The training horizons are $S_{\max} = 16$ and 32 for $W = 8$ and 16, respectively. For $W = 8$ and 16, respectively, maximum recurrent rows/call are $(16, 32)$ for Linear-SS, $(81, 289)$ for Quadratic-SS, $(25, 49)$ for Packed-SS, and $(16, 28)$ for Native. Native commits at confidence threshold 0.9 and has no verification order (Appendix E.6).

F.7 Accuracy and output trajectories

Under greedy decoding, the three Nemotron schedules target the same AR decoder, but their different matrix shapes can change BF16 rounding and reduction order. Near-tied logits can then produce different argmax tokens and continuations, causing differences in measured accuracy despite exact-arithmetic equivalence. We report task scores to check these numerical effects. At the 1,024-token mathematics cap a diverted continuation can additionally cross or avoid the cap, which widens the spread these diagnostics can show: the largest greedy gap is 3B MATH-500, where Packed-SS scores 3.0 points above Linear-SS with paired interval $[0.8, 5.2]$ on 500 questions. We report it for completeness and do not claim the schedule improves accuracy. In the FP32 SetDLM runs, all three LTR methods have identical per-example correctness labels within each checkpoint, which does not imply bitwise-identical continuations: output sequences differ among the three methods on 5 of 1,319 smax16 examples and 13 of 1,319 smax32 examples without changing their correctness labels. At temperature 1, measured accuracies also vary because the schedules generate different random samples; distributional equivalence does not require identical sampled continuations. Confidence order differs in kind, since the permutation is computed from each schedule's current speculative state and different schedules can realize different position orders even in exact arithmetic, making its task score an end-to-end adaptive-decoder outcome.

Responses that still reach 1,024 tokens are retained, not discarded; Table 6 therefore includes cap-limited responses, 10.4% and 30.6% of MATH-500 under Packed-SS at 3B and 8B and under 6% of GSM8K at either scale. WritingBench reaches its cap in 94% of generations and is reported as a generation-speed study.

Table 9 gives all 48 paired Nemotron accuracy comparisons. Differences are in percentage points; brackets give 95% confidence intervals.

For SetDLM, the three LTR methods have zero paired accuracy differences. Each exceeds Native by 8.72 percentage points at $W = 8$, with 95% confidence interval $[6.37, 11.07]$, and by 7.05 points at $W = 16$, with interval $[4.78, 9.33]$.

Table 9: **Paired Nemotron accuracy differences.**

Setting	Packed-SS − Linear-SS	Packed-SS − Quadratic-SS	Linear-SS − Quadratic-SS
Temperature 0			
3B MBPP	-0.26 [-1.32, +0.79]	+0.26 [-0.79, +1.32]	+0.53 [-0.53, +1.59]
3B HumanEval	-1.22 [-3.66, +1.22]	-1.83 [-4.27, +0.00]	-0.61 [-1.83, +0.00]
3B GSM8K	-0.30 [-1.14, +0.53]	-0.15 [-0.99, +0.68]	+0.15 [-0.68, +0.99]
3B MATH-500	+3.00 [+0.80, +5.20]	+0.20 [-1.60, +2.00]	-2.80 [-4.60, -1.00]
8B MBPP	-0.26 [-1.59, +0.79]	+0.00 [-0.79, +0.79]	+0.26 [-0.53, +1.06]
8B HumanEval	+1.22 [-1.22, +3.66]	-0.61 [-3.05, +1.22]	-1.83 [-4.27, +0.00]
8B GSM8K	+0.15 [-0.83, +1.14]	-0.53 [-1.59, +0.53]	-0.68 [-1.82, +0.38]
8B MATH-500	+1.40 [-0.60, +3.60]	+1.60 [-0.40, +3.80]	+0.20 [-2.00, +2.40]
Temperature 1			
3B MBPP	+0.64 [-0.98, +2.31]	-0.91 [-2.61, +0.83]	-1.55 [-3.17, +0.08]
3B HumanEval	+1.46 [+0.06, +2.90]	+1.07 [-0.34, +2.47]	-0.40 [-1.74, +0.92]
3B GSM8K	-0.68 [-2.65, +1.36]	-1.36 [-3.34, +0.61]	-0.68 [-2.73, +1.36]
3B MATH-500	-1.00 [-4.60, +2.60]	+0.80 [-2.80, +4.40]	+1.80 [-1.80, +5.40]
8B MBPP	-1.40 [-2.95, +0.15]	+0.38 [-1.32, +2.12]	+1.78 [+0.11, +3.44]
8B HumanEval	+0.64 [-0.91, +2.20]	+0.06 [-1.49, +1.59]	-0.58 [-2.13, +0.95]
8B GSM8K	-2.05 [-4.02, -0.08]	-1.21 [-3.26, +0.83]	+0.83 [-1.06, +2.73]
8B MATH-500	+0.00 [-4.00, +4.00]	-1.00 [-4.60, +2.80]	-1.00 [-4.80, +2.80]

F.8 Paired efficiency comparisons

Tables 10 to 12 report Packed-SS minus baseline wall time and Packed-SS/baseline throughput, with 95% paired intervals. Negative differences and ratios above one favor Packed-SS.

Table 10: **Paired SetDLM efficiency comparisons.**

Setting	Baseline	Δ wall (s) [95% CI]	Throughput ratio [95% CI]
$W = 8$ LTR	Linear-SS	-0.1529 [-0.1588, -0.1470]	1.060 [1.058, 1.062]
	Quadratic-SS	-0.4248 [-0.4377, -0.4124]	1.166 [1.164, 1.169]
$W = 8$ Confidence	Linear-SS	-0.2581 [-0.2953, -0.2217]	1.102 [1.098, 1.107]
	Quadratic-SS	-0.3521 [-0.3836, -0.3198]	1.140 [1.136, 1.143]
$W = 16$ LTR	Linear-SS	-0.0382 [-0.0432, -0.0330]	1.015 [1.013, 1.017]
	Quadratic-SS	-3.6686 [-3.7869, -3.5590]	2.411 [2.399, 2.424]
$W = 16$ Confidence	Linear-SS	-0.2428 [-0.2757, -0.2091]	1.128 [1.123, 1.134]
	Quadratic-SS	-2.7033 [-2.7766, -2.6335]	2.196 [2.185, 2.207]

Table 11: **Paired Nemotron efficiency comparisons at temperature 0.**

Setting	Baseline	Δ wall (s) [95% CI]	Throughput ratio [95% CI]
3B MBPP	Linear-SS	-0.0092 [-0.0268, +0.0123]	1.044 [1.007, 1.075]
	Quadratic-SS	-0.0308 [-0.0563, -0.0027]	1.102 [1.079, 1.127]
3B HumanEval	Linear-SS	-0.0207 [-0.0341, -0.0100]	1.052 [1.033, 1.072]
	Quadratic-SS	-0.0703 [-0.0886, -0.0539]	1.181 [1.149, 1.213]
3B GSM8K	Linear-SS	-0.0434 [-0.0605, -0.0243]	1.061 [1.049, 1.071]
	Quadratic-SS	-0.1003 [-0.1187, -0.0798]	1.114 [1.103, 1.125]
3B MATH-500	Linear-SS	-0.0565 [-0.1145, +0.0040]	1.035 [1.021, 1.048]
	Quadratic-SS	-0.3872 [-0.4531, -0.3221]	1.192 [1.174, 1.209]
3B WritingBench	Linear-SS	-0.4969 [-0.5335, -0.4604]	1.093 [1.087, 1.099]
	Quadratic-SS	-0.1753 [-0.2068, -0.1435]	1.034 [1.029, 1.040]
8B MBPP	Linear-SS	-0.0058 [-0.0222, +0.0156]	1.040 [1.024, 1.056]
	Quadratic-SS	-0.0711 [-0.0921, -0.0520]	1.172 [1.144, 1.200]
8B HumanEval	Linear-SS	-0.0294 [-0.0509, -0.0077]	1.050 [1.022, 1.078]
	Quadratic-SS	-0.1372 [-0.1634, -0.1121]	1.323 [1.274, 1.376]
8B GSM8K	Linear-SS	-0.0908 [-0.1499, -0.0318]	1.054 [1.042, 1.066]
	Quadratic-SS	-0.1856 [-0.2433, -0.1277]	1.092 [1.082, 1.104]
8B MATH-500	Linear-SS	-0.2656 [-0.3560, -0.1778]	1.056 [1.040, 1.071]
	Quadratic-SS	-0.6361 [-0.7332, -0.5367]	1.163 [1.145, 1.181]
8B WritingBench	Linear-SS	-0.5538 [-0.6101, -0.4958]	1.081 [1.073, 1.089]
	Quadratic-SS	-0.3866 [-0.4433, -0.3300]	1.057 [1.050, 1.065]

Table 12: **Paired Nemotron efficiency comparisons at temperature 1.**

Setting	Baseline	Δ wall (s) [95% CI]	Throughput ratio [95% CI]
3B MBPP	Linear-SS	-0.0389 [-0.0678, -0.0107]	1.039 [1.026, 1.052]
	Quadratic-SS	-0.0287 [-0.0832, +0.0248]	0.969 [0.956, 0.984]
3B HumanEval	Linear-SS	-0.0076 [-0.0250, +0.0093]	1.021 [1.005, 1.038]
	Quadratic-SS	+0.0033 [-0.0121, +0.0189]	1.015 [0.997, 1.034]
3B GSM8K	Linear-SS	-0.0227 [-0.0832, +0.0387]	1.020 [1.001, 1.037]
	Quadratic-SS	+0.0000 [-0.0640, +0.0652]	0.996 [0.979, 1.013]
3B MATH-500	Linear-SS	-0.0820 [-0.2440, +0.0791]	1.036 [1.012, 1.060]
	Quadratic-SS	-0.0895 [-0.2481, +0.0661]	1.043 [1.021, 1.066]
3B WritingBench	Linear-SS	-0.2455 [-0.3181, -0.1699]	1.041 [1.034, 1.048]
	Quadratic-SS	+0.0536 [-0.0187, +0.1283]	0.998 [0.990, 1.006]
8B MBPP	Linear-SS	-0.0467 [-0.0970, +0.0024]	1.061 [1.045, 1.076]
	Quadratic-SS	+0.0619 [+0.0072, +0.1164]	0.982 [0.969, 0.995]
8B HumanEval	Linear-SS	-0.0767 [-0.1104, -0.0443]	1.059 [1.042, 1.077]
	Quadratic-SS	-0.0423 [-0.0786, -0.0059]	1.024 [1.007, 1.042]
8B GSM8K	Linear-SS	-0.3104 [-0.4833, -0.1363]	1.063 [1.046, 1.081]
	Quadratic-SS	-0.0025 [-0.1731, +0.1618]	0.992 [0.976, 1.007]
8B MATH-500	Linear-SS	-0.2686 [-0.5450, +0.0071]	1.050 [1.029, 1.073]
	Quadratic-SS	+0.0831 [-0.1779, +0.3394]	0.997 [0.978, 1.018]
8B WritingBench	Linear-SS	-0.5121 [-0.6381, -0.3880]	1.052 [1.043, 1.061]
	Quadratic-SS	-0.3029 [-0.4490, -0.1578]	1.039 [1.030, 1.047]

All ten paired throughput intervals against Linear-SS lie above one (Table 12), supporting a throughput advantage in the measured settings. Against the Quadratic-SS extension, three intervals lie above one, two lie below it, and five include it; the last group is inconclusive rather than evidence of equivalence.

Reference schedules. At temperature 0, Quadratic-SS has lower throughput than Linear-SS on all eight code and mathematics settings but higher throughput on WritingBench; at temperature 1, it has higher throughput on seven of the eight code and mathematics settings. Shorter accepted prefixes require more iterations, so eliminating a second call can outweigh the cost of its fixed 72-row pass (Appendix F.5). On SetDLM, Linear-SS and Packed-SS are faster than Native at both windows; Quadratic-SS is faster only at $W = 8$ (Table 8).