LATTICESPEC: FROM DRAFT CHAINS TO VERIFICATION LATTICES FOR EFFICIENT DIFFUSION LANGUAGE MODEL DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Masked diffusion language models can predict updates at many positions in parallel, but turning these predictions into faster decoding requires deciding which updates to accept and in what order. Speculative decoding addresses this challenge by proposing updates in advance and verifying them against a target decoder. Yet a draft can contain the right updates in the wrong order: acceptance along a fixed draft chain stops at an ordering mismatch, leaving useful candidates unaccepted. Allowing alternative orders can recover this progress, but their verification costs can erase the benefit. We introduce LATTICESPEC, a training-free, plug-and-play extension that selectively relaxes local draft ordering constraints within a small verification budget. It retains the existing drafting procedure and organizes alternative orders into a lattice, sharing verification states whenever different paths commit the same updates. Across 5 tasks and 6 model configurations from the LLaDA, Dream, and SDAR families, LATTICESPEC improves end-to-end throughput over the corresponding SSD baselines by 5.7% on average. Separate analyses show that local relaxation recovers updates rejected by chain verification and that state sharing reduces the cost of checking the same alternatives. These results show that recovering more value from existing drafts through limited order flexibility can yield practical decoding gains.

1 INTRODUCTION

Masked diffusion language models (dLLMs) generate text by iteratively filling masked positions, offering flexibility in both the positions selected and the number of updates considered at each step (Nie et al., 2025). Absorbing-state discrete denoising provides a foundation for masked text generation (Austin et al., 2021a), with score- and cross-entropy-based formulations supporting language modeling (Lou et al., 2024; Shi et al., 2024). Cache reuse and adaptive parallel updates provide complementary acceleration routes (Ma et al., 2025; Israel et al., 2025). We study how to exploit dLLMs' parallel predictions for more efficient decoding while preserving the output of a target decoder that selects 1 position–token update at each successive decoding step.

Speculative decoding proposes updates in advance and verifies candidate states in parallel, allowing several sequential steps to be completed within 1 verification round (Leviathan et al., 2023). For autoregressive targets, drafting can use layer skipping (Zhang et al., 2024), learned features (Li et al., 2024a), or Jacobi trajectories (Fu et al., 2024). These choices change drafting, whereas we study verification order for a fixed candidate set. Methods such as SSD (Gao et al., 2025) and FreeDave (Wu & Zhang, 2025) use the model's own predictions for drafting. Their verification must account for both the proposed updates and the order in which the target selects them.

Correct candidate updates need not arrive in the correct draft order. Committing an update changes the decoding state and can change remaining candidates' relative confidence, so draft order need not match subsequent target choices. Suppose a drafter proposes position–token updates A, B, C, D at distinct masked positions in that order, while the target selects A, C, B, D. After accepting A, verification restricted to the draft chain cannot continue draft acceptance along C, although all 4 candidates occur on the target decoding path. SSD reports this out-of-order behavior and studies limited alternative checks through mix-order verification. We separate candidate availability from

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

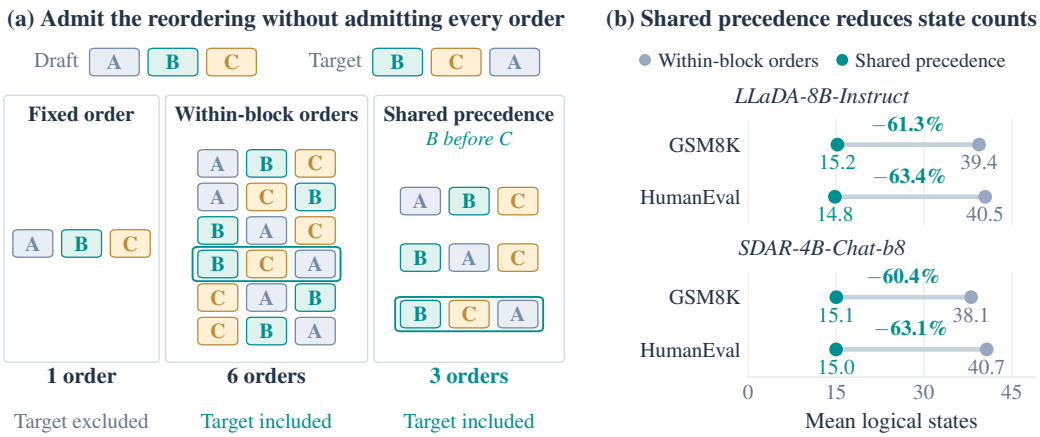


Figure 1: Admitting a target reordering need not require allowing every order. **(a)** Both the draft and target place B before C. Keeping this relation admits the target while excluding other permutations. **(b)** For observed draft–target pairs with identical updates but different orders, shared precedence reduces mean logical state counts from 38–41 (all within-block orders) to about 15. These counts measure representation size using observed target continuations (Appendix C.1).

acceptance under a fixed order by comparing the draft chain with the longest target prefix supported by the same candidates. Our shared-root analysis on LLaDA-8B-Instruct and GSM8K measures progress recovered by order relaxation from the same draft (Section 4.3).

Recovering this progress requires controlling the cost of additional verification states. Dynamic and budgeted verification trees likewise balance additional candidates against verification work (Li et al., 2024b; Chen et al., 2024). Graph-based methods already provide multiple paths and shared states: Spiffy calibrates draft graphs offline and prunes them using current predictions (Agrawal et al., 2026), while PSD uses confidence-ranked candidates on an offline-calibrated directed acyclic graph (DAG) whose topology remains fixed during inference (Sun et al., 2026). The design question is which local draft-order constraints to relax and how many distinct states those alternatives require. Our analysis of observed reorderings supports retaining shared precedence to limit the distinct states introduced by local relaxation (Figure 1). Representing the remaining alternatives as separate chains can still repeat states reached by committing the same updates in different orders, increasing verification work without admitting additional candidate orders. These observations motivate controlling the distinct states introduced by order relaxation and sharing them across admissible paths.

We introduce LatticeSpec, a training-free, plug-and-play verification extension that retains the drafting procedure. It controls the cost of local order relaxation by budgeting additional states and sharing them across update sequences that reach the same committed subset. In the 4-update example above, relaxing the order between B and C adds the state $\{A, C\}$, allowing the target path to rejoin the draft path at $\{A, B, C\}$ and continue with D without duplicating the shared suffix (Figure 2). The legal committed subsets form the order ideals of the relaxed partial order, and their lattice represents each distinct verification state once. This count makes the structural cost of relaxation explicit: block selection limits the additional nodes relative to the draft chain to a prescribed budget. Target predictions at the resulting incomplete states guide verification along exactly matching edges. If decoding is incomplete when traversal stops, we commit 1 additional update selected by the target decoder. This traversal preserves the target output under the execution assumptions in Appendix A.3, independently of how the local order was constructed.

Our oracle study shows that selecting local order relaxations using reference continuations can improve throughput, revealing the potential of admitting useful alternative orders. Realizing this potential online requires selecting relaxations whose recovered progress offsets their selection and verification costs. We therefore explore a low-budget strategy for this trade-off: calibrated construction with gain-based selection that relaxes at most 1 adjacent pair per round. By recovering progress through a small set of shared verification states, this strategy delivers end-to-end gains over the corresponding SSD baselines across the LLaDA, Dream, and SDAR families. Simpler construction and selection rules also achieve competitive throughput in the evaluated single-block settings, suggesting

that this practical benefit extends to alternative choices within the same low-budget design. These findings motivate LATTICESPEC's focus on useful local order flexibility with controlled verification work. Our contributions are as follows:

- **Structural insights.** We quantify the gap between candidate availability and fixed-order acceptance, showing how local order relaxation recovers target updates from the same draft.
- **Budgeted order relaxation.** We select local order relaxations using their additional node counts, admitting alternative candidate orders within a fixed budget beyond the draft chain.
- **Shared-state verification.** We use an order-ideal lattice to share states across prefixes committing the same candidate subset, avoiding repeated target evaluation across alternative orders.
- **Practical acceleration.** We improve end-to-end throughput over SSD baselines by 2.9–8.7% across 5 tasks and 6 configurations from 3 model families with our low-budget design.

2 RELATED WORK

Masked diffusion models and efficient decoding. Masked diffusion models generate text through iterative unmasking (Sahoo et al., 2024; Nie et al., 2025; Ye et al., 2025), while block diffusion combines autoregressive generation across blocks with denoising within them (Arriola et al., 2025; Cheng et al., 2026). Inference accelerators exploit cache reuse and confidence-based parallel updates (Wu et al., 2026), or adaptive parallel sampling (Israel et al., 2025). LATTICESPEC studies acceleration of a deterministic decoder committing 1 position–token update per step.

Speculative decoding for dLLMs. Parallel proposal verification supports greedy decoding (Stern et al., 2018) and distribution-preserving autoregressive speculation (Leviathan et al., 2023; Chen et al., 2023). DiffuSpec uses diffusion proposals for an autoregressive target (Li et al., 2026). For diffusion targets, verification must match positions as well as tokens. SSD studies self-drafting and mix-order checks to recover out-of-order updates (Gao et al., 2025), while FreeDave reuses verification outputs for subsequent drafts (Wu & Zhang, 2025). S2D2 couples diffusion proposals with local autoregressive verification in a hybrid decoding trajectory (Han et al., 2026). LATTICESPEC retains drafting and changes which candidate update orders are verified.

Verification trees and graphs. SpecInfer and Medusa verify candidate trees (Miao et al., 2024; Cai et al., 2024), while EAGLE-2 and Sequoia study dynamic trees and budgeted expansion (Li et al., 2024b; Chen et al., 2024). Trajectory-level speculation explores confidence-stratified denoising trees (Pan et al., 2026). Spiffy calibrates shared draft graphs offline and prunes them online (Agrawal et al., 2026), whereas PSD uses confidence-ranked subsets with a fixed inference topology (Sun et al., 2026). Both support multiple acceptance paths and shared states. LATTICESPEC selects local precedence relaxations within contiguous draft blocks. Order ideals identify shared committed subsets, and the node count beyond the chain limits verification expansion.

3 METHOD

LATTICESPEC is a training-free, plug-and-play extension of speculative verification that retains the drafting procedure. It relaxes local ordering constraints under an additional-node budget and shares states across alternative orders (Section 3.2 and Section 3.3), as shown in Figure 2 and Algorithm 1. Target updates computed for these states guide exact-match traversal. Construction and selection determine available alternatives, using offline statistics in the calibrated configuration (Section 3.4).

3.1 PROBLEM SETUP

A decoding state s comprises a partially masked sequence and the information needed to determine its next update, including the remaining masked positions (Nie et al., 2025). Absorbing diffusion admits a conditional-distribution view of prediction from the unmasked tokens (Ou et al., 2025). Let F denote the *target decoder*, which deterministically selects 1 update from each incomplete state s. The target update $F(s)$ is a position–token pair specifying the masked position to fill and its token. Successively applying these updates defines the *target decoding path*.

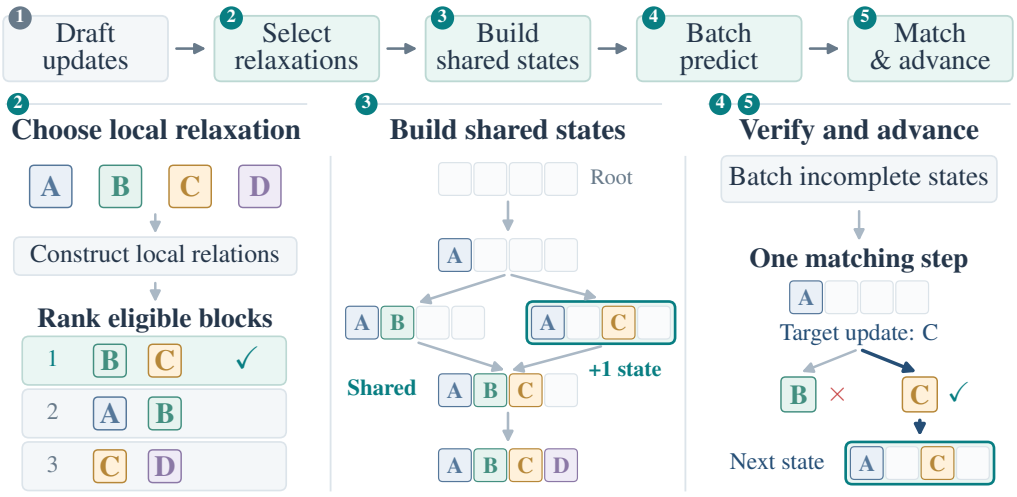


Figure 2: LATTICESPEC admits alternative update orders through budgeted state sharing. **Left:** three eligible blocks are ranked by estimated gain per additional state, with B/C selected under a budget of 1 additional state. The ranking is illustrative. **Middle:** the added state $\{A, C\}$ allows the reordered path to rejoin at $\{A, B, C\}$. **Right:** at state $\{A\}$, the target selects C rather than B, and traversal follows the matching edge to $\{A, C\}$. Letters denote position–token updates, and state icons show only the 4 candidate positions.

Given a root state s, the starting state of a speculative decoding iteration, a drafter D proposes a set $A = \{a_1, \ldots, a_N\}$ of $N \geq 1$ candidate position–token updates with confidence scores $c_1, \ldots, c_N$. The candidates occupy distinct positions and are indexed in decreasing order of confidence, $c_1 \geq \cdots \geq c_N$, with fixed tie-breaking. Under *chain verification*, this order imposes the precedence constraints $a_1 \prec \cdots \prec a_N$. Acceptance stops if the target selects a later candidate rather than the next candidate in this chain, even when both occur along the target decoding path. A single candidate has no ordering constraint to relax.

3.2 LATTICE REPRESENTATION

To continue verification when the target selects candidates in a different order, LATTICESPEC relaxes the draft chain into a partial order $P = (A, \prec_P)$. Here $a_i \prec_P a_j$ requires a_i to be committed before a_j. For example, relaxing the order of a_2 and a_3 (labeled B and C in Figure 2) admits both a_1, a_2, a_3, a_4 and a_1, a_3, a_2, a_4, allowing verification to follow either order. These paths differ after the 2nd update but commit the same subset $\{a_1, a_2, a_3\}$ after the 3rd, motivating a representation that shares their subsequent verification.

To identify where such paths can share a state, we represent each prefix by its committed subset. A subset reachable under P must contain every predecessor of each candidate it includes, making it an *order ideal* $I \subseteq A$ (Stanley, 2011). The lattice has 1 node per ideal in $\mathcal{I}(P)$, the set of all ideals of P. Prefixes committing the same ideal produce the same partially masked sequence, despite different update orders. Sharing a verification state also requires these prefixes to agree on any other information the target decoder uses to select its next update (Appendix A.3). Under this condition, each ideal I corresponds to a shared verification state s_I, obtained by applying its updates to the root state s, with $s_\emptyset = s$. A single lattice node can therefore represent all prefixes committing I, with 1 target evaluation supplying their next update whenever masked positions remain.

From each such node, the outgoing edges specify which candidate updates are permitted next under P. An edge connects I to $I \cup \{a\}$ when $a \notin I$ and all its predecessors are in I. In the example, the new node $\{a_1, a_3\}$ lets verification continue when the target selects a_3 after a_1. If the target selects a_2 next, verification returns to the existing state for $\{a_1, a_2, a_3\}$ and continues along the shared suffix. Supporting this alternative therefore requires only 1 additional node. The lattice thus allows verification to follow orders admitted by P while representing each committed subset once.

Acceptance under order relaxation. For a fixed root and draft, removing precedence constraints preserves the chain's admissible paths, so exact-match traversal cannot accept fewer draft updates under the execution assumptions in Appendix A.3. Relaxation can therefore recover drafted updates that the chain rejects solely because of their order, but it cannot supply a target update missing from the draft. If such an update is selected, draft acceptance must stop even when later target updates are available among the candidates (Appendix A.4). Supporting alternative paths to recover available updates can introduce additional verification states, whose cost may outweigh the benefit of accepting more updates. The practical objective is therefore to recover useful progress within limited verification work, motivating the node budget below.

3.3 Controlling Verification Work

Relaxing precedence constraints can expand the lattice despite state sharing. We limit this expansion through an additional-node budget B_K beyond the $N + 1$ chain nodes.

An order-construction rule specifies which draft precedence constraints to retain. A candidate block G consists of consecutive updates in the confidence-ordered draft, whose sequence positions need not be consecutive. To limit the extent of each local relaxation, candidate blocks contain at most $w_{\max}$ candidates. They must also lie within 1 connected component of the incomparability graph, which connects pairs left incomparable by the construction rule.

We denote the construction relation restricted to G by P_G. Selecting G replaces a chain segment with $|G| + 1$ nodes by a local lattice with $|\mathcal{I}(P_G)|$ nodes. Its additional cost is therefore

$$\Delta K_G = |\mathcal{I}(P_G)| - |G| - 1 \leq 2^{|G|} - |G| - 1. \tag{1}$$

The upper bound is attained when all candidates in G are incomparable. For example, removing the precedence constraint between 2 adjacent draft candidates adds 1 intermediate state, increasing their local lattice from 3 nodes to 4 and giving $\Delta K_G = 1$ (Figure 2, middle). Blocks with $\Delta K_G = 0$ add no alternative prefix and are excluded.

The rules in Section 3.4 select a pairwise-disjoint set $\mathcal{G}$ of candidate blocks. Preserving draft order between these blocks makes their additional node counts additive. The resulting $N + 1 + \sum_{G\in\mathcal{G}} \Delta K_G$ lattice nodes include any complete states requiring no target evaluation. We require the total additional node count to satisfy $\sum_{G\in\mathcal{G}} \Delta K_G \leq B_K$. The final partial order P retains P_G within each selected block and draft precedence for every other pair, followed by transitive closure.

Together, the block constraints and node budget allow LatticeSpec to admit alternative candidate orders while limiting the resulting lattice to at most $N + 1 + B_K$ nodes.

Verification. Following SSD, we reuse the target update at the root when available and batch-evaluate the remaining incomplete lattice states. This provides one target-selected position–token update per incomplete state. We then traverse from the root along edges that exactly match their source state's computed update until no edge matches or decoding is complete. If traversal stops at an incomplete state, we commit its computed target update as the *bonus token*. This procedure preserves 1-token-per-step target decoding under the state-construction and pointwise-evaluation assumptions in Appendix A.3, independently of the order-construction rule.

3.4 Constructing Local Orders

The framework leaves 2 choices open: which draft precedence constraints to retain and which candidate blocks G to select under the node budget. We instantiate them with 2 construction rules and the gain-based selection rule used in the main evaluation. These rules determine which additional states are evaluated within the same lattice representation and node budget.

Fixed-threshold construction. For $\delta \geq 0$ and $i < j$, this relation retains $a_i \prec a_j$ when $c_i - c_j \geq \delta$ and otherwise leaves the pair incomparable. It relaxes constraints between candidates with similar confidence scores. Block constraints and the node budget limit expansion (Section 3.3).

Calibrated rank intervals. Candidate a_i has *draft rank* i in the confidence-ordered list. Its *target rank* is the index of the update matching both its position and token among the next N target updates.

An offline radius q_{α,z_i} is fitted from matched rank errors, where α is the miscoverage level and z_i groups candidates by draft length, draft rank, and confidence gap. We fit the radius using split-conformal quantiles of sequence-level maximum matched rank errors (Vovk et al., 2005; Lei et al., 2018). The clipped rank interval is given by:

$$J_i = [\ell_i, u_i] = \big[\max(1, i - q_{\alpha,z_i}), \min(N, i + q_{\alpha,z_i})\big]. \tag{2}$$

The interval-derived relation retains

$$a_i \prec_{\text{int}} a_j \quad \Longleftrightarrow \quad u_i < \ell_j, \tag{3}$$

leaving candidates with overlapping intervals incomparable. Appendix A.5 gives the fitting procedure and its fixed-pool marginal coverage guarantee under exchangeability (Angelopoulos & Bates, 2023). This statistical guarantee does not imply that the budgeted structure admits the target prefix.

Gain-based selection. For each candidate block G, we estimate the mean increase in accepted target-prefix length from relaxing that block alone relative to the chain using offline calibration states (Appendix B.3). This gain measures accepted draft updates, excluding the bonus token, and serves as a proxy for progress rather than end-to-end throughput. This estimate $\widehat{g}_G$ remains fixed throughout selection and is used to rank blocks with positive additional cost once per iteration by decreasing $\widehat{g}_G/\Delta K_G$. The illustrative ranking in Figure 2 (left) places the B/C block first. Ties are resolved by preferring fewer additional nodes followed by earlier draft position and finally shorter block width. We scan the resulting order to select blocks with positive gain that do not overlap previously selected blocks while keeping the cumulative additional node count within B_K. The scan stops after considering all blocks or selecting the configured maximum number of blocks.

Our main experiments use calibrated rank intervals and gain-based selection with $N = 4$, $w_{\max} = 2$, at most 1 selected block, and $B_K = 1$. We also compare both constructions with fixed-position selection, which relaxes a calibration-selected adjacent pair when it is eligible under the construction rule and otherwise retains the draft chain (Section 4.5). These choices leave the lattice representation and exact-match traversal unchanged.

4 Experiments

We evaluate end-to-end efficiency, recovered draft progress, and the cost of shared-state verification, followed by construction, selection, and budget comparisons.

4.1 Experimental Setup

Models and tasks. The runtime evaluation covers LLaDA-8B-Instruct, Dream-7B, SDAR-4B-Chat, and SDAR-8B-Chat on GSM8K and the MATH-500 subset of MATH for mathematics (Cobbe et al., 2021; Hendrycks et al., 2021; Lightman et al., 2024), HumanEval and MBPP for code (Chen et al., 2021; Austin et al., 2021b), and MT-Bench for multi-turn chat (Zheng et al., 2023). Unless otherwise specified, experiments use FP16 on NVIDIA A100 80GB GPUs. The evaluation is implemented in PyTorch (Paszke et al., 2019), using Transformers for checkpoint and tokenizer loading (Wolf et al., 2020). Protocols for the mechanism diagnostics, runtime comparisons, and structure extensions are provided in Appendix C, Appendix D, and Appendix E, respectively. All experiments use 1 active request per GPU, with speculative states batched within that request. We implement the deterministic reference rule with greedy decoding, selecting the highest-probability token at the highest-confidence eligible position at each step. Appendix B provides further implementation details, including the execution setup, calibration procedure, and measurement protocol.

Baselines. We compare 1-token target decoding, SSD, and LatticeSpec. Here SSD denotes the greedy chain configuration used for its main results, and LatticeSpec uses the same drafting procedure. For SDAR, we adapt SSD to its native blockwise decoding procedure while preserving SDAR's existing attention and cache mechanisms. The SDAR timing runs use FlashAttention (Dao et al., 2022). For a fair comparison, LatticeSpec retains the baseline's candidate generation and cache policy to isolate the effects of order selection and verification structure. In our main experiments, all LatticeSpec configurations use calibrated rank intervals and gain-based selection with draft length $N = 4$, maximum block width $w_{\max} = 2$, at most 1 selected block, and additional-node

Table 1: End-to-end generation efficiency. TPS reports tokens/s, with speedup over the panel's Vanilla baseline (1-token target decoding) in parentheses. Avg reports the arithmetic mean of the 5 task-wise speedups, with subscripts indicating the percentage increase in this average over SSD. NFE reports model forward calls as a percentage of the panel's Vanilla baseline.

	GSM8K		MATH		HumanEval		MBPP		MT-Bench		Avg
Method	TPS↑	NFE (%)↓	TPS↑	NFE (%)↓	TPS↑	NFE (%)↓	TPS↑	NFE (%)↓	TPS↑	NFE (%)↓	TPS Speedup↑
LLaDA-8B-Instruct (generation block length = 16)											
Vanilla	40.3 (1.00×)	100.0	39.6 (1.00×)	100.0	39.0 (1.00×)	100.0	38.6 (1.00×)	100.0	37.9 (1.00×)	100.0	1.00×
SSD	59.2 (1.47×)	41.2	55.8 (1.41×)	43.4	54.0 (1.39×)	41.6	54.9 (1.42×)	41.4	45.7 (1.21×)	47.0	1.38×
LATTICESPEC	63.7 (1.58×)	38.6	60.6 (1.53×)	40.2	57.8 (1.48×)	38.7	58.8 (1.52×)	38.6	49.3 (1.30×)	43.3	1.48×$_{+7.6\%}$
Dream-7B (generation block length = 16)											
Vanilla	49.6 (1.00×)	100.0	50.0 (1.00×)	100.0	47.3 (1.00×)	100.0	47.1 (1.00×)	100.0	47.4 (1.00×)	100.0	1.00×
SSD	74.8 (1.51×)	42.7	73.5 (1.47×)	44.2	72.0 (1.52×)	41.9	66.9 (1.42×)	45.0	66.3 (1.40×)	46.0	1.46×
LATTICESPEC	77.9 (1.57×)	39.7	77.2 (1.54×)	40.6	74.1 (1.57×)	39.2	69.7 (1.48×)	41.6	69.3 (1.46×)	42.3	1.52×$_{+4.2\%}$
SDAR-4B-Chat (generation block length = 4)											
Vanilla	13.0 (1.00×)	100.0	13.2 (1.00×)	100.0	13.0 (1.00×)	100.0	12.5 (1.00×)	100.0	13.1 (1.00×)	100.0	1.00×
SSD	18.1 (1.39×)	72.1	18.4 (1.39×)	72.3	17.6 (1.35×)	74.3	16.2 (1.30×)	77.0	17.1 (1.30×)	76.7	1.34×
LATTICESPEC	19.1 (1.47×)	68.2	19.4 (1.47×)	68.3	18.7 (1.43×)	69.8	17.5 (1.40×)	71.2	18.1 (1.38×)	72.1	1.43×$_{+6.4\%}$
SDAR-4B-Chat-b8 (generation block length = 8)											
Vanilla	14.3 (1.00×)	100.0	14.7 (1.00×)	100.0	14.1 (1.00×)	100.0	13.3 (1.00×)	100.0	14.5 (1.00×)	100.0	1.00×
SSD	22.1 (1.54×)	64.6	22.9 (1.56×)	63.8	21.7 (1.54×)	64.7	19.4 (1.45×)	68.6	21.0 (1.45×)	68.5	1.51×
LATTICESPEC	23.2 (1.61×)	61.7	23.9 (1.63×)	61.2	22.8 (1.61×)	61.8	20.6 (1.54×)	64.6	22.0 (1.53×)	65.2	1.58×$_{+5.0\%}$
SDAR-8B-Chat (generation block length = 4)											
Vanilla	13.0 (1.00×)	100.0	13.1 (1.00×)	100.0	12.9 (1.00×)	100.0	12.4 (1.00×)	100.0	13.0 (1.00×)	100.0	1.00×
SSD	18.1 (1.39×)	71.6	18.7 (1.42×)	70.3	17.2 (1.33×)	75.1	15.9 (1.29×)	77.6	16.9 (1.30×)	77.0	1.35×
LATTICESPEC	19.2 (1.47×)	67.8	19.5 (1.49×)	67.1	18.4 (1.43×)	69.9	17.1 (1.38×)	72.1	17.9 (1.38×)	72.4	1.43×$_{+6.2\%}$
SDAR-8B-Chat-b8 (generation block length = 8)											
Vanilla	14.2 (1.00×)	100.0	14.5 (1.00×)	100.0	13.9 (1.00×)	100.0	13.1 (1.00×)	100.0	14.3 (1.00×)	100.0	1.00×
SSD	21.6 (1.52×)	65.4	22.4 (1.55×)	64.4	21.3 (1.53×)	65.2	18.8 (1.44×)	69.3	20.7 (1.45×)	68.5	1.50×
LATTICESPEC	22.6 (1.59×)	62.4	23.4 (1.62×)	61.5	22.3 (1.60×)	61.9	20.0 (1.53×)	65.1	21.7 (1.52×)	65.1	1.57×$_{+5.0\%}$

budget $B_K = 1$. We use a held-out calibration set to estimate rank and gain statistics. Generation block lengths are 16 for LLaDA and Dream and 4 or 8 for SDAR, distinct from the candidate-block width $w_{\max}$. SSD draft length is selected by calibration latency for LLaDA and Dream and fixed to 4 for SDAR. Appendix B.1 and Appendix B.3 provide the data splits used for calibration and evaluation, along with the draft lengths and other settings for each model configuration.

Metrics. The primary metric is generated tokens per second (TPS) for 1 active request per GPU, with speedup relative to 1-token target decoding. Both are computed from end-to-end wall-clock time, including prefill and all online drafting, selection, state construction, verification, traversal, and cache updates. Appendix B.4 details the timing boundaries and metric definitions.

4.2 END-TO-END EFFICIENCY

Table 1 evaluates the calibrated single-block configuration across 6 model configurations and 5 tasks. LATTICESPEC improves end-to-end throughput over the corresponding SSD baseline in all 30 configuration–task pairs, with an average gain of 5.7% and gains ranging from 2.9% to 8.7%. These improvements build on the acceleration already provided by SSD and cover mathematical reasoning, code generation, and multi-turn conversation. The throughput measurements account for the online costs of selecting local relaxations and verifying the resulting states.

The gains occur on LLaDA and Dream as well as the SDAR models with generation block lengths of 4 and 8. For example, throughput on MATH increases from 55.78 to 60.64 tokens/s for LLaDA-8B-Instruct and from 22.45 to 23.44 tokens/s for SDAR-8B-Chat-b8. With each comparison retaining its baseline's candidate generation and cache policy, these results demonstrate the benefit of local order relaxation and lattice verification within the existing decoding procedure. Across these configurations and tasks, throughput is 1.30–1.63× that of the corresponding 1-token target decoder.

Table 2: Mean accepted draft length.

Verification structure	Accepted length
Draft chain	1.8203
Calibrated single-block selection	2.0964
Single-pair oracle	2.2682
Larger-family oracle, $B_K = 2$	2.5234
Larger-family oracle, $B_K = 4$	2.5651
Candidate bound	2.6302

Table 3: Shared-root verification cost.

	GSM8K	HumanEval
Evaluated states		
Lattice	1993	2044
Unmerged tree	2852	2942
Verification time		
Tree/lattice ratio	1.2767	1.3000

LATTICESPEC also reduces forward-call counts in every comparison. In the 2 MATH examples, NFE decreases from 43.4% to 40.2% of Vanilla for LLaDA-8B-Instruct and from 64.4% to 61.5% for SDAR-8B-Chat-b8. Together with the measured throughput gains, these results show that relaxing at most 1 adjacent pair per round can reduce model calls while providing a net benefit after online costs. The following analyses examine recovered draft progress and verification cost separately on LLaDA to clarify how local order relaxation and state sharing contribute to this trade-off.

4.3 RECOVERING PROGRESS THROUGH ORDER RELAXATION

Local order relaxation recovers drafted target updates rejected by chain verification. We measure recovery at 384 roots sampled from native SSD trajectories on LLaDA-8B-Instruct and GSM8K without selecting for order mismatches or gain. At each root, we use the same draft and serial target continuation to compute chain acceptance and the *candidate bound*. This bound is the length of the longest consecutive target prefix contained in the draft, up to the draft length or termination. The gap between chain acceptance and this bound measures the progress potentially recoverable by changing verification order while retaining the same candidates (Appendix A.4). As shown in Table 2, where all accepted draft lengths exclude the bonus token, the calibrated single-block configuration increases mean accepted draft length from 1.8203 to 2.0964 against a candidate bound of 2.6302, recovering 34.1% of the chain-to-bound gap.

At the same sampled roots, oracle selection uses the known target continuation to maximize accepted draft length within each structure family. Choosing among the chain and all adjacent pairs recovers 55.3% of the chain-to-bound gap, showing additional potential within the single-pair structure family. Allowing the larger family with $B_K = 2$ increases recovery to 86.8%. These comparisons identify opportunities in both the choice of local relaxation and the structure family available to represent it. Appendix C.2 provides the sampling protocol and structure definitions.

A separate structural diagnostic examines 8-update drafts on LLaDA-8B-Instruct and SDAR-4B-Chat-b8, each on GSM8K and HumanEval. We use the first 64 problems in each task's fixed evaluation order and retain sampled roots whose draft and target continuation contain identical position–token updates in different orders. Retaining the pairwise orderings shared by these 2 sequences yields 14.8–15.2 logical subset states on average, compared with 38.1–40.7 when all within-block orders are allowed (Figure 1). The corresponding reductions of 60.4–63.4% quantify representation compactness using observed target continuations, rather than online verification cost. These results support LATTICESPEC's design of selectively relaxing local order while retaining precedence constraints to limit the number of distinct verification states, complementing the accepted-progress analysis above. Appendix C.1 provides the sampling protocol and counts.

4.4 VERIFICATION COST OF SHARED STATES

Sharing repeated states reduces the verification cost of representing alternative orders in LATTICESPEC. To isolate this benefit, we compare the lattice with an unmerged tree that shares ordered prefixes but keeps different orders separate after they reach the same committed subset. We evaluate 384 roots per task from native LLaDA-8B-Instruct trajectories on GSM8K and HumanEval using the calibrated single-block configuration with $B_K = 1$. Each pair uses the same root state, cache, draft, and admissible orders. The comparison includes all sampled roots, including those without a selected block. Evaluated-state counts are summed over the 384 roots for each task, counting root states but excluding complete states. Appendix C.3 provides the full protocol.

As shown in Table 3, the lattice evaluates fewer states than the unmerged tree: 1,993 vs. 2,852 on GSM8K and 2,044 vs. 2,942 on HumanEval. The tree/lattice ratios of total verification time are 1.2767 on GSM8K and 1.3000 on HumanEval, corresponding to verification time reductions of 21.7% and 23.1%, respectively. These timings include representation and state construction, target evaluation, and traversal. These results show that merging prefixes with the same committed subset reduces verification cost in this shared-root comparison.

4.5 Local Order Ablations

We compare calibrated rank intervals and fixed-threshold construction with gain-based and fixed-position selection in a unified four-way comparison. The study covers LLaDA-8B-Instruct, Dream-7B, and the SDAR-4B and SDAR-8B block-8 configurations on GSM8K and HumanEval. All combinations retain the same drafting and lattice verification procedures, with $N = 4$, maximum block width 2, at most 1 selected block, and $B_K = 1$. Construction-specific gain statistics and the threshold and fixed position are determined on calibration data before evaluation (Appendix D.2).

The results identify competitive alternatives within LatticeSpec's low-budget design and show that the effect of selection depends on the construction rule. Replacing both rank intervals and gain-based selection with a threshold and a fixed pair position changes measured throughput by less than 1% in all 8 comparisons (Table 9). Fixed-threshold construction with gain-based selection has the highest throughput point estimate in each comparison, although some differences are small. Gain-based selection improves measured throughput over fixed-position selection by 0.2–1.0% under the threshold construction, while its effect under rank intervals is mixed.

The simpler combination also retains an end-to-end advantage over chain verification. In the separate shared-baseline comparisons in Appendix E.2, the threshold and fixed-position combination improves throughput over the fixed-$N = 4$ SSD chain by 4.4–7.9% across the same 8 settings.

4.6 Scaling Verification Structures

Larger verification structures can recover more updates without improving throughput, motivating LatticeSpec's node budget. In a fixed-family budget sweep on LLaDA-8B-Instruct, budget 1 has the highest measured throughput on GSM8K and HumanEval under the evaluated construction and selection rule (Appendix E.1). On GSM8K, raising the budget from 1 to 2 reduces forward calls but increases evaluated query tokens, lowering throughput from 64.005 to 62.806 tokens/s.

Better selection can nevertheless make larger families useful. In a separate LLaDA-8B-Instruct GSM8K experiment, the budget-4 oracle improves throughput by 11.69% over calibrated single-block selection and 5.07% over the single-pair oracle (Appendix E.3). These measurements include actual drafting and verification but exclude reference collection and oracle search. The comparison with the single-pair oracle identifies potential beyond selecting a better adjacent pair. Realizing this potential online requires useful relaxations whose benefits offset both selection and verification costs. The evaluated low-budget configuration of LatticeSpec already achieves this balance, recovering draft progress through a bounded set of shared verification states.

5 Conclusion

We present LatticeSpec, which connects local order relaxation to the distinct states needed to verify alternative draft orders. Its lattice shares committed subsets under a node budget, while exact-match traversal preserves target output under the stated execution assumptions. The evaluated low-budget configuration accelerates the corresponding SSD baselines across LLaDA, Dream, and SDAR models, while shared-root comparisons on LLaDA demonstrate reduced verification cost at matched accepted progress. Unified construction and selection comparisons identify competitive simple alternatives within this design. Selecting useful larger structures remains an open direction, since additional accepted progress does not by itself ensure higher throughput. Future work could explore adaptive node budgets, dynamic choices of lattice width and depth, and structure selection informed by hardware-specific verification costs. Our experiments show that local order relaxation with shared verification states can accelerate end-to-end decoding under a small node budget.

AI USE STATEMENT

The authors developed the manuscript structure and wrote the initial draft. Generative AI tools were used to polish the language and improve formatting. AI tools also assisted with the preliminary literature review, including identifying related work and clarifying key concepts. The authors designed the core experiments and implemented the code, while AI tools assisted with experiment monitoring and analysis of the results. All code was manually reviewed by the authors. The authors take responsibility for the final content of this work, including all AI-assisted content.

REPRODUCIBILITY STATEMENT

We provide the source code for LatticeSpec in the supplementary materials to facilitate reproducibility of our experimental results.

REFERENCES

Sudhanshu Agrawal, Risheek Garrepalli, Raghavv Goel, Christopher Lott, Fatih Porikli, and Mingu Lee. Structuring the future: Diffusion LLM speculative decoding via calibrated draft graphs. In *ICML 2026 Workshop on Foundations of Deep Generative Models: Understanding Memorization, Generalization, and Reasoning*, 2026.

Anastasios N Angelopoulos and Stephen Bates. Conformal prediction: A gentle introduction. *Foundations and Trends in Machine Learning*, 16(4):494–591, 2023.

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 Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 50726–50753, 2025.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993. Curran Associates, Inc., 2021a.

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.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

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.

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.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 129531–129563. Curran Associates, Inc., 2024. doi: 10.52202/079017-4116.

Shuang Cheng, Yihan Bian, Dawei Liu, Yuhua Jiang, Yihao Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, et al. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 22058–22075, 2026.

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.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359. Curran Associates, Inc., 2022. doi: 10.52202/068431-1189.

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*, ICML'24. JMLR.org, 2024.

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.

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.

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 *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 52870–52888. Curran Associates, Inc., 2025. doi: 10.52202/085713-1763.

Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J Tibshirani, and Larry Wasserman. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113 (523):1094–1111, 2018.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pp. 19274–19286. PMLR, 2023.

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. Diffuspec: Unlocking diffusion language models for speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 20896–20910, 2026.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 7421–7432, 2024b.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 149009–149033. Curran Associates, Inc., 2025. doi: 10.52202/085713-4984.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. 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*, pp. 932–949, 2024.

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. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 50608–50646. Curran Associates, Inc., 2025. doi: 10.52202/085713-1689.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *The Thirteenth International Conference on Learning Representations*, 2025.

Tianxiang Pan, Baitao Gong, Mo Guang, Hongwei Yong, Tianpeng Jiang, Yaqian Li, Zheng Cao, and Kaiwen Long. Trajectory-level speculative decoding for diffusion language models. *arXiv preprint arXiv:2608.27514*, 2026.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

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 A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 130136–130184. Curran Associates, Inc., 2024. doi: 10.52202/079017-4135.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 103131–103167. Curran Associates, Inc., 2024. doi: 10.52202/079017-3277.

Richard P. Stanley. *Enumerative Combinatorics*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2 edition, 2011.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Shengyin Sun, Yiming Li, Renxi Liu, Xinqi Li, Hui-Ling Zhen, Weizhe Lin, Chen Chen, Xianzhi Yu, Mingxuan Yuan, and Chen Ma. Psd: Pushing the pareto frontier of diffusion llms via parallel speculative decoding. *arXiv preprint arXiv:2605.15609*, 2026.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer, 2005.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. In *International Conference on Learning Representations*, volume 2026, pp. 57027–57051, 2026.

Shutong Wu and Jiawei Zhang. Free draft-and-verification: Toward lossless parallel decoding for diffusion large language models. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025.

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.

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)*, pp. 11263–11282, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623. Curran Associates, Inc., 2023. doi: 10.52202/075280-2020.

A METHOD DETAILS AND THEORETICAL PROPERTIES

A.1 DECODING PROCEDURE

Algorithm 1 specifies the online iteration summarized in Figure 2. It separates budgeted structure construction from target prediction and exact-match traversal.

Algorithm 1 One decoding iteration with LATTICESPEC

Require: Incomplete root s, drafter D, target decoder F, configured rules and limits (Sections 3.3 and 3.4)
Ensure: Updated decoding state

$\big((a_i, c_i)\big)_{i=1}^{N} \leftarrow D(s)$
Construct local precedence relations and select candidate blocks $\mathcal{G}$
Form the final P from selected block relations P_G and retained draft precedence
Enumerate $\mathcal{I}(P)$ and construct states s_I
Reuse the target update at the root when available and batch-evaluate the remaining incomplete s_I with F
Store the resulting target updates $\widehat{a}_I$
$I \leftarrow \emptyset$
while s_I is incomplete and an outgoing edge to I' exactly matches $\widehat{a}_I$ **do**
    $I \leftarrow I'$
end while
if s_I is incomplete **then**
    return s_I with $\widehat{a}_I$ applied ▷ bonus token
end if
return s_I

A.2 ENUMERATING ORDER IDEALS

The total orders consistent with P are its linear extensions, and each order ideal is the committed subset of a prefix of at least 1 such order. An unconstrained block of k candidates admits $k!$ orders and 2^k subsets. Representing each subset once avoids the duplicate states introduced by a tree that stores permutation prefixes separately.

To enumerate the ideals and their labeled edges, start from the empty subset and add each uncommitted candidate whose predecessors are already present. Deduplicate by committed subset, expanding each newly discovered subset once. Every generated subset is an ideal, and removing a maximal element from any nonempty ideal leaves an ideal, so induction on cardinality establishes completeness.

A.3 OUTPUT EQUIVALENCE OF LATTICE VERIFICATION

Let $T(s, a)$ denote application of update a to state s. Starting from a state $s^{(0)}$, 1-token-per-step decoding with the target model follows

$$a^{(t)} = F(s^{(t)}), \qquad s^{(t+1)} = T(s^{(t)}, a^{(t)}). \tag{4}$$

Consider 1 iteration of Algorithm 1 rooted at $s^{(0)}$. State construction must satisfy $s_\emptyset = s^{(0)}$ and $s_{I \cup \{a\}} = T(s_I, a)$ on every edge, including any model cache contents or position information that affect the next update. Assume that each reused or batch-computed update agrees pointwise with the target decoder, $\widehat{a}_I = F(s_I)$. The root lies on the target path. If a reached state s_I lies on that path, any traversed edge matches $\widehat{a}_I = F(s_I)$ and therefore reaches its next state. If no edge matches and decoding is incomplete, the bonus update applies the same target transition. Induction over traversed edges and successive iterations proves output equivalence, independently of the order-construction rule under these assumptions.

A.4 Acceptance under Order Relaxation

Fix a root s, a set A of N distinct-position candidates, and the target decoding path under the assumptions of Appendix A.3. Let $b_1, \ldots, b_m$, with $m \leq N$, be the next target updates up to N steps or termination, and define

$$H_A(s) = \max\{h \in \{0, \ldots, m\} : b_1, \ldots, b_h \in A\}. \tag{5}$$

For a fixed draft, $H_A(s)$ is the longest target prefix that can be accepted using these candidates. Let $L(P; s)$ denote the accepted draft length under partial order P, excluding the bonus token. Every accepted edge must match the next target update and belong to A, so acceptance stops no later than the first target update outside A. Thus $L(P; s) \leq H_A(s)$. When P is an antichain, every subset is an ideal, and verification can follow every update in this prefix. The full subset lattice therefore attains $H_A(s)$. The difference between $H_A(s)$ and chain acceptance measures the progress that alternative orders could recover.

If P_2 retains a subset of the precedence relations of P_1, every ideal and edge of P_1 remains available in P_2. Exact-match traversal can therefore accept at least the same prefix, giving $L(P_2; s) \geq L(P_1; s)$. In particular, the relations constructed in Section 3 retain only draft-order precedence, so $L(P; s) \geq L(P_{\text{chain}}; s)$. These statements concern accepted draft length at the same root, excluding the bonus token. Additional states may increase execution cost even when acceptance increases.

A.5 Rank Calibration

This appendix specifies the calibrated rank intervals used by the main experimental configuration in Section 3.4. They estimate the ranks of matched draft candidates along the target decoding path and guide which precedence relations to retain. Acceptance is determined by exact matching against target-selected updates.

For each independently sampled sequence, we collect the decoding states used for calibration, their ordered draft candidates, and the next N updates produced by the target decoder from each state. Each sequence forms 1 calibration example, so all states from the same generation contribute jointly. A draft candidate is matched when an update with the same position and token appears among these target updates. Its position in this update sequence defines its *target rank*, recording when it is selected along the target decoding path. For a matched candidate a_i with draft rank i and target rank $r_i^\star$, we define the rank error as

$$e_i = |r_i^\star - i|. \tag{6}$$

Candidates absent from these updates are excluded, so calibration models rank errors among matched candidates. For stratification, each candidate is grouped by the draft length, its draft rank, and a predefined bin of the smallest absolute difference between its score and another candidate score. We denote this stratum by z_i.

We pool sparse strata by dropping the confidence-gap bin, then draft rank, and finally draft length, selecting the narrowest pool with sufficient sequence support. Each sequence contributes its maximum matched rank error to each applicable pool, and the selected pool's split-conformal quantile gives the radius $q_{\alpha,z}$. The resulting radius table Q_α is computed offline for each drafter–target configuration, using an infinite radius when calibration support is insufficient. Algorithm 2 specifies the pooling hierarchy, quantile, and support requirement. For a pool fixed independently of the scores used for its guarantee, exchangeable calibration and test scores yield marginal coverage of at least $1 - \alpha$ for all matched candidates assigned to that pool in a test example containing such candidates (Vovk et al., 2005). This follows from the split-conformal quantile of the example-level maxima. It does not establish fine-stratum conditional coverage, simultaneous coverage across pools, coverage after data-dependent pool selection, or inclusion of the target prefix after budgeted block selection.

B Experimental Configuration and Measurement

B.1 Data Splits and Evaluation Scope

Calibration and evaluation have disjoint problem or conversation IDs, with all states from a problem and both turns of a conversation kept together. Table 4 summarizes the calibration and eval-

Algorithm 2 Offline calibration of rank intervals

Require: Calibration examples $\mathcal{D}_{\text{cal}}$, miscoverage α, fixed pooling hierarchy, minimum pool size m
Ensure: Radius table Q_α

initialize a multiset $\mathcal{E}_u$ for every calibration pool u
for all example $\xi \in \mathcal{D}_{\text{cal}}$ **do**
    compute matched errors e_i and their pool memberships across all states in ξ
    for all pool u with a matched candidate assigned to u in ξ **do**
        append the maximum e_i among those candidates to $\mathcal{E}_u$
    end for
end for
for all fine strata z **do**
    choose the narrowest pool $u(z)$ with $|\mathcal{E}_{u(z)}| \geq m$
    if no such pool exists **then**
        $Q_\alpha[z] \leftarrow +\infty$
    else
        $k \leftarrow \lceil (|\mathcal{E}_{u(z)}| + 1)(1 - \alpha) \rceil$
        $Q_\alpha[z] \leftarrow \mathcal{E}_{u(z),(k)}$, using $+\infty$ when $k = |\mathcal{E}_{u(z)}| + 1$
    end if
end for
return Q_α

Table 4: Calibration and evaluation data. HumanEval subsets follow dataset order, and MT-Bench counts complete 2-turn conversations.

Task	Calibration data	Evaluation data
GSM8K	64 training problems	131 test problems
MATH	64 training problems across 7 subjects in `hendrycks_math`	All 500 MATH-500 problems
HumanEval	Last 50 problems	First 100 problems
MBPP	64 training problems from the original full release	All 500 original test problems
MT-Bench	64 ShareGPT conversations	All 80 MT-Bench conversations

uation data for all evaluated configurations. The intervening 14 HumanEval problems are unused. Random sampling uses seed 42. The MT-Bench calibration conversations come from `ShareGPT_Vicuna_unfiltered`. We sample ShareGPT conversations with 2 user turns separated by an assistant response, retaining nonempty user turns with at most 4,000 characters in total. Before sampling, we deduplicate user-turn pairs and exclude conversations containing a user turn identical to an MT-Bench prompt. MT-Bench is a generation-efficiency workload, with a complete 2-turn conversation as the evaluation unit. Each method uses its own 1st response in the 2nd-turn history, with fresh prefill for each turn and both generation times summed.

B.2 Execution Setup

All configurations use seed 42, checkpoint chat templates, and 1 active request per GPU across 8 A100 80GB GPUs, with all methods for a request assigned to the same GPU. The reference rule fixes position selection, token selection, and tie breaking across methods. Table 5 lists generation limits, and Appendix B.3 specifies draft-length selection.

LLaDA and Dream. All methods use the checkpoint chat template without few-shot examples and a separately measured greedy 1-token reference with the same generation block length. Generation stops after a completed block contains EOS or EOT. Dream-7B uses Dream-v0-Instruct-7B with its ChatML template.

SDAR. The block-specific models differ across generation block lengths (Cheng et al., 2026), so these comparisons do not isolate block length on a fixed checkpoint. The 1-token reference selects

Table 5: Generation settings for the main runtime evaluation. Lengths are in tokens per answer. SDAR requests 256 tokens and rounds up to complete a block, whereas LLaDA and Dream use a 256-token cap.

Model	Block length	Generation limit
LLaDA-8B-Instruct	16	256 (cap)
Dream-7B	16	256 (cap)
SDAR-4B/8B-Chat	4	Up to 259
SDAR-4B/8B-Chat-b8	8	Up to 263

the argmax token and its original softmax probability at each masked position, then commits the highest-confidence position, breaking ties by the first index. Each block permits enough iterations to fill all masks before writing its cache. All 3 methods use the common inference implementation distributed with the other SDAR models in place of the default 4B inference code, leaving weights and tokenizers unchanged. Whole-block upward rounding and block-end EOS/EOT checks are retained, with allocation limits depending on prompt alignment. SSD drafts within the current block using root-confidence ranking and fixed ties. Verification reuses the root prediction and batches remaining nonterminal states with read-only views of the completed-prefix key–value cache, updating the cache only after committing the actual completed block. LATTICESPEC retains this candidate generation and cache policy while adding the selected lattice states. A complete verification state here has no masks left in the current generation block. If matched traversal fills the current block, no bonus token is added. Decoding the next block requires a new root evaluation after the completed block is written to the cache.

B.3 CALIBRATION STATISTICS AND CONFIGURATION SELECTION

Calibration statistics. Rank calibration follows Algorithm 2, with parameters and support units in Table 6. The same calibration split supplies rank and gain statistics, fitted separately for each model, task, and drafting configuration, including each SDAR generation-block configuration. States come from the drafting baseline for LLaDA and Dream and from native 1-token trajectories within the current block for SDAR. SDAR collection uses the minimum calibration-unit count in Table 6.

For each candidate block G, the recorded next N target updates define $C(\{G\})$, the longest prefix supported by relaxing only G, and the corresponding chain length $C(\emptyset)$, matching each update by position and token. The gain estimate is the empirical mean of $\max\{0, C(\{G\}) - C(\emptyset)\}$. These future updates are used only offline. Gain observations are grouped by construction variant, block start and width, draft length, and confidence-gap bin. The gain feature is the largest minus smallest confidence across the entire draft, using the bin boundaries in Table 6. Below the gain-support threshold, lookup drops block start and gap bin, then draft length, then block width, finally using all observations for that variant or returning 0 if none exist. These block-level means differ from the sequence-level maxima used for rank calibration. Offline collection and fitting are excluded from online timing.

Configuration selection. For SSD and LATTICESPEC on LLaDA and Dream, we compare 2 draft lengths, $N \in \{4, 6\}$, independently for each model and task. Each method selects the length with lower total end-to-end latency on the calibration requests. The selected length is the same for every task on these 2 models, as reported in Table 6. SDAR fixes its draft length before calibration.

The structural limits, α, the support thresholds, and the confidence-gap bins in Table 6 are fixed throughout parameter tuning. The selected settings and offline statistics are frozen before evaluation.

Offline calibration and tuning time. For the LLaDA-8B-Instruct FP16 SSD-based experiments, cumulative request time ranges from 2,095.7 to 5,797.2 seconds per task for calibration and from 938.7 to 2,624.8 seconds for configuration tuning across the 5 tasks. Calibration includes baseline runs and diagnostic collection for draft lengths 4 and 6. These values sum individual request times across GPU shards and include collection probes, but exclude model loading, warmup, and fitting. Summed request time differs from wall-clock time when requests execute in parallel.

Table 6: Parameters for the main runtime evaluation. Separate diagnostics and ablations specify their own settings. Support thresholds use the distinct observation units shown below.

Setting	Value	Scope
Draft length N	4	SSD and LATTICESPEC
Maximum candidate-block width $w_{\max}$	2	LATTICESPEC
Maximum selected blocks per round	1	LATTICESPEC
Additional-node budget B_K	1	LATTICESPEC
Rank-calibration α	0.3	Rank statistics
Confidence-gap bin boundaries	0.005, 0.02, 0.1	Rank and gain statistics
Minimum rank-pool support	30 sequences	Before rank fallback
Minimum gain-lookup support	30 block observations	Before gain fallback
Minimum calibration units per task	30 complete requests or conversations	SDAR collection
Physical verification-state cap	16	SDAR only

B.4 MEASUREMENT PROTOCOL

LLaDA and Dream. For LLaDA and Dream, the primary speed measurements use uninstrumented runs. We measure end-to-end request time after warmup, including prompt construction and tokenization, online setup, prefill, generation, and benchmark result extraction, with device synchronization before and after generation. Model loading, offline calibration, and result-file writes are excluded. Generated-token counts exclude prompts, the first EOS or EOT, and all subsequent tokens. TPS is total generated tokens divided by total request time, and speedup is the baseline's total time divided by the method's total time on the same requests. Total request time is the sum of individual request times across GPU shards. NFE counts model forward calls, including cache-refresh calls, with each batched verification call counted once regardless of the number of states it evaluates. Table 1 reports the ratio of the method's total NFE to Vanilla's total NFE, expressed as a percentage. For MT-Bench, these totals include both turns.

SDAR. End-to-end timing includes prompt templating, tokenization, input transfer, prefill, generation, raw-token extraction, and stop processing. All online drafting, selection, state construction, verification, traversal, and cache updates are included. Model loading, seed setup, offline fitting, text decoding, and result-file writes are excluded. The LLaDA and Dream timing boundary includes text decoding during benchmark result extraction, so absolute TPS across backends has different measurement boundaries. Within each panel, all methods use the same timing boundary. TPS, speedup, token counting, and NFE normalization follow the definitions above. Lightweight counters are enabled for all methods during timing.

C SUPPORTING MECHANISM ANALYSIS

C.1 STRUCTURAL DIAGNOSTIC FOR THE MOTIVATION FIGURE

Figure 1 uses a separate structural diagnostic on LLaDA-8B-Instruct and SDAR-4B-Chat-b8, each evaluated on GSM8K and HumanEval. For each task, we take the first 64 problems in the fixed evaluation order. LLaDA uses a 16-position active window, and SDAR uses an 8-position generation block. The configured draft length is 8 in both cases. Runs use FP16, batch size 1, seed 42, and a 256-token generation cap. We sample at most 24 roots per problem at uniformly spaced native decoding rounds, with the sampling schedule fixed before collecting serial target continuations from cloned root states. Instrumented and uninstrumented native outputs agree on all 64 problems in each setting, and no sampled root is excluded for missing target labels.

We first retain roots with an actual draft length of 8, then condition on the draft and the next 8 serial target updates containing the same position–token update set in different orders. Table 7 reports the counts before and after this conditioning. For each retained root, we partition the draft and target sequences at every boundary where their prefix update sets coincide. The comparison allowing all within-block orders removes all precedence constraints within each resulting contiguous block while preserving order between blocks. The shared-precedence partial order retains a pairwise

Table 7: Sampling counts and mean logical states for Figure 1. Full depth counts roots with 8 draft updates, same set additionally requires identical draft and target update sets, and reordered additionally requires different orders. Both state-count means use only the reordered roots. All orders preserves order between blocks.

Model	Task	Full depth	Same set	Reordered	All orders	Shared
LLaDA-8B-Instruct	GSM8K	750	212	173	39.41	15.25
	HumanEval	720	203	152	40.45	14.82
SDAR-4B-Chat-b8	GSM8K	643	420	332	38.05	15.05
	HumanEval	632	523	434	40.71	15.01

relation exactly when the draft and the same-root target continuation agree on it. We enumerate all predecessor-closed subsets, including the empty and full subsets, to obtain logical state counts. The means weight retained roots equally, and each percentage reduction is 1 minus the ratio of the shared-precedence mean to the all-within-block mean. These counts describe representation size using observed target continuations, not target rows executed, forward calls, or online latency. The diagnostic does not verify every additional path admitted by the resulting partial order as a target continuation.

C.2 Observed Acceptance under Order Relaxation

We sample 384 verification roots uniformly within 32 native generation trajectories on GSM8K, without selecting for order mismatches or gain. This diagnostic uses LLaDA-8B-Instruct, draft length configured to 4 (actual drafts can be shorter), and the generation block lengths and cache policies specified in Appendix B. At each root, we use a shared draft and a serial target continuation to compare chain acceptance, the selected partial order, and the candidate-supported prefix bound $H_A(s)$ from Appendix A.4. Accepted draft length excludes the bonus token. The calibrated configuration uses $\alpha = 0.3$, maximum block width 2, at most 1 block, and $B_K = 1$. Its rank and gain statistics are fitted for this study, separately from the complete-generation representation comparison in Table 8.

Table 2 separates candidate availability, structure coverage, and online selection. Here, *oracle selection* uses advance knowledge of the target decoder's serial continuation to choose a verification structure. The single-pair oracle maximizes accepted draft length over the chain and all adjacent pairs. The larger family additionally includes 2 disjoint pairs and partial orders on a contiguous 3-update block whose relations follow the draft order, subject to the stated additional-node budget. Order between blocks is retained. These choices use reference continuations to maximize acceptance within each structure family at fixed roots.

At the first chain rejection, the target selects another available draft update at 127 roots, a different token at a draft position at 24 roots, and a position outside the draft at 85 roots. The remaining 148 roots have no chain rejection. All counts use the full 384-root population. Selecting another available update indicates an order mismatch, but does not imply that a single adjacent-pair relaxation can recover it.

C.3 Verification Cost at Shared Roots

We compare the lattice and unmerged tree on GSM8K and HumanEval using LLaDA-8B-Instruct, the same backends, and draft length 4. For each task, we sample 384 roots from 32 native trajectories and apply the calibrated single-block selection with $\alpha = 0.3$ and $B_K = 1$. Each pair uses identical root state, cache, draft, and admissible orders, with independent cache copies and alternating execution order. The comparison includes all sampled roots, including those without a selected block. For this SSD implementation, the evaluated states include the root and exclude complete states, giving at most 6 evaluated lattice states per root when $N = 4$ and $B_K = 1$. The SDAR implementation instead reuses the root prediction, so its verification batch contains at most 5 states under the same node budget (Appendix B.2). Verification time includes representation construction, materialization, target evaluation, and traversal, excluding the protective cache copies.

Table 8: Complete-generation representation comparison on LLaDA-8B-Instruct and GSM8K. Time ratios divide variant time by calibrated lattice time, with paired request-bootstrap 95% intervals. Intermediate trajectories can differ.

Representation	TPS	NFE/request	States/round	Time ratio [95% interval]
Calibrated lattice	64.277	93.774	5.476	1.000000
Unmerged tree	52.074	93.790	7.854	1.234335 [1.229113, 1.239568]

Both representations agree on the accepted updates and additional target update at every sampled root, and on target actions at their shared subsets. In this comparison, the lattice accepts the same updates as the unmerged tree in less verification time.

D Supplementary Runtime Analysis

D.1 Complete-Generation Unmerged-Tree Comparison

This supplementary comparison evaluates complete executions with lattice and unmerged-tree representations on LLaDA-8B-Instruct and GSM8K. It is separate from the shared-root comparison in Appendix C.3 and the unified construction and selection study below. Both representations use $N = 4$, maximum block width 2, at most 1 selected block, and $B_K = 1$. For this comparison, calibration selects $\alpha = 0.2$ from $\{0.1, 0.2, 0.3\}$. The unmerged tree inherits its rank and gain statistics without separate tuning. Gain lookup uses the confidence range within each candidate block, whereas the main evaluation and the unified study below use the range over the entire draft.

We compute paired 95% percentile intervals by resampling problems 10,000 times with seed 42. The intervals describe variation across requests. NFE includes cache-refresh calls, and states per round counts evaluated target states during verification.

Intermediate trajectories. The unmerged tree shares ordered prefixes but keeps different prefixes separate after they reach the same committed subset. At a fixed root with the same draft and selected blocks, it represents the same admissible orders using the same backend, cache policy, and terminal-state treatment. The lattice and unmerged-tree executions follow different intermediate trajectories on 7 of the 124 evaluated problems. This comparison measures complete executions with differences in both representation and intermediate trajectory, rather than strictly matched per-round progress. States/round includes all verification rounds, including those without a relaxed block.

D.2 Construction and Selection Rules

We evaluate all 4 combinations of rank-interval or fixed-threshold construction and gain-based or fixed-position selection. The 4 model configurations in Table 9 use FP16, greedy decoding, draft length 4, maximum block width 2, at most 1 selected block, and $B_K = 1$. LLaDA and Dream use generation blocks of 16, and the SDAR configurations retain their block-8 models and native block completion rules. The fixed splits contain 64 calibration and 131 test problems for GSM8K, and 50 calibration and 100 test problems for HumanEval.

Rank intervals use $\alpha = 0.3$ and the main evaluation's pooling rules. We fit gain statistics separately for each construction using the same calibration trajectories, the whole-draft confidence range, and a minimum support of 30 block observations, following the pooling hierarchy in Appendix B.3. We select a threshold and fixed pair position jointly from 9 calibration trials: $\delta \in \{0.005, 0.02, 0.1\}$ and the 3 adjacent-pair starting positions in a 4-update draft. Selection minimizes total end-to-end time across all 9 calibration trials, with ties resolved in the listed threshold order and then by earlier position. Both threshold variants share the selected δ, and both fixed-position variants share the selected position. Thus the rank-based fixed-position variant uses the position selected with the threshold construction, rather than a separately optimized rank-based position. All 8 settings select $\delta = 0.1$; LLaDA and Dream select position 1, and SDAR selects position 2. The fixed-position rule relaxes that pair only when it exists and is eligible, without applying a gain filter or substituting another pair. All parameters are frozen before testing.

Table 9: Unified construction and selection comparison. Entries report TPS for the 4 combinations. Rank denotes rank intervals, and threshold denotes fixed-threshold construction. Gain and fixed denote gain-based and fixed-position selection. These measurements are independent of the main evaluation.

		Rank		Threshold	
Model	Task	Gain	Fixed	Gain	Fixed
LLaDA-8B	GSM8K	63.614	63.530	64.357	63.805
LLaDA-8B	HumanEval	58.107	58.109	58.661	58.076
Dream-7B	GSM8K	77.630	77.771	78.795	78.348
Dream-7B	HumanEval	73.764	74.059	74.890	74.471
SDAR-4B-b8	GSM8K	22.306	22.305	22.344	22.244
SDAR-4B-b8	HumanEval	21.567	21.581	21.641	21.568
SDAR-8B-b8	GSM8K	21.899	21.896	21.955	21.870
SDAR-8B-b8	HumanEval	21.821	21.788	21.822	21.775

Table 10: Budget sensitivity with the order construction and selection rule fixed. Each budget cell reports TPS and mean additional logical nodes per verification round in parentheses. The baseline is SSD. All rows use LLaDA-8B-Instruct.

Task	Baseline TPS	$B_K = 0$	$B_K = 1$	$B_K = 2$	$B_K = 4$
GSM8K	58.871	58.731 (0.00)	64.005 (0.90)	62.806 (1.67)	62.814 (1.67)
HumanEval	53.842	53.774 (0.00)	58.214 (0.92)	56.877 (1.71)	56.484 (1.77)

Methods run on the same GPU for each request, with method order rotated across the 8 request shards after warming up every method. The timing boundaries follow Appendix B.4 and include online selection and verification work.

Threshold construction with gain-based selection has the highest TPS point estimate in all 8 comparisons, with gains over rank intervals and gain-based selection ranging from 0.008% to 1.527%. The threshold and fixed-position combination differs from rank intervals and gain-based selection by -0.276% to $+0.959\%$. Gain-based selection changes throughput relative to fixed-position selection by -0.398% to $+0.147\%$ under rank intervals and by $+0.217\%$ to $+1.008\%$ under threshold construction. These observations support competitive simple configurations within the evaluated single-block setting, rather than a universally preferable construction or selection rule.

E Scaling Verification Structures

E.1 Budget Sensitivity and Online Extensions

We evaluate LLaDA-8B-Instruct on GSM8K and HumanEval with the LLaDA splits, backends, draft length 4, and generation limits specified above. We fix maximum block width 3, at most 2 selected blocks, $\alpha = 0.3$, and the same fitted rank and gain statistics and candidate ordering across budgets. Only $B_K \in \{0, 1, 2, 4\}$ changes. Gain lookup uses the within-block confidence range, as in the complete-generation representation comparison. The budget-1 family can include 3-update partial orders costing 1 additional node, so it differs from the width-2 configuration used in Table 1. At budget 0, candidate construction and selection still incur overhead, whereas the drafting baseline uses its native path.

Budget 1 achieves the highest measured throughput in both tasks. On GSM8K, increasing the budget from 1 to 2 lowers total NFE from 11,364 to 11,054 but increases evaluated query tokens from 1,446,015 to 1,533,183. The additional verification work outweighs the reduction in forward calls for this selection rule. Budgets 2 and 4 produce the same recorded trajectory and work on GSM8K, so their small timing difference is not an additional structural benefit. These results motivate a low-budget operating point for the evaluated selection rule.

Online structure extensions. We also conduct a separate 7-variant comparison on LLaDA-8B-Instruct and GSM8K. It includes the baseline, calibrated single-block and fixed-position configura-

Table 11: Expanded structure families relative to calibrated single-block selection. Entries are percentage changes in NFE, evaluated query tokens (Query), and TPS. The expanded families are measured in a separate batch after the single-block variants.

		$B_K = 2$, width ≤ 2			$B_K = 4$, width ≤ 3		
Model	Task	NFE	Query	TPS	NFE	Query	TPS
LLaDA-8B	GSM8K	-2.90	+6.62	-1.82	-3.19	+6.74	-1.98
LLaDA-8B	HumanEval	-3.14	+5.94	-2.46	-3.14	+5.94	-2.46
Dream-7B	GSM8K	-3.14	+6.52	+0.69	-3.14	+6.52	+0.76
Dream-7B	HumanEval	-3.24	+6.17	+0.58	-3.29	+6.27	+0.74
SDAR-4B-b8	GSM8K	+0.00	-0.00	+0.71	-0.05	+0.09	+0.74
SDAR-4B-b8	HumanEval	+0.04	+0.06	+1.97	+0.04	+0.06	+1.95
SDAR-8B-b8	GSM8K	-0.03	-0.01	-0.97	-0.03	-0.01	-0.95
SDAR-8B-b8	HumanEval	-0.03	+0.00	-0.42	-0.03	+0.08	-0.45

tions, 2 width-2 blocks at budget 2, a block of width up to 3 at budgets 2 and 4, and a within-block antichain obtained after selecting a width-3 block at budget 4. All extensions retain calibrated construction and gain-based selection unless explicitly replaced, with wider-block gain statistics fitted from wider-block observations. These runs are separate from the budget sweep, main runtime evaluation, and construction and selection comparison in Table 9. The 2-block variant attains 63.358 TPS versus 64.375 for calibrated single-block selection, selecting 2 blocks in 6,899/9,213 rounds and accepting 132 updates on paths unavailable to any single-pair relaxation. Thus additional accepted progress does not by itself guarantee a speedup. The width-3 variants attain 64.322 and 64.395 TPS at budgets 2 and 4, and the within-block antichain attains 64.346 TPS, showing no clear advantage over single-block selection. The 2 wider-block budgets produce identical recorded graphs and work, so their timing difference does not establish a structural benefit.

E.2 Cross-Model Structure Extensions

This study uses the 4 model configurations, 2 tasks, splits, and execution settings of Appendix D.2. It compares 6 variants: a fixed-$N = 4$ SSD chain, the calibrated single-block configuration, the 2 fixed-position variants, and 2 expanded families. The budget-2 family permits up to 2 width-2 blocks, and the budget-4 family permits up to 2 blocks of width at most 3. Both retain calibrated rank construction and the positive-gain selection rule, with wider-block gain statistics fitted from calibration observations for the expanded blocks. The single-block configuration uses width 2 and at most 1 block, so this comparison changes the allowed structure family as well as the budget. It is distinct from the fixed-family budget sweep in Appendix E.1.

The chain and single-block variants reuse the runs underlying Appendix D.2. The expanded variants are measured afterward on the same GPU assignments, rotating their order across shards, rather than interleaving them with the single-block variants. Small TPS differences therefore do not isolate the effect of changing structures. NFE includes prefill and cache updates, and query tokens count actual model input positions, including their batch multiplicity. On LLaDA and Dream, the expanded families reduce NFE but increase query work. The observed TPS changes are negative on LLaDA and positive on Dream in these runs (Table 11). In this comparison, the threshold and fixed-position configuration improves TPS over the fixed-$N = 4$ chain by 4.36–7.87% across the 8 settings. This chain is the controlled baseline for this study, not the independently tuned SSD configuration in the main table.

To distinguish available from selected structures, we replay the frozen selectors on the saved single-block roots, keeping the draft and remaining mask count fixed. The budget-2 selector produces the same ideal graphs at all 18,651 saved SDAR roots, so the corresponding online timing differences cannot be attributed to additional order relaxation at those roots. Across all 44,878 saved roots, the budget-4 selector chooses a width-3 block at only 292 roots. These counts describe selected graphs on fixed roots, not counterfactual acceptance or throughput. They illustrate why a larger structural allowance need not translate into additional useful verification paths.

Table 12: Throughput with oracle structure selection on LLaDA-8B-Instruct and GSM8K. Ratios are relative to calibrated single-block selection within this experiment. Reference information and oracle search are free, while actual verification and generation costs are measured.

Selection	TPS	Ratio
Drafting baseline	59.203	0.923
Calibrated single-block selection	64.119	1.000
Single-pair oracle	68.160	1.063
Larger-family oracle, $B_K = 2$	70.927	1.106
Larger-family oracle, $B_K = 4$	71.613	1.117
Cost-aware oracle, $B_K = 4$	71.443	1.114

E.3 Throughput Potential under Oracle Selection

We test whether better structure selection can turn additional reference acceptance into higher end-to-end throughput. The study uses LLaDA-8B-Instruct on the 131 GSM8K test problems with $N = 4$ and the LLaDA generation limits and backend-specific cache policies specified above. The calibrated single-block configuration uses $\alpha = 0.3$, $w_{\max} = 2$, and $B_K = 1$. For this oracle comparison, we measure all methods on 8 A100 80GB GPUs, with the methods for each request assigned to the same GPU.

The single-pair oracle chooses the chain or an adjacent pair to maximize accepted draft length along the serial target continuation. The budget-2 and budget-4 oracles use the larger families defined in Appendix C.2, which contain the chain and all single-pair choices and need not exhaust the budget. Ties prefer lower predicted verification time, then fewer evaluated states and additional nodes, with a fixed structural tie break. A cost-aware budget-4 oracle instead maximizes expected committed updates per predicted verification time. Verification costs are estimated from up to 8 uniformly sampled roots per calibration problem and frozen before testing. The cost lookup groups measurements by backend, draft length, remaining masks, evaluated states, and native versus extended verification path.

Each oracle first generates its own recursive trajectory, collecting target continuations on independent state and cache copies and saving the chosen structure at each root. The timed execution includes prompt preparation and online setup, then replays these choices from a fresh prefill while performing actual drafting, state construction, verification, traversal, and cache updates. It includes decision lookup but excludes reference-continuation collection and oracle search. Replay checks require agreement with the prepared roots, drafts, transitions, and outputs. The experiment therefore measures execution with free reference information within the evaluated structure families. Realizing these gains online requires accounting for the cost of obtaining and applying the selection decisions.

Relative to calibrated single-block selection, the single-pair oracle improves throughput by 6.30%, and the budget-4 oracle by 11.69%. The latter remains 5.07% faster than the single-pair oracle, demonstrating additional throughput potential from the broader structure family. It uses all 4 additional nodes at only 183/8,932 roots across the full test set. Its total forward calls decrease by 11.56% relative to calibrated selection, and total evaluated query tokens decrease by 6.15%. Allowing a larger family can therefore reduce total generation work when the structure is chosen appropriately.

The cost-aware oracle selects the same graphs as the acceptance-maximizing budget-4 oracle on the observed roots, so their small timing difference does not establish an advantage for either objective. The results motivate improved online selection that converts this reference-assisted throughput potential into practical gains.