Structuring The Future: Diffusion LLM Speculative Decoding via Calibrated Draft Graphs

Sudhanshu Agrawal [1] **Risheek Garrepalli** [1] **Raghavv Goel** [1] **Christopher Lott** [1] **Fatih Porikli** [1] **Mingu Lee** [1]

Abstract

Diffusion LLMs (dLLMs) have recently emerged as a powerful alternative to autoregressive LLMs (AR-LLMs) with the potential to operate at significantly higher token-generation rates. To unlock this potential, we present Spiffy, a speculative decoding algorithm to accelerate dLLM inference while provably preserving the model's output distribution. This work addresses the unique challenges involved in applying ideas from speculative decoding of AR-LLMs to dLLMs. Spiffy performs auto-speculation to eliminate the overheads of an independent draft model, structuring draft states in the form of a novel directed draft graph to take advantage of the bidirectional, blockwise nature of dLLM generation. These draft graphs are calibrated offline to maximize acceptance rates by capturing the underlying decoding dynamics, and are dynamically pruned during inference for improved computational efficiency. We present a detailed formulation of Spiffy and demonstrate its acceleration of LLaDA, Dream, and SDAR models in combination with KV caching and threshold-based dynamic unmasking, leading to up to $8.6\times$ reduction in model inferences and $6.3\times$ acceleration in token rate.

1. Introduction

The majority of large language models (LLMs) today are autoregressive LLMs (AR-LLMs) (Brown et al., 2020). Such models operate left-to-right, generating new tokens conditioned on the previous tokens in the sequence. AR-LLMs have demonstrated impressive capabilities in a variety of domains but, due to their autoregressive nature, are constrained to generating a single token per model inference. As overall latency is dominated by the total model inference time, this constraint is a bottleneck in LLM decoding speeds.

[1]Qualcomm AI Research. Correspondence to: Sudhanshu Agrawal <sudhagra@qti.qualcomm.com>.

Published as a paper at the 1st FoGen workshop, ICML 2026, Seoul, South Korea, 2026.

More recently, the emergence of diffusion LLMs (dLLMs) (Nie et al., 2025; Inception Labs et al., 2025) represents a paradigm shift in language modeling. dLLMs are inherently parallel in nature and are not bound by a strict causal factorization. Instead, they model the joint distribution over token blocks, leveraging bidirectional attention (Devlin et al., 2019), and enabling arbitrary-order multi-token decoding. While commercial dLLMs such as Mercury (Inception Labs et al., 2025), Gemini Diffusion (Google, 2025), and Seed Diffusion (Song et al., 2025) may achieve token rates in excess of 1,000 tokens/second, open-sourced dLLMs such as LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) do not yet achieve these decoding speeds. In particular, to maintain output quality, these dLLMs generate only a *single token* per model inference by default, restricting their efficiency. Recent works seek to improve dLLM inference efficiency via KV caching (Wu et al., 2025; Ma et al., 2025) and threshold-based dynamic unmasking (Wu et al., 2025).

On the other hand, recent work on AR-LLMs circumvents their autoregressive constraints via speculative decoding (Leviathan et al., 2023; Chen et al., 2023). This has emerged as a popular and powerful method to accelerate AR-LLMs due to its accuracy guarantees and potential for efficient deployment. However, transferring the ideas developed for AR-LLM speculation to dLLMs encounters fundamental challenges. Since dLLM token distributions do not yield a causal factorization, draft verification must be adapted to remain lossless. Typical drafting strategies, such as tree-based drafting, also require substantial rethinking. Further, the choice of a compatible and efficient drafter for dLLMs is essential. In this work, we address these challenges and develop speculative decoding for dLLMs.

Spiffy (**Sp**eculation for D**iff**usion LLM Efficienc**y**), is a novel speculative decoding algorithm for dLLMs. During generation, Spiffy samples draft states from the target dLLM distribution itself, avoiding the need for, and costs of an auxiliary draft model. Spiffy structures these drafts as a directed draft graph to maximize their acceptance. While these draft graphs are similar in motivation to draft trees (Li et al., 2024; 2025b; Miao et al., 2024; Jeon et al., 2024), Spiffy's draft graphs respect and also take advantage of the bidirectional nature of dLLM generation. To optimize these graph

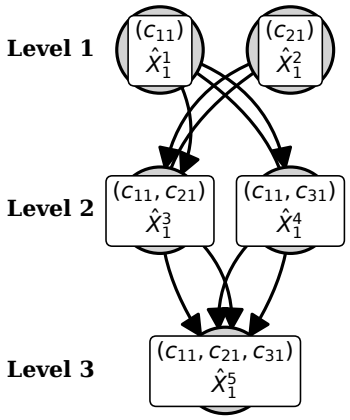

Figure 1. Spiffy defines and calibrates directed draft graph structures to capture multi-token unmasking dynamics. Spiffy's draft graphs leverage the bidirectional nature of dLLM decoding, allowing a draft state to be accepted via multiple potential pathways.

structures, we introduce a novel offline calibration procedure which determines effective draft graph configurations. These calibrated graphs significantly increase acceptance rates during generation and are dynamically pruned during inference for computational efficiency. In this work, we define speculative decoding for dLLMs and demonstrate our algorithm's ability to unlock accelerated dLLM inference. We summarize our key contributions below.

Summary of Contributions

1. **Speculative decoding of dLLMs:** We introduce Spiffy, a novel speculative decoding algorithm. We define drafting and verification for dLLMs as seamless, inexpensive procedures, and prove Spiffy's lossless-acceleration property formally and experimentally.

2. **Directed draft graphs:** We define directed draft graphs, a generalization of speculative draft trees, designed to take advantage of the bidirectional nature of dLLMs and maximize draft acceptance.

3. **Graph structure calibration and pruning:** We develop a novel calibration algorithm to generate optimized draft graphs that capture token-unmasking dynamics, boosting acceptance rates during speculation. During inference, these draft graphs are dynamically pruned to improve computational efficiency.

4. **Auto-speculation:** Spiffy leverages the target dLLM distribution at every timestep to speculate draft states, thus avoiding the need to train and run a draft model.

5. **Experimental validation:** We validate our methods through comprehensive experimentation with multiple open-source dLLMs (LLaDA-8B-Instruct, Dream-7B-Instruct, SDAR-8B-Chat-b32), evaluated on standard benchmarks (GSM8K, HumanEval, MATH500, MBPP), in combination with prefix KV caching and threshold-based dynamic unmasking, enabling up to $8.6\times$ reduction in model inferences and $6.3\times$ speedups in token rate.

2. Related Work

2.1. Diffusion LLMs

Diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020) have become a dominant approach in generative modeling, particularly excelling in continuous domains, such as image synthesis (Song et al., 2020; Saharia et al., 2022; Esser et al., 2024; Black Forest Labs et al., 2025). Their capacity for parallel generation has motivated their adaptation to language modeling, a discrete domain. Early works explored structured denoising and discrete diffusion processes for text generation (Austin et al., 2021a; Lou et al., 2023), while more recent efforts have focused on token-level masking-based forward processes (Sahoo et al., 2024; Shi et al., 2024) and blockwise decoding (Arriola et al., 2025; Cheng et al., 2025). Current open-source models implementing these ideas at scale include LLaDA (Nie et al., 2025; Bie et al., 2025), Dream (Ye et al., 2025; Xie et al., 2025), SDAR (Cheng et al., 2025), DiffuCoder (Gong et al., 2025), and OpenMoE (Ni & team, 2025).

2.2. Accelerating Diffusion LLMs

While dLLMs have the potential for parallel generation, in practice, many still decode one token at a time, thus requiring a large number of denoising steps. To overcome this, recent work has explored simultaneous token updates based on notions of confidence determined intrinsically (Wu et al., 2025) or by an auxiliary AR-LLM (Israel et al., 2025), as well as the use of modified solvers to accelerate the reverse diffusion process (Shaul et al., 2024; Li et al., 2025a; Luxembourg et al., 2025). Recent work also proposes drafting and verification for dLLMs (Hong et al., 2025; Gao et al., 2025), but without the use of structured directed draft graphs. Additionally, efforts have been made to integrate KV caching mechanisms to reduce redundant computation (Ma et al., 2025; Liu et al., 2025a;b). An alternate line of work focuses on improving efficiency through step distillation (Deschenaux & Gulcehre, 2025; Qian et al., 2026).

2.3. Speculative Decoding of AR-LLMs

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) accelerates the inference of a large, target model with

an efficient drafter. Generations from the drafter have seen several forms, most often tree-based (Miao et al., 2024; Jeon et al., 2024; Li et al., 2024). Other works reduce the overhead of the drafter (Lin et al., 2025) or have tried to do away with it entirely (Fu et al., 2024b; Zhang et al., 2023; Fu et al., 2024a). Recent methods have proposed improvements to the draft generation quality via training improvements (Goel et al., 2024), by allowing the drafter to leverage the final hidden features of the target model (Li et al., 2024), and by adding additional linear modules to predict multiple draft tokens at once (Cai et al., 2024). Recent works also explore the use of a small diffusion language model as a draft model for a target AR-LLM (Christopher et al., 2024).

2.4. Speculation for Image Diffusion Models

(Wang et al., 2024) and (De Bortoli et al., 2025) propose modifications to rejection sampling in the case of continuous-valued diffusion models and achieve impressive speedups in image generation latency. While Wang et al. use an independent diffusion model as a draft model, De Bortoli et al. and Hu et al. use the target model to speculate future states. Other works explore the use of multi-sample generation and gradient-based MCMC within energy-based formulations (Du et al., 2020; Singhal et al., 2025).

3. Preliminaries

3.1. Masked Diffusion Language Models

Masked diffusion language models, at inference time, define a reverse denoising process where a token sequence $X(T)$ is iteratively unmasked to a valid sequence $X(0)$. Practically, this is implemented by setting a generation length W and initializing the sequence with a specially designated `MASK` token as $X(T) = [\texttt{MASK}, \cdots, \texttt{MASK}]$. This sequence is divided into N blocks of size L each, and we denote the i^{th} block of X by X_i. The dLLM then fully unmasks each block one by one, from $X_i(T) \to X_i(0)$. In particular, at each timestep $t+1 \in \{T, \cdots, 0\}$, the dLLM models:

$$p_\theta\left(X_i(t) \mid X(t+1)\right) \tag{1}$$

By sampling $X_i(t)$ from (1), the next candidate state of the sequence with per-token probabilities $p_1, \cdots, p_L$ is obtained. We define the rate of unmasking as S_t, and the top-S_t tokens, according to their probabilities, are unmasked while the others remain masked. S_t thus represents the rate at which a block is denoised. Open-source dLLMs such as LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) often set this parameter as $S_t = S = 1$ for all timesteps t, consequently unmasking one token per iteration. This setting allows these models to achieve accuracies comparable to AR-LLMs.

Recently, **threshold-based dynamic unmasking** (Wu et al., 2025) has proved to be an effective solution to this issue. A fixed threshold τ is determined, and at each iteration, all tokens with probability $p_i > \tau$ are unmasked, leading to a variable unmasking rate S_t per iteration. Wu et al. show that this method preserves accuracy while speeding up dLLM generations significantly. Other recent work involves KV caching of dLLMs via **prefix-caching** (Wu et al., 2025; Ma et al., 2025) where the KVs of all $X_{<i}$ remain fixed while decoding X_i. We present the speedup of our speculative decoding algorithm in combination with these methods and demonstrate its ability to improve upon their acceleration.

3.2. Speculative Decoding

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) is a powerful technique designed to parallelize the decoding process of AR-LLMs while provably preserving their output distribution. A typical speculative decoding setup consists of a large *target* model that one wishes to accelerate and a second, more efficient *drafter* that approximates it. This drafter may take many forms, as discussed in Sec. 2, and proposes a set of draft completions for the target model to verify in parallel. Concretely, given a prompt $X_P = [x_1, \cdots, x_t]$, the target model has a target distribution p_{TD} and autoregressively models $p_{\text{TD}}\left(x_i \mid x_1, \cdots, x_{i-1}\right)$ for $i \in \{t+1, \cdots, t+L\}$, thus generating L new tokens, $[x_{t+1}, \cdots, x_{t+L}]$. This process is slow, so, to accelerate it, the drafter models a draft distribution $p_{\text{DD}}\left(x_i \mid x_{<i}\right)$ generating draft tokens $[\hat{x}_{t+1}, \cdots, \hat{x}_{t+D}]$ for some draft length D. These draft tokens may then be *verified* in parallel by obtaining $p_{\text{TD}}(\cdot \mid x_{<i}, \hat{x}_{<i})$ and performing rejection sampling with the candidate draft tokens. In a greedy-decoding setting, we may simply sample $x_i \sim p_{\text{TD}}(\cdot \mid x_{<i}, \hat{x}_{<i})$ and accept $\hat{x}_i$ if it equals x_i. If we accept M of these draft tokens, we have effectively sampled $[x_{t+1}, \cdots, x_{t+M}]$ using only a single target model inference.

4. Method

Spiffy casts the denoising process of dLLMs in a speculative decoding framework. Rather than speculating on individual tokens, Spiffy speculates the state of the sequence over denoising timesteps, allowing us to *skip ahead* in the unmasking process while maintaining output quality.

4.1. Notation

Suppose, in the setting defined in Sec. 3, we are currently denoising block $k \in \{1, \cdots, N\}$ and X_k is observed at a denoising timestep $t \in [T, 0]$ as $X_k(t)$. Since the rest of the sequence is unchanged while X_k is being unmasked, we refer to the rest of the sequence collectively as X'. Thus, the token sequence as a whole is $X(t) = X'; X_k(t)$ (the sequence with $X_k(t)$ taking the position of the k^{th} block).

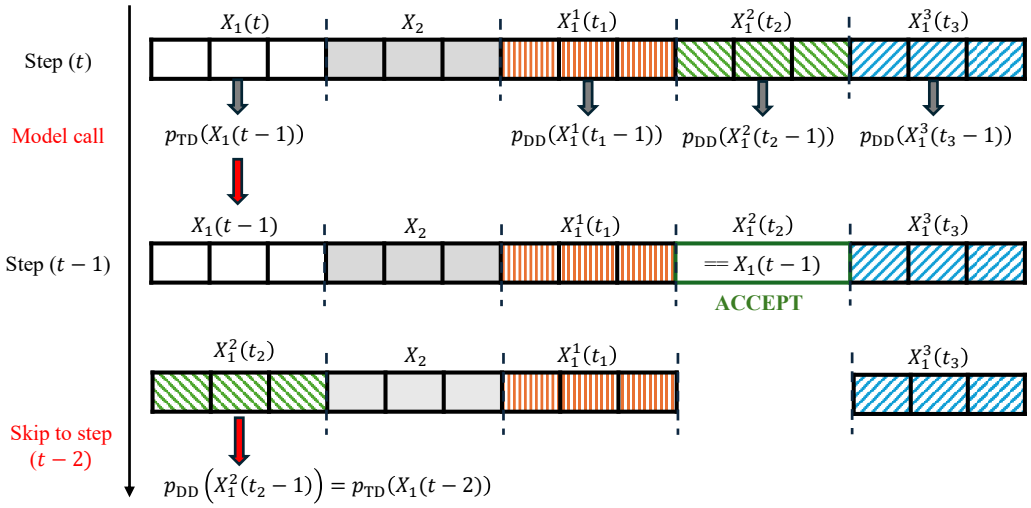

Figure 2. Spiffy's lossless verification allows $X_1(t)$ to advance to state $X_1(t-2)$ using a single model inference. Draft blocks X_1^1, X_1^2, X_1^3 are appended to the sequence, and model inference is performed in parallel. Each draft is compared against $X_1(t-1)$ to check for acceptance. If a draft is accepted, we use its predicted draft distribution to skip ahead and predict $X_1(t-2)$. We then repeat this process with the remaining drafts to check for additional acceptances.

4.2. Draft Blocks

When unmasking any given token, it is essential to consider the influence of every other token in the sequence. This is a characteristic property of bidirectional attention (Devlin et al., 2019) and to satisfy this condition, we consider drafts at the block level. We denote a **draft block** for X_k as $\hat{X}_k^m(t_m)$ with $m \in \{1, \cdots, D\}$ for a number of drafts, D. Each draft block represents a speculation of the state of the block at a timestep t_m and we refer to the set of current draft blocks as $\{\hat{X}_k^m\}$.

4.3. Verification

At timestep t, the current state of the target (true) sequence is $X'; X_k(t)$. We use the dLLM to model two separate distributions, the target distribution at the next timestep:

$$p_{\text{TD}}(t-1) = p_\theta\left(X_k(t-1) \mid X'; X_k(t)\right) \tag{2}$$

and the draft distributions at their corresponding timesteps:

$$p_{\text{DD}}^m(t_m - 1) = p_\theta\left(X_k(t_m - 1) \mid X'; \hat{X}_k^m\right) \tag{3}$$

This can be achieved in parallel using a single model call with a custom attention mask and appropriate positional embeddings, similar to the strategies adopted by (Miao et al., 2024; Jeon et al., 2024; Li et al., 2024). We include an example of such a mask in Fig. 3.

To advance the sequence to the next timestep, S_{t-1} tokens are sampled from the target distribution and are unmasked. The state of the block is then compared to the draft candidates and any draft that matches is accepted. Since we already have access to the draft distribution of the accepted block, we may skip ahead in the denoising and repeat the procedure with the remaining drafts. This verification method is described in detail in Algorithm 1 and visualized in Fig. 2.

Thus, if in the process of unmasking a block at a rate of S_t tokens, we accept drafts for timesteps $t_1, \cdots, t_M$, we decrease the number of model inferences required by a factor of $(T - M)/T$.

We show that Algorithm 1 is *lossless*, that is, it preserves the output distribution of the model. We include a proof of this statement in Appendix A.1. In certain settings, producing the exact draft probabilities $\{p_{\text{DD}}^m\}$ is computationally expensive, and achieving exact decoding requires construction of a batched attention mask and batched token sequence which would proportionally increase the computational requirements of the system. Instead, we choose to create a blockwise tree-attention mask structure that achieves near-lossless performance for full-sequence bidirectional dLLMs such as LLaDA. Moreover, using this simple mask, we can achieve exact lossless speculative decoding when utilizing dual caches (Wu et al., 2025) or when decoding blockwise causal dLLMs such as SDAR (Cheng et al., 2025). Further discussion, including the impact of floating-point precision loss, is included in Appendix A.2.

4.4. Drafting

4.4.1. Draft Graphs

We now desire draft blocks $\{\hat{X}_k^m\}$ for $m \in \{1, \cdots, D\}$ such that their *acceptance is maximized*. In particular, we

Algorithm 1 Spiffy Draft Block Verification

Given: unmasking rate $\{S_t\}$, vocabulary of size V
Input: current t, current $X_k(t)$, target p_{TD}, draft $\{p^m_{\text{DD}}\}$, drafts $\{\hat{X}^m_k\}$
(1) Extract marginals p_n for each token position
$p_n \leftarrow p_{\text{TD}}(t-1)[n] \in [0,1]^V$
(2) top-1 probability for each token position n
$p^1_n, j_n \leftarrow \max, \arg\max_{v \in V} p_n(v) \; \forall \; n \in [1, L]$
(3) Unmask top-S_{t-1} positions
$n_1, \ldots, n_{S_{t-1}} \leftarrow \text{topKIndex}(S_{t-1}, [p^1_1, \ldots, p^1_L])$
$c_{n_i} \leftarrow \text{vocabulary}[j_{n_i}] \quad \forall i \in 1, \cdots, S_{t-1}$
(4) Update the block with the new tokens
$X_k(t-1) \leftarrow X_k(t)$
$X_k(t-1)[n_i] \leftarrow c_{n_i} \quad \forall \; i \in 1, \cdots, S_{t-1}$
$t \leftarrow t-1$
(5) Draft verification loop
for $m \in \{1, \ldots, D\}$ **do**
  if $t_m \neq t$ **then**
    continue
  end if
  if $\hat{X}^m_k = X_k(t)$ **then**
    # Accept
    $p_{\text{TD}}(t-1) \leftarrow p^m_{\text{DD}}(t-1)$
    Pop $\hat{X}^m_k$ from the draft list
    # Restart the algorithm from this state
      using $children(\hat{X}^m_k)$ (4) (5) (6)
    $\text{VERIFY}(t, X_k(t), p_{\text{TD}},$
      $\{p_{\text{DD}}\}_{\in children(\hat{X}^m_k)}, \{\hat{X}_k\}_{\in children(\hat{X}^m_k)})$
  end if
end for
Output: t, $X_k(t)$

would like the opportunity to have *multiple* drafts accepted per model call. To this end, we structure the draft blocks to have a directed, *parent-child* relationship with each other and to the current block $X_k(t)$. Block A at timestep t_a is defined to be a **parent** of block B at timestep t_b if:

$$t_b = t_a - 1 \tag{4}$$

$$|unmasked(A)| + S_{t_b} = |unmasked(B)| \tag{5}$$

$$unmasked(A) \subset unmasked(B) \tag{6}$$

Intuitively, block B should be a potential *next step* in the denoising process starting from block A. This property allows us to accept multiple drafts per model call since the acceptance of draft A means that we may potentially accept any children of A that were also drafted. In this manner, we may skip ahead multiple timesteps with a single model call.

We define $children(A)$ as the set of all blocks B that satisfy (4, 5, 6). We first choose a set of draft blocks, $\{\hat{X}^m_k\}(\text{level } 1) \subset children(X_k(t))$, representing the *first level* of drafts. Draft blocks for subsequent levels are then chosen in a similar manner:

$$\{\hat{X}^m_k\}(\text{level } i) \quad \subset \bigcup_{A \in \{\hat{X}^m_k\}(\text{level } i-1)} children(A) \tag{7}$$

Together, our draft blocks $\{\hat{X}^m_k\}$ take on the form of a **directed draft graph**. Notably, unlike draft trees used for speculative decoding of AR-LLMs, each draft block may have *multiple parents* and thus *multiple pathways to being accepted*, a unique advantage of bidirectional attention. A visualization of such a draft graph is provided in Fig. 1.

	X_1	X_2	$\hat{X}^1_1$	$\hat{X}^2_1$	$\hat{X}^3_1$	$\hat{X}^4_1$	$\hat{X}^5_1$
X_1	✓	✓					
X_2	✓	✓					
$\hat{X}^1_1$		✓	✓				
$\hat{X}^2_1$		✓		✓			
$\hat{X}^3_1$		✓			✓		
$\hat{X}^4_1$		✓				✓	
$\hat{X}^5_1$		✓					✓

Figure 3. Spiffy constructs a blockwise attention mask to verify the draft candidate states. The true sequence continues to attend to itself. Each draft block attends to itself and all blocks in the true sequence except the original block for which it is drafting. For blockwise-causal dLLMs, this mask is updated to have a blockwise-causal masking structure with respect to the true tokens.

4.4.2. Drafting Source

Given this desired drafting structure, we now need a process by which the contents, i.e., unmasked tokens, of a draft block may be determined. At any timestep, we have access to $p_{\text{TD}} = p_\theta\left(X_k(t-1) \mid X'; X_k(t)\right)$, which we leverage to construct draft blocks. For brevity, ArgSort is assumed to be in descending order in the following relations.

First, with the marginals $p_1, \cdots, p_L$ of the current distribution, we obtain an ordering of token positions:

$$n_1, \ldots, n_L \leftarrow \text{ArgSort}\left(\max\left(p_1\right), \ldots, \max\left(p_L\right)\right) \tag{8}$$

Second, for each sorted position n_i, we determine the top-k token choices from the vocabulary of size V:

$$c_{i1}, \cdots, c_{ik} \leftarrow \text{ArgSort}(p_{n_i}(v_1), \cdots, p_{n_i}(v_V))[:k] \tag{9}$$

Thus, when considering candidates for unmasking, the 'ranking' of a candidate depends on both its position in the token sequence and its likelihood among token choices for that

position. We can characterize this ranking by defining two new terms. For any token position $n \in \{1, \cdots, L\}$ that we are considering unmasking, we define its **token position rank** and denote it by i

$$i := \text{ArgSort}\left(\max(p_1), \cdots, \max(p_L)\right)[\, n \,] \tag{10}$$

Further, for token position n, for any particular token $v \in \{1, \cdots, V\}$ that we could unmask in this position, we define the **vocabulary rank** of this choice v and denote it by j

$$j := \text{ArgSort}\left(p_n(1), \cdots, p_n(V)\right)[\, v \,] \tag{11}$$

So any candidate token that may be unmasked at a sequence position can be denoted by c_{ij} in terms of its token position rank and its vocabulary rank. These indices (i, j) and the sorting method is visualized in Fig. 4. In this way, we define each of our draft blocks $\hat{X}_k^m(t_m)$ as consisting of the current block state $X_k(t)$ with a set of additionally unmasked tokens $\{c_{ij}\}$, such that $|\{c_{ij}\}| = \sum_{t<t'\leq t_m} S_{t'}$. By leveraging the current target distribution in the construction of draft blocks, we avoid the need for an independent draft model.

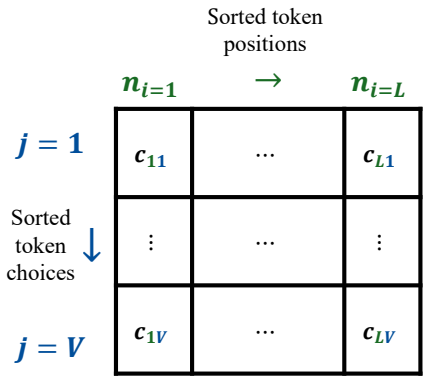

Figure 4. By leveraging the marginals of $p_{\text{TM}}(X(t))$ for each position in the block, we sort token candidates by i = token position rank and j = vocabulary rank.

4.4.3. Draft Graph Calibration

Since we now have a source for draft blocks, the next step is to determine which candidate states are *most promising*. We know that it is to our advantage to structure these drafts as a directed graph, but since there are L possible unmasking positions, each with V candidates to choose from, we desire a method to produce *optimized* draft graph structures.

To achieve this, we present a novel calibration algorithm to procedurally determine draft graph structures. A draft graph is defined by a set of D **draft formulas**, where each formula is a set $\{(i, j)\}$, representing the tokens $\{c_{ij}\}$ that will populate the draft block. Intuitively, we wish to create drafts with assigned choices of (i, j) so that we may select their unmasked token contents once the current timestep's probability distribution is available. Algorithm 2 details the calibration procedure, which involves running the vanilla dLLM on a small dataset and using the frequency of various sequences of (i, j) to create optimized directed draft graphs. We also present a simplified pseudocode of Algorithm 2 in Appendix D.4 Algorithm 3.

In this work, we run calibration only once per model on a combined set of 25 samples each from MATH500 and MBPP. An example of a calibrated draft graph with $D = 5$ is seen in Fig. 1. We provide further intuition for this calibration strategy in Appendix D.1 and discuss the costs of running this algorithm in Appendix D.2, observing that it requires ~ 10 minutes to perform offline on a single GPU. In Appendix D.3 we study the effect of changing the calibration dataset to a general text dataset, ShareGPT, observing comparable performance to the default calibration.

4.4.4. Draft Graph Pruning

Draft graph calibration through Alg. 2 produces a structure with D draft nodes. D represents the number of draft blocks that will be appended to the sequence during verification. Although the inference of LLMs is memory-bound in general, it is beneficial to *prune* the draft graph dynamically during inference to a smaller budget $D^* < D$ to save on computation. Since the draft nodes each represent sequences of tokens with a parent-child structure (Sec. 4.4.1), it is desirable to have a pruning metric that captures the joint probability of that sequence and the relationships between nodes. To this end, we define a metric based on the geometric mean of the token probabilities corresponding to each node. Suppose a node q_i consists of tokens $c_{i1}, \cdots, c_{ik}$ with corresponding probabilities $p_{i1}, \cdots, p_{ik}$. We define the local score of a node via its geometric mean as,

$$localScore(q_i) = GM(p_{i1}, \cdots, p_{ik}) = \left(\prod_{j=1}^{k} p_{ij}\right)^{1/k} \tag{12}$$

and the aggregated score of its children as,

$$childScore(q_i) = GM\left(\{localScore(x) \text{ for } x \in children(q_i)\}\right) \tag{13}$$

to compute the total score of the node,

$$score(q_i) = GM\left(localScore(q_i), childScore(q_i)\right) \tag{14}$$

GM is used consistently as the aggregation metric to respect the multiplicative nature of probabilities. With this scoring metric, the top-D^* nodes according to the current timestep's probability distribution can be flexibly chosen to be draft states. We see in Sec. 5.2 that this metric enables near-complete recovery of the full draft graph's acceptance rate while reducing its computational overhead.

Algorithm 2 Draft Graph Structure Determination via Offline Calibration

Inputs: dLLM ϕ, sample $Dataset$, 'look-ahead' length for calibration $\mathcal{L}$, number of drafts budget D

 # Part 1: Collect calibration data
results $\leftarrow$ [ ]
for sample $\in Dataset$ **do**
  $X \leftarrow \phi(sample)$ with:
    denoising interval $[T, 0]$
    recording of $p_{\text{TD}}(t) \quad \forall t$
  # Rewind the generation of X
  sequence $\leftarrow$ [ ]
  for every timestep $t \in [T, 0]$ **do**
    for $t' \in [t, t + \mathcal{L}]$ **do**
    # Unmasked tokens at current lookahead
      $x_1, \ldots, x_k \leftarrow$ tokens unmasked at t'
      Get (i, j) for each token using (10, 11)
      Append $[(L = t' - t, \{(i, j)\})]$ to sequence
    end for
  end for
end for
# results = [seq $= [(L = k, \{(i, j)\})]$]

 # Part 2: All-possibilities-graph construction
for $k \in 1, \ldots, \mathcal{L}$ **do**
  Possibilities $\leftarrow$ sequences in $results$ containing $L = k$
  For each $seq \in$ Possibilities,
    # Each node is an unordered set $\{(i, j)\}$
    count(node) := frequency of occurrence of node
    current_node = $\bigcup_{e \in seq \mid e.L \leq k} \{(i, j)\}$
    previous_node = $\bigcup_{e \in seq \mid e.L < k} \{(i, j)\}$
  top_nodes = {top-3 most frequent current_nodes}
  top_in_edges = {top-3 most frequent previous_nodes}
  $G \leftarrow$ top_nodes, top_edges.
  # Add additional edges
  $G \leftarrow \{(p, q) \; \forall \; p, q \in G \mid p \subset q\}$
end for
# $G \leftarrow$ nodes $= \{q_{kv}\}$ for $v \in 1, 2, 3, \; k \in 1, \cdots, \mathcal{L}$
# $G \leftarrow$ edges $= \{(p_{k_1 v}, q_{k_2 u})\}$
 # Each $q_{kv} = \{(i, j)\}$ **is a draft formula**

 # Part 3: Optimal subgraph selection
$G' \leftarrow$ all connected subgraphs of G with size D where $k = 1, 2$ are root levels.
 # Find the optimal subgraph G^*
$G^* = \arg\max_{Q \in G'} \sum_{q \in Q} (count(q))$
$G^* = \{q_{kv}\}$**, a set of** D **draft formulas**

Output G^*

Thus, Spiffy's method involves first running the calibration algorithm offline to determine an optimized draft graph structure. Speculation then begins, using this graph configuration to determine the token candidates c_{ij} that will appear in each draft block. The graph is pruned according to a desired budget and is then verified in parallel to accept draft states, resulting in accelerated inference. We provide a detailed profile of the overheads of Spiffy's drafting, pruning, and verification steps in Appendix C and see that they are negligible compared to the model inference time.

5. Experiments

Following the settings of (Wu et al., 2025) and (Wang et al., 2025), we demonstrate the acceleration provided by Spiffy for LLaDA-8B-Instruct (Nie et al., 2025), Dream-7B-Instruct (Ye et al., 2025), and SDAR-8B-Chat-b32 (Cheng et al., 2025) on standard coding and math tasks, GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021), MBPP (Austin et al., 2021b), and HumanEval (Chen et al., 2021) with block size 32. For LLaDA and Dream, the generation length is set to 256 and the temperature to 0.0 by default. We set SDAR's sampling temperature to 1.0, following its recommended setting, and its generation length to 512. We additionally compute the accuracy of the model outputs to validate Spiffy's lossless property. Prefix KV caching (Wu et al., 2025; Ma et al., 2025) is enabled in all settings and the baseline is set to be static unmasking with one token decoded per iteration. We compare this baseline to the same setting with dynamic unmasking with threshold $= 0.9$ (Wu et al., 2025). Spiffy is then enabled in this setting to demonstrate its combined speedup.

To calibrate Spiffy, we use 25 samples from each of MBPP and MATH500, isolated from the test set to avoid contamination, and produce a single directed draft graph with $D = 10$ for each model. This graph is then dynamically pruned to $D^* = 3$ during inference. We calculate speedup with respect to the baseline in terms of increase in wall-clock token rate (TPS) and in terms of reduction in the Number of Function Evaluations (NFEs).

5.1. Main Experiment

Table 1 displays the acceleration provided by Spiffy for LLaDA-8B-Instruct, Dream-7B-Instruct, and SDAR-8B-Chat-b32, and we see that across tasks and models, Spiffy enables NFE reductions of up to $9\times$ and token rate speedups of up to $6\times$ while preserving the model's accuracy. We include additional experiments on open-ended text-generation tasks from MMLU in Appendix B Table 2.

5.2. Number of Draft Blocks

Using the draft pruning procedure in Sec. 4.4.4, we demonstrate in Fig. 5 the effect of the number of draft blocks on TPS speedup and NFE reduction. Using a higher number of draft blocks will result in more acceptances and further

Table 1. Spiffy successfully results in NFE reductions of up to 8.6×, with TPS improvements of up to 6.3×, while preserving accuracy across dLLMs and tasks.

	GSM8K	HumanEval	MATH500	MBPP
Baseline (LLaDA-8B-Instruct)	1.00× (1.00×), 0.79	1.00× (1.00×), 0.41	1.00× (1.00×), 0.34	1.00× (1.00×), 0.36
+ Dynamic Unmasking	3.31× (3.20×), 0.79	3.51× (2.89×), 0.41	2.52× (2.63×), 0.35	5.73× (5.01×), 0.36
+ Spiffy	4.97× (3.55×), 0.79	5.80× (3.22×), 0.38	4.11× (3.00×), 0.35	8.58× (5.23×), 0.36
Baseline (Dream-7B-Instruct)	1.00× (1.00×), 0.79	1.00× (1.00×), 0.54	1.00× (1.00×), 0.41	1.00× (1.00×), 0.49
+ Dynamic Unmasking	2.35× (2.28×), 0.78	2.68× (2.73×), 0.52	2.17× (2.16×), 0.42	2.45× (2.63×), 0.50
+ Spiffy	3.76× (2.56×), 0.79	4.42× (2.90×), 0.55	3.57× (2.57×), 0.42	4.00× (2.97×), 0.49
Baseline (SDAR-8B-Chat-b32)	1.00× (1.00×), 0.90	1.00× (1.00×), 0.68	1.00× (1.00×), 0.49	1.00× (1.00×), 0.26
+ Dynamic Unmasking	6.71× (5.67×), 0.88	5.54× (4.24×), 0.71	4.38× (4.11×), 0.47	3.99× (3.45×), 0.24
+ Spiffy	8.25× (6.28×), 0.88	6.96× (4.77×), 0.71	5.46× (4.60×), 0.47	5.29× (4.21×), 0.25

reductions in NFEs, while fewer draft blocks will result in decreased acceptance, but lower computational costs. By varying the draft budget using our draft pruning procedure, we can strike a balance between these metrics according to the system's requirements.

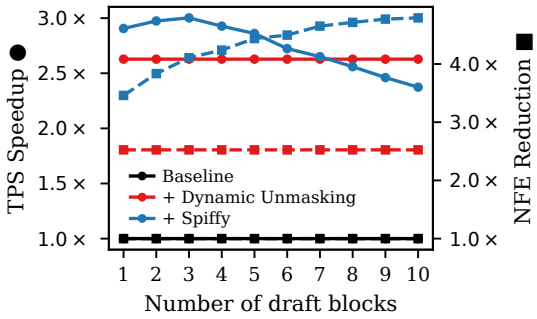

Figure 5. Spiffy's draft pruning procedure achieves a balance between reduction in NFEs (squares) and wall-clock token rate speedup (circles) depending on the preferred metric. The baseline is LLaDA-8B-Instruct on MATH500 with prefix-caching.

5.3. Effect of Temperature

In Fig. 6, we study the effect of temperature on Spiffy's speedup for LLaDA-8B-Instruct with prefix caching. We reuse the calibrated draft graph generated with temperature $= 0.0$ for all settings and see that with increasing sampling temperature, draft acceptance is more challenging, yet Spiffy can achieve higher TPS than the baseline. We expect that re-calibrating the draft graph for the desired temperature setting would improve performance further.

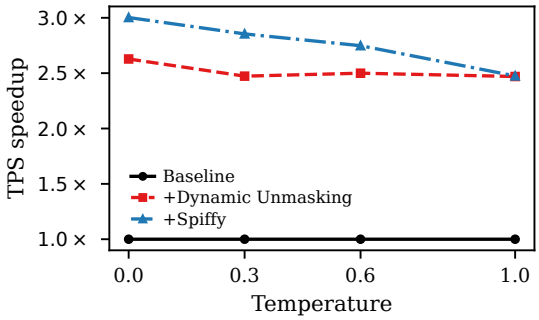

Figure 6. Spiffy remains effective at speeding up LLaDA-8B-Instruct inference even with increasing sampling temperature.

6. Discussion and Conclusion

Spiffy is a novel speculative decoding algorithm for diffusion LLMs that uses structured draft graphs to capture token unmasking dynamics. In this work, we define directed draft graphs that take advantage of the bidirectional nature of dLLM decoding to maximize acceptance. Verification of these draft graphs occurs in parallel and results in preserved model accuracy. We develop a novel, inexpensive calibration algorithm to optimize the draft graph structures and a pruning procedure that reduces its computational requirements. This framework is implemented via auto-speculation, eliminating the overheads of training and deploying an independent draft model. Spiffy is shown to be effective for multiple open-source dLLM model families, LLaDA, Dream, and SDAR, evaluated on multiple tasks. We further ablate on the impact of the number of draft blocks and temperature on Spiffy's acceleration benefits. Future investigations could explore alternate calibration and pruning metrics, and substituting Spiffy's auto-speculation implementation with the use of an auxiliary draft model. We demonstrate that Spiffy's formulation generalizes to arbitrary denoising rates and as a result, as future dLLMs require decreasing numbers of denoising steps, Spiffy will remain an effective acceleration solution.

References

Arriola, M., Gokaslan, A., Chiu, J. T., Yang, Z., Qi, Z., Han, J., Sahoo, S. S., and Kuleshov, V. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and Van Den Berg, R. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021a.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021b.

Bie, T., Cao, M., Chen, K., Du, L., Gong, M., Gong, Z., Gu, Y., Hu, J., Huang, Z., Lan, Z., et al. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

Black Forest Labs, Batifol, S., Blattmann, A., Boesel, F., Consul, S., Diagne, C., Dockhorn, T., English, J., English, Z., Esser, P., et al. Flux. 1 kontext: Flow matching for in-context image generation and editing in latent space. *arXiv preprint arXiv:2506.15742*, 2025.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Cheng, S., Bian, Y., Liu, D., Zhang, L., Yao, Q., Tian, Z., Wang, W., Guo, Q., Chen, K., Qi, B., and Zhou, B. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation, 2025. URL https://arxiv.org/abs/2510.06303.

Christopher, J. K., Bartoldson, B. R., Ben-Nun, T., Cardei, M., Kailkhura, B., and Fioretto, F. Speculative diffusion decoding: Accelerating language generation through diffusion. *arXiv preprint arXiv:2408.05636*, 2024.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

De Bortoli, V., Galashov, A., Gretton, A., and Doucet, A. Accelerated diffusion models via speculative sampling. *arXiv preprint arXiv:2501.05370*, 2025.

Deschenaux, J. and Gulcehre, C. Beyond autoregression: Fast LLMs via self-distillation through time. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=uZ5K4HeNwd.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Du, Y., Li, S., and Mordatch, I. Compositional visual generation with energy based models. *Advances in Neural Information Processing Systems*, 33:6637–6647, 2020.

Esser, P., Kulal, S., Blattmann, A., Entezari, R., Müller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F., Podell, D., Dockhorn, T., English, Z., and Rombach, R. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=FPnUhsQJ5B.

Fu, Y., Bailis, P., Stoica, I., and Zhang, H. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024a.

Fu, Y., Bailis, P., Stoica, I., and Zhang, H. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024b.

Gao, Y., Ji, Z., Wang, Y., Qi, B., Xu, H., and Zhang, L. Self speculative decoding for diffusion large language models, 2025. URL https://arxiv.org/abs/2510.04147.

Goel, R., Gagrani, M., Jeon, W., Park, J., Lee, M., and Lott, C. Direct alignment of draft model for speculative decoding with chat-fine-tuned llms. *arXiv preprint arXiv:2403.00858*, 2024.

Gong, S., Zhang, R., Zheng, H., Gu, J., Jaitly, N., Kong, L., and Zhang, Y. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.

Google. Gemini diffusion, 2025. Accessed: 2025-07-25.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Hong, F., Yu, G., Ye, Y., Huang, H., Zheng, H., Zhang, Y., Wang, Y., and Yao, J. Wide-in, narrow-out: Revokable decoding for efficient and effective dllms. *arXiv preprint arXiv:2507.18578*, 2025.

Hu, H., Das, A., Sadigh, D., and Anari, N. Diffusion models are secretly exchangeable: Parallelizing ddpms via autospeculation. *arXiv preprint arXiv:2505.03983*, 2025.

Inception Labs, Khanna, S., Kharbanda, S., Li, S., Varma, H., Wang, E., Birnbaum, S., Luo, Z., Miraoui, Y., Palrecha, A., Ermon, S., Grover, A., and Kuleshov, V. Mercury: Ultra-fast language models based on diffusion, 2025. URL https://arxiv.org/abs/2506.17298.

Israel, D., Broeck, G. V. d., and Grover, A. Accelerating diffusion llms via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025.

Jeon, W., Gagrani, M., Goel, R., Park, J., Lee, M., and Lott, C. Recursive speculative decoding: Accelerating llm inference via sampling without replacement. *arXiv preprint arXiv:2402.14160*, 2024.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Li, S., Kallidromitis, K., Bansal, H., Gokul, A., Kato, Y., Kozuka, K., Kuen, J., Lin, Z., Chang, K.-W., and Grover, A. Lavida: A large diffusion language model for multimodal understanding. *arXiv preprint arXiv:2505.16839*, 2025a.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 2024.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-3: Scaling up inference acceleration of large language models via training-time test, 2025b. URL https://arxiv.org/abs/2503.01840.

Lin, F., Yi, H., Yang, Y., Li, H., Yu, X., Lu, G., and Xiao, R. Bita: Bi-directional tuning for lossless acceleration in large language models. *Expert Systems with Applications*, 279:127305, 2025.

Liu, Z., Yang, Y., Zhang, Y., Chen, J., Zou, C., Wei, Q., Wang, S., and Zhang, L. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025a.

Liu, Z., Yang, Y., Zhang, Y., Chen, J., Zou, C., Wei, Q., Wang, S., and Zhang, L. dllm-cache: Accelerating diffusion large language models with adaptive caching, 2025b. URL https://arxiv.org/abs/2506.06295.

Lou, A., Meng, C., and Ermon, S. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

Luxembourg, O., Permuter, H., and Nachmani, E. Plan for speed–dilated scheduling for masked diffusion language models. *arXiv preprint arXiv:2506.19037*, 2025.

Ma, X., Yu, R., Fang, G., and Wang, X. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Wang, Z., Zhang, Z., Wong, R. Y. Y., Zhu, A., Yang, L., Shi, X., 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.

Ni, J. and team. Openmoe 2: Sparse diffusion language models. https://github.com/JinjieNi/OpenMoE2, 2025.

Nie, S., Zhu, F., You, Z., Zhang, X., Ou, J., Hu, J., Zhou, J., Lin, Y., Wen, J.-R., and Li, C. Large language diffusion models, 2025. URL https://arxiv.org/abs/2502.09992.

Qian, Y.-Y., Su, J., Hu, L., Zhang, P., Deng, Z., Zhao, P., and Zhang, H. d3llm: Ultra-fast diffusion llm using pseudo-trajectory distillation. *arXiv preprint arXiv:2601.07568*, 2026.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Gontijo-Lopes, R., Ayan, B. K., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M. Photorealistic text-to-image diffusion models with deep language understanding. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=08Yk-n5l2Al.

Sahoo, S., Arriola, M., Schiff, Y., Gokaslan, A., Marroquin, E., Chiu, J., Rush, A., and Kuleshov, V. Simple and

effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Shaul, N., Gat, I., Havasi, M., Severo, D., Sriram, A., Holderrieth, P., Karrer, B., Lipman, Y., and Chen, R. T. Flow matching with general discrete paths: A kinetic-optimal perspective. *arXiv preprint arXiv:2412.03487*, 2024.

Shi, J., Han, K., Wang, Z., Doucet, A., and Titsias, M. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

Singhal, R., Horvitz, Z., Teehan, R., Ren, M., Yu, Z., McKeown, K., and Ranganath, R. A general framework for inference-time scaling and steering of diffusion models. *arXiv preprint arXiv:2501.06848*, 2025.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 2256–2265, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/sohl-dickstein15.html.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Song, Y., Zhang, Z., Luo, C., Gao, P., Xia, F., Luo, H., Li, Z., Yang, Y., Yu, H., Qu, X., Fu, Y., Su, J., Zhang, G., Huang, W., Wang, M., Yan, L., Jia, X., Liu, J., Ma, W.-Y., Zhang, Y.-Q., Wu, Y., and Zhou, H. Seed diffusion: A large-scale diffusion language model with high-speed inference, 2025. URL https://arxiv.org/abs/2508.02193.

Wang, Y., Yang, L., Li, B., Tian, Y., Shen, K., and Wang, M. Revolutionizing reinforcement learning framework for diffusion large language models. *arXiv preprint arXiv:2509.06949*, 2025.

Wang, Z., Zhang, R., Ding, K., Yang, Q., Li, F., and Xiang, S. Continuous speculative decoding for autoregressive image generation. *arXiv preprint arXiv:2411.11925*, 2024.

Wu, C., Zhang, H., Xue, S., Liu, Z., Diao, S., Zhu, L., Luo, P., Han, S., and Xie, E. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding, 2025. URL https://arxiv.org/abs/2505.22618.

Xie, Z., Ye, J., Zheng, L., Gao, J., Dong, J., Wu, Z., Zhao, X., Gong, S., Jiang, X., Li, Z., et al. Dream-coder 7b: An open diffusion language model for code. *arXiv preprint arXiv:2509.01142*, 2025.

Ye, J., Xie, Z., Zheng, L., Gao, J., Wu, Z., Jiang, X., Li, Z., and Kong, L. Dream 7b, 2025. URL https://hkunlp.github.io/blog/2025/dream.

Zhang, J., Wang, J., Li, H., Shou, L., Chen, K., Chen, G., and Mehrotra, S. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint arXiv:2309.08168*, 2023.

A. Losslessness

Let the current state of the sequence be $X(t_0) = X'; X_k(t_0)$. We wish to show that when the denoising progresses from $t_0 \to t_0 - 2$, we achieve the same result whether we had used only the true (target) probabilities or accepted a draft block using Spiffy and skipped to timestep $t_0 - 2$.

First, define the process of sampling the next state of a block X from some probability distribution p as a function `Next(`X`,` p`)`. This is simply the process of using p to performing ranking and picking of token positions to be unmasked in X described in `Steps 1-4` of Alg. 1 and is common to both speculative and non-speculative implementations.

A.1. Proof

Vanilla decoding output: Let $X_k^*(t_0 - 2)$ be the final state of the sequence with no speculation. From $t_0 \to t_0 - 1$, we may obtain the next state as:

$$p_{\text{TD}}(t_0 - 1) \leftarrow p_\theta\left(X_k(t_0 - 1) \mid X'; X_k(t_0)\right) \tag{15}$$

$$X_k(t_0 - 1) \leftarrow \texttt{Next}\left(X_k(t_0), p_{\text{TD}}(t_0 - 1)\right) \tag{16}$$

This state becomes the true state of the block, and inference is performed on it once more by the dLLM to advance to $t_0 - 2$

$$p_{\text{TD}}(t_0 - 2) \leftarrow p_\theta\left(X_k(t_0 - 2) \mid X'; X_k(t_0 - 1)\right) \tag{17}$$

$$X_k^*(t_0 - 2) \leftarrow \texttt{Next}\left(X_k(t_0 - 1), p_{\text{TD}}(t_0 - 2)\right) \tag{18}$$

Spiffy output: Let $X_k^+(t_0 - 2)$ be the state of the sequence if we first accept a draft block $\hat{X}_k^m$ at timestep t_m using Alg. 1, then treat it as the true block, and then continue to the next iteration of the algorithm where we perform unmasking once more.

First, in the case of speculation, we also sample $X_k(t_0 - 1)$ as usual in `Steps 1-4`, as is done in (16). The next step of the algorithm is to check for acceptance or rejection.

Case 1 (rejection): For any of the drafts $m \in \{1, \cdots, D\}$, if $\hat{X}_k^m$ is rejected, the algorithm continues to the next available draft and checks again for acceptance or rejection. If all drafts are rejected, we simply break and output $X_k(t_0 - 1)$ as the true block state. At this point, in the next iteration, we will call the dLLM once more and this state $X_k(t_0 - 1)$ will be used to compute p_{TD} as is done in (17). Thus, we obtain $X_k^+(t_0 - 2)$ in the same way, and from the same distribution, as $X_k^*(t_0 - 2)$ and thus they are equal.

Case 2 (acceptance): If $\hat{X}_k^m$ is accepted, we now set $p_{\text{TD}} \leftarrow p_{\text{DD}}^m(t_m - 1)$ and restart the algorithm. In this next step, we sample from this distribution to obtain

$$X_k^+(t_0 - 2) \leftarrow \texttt{Next}\left(\hat{X}_k^m(t_m - 1), p_{\text{DD}}^m(t_m)\right) \tag{19}$$

Since we previously had acceptance, we first have that $t_m == t_0 - 1$. Moreover, we have that $\hat{X}_k^m == X_k(t_0 - 1)$ from the acceptance condition. So we see that:

$$p_{\text{DD}}^m(t_m - 1) = p_{\text{DD}}^m(t_0 - 2)$$

and, since by definition

$$p_{\text{DD}}^m(t_m - 1) = p_\theta\left(X_k(t_m - 1) \mid X'; \hat{X}_k^m\right)$$

we have that

$$p_{\text{DD}}^m(t_0 - 2) = p_\theta\left(X_k(t_0 - 2) \mid X'; X_k(t_0 - 1)\right) \tag{20}$$

$$= p_{\text{TD}}(t_0 - 2) \tag{21}$$

So, in reality, we sample

$$
\begin{aligned}
&p^m_{\text{DD}}(t_0-2) \leftarrow \\
&\quad p_\theta\left(X_k(t_0-2) \mid X'; X_k(t_0-1)\right) \\
&\quad = p_{\text{TD}}(t_0-2) \qquad (22)\\
&X^+_k(t_0-2) \leftarrow \\
&\qquad \texttt{Next}\left(X^m_k(t_0-2), p^m_{\text{DD}}(t_0-2)\right) \\
&\quad = \texttt{Next}\left(X^m_k(t_0-2), p_{\text{TD}}(t_0-2)\right) \qquad (23)
\end{aligned}
$$

which is the same as obtaining $X^*_k(t_0-2)$ in (17) and (18)

□

A.2. Experimental Verification

We note that achieving the losslessness described above requires generating draft probabilities such that $p^m_{\text{DD}}(t_0-2) = p_{\text{TD}}(t_0-2)$. That is, the output distribution of the draft block should be identical to the distribution produced by the same block during true vanilla inference. In some settings, ensuring this is computationally expensive. Concretely, suppose we have 3 blocks X_1, X_2, X_3 and we are currently decoding $X_2(t)$ with a draft block $X^*_2(t-1)$. Since X^*_2 attends to the entire sequence (excluding X_2), but X_1 and X_3 in turn attend to X_2, the draft block is indirectly attending to the current block at a previous timestep t. This leakage of information would lead to drift in the output hidden states of the draft block across layers. However, two factors mitigate this deviation. First, X_1 has already been fully decoded in semi-autoregressive inference, so its hidden states remain stable during decoding (Wu et al., 2025). Second, since X_3 consists entirely of masked tokens, both X_2 and X^*_2 place low attention mass on this block. Due to these factors, the shift in output distribution is minimal.

In this framework, we may also conclude that when a dLLM is run with blockwise-causal masking, as is done in the SDAR model family (Cheng et al., 2025), the previous hidden states do not attend to future hidden states, which will ensure that the draft and true distribution remain identical since X_3 will not attend to X_2. Similarly, if a dLLM is run with dual KV caches, before and after the current block (Wu et al., 2025), the keys and values of X_1 and X_3 remain constant, resulting in an exact draft distribution.

We validate our claims by computing the downstream accuracy of the LLaDA, Dream, and SDAR models with and without Spiffy and comparing their resulting metric scores. These results are displayed in Table 1 and demonstrate Spiffy's ability to preserve the model's accuracy.

We further investigate minor differences in the metric values and attribute them to minor variations in generated outputs due to precision loss caused by floating-point matrix multiplication on CUDA GPUs. This is caused by differing accumulation orders for tensors of differing sizes. In particular, even though each of the 3 draft blocks used by Spiffy are independent, matrix multiplications with this extended tensor will have a slightly different accumulation order as compared to an input tensor with no draft blocks appended. This leads to minor differences in generations and due to the cumulative nature of errors, may lead to minor deviations in the final generations. For applications where precision is critical, we recommend padding the input tensor to 4096 to ensure predictable accumulation on CUDA GPUs in reduced precision. We do not adopt this method here as it is prohibitively expensive and minor differences do not affect the output significantly as demonstrated by Table 1. The PyTorch documentation on numerical accuracy may be referred to for further explanation of floating point error accumulation in such settings.

B. Extended Experiments

In Table 2, we use the graphs calibrated on 50 samples of MATH500 and MBPP and evaluate Spiffy on MMLU tasks from medicine, ethics, and law. We see that for diverse evaluation domains, Spiffy's calibration algorithm produces draft graph topologies that effectively capture the dLLM's decoding behavior, resulting in consistent acceleration.

Table 2. Spiffy results in consistent NFE reductions and TPS speedup with preserved accuracy across tasks from medicine, ethics, and law.

	MMLU (clinical knowledge)	MMLU (moral scenarios)	MMLU (professional law)
Baseline (LLaDA-8B-Instruct)	1.00× (1.00×), 0.55	1.00× (1.00×), 0.35	1.00× (1.00×), 0.31
+ Dynamic Unmasking	1.87× (1.89×), 0.54	1.62× (1.60×), 0.36	1.52× (1.47×), 0.32
+ Spiffy	3.18× (2.38×), 0.52	2.82× (2.07×), 0.38	2.71× (1.93×), 0.28

C. Overheads of Spiffy

In this section, we examine the overheads of various stages of Spiffy including drafting, custom attention mask construction, custom position ID construction, and the increase in model inference time due to the extra computation of the draft blocks. We normalize these values to be a percentage of the vanilla model inference time for ease of comparison. This data is collected with LLaDA-8B-Instruct using the MBPP dataset, with the average value reported. All profiling was done using a single 80GB NVIDIA A100 GPU with `torch==2.6.0` and `transformers==4.52.4`, with vectorized PyTorch operations and `torch.nn.scaled_dot_product_attention` with the `SDPBackend.MATH` kernel.

We see in Table 3 that the costs of the various stages of generation with Spiffy are low compared to the model inference time, with drafting in total taking $\sim 6\%$ of the model inference time and draft acceptance taking $\sim 5\%$ of the model inference time. These costs may be reduced via further optimizations to the codebase including additional vectorization and multi-processing for latency-critical scenarios. The model forward pass is 25% more expensive than vanilla inference due to the extra computational requirements and may be mitigated by implementing a custom attention kernel designed for Spiffy's custom attention mask.

Table 3. Overheads of Spiffy, normalized to a % of the model inference time for # draft blocks $D = 3$. $D = 0$ corresponds to vanilla inference. All values are normalized to be a percentage of the vanilla model inference time.

Overheads as % of model call time		
	Number of draft blocks	
	0	3
Model	100.00	124.85
Draft Graph Construction	0.00	0.71
Draft Pruning	0.00	1.77
Draft Sequence Construction	0.00	0.76
Attention Mask Construction	0.00	0.16
Position IDs Construction	0.00	0.05
Draft Acceptance	0.00	6.18

D. Calibration of Directed Draft Graphs

D.1. Intuition

In Alg. 2, we first establish a set of graph candidates G' that satisfy our constraints of budget and connectedness:

$$\text{A node } q_{kv} \text{ is } \textbf{connected} \text{ to a graph } g \text{ if } k \in \{1, 2\} \text{ or } parents(q_{kv}) \cap g \neq \varnothing \tag{24}$$

$$\text{A graph } g \in G' \text{ is connected if it consists of nodes } \{q_{kv}\} \text{ which are connected} \tag{25}$$

$$G' \leftarrow \{g \in G \mid g \text{ is connected AND } |g| = D\} \tag{26}$$

Intuitively, this means that to be selected, a node must be in the first two levels or descended from a node in the first two levels. The first two levels account for cases in which multi-token decoding is the most conservative. Hence, prioritizing having draft states for such cases is useful. With this constraint, we optimize the following expression:

$$G^* = \arg\max_{Q \in G'} \sum_{q \in Q} (count(q)) \tag{27}$$

Each node $q \in Q$ represents a sequence of tokens that has appeared in order. For example, $q = (c_{11}, c_{21}, c_{31})$. $count(q)$ thus represents how often this particular sequence has occurred in the collected calibration data.

Since we create the full graph G by filtering for the top nodes per level and the top in_edges per level, we are implicitly introducing parent-child frequency relationship. Moreover, since we may only choose graphs that are connected, we are reinforcing this cross-node connection. For example, (c_{11}, c_{12}) or (c_{11}, c_{31}) or (c_{21}, c_{31}) must have first occurred for (c_{11}, c_{21}, c_{31}) to have occurred. So a node is more likely to have occurred if its parents are likely to occur. Thus, using a simple frequency scoring metric, as shown above, suffices to represent the overall score of a candidate draft graph.

D.2. Time to Perform Calibration

In our experiments calibrating a draft graph for LLaDA-8B-Instruct on 50 samples, the time to process a single sample is $\sim$ 9 seconds for a total calibration time of $\sim$ 7 minutes on a single NVIDIA A100 GPU. The time taken to perform the optimization step in (27) using a brute-force breadth-first-search approach is $\sim$ 3 minutes. In sum, calibration is an inexpensive, 10-minute procedure that can lead to high-quality draft graph structures.

D.3. Calibration Domain

In Table 4, we analyze Spiffy's acceleration when the calibration dataset source is changed from MATH500 and MBPP to ShareGPT. 50 samples total from these datasets are used in either case. We see that Spiffy's calibration algorithm is able to effectively capture the model's underlying decoding dynamics regardless of the calibration domain, resulting in consistent speedups.

Table 4. Regardless of the calibration domain, Spiffy results in NFE reductions and TPS speedup with preserved accuracy across datasets.

	GSM8K	HumanEval	MATH500	MBPP
Baseline (LLaDA-8B-Instruct)	1.00× (1.00×), 0.79	1.00× (1.00×), 0.41	1.00× (1.00×), 0.34	1.00× (1.00×), 0.37
+ Dynamic Unmasking	3.31× (3.43×), 0.79	3.53× (3.13×), 0.41	2.52× (2.64×), 0.35	5.73× (5.46×), 0.37
+ Spiffy (MATH500-MBPP calib.)	4.96× (3.79×), 0.79	5.85× (3.59×), 0.38	4.11× (3.15×), 0.35	8.58× (5.89×), 0.37
+ Spiffy (ShareGPT calib.)	4.97× (3.73×), 0.79	5.85× (3.49×), 0.39	4.12× (3.07×), 0.35	8.61× (5.72×), 0.37

D.4. Calibration Algorithm Pseudocode

Algorithm 3 Draft Graph Calibration Pseudocode

```
def calibrate_draft_graph(dLLM, dataset, lookahead L, draft_budget D):

    sequences = [record_unmasking_trajectory(dLLM, sample, L)
                 for sample in dataset]
    # Each trajectory = [ (lookahead_depth, { (position_rank(i), vocab_rank(j)) } ) ]

    G = build_candidate_graph(sequences, L)
    # G = graph with L levels
    # Nodes = most frequent unmasked token sets at each level = lookahead depth
    # Eg. level 1 = (1, 1), (2, 1)
    # Edges = most frequent token set transitions
    # Eg. {(1, 1)} -> {(1, 1), (2, 1), (3, 1)}

    G_opt = select_best_subgraph(G, D)
    # G* = argmax_(Q in {size-D connected subgraphs of G}) (sum of node frequencies of Q)

    return G_opt
    # list of D draft nodes, each node is a set of {(pos_rank, vocab_rank)}
```