Free Draft-and-Verification: Toward Lossless Parallel Decoding for Diffusion Large Language Models

Shutong Wu **Jiawei Zhang**
Department of Computer Sciences, University of Wisconsin – Madison
{swu494, jzhang2924}@wisc.edu

Abstract

Diffusion Large Language Models (DLLMs) have emerged as a new paradigm of language modeling beyond autoregressive next-token prediction. Thanks to their bidirectional attention mechanism, DLLMs are more capable of capturing the connection of context, and thus show unique advantages in challenges like the famous "reversal curse" or learning under data-constrained scenarios. On the other hand, taking advantage of their inherent modeling foundations, parallel decoding algorithms enable multi-token prediction per step for DLLMs, which can accelerate the inference to the next level. However, the high generation quality often requires a large number of decoding steps, which is usually equal to the sequence length, and parallel decoding brings inference speedup at the cost of non-negligible performance degradation. To overcome this challenge, we introduce **Free D**raft-**a**nd-**V**erification (**FreeDave**), a novel fast sampling algorithm tailored for DLLMs that achieves lossless parallel decoding. Specifically, we propose a pipeline of parallel-decoded candidate generation and verification, which is guaranteed to reproduce the same sequence generated by static decoding, without external modules, extra model forward calls, or any post-training stage. By extensive evaluations on math reasoning and code generation benchmarks across different DLLMs, FreeDave is proven to boost the inference throughput up to $3.78\times$ without performance degradation. Code is available at `https://github.com/cychomatica/FreeDave`.

1 Introduction

The advent of Large Language Models (LLMs) [1, 2, 3, 4, 5, 6] built on large transformers[7] and autoregressive (AR) next-token prediction has marked a revolutionary milestone on the way to artificial general intelligence. They have demonstrated an extraordinary capacity for text processing and generation, achieving near-human performance on a vast spectrum of natural language tasks. In recent years, their capabilities have been extending far beyond basic text completion, showing remarkable proficiency in highly specialized and challenging tasks. For instance, LLMs have made significant strides in mathematical reasoning [8, 9, 10], which requires deep logical and symbolic understanding. Similarly, in the domain of software engineering, LLMs have shown promising achievements on code generation [6, 11, 12, 13, 14], offering powerful tools to automate development tasks and improve development productivity.

Diffusion Large Language Models (DLLMs) [15, 16, 17] have emerged as a compelling new paradigm of language modeling. Inspired by their profound success in continuous data domains such as high-fidelity image and audio synthesis [18, 19, 20], diffusion models have been adapted to the discrete domain, especially language modeling [21, 22, 23, 24, 25, 26]. Due to their bidirectional attention mechanism, DLLMs are more capable of capturing the connection of context, and thus show their unique advantages in challenges like the famous "reversal curse" [15] or learning under data-constrained scenarios [27]. On the other hand, taking advantage of their inherent modeling

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning.

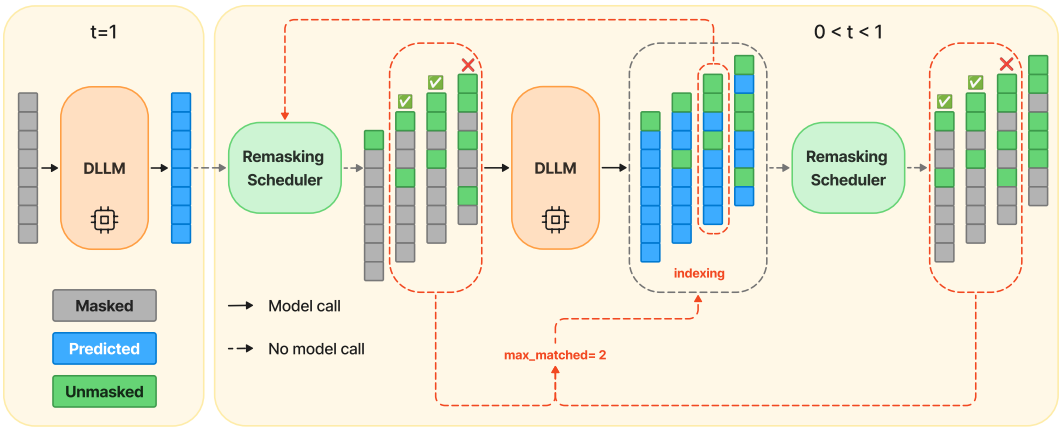

Figure 1: The overview of FreeDave decoding for DLLMs. Based on the estimated distribution predicted by the DLLM at the current step, the remasking scheduler looks multiple steps ahead and returns multiple draft candidates at those timesteps. Then, at the next step, the DLLM takes those candidates as a batch of inputs in parallel and gets the estimated distribution for each candidate, which is further processed by the remasking scheduler for one more step to get a target sequence. The candidates, as well as their estimated distributions, are then accepted or rejected by matching their targets. The generation and verification of the draft candidates can be understood as byproducts during the normal static decoding without introducing extra cost, except for a slight memory overhead from the batch forward. Empirically, with a high potential, the inference will get an appreciable speedup.

foundations, parallel decoding algorithms [28, 29] enable multi-token prediction per step for DLLMs, which can accelerate their inference to the next level.

However, despite their great potential, the practical application of DLLMs is hampered by the challenge that a high generation quality often requires a large number of decoding steps, which is usually equal to the sequence length, and parallel decoding brings inference speedup at the cost of non-negligible performance degradation. On the other hand, without parallel decoding, the inference efficiency is limited due to their bidirectional attention mechanism, which is incompatible with KV Cache. Although specialized caching strategies[26, 28, 30] are proposed as a solution, the necessity of frequently refreshing the KV Cache remains a drag on the inference speed. Thus, unleashing the great potential of parallel decoding without sacrificing the generation quality remains a critical challenge for DLLMs.

To cross this dilemma, we introduce **Free D**raft-**a**nd-**Ve**rification (**FreeDave**), a novel fast sampling algorithm for DLLMs that achieves lossless parallel decoding. As a training-free and model-free method, FreeDave requires no model modification or any extra modules, and can be seamlessly incorporated with any existing efficient inference frameworks like DLLM caching[26, 28, 30]. Taking advantage of the inherent capability of parallel decoding of DLLMs, and inspired by speculative decoding applied on autoregressive LLMs for fast sampling[31], we reveal that:

Your DLLM is secretly a self-verifiable parallel decoder without extra cost.

We prove, by both theoretical analysis and empirical evaluations, that FreeDave can substantially accelerate the inference of DLLMs without sacrificing the generation quality. Specifically, for TraDo models[17], currently the state-of-the-art DLLM family, FreeDave can boost the throughput up to 3.78×, while maintaining the performance on math reasoning (including MATH500[32], GSM8K[33], and AIME2024[34]) and code generation (including HumanEval[35] and MBPP[36]) benchmarks.

2 Related Work

2.1 Diffusion Large Language Models

Diffusion Large Language Models (DLLMs) emerge as a new language modeling paradigm beyond autoregressive next token prediction. Early foundational frameworks like D3PM[21] generalize diffusion models from the continuous domain to discrete state-spaces, and a key finding was that using a special token as an absorbing state consistently yielded the best performance. This paved the way for Masked Diffusion Language Models (MDLMs)[22, 23], with the simplification of the complex Evidence Lower Bound (ELBO) training objective into a more stable weighted mixture of masked language modeling (MLM) losses. Recent studies on DLLMs [15, 24, 16, 17], which scale up to billions of parameters, have been making remarkable progress on closing the performance gap with AR LLMs.

Compared to AR LLMs, the bidirectional nature of DLLMs allows them to overcome challenges like the "reversal curse"[37]. Furthermore, recent studies have shown that DLLMs are significantly more data-efficient in data-constrained settings[38]. On the other hand, the primary drawback of DLLMs remains their slower inference speed due to the inherent incompatibility with KV Cache, which also stems from their bidirectional nature. To address this problem, strategies like Block Diffusion[26] interpolate between AR and diffusion paradigms by generating text in blocks, enabling the use of inter-block KV caching. dLLM-Cache[30] introduces an adaptive caching mechanism by exploiting the distinct redundancies in the internal features of DLLMs. Fast-dLLM[28] incorporates a novel block-wise approximate KV Cache mechanism and a confidence-aware parallel decoding strategy to further bridge this gap in inference speed with AR LLMs.

2.2 Parallel Decoding

Before the advent of DLLMs, to accelerate the inference of AR LLMs, speculative decoding [31, 39, 40, 41, 42] is designed to perform multi-token decoding in parallel with verifiable acceptance. Generally, a lightweight draft model, which is much faster than the LLM, autoregressively performs next-token prediction for multiple steps to generate multiple candidates, and then those candidates are fed to the target model, *i.e.* the LLM itself, for verification, usually by rejection sampling, to get accepted or rejected. After that, the process of candidate generation and verification is repeated, starting from the first rejected token in the last turn. However, the draft model is required to be aligned with the target models as much as possible, otherwise, the low acceptance rate of candidates will instead bring inefficiency to the LLM inference by introducing extra draft model calls while being unable to appreciably reduce the LLM calls.

For DLLMs, speculative decoding can be directly adopted in a similar manner, which still requires an external draft model and an extra draft generation stage [43]. On the other hand, parallel decoding tailored to the unique characteristics of DLLMs is a more promising research line. A popular method is threshold-based parallel decoding: at each step, all unmasked positions where the confidence of the predicted token exceeds a predefined threshold will be unmasked [28, 29]. However, compared with static decoding that unmasks a fixed number of tokens with the highest confidence at each step, threshold-based parallel decoding usually brings a non-negligible drop in generation quality. Israel et al. [44] introduce Adaptive Parallel Decoding, which utilizes the DLLM itself and an external autoregressive model to craft a multiplicative mixture of distributions, and then compare the parallel-decoded tokens from the DLLM with tokens sampled from this multiplicative mixture to decide which tokens should be accepted. However, it will still lead to a performance drop when the mixture weight is too large, and the trade-off between throughput and generation quality remains.

3 Methodology

3.1 Preliminaries

Autoregressive language models In the past years, autoregressive (AR) language modeling has been the predominant paradigm of LLMs. The joint distribution of generated tokens is modeled by

$$p_\theta(x) = \prod_{i=1}^{L} p_\theta(x^i | x^{<i}), \quad x^{<i} = \begin{cases} \{x^j | j = 1, \ldots, i-1\} & \text{if } i \geq 2 \\ \emptyset & \text{if } i = 1 \end{cases}, \tag{1}$$

where x is a response sequence of length L, x^i is the one-hot vector of i-th token in the vocabulary space, and $x^{<i}$ is the subsequence of all tokens before the i-th token.

Diffusion language models Like diffusion models on continuous spaces, DLLMs model the transition from data distribution to noise distribution on discrete spaces by a forward noising process, and generate data from noise by the corresponding reverse denoising process. Generally, the forward process diffuses the data distribution toward a uniform distribution over the discrete space, or an absorbing state, and the latter is usually adopted for language modeling[22, 23, 15, 16, 17]. Specifically, for a time level t evolving from 0 to 1, the forward process progressively replaces each unmasked token in the original sequence x_0 independently with a special mask token **m** with an increasing probability *w.r.t.* $t \in [0, 1]$, and once a token is masked, it will remain masked till $t = 1$. Eventually, at $t = 1$, all the tokens in the sequence will be **m**. This forward process with the absorbing state **m** can be formulated as

$$q(x_t|x_0) = \prod_{i=1}^{L} q(x_t^i|x_0^i) = \prod_{i=1}^{L} \text{Cat}\left(x_t^i; \alpha_t x_0^i + (1-\alpha_t)\mathbf{m}\right), \tag{2}$$

where x_t is the transferred sequence at the time level t, α_t is the noising schedule monotonically decreasing *w.r.t.* $t \in [0, 1]$ and satisfies $\alpha_0 = 1$ and $\alpha_1 = 0$, and Cat represents the categorical distribution.

The reverse process is induced to progressively recover the original sequence from an all-mask sequence at $t = 1$. Specifically, the reverse process transferring the sequence x_t at a time level t conditioned on x_0 to the sequence x_s at an earlier time level s (where $0 \le s < t \le 1$) can be formulated as

$$q(x_s|x_t, x_0) = \prod_{i=1}^{L} q(x_s^i|x_t^i, x_0^i) \tag{3}$$

$$q(x_s^i|x_t^i, x_0^i) = \begin{cases} \text{Cat}(x_s^i; x_t^i) & \text{if } x_t^i \neq \mathbf{m} \\ \text{Cat}(x_s^i; \frac{(1-\alpha_t)\mathbf{m} + (\alpha_s - \alpha_t)x_0^i}{1-\alpha_t}) & \text{if } x_t^i = \mathbf{m} \end{cases}, \tag{4}$$

and once a token is unmasked, it will remain unchanged till $t = 0$. If we train a parameterized model $f_\theta(\cdot, \cdot) : \mathbb{R}^{L\times|V|} \times \mathbb{R} \to \mathbb{R}^{L\times|V|}$ to estimate the distribution of x_0 by $f_\theta(x_t, t)$, given the sequence x_t at time level t, the reverse process can be formulated as

$$p_\theta(x_s^i|x_t^i) = q(x_s^i|x_t^i, x_0^i = f_\theta(x_t^i, t)) = \begin{cases} \text{Cat}(x_s^i; x_t^i) & \text{if } x_t^i \neq \mathbf{m} \\ \text{Cat}(x_s^i; \frac{(1-\alpha_t)\mathbf{m} + (\alpha_s - \alpha_t)f_\theta^i(x_t, t)}{1-\alpha_t}) & \text{if } x_t^i = \mathbf{m} \end{cases} \tag{5}$$

$$p_\theta(x_s|x_t) = \prod_{i=1}^{L} p_\theta(x_s^i|x_t^i) \tag{6}$$

Generally, for better generation quality, the transition from x_t to x_s is not by directly sampling x_s from $p_\theta(x_s|x_t)$. Instead, we have a remasking scheduler g (usually rule-based and unparameterized) to decide which **m** tokens to be transferred to unmasked tokens at the current time level. [1] The remasking scheduler g takes as input a source time level t, a target time level s, and the estimated distribution of x_0 at the source time level t (*i.e.* $f_\theta(x_t, t)$ here), and returns a decision set $\mathcal{I}_{t\to s} = \{(a_1, z_1), \ldots, (a_n, z_n)\}$, where in each tuple a_j is the transferred index and $z_j \sim \text{Cat}(f_\theta^{a_j}(x_t, t))$ is the transferring target token sampled from the estimated distribution. Algorithm 1 demonstrates the whole sampling pipeline by the reverse process with model f_θ. Generally, for a high generation quality, the total number of sampling steps is set equal to the response length, and at each step, among all the masked positions, the one where the predicted token has the highest confidence is unmasked. We denote this decoding strategy as static decoding.

[1] Empirically, greedily unmasking positions with the highest-confidence unmasked tokens usually yields good quality of generation. Unless specified, in this paper, we focus on the greedy remasking scheduler.

Algorithm 1 DLLM Static Sampling

Require: Model f_θ, remasking scheduler g, total generation length L, total sampling steps N, time schedule array $\mathbf{T} = \{t_0, \ldots, t_N | 1 = t_0 > t_1 > \cdots > t_{N-1} > t_N = 0\}$
Ensure: The generated sequence x.

$x_{t_0} \leftarrow \mathbf{m}^{1:L}$
for i in $0 : N-1$ **do**
  $\tilde{x}_0 \leftarrow f_\theta(x_{t_i}, t_i)$
  $\mathcal{I}_{t_i \to t_{i+1}} \leftarrow g(\tilde{x}_0, t_i, t_{i+1})$ ▷ decision returned by the remasking scheduler
  $x_{t_{i+1}} \leftarrow x_{t_i}$
  for each (a_j, z_j) in $\mathcal{I}_{t_i \to t_{i+1}}$ **do**
    $x_{t_{i+1}}^{a_j} \leftarrow z_j$ ▷ replace masked tokens according to the decision
  end for
end for
$x \leftarrow x_0$
return x

3.2 Free Draft-and-Verification Sampling

Definition 1 (Sampling Path and Oracle Path) *For the reverse process of DLLM f_θ, a sampling path of N steps following a time schedule array $[t_0, \ldots, t_N | 1 = t_0 > t_1 > \cdots > t_{N-1} > t_N = 0]$ is defined as $\mathbf{p} = [\mathcal{I}_{t_0 \to t_1}, \ldots, \mathcal{I}_{t_{N-1} \to t_N}]$, where $\mathcal{I}_{t_i \to t_{i+1}}$ is the decision set given at time step t_i. Given a remasking scheduler g and a predefined time schedule array $\mathbf{T} = [t_0, \ldots, t_N | 1 = t_0 > t_1 > \cdots > t_{N-1} > t_N = 0]$, we define the oracle path as $\mathbf{p}_{oracle} = [\mathcal{I}_{t_0 \to t_1}, \ldots, \mathcal{I}_{t_{N-1} \to t_N}]$, where $\mathcal{I}_{t_i \to t_{i+1}} = g(f_\theta(x_{t_i}, t_i), t_i, t_{i+1})$ is the decision set returned by the remasking scheduler g at time step t_i.*

Definition 2 (Feasible Path and Optimal Path) *For the reverse process of DLLM f_θ, given a remasking scheduler g and a predefined time schedule array $\mathbf{T} = [t_0, \ldots, t_N | 1 = t_0 > t_1 > \cdots > t_{N-1} > t_N = 0]$, if we can find a M-step subarray $[t_{a_0}, \ldots, t_{a_M} | 0 = a_0 < a_1 < \cdots < a_{M-1} < a_M = N, a_i \in \{0, \ldots, N\}]$, such that*

$$
\begin{aligned}
&\mathcal{I}_{t_{a_i} \to t_{a_{i+1}}} \\
&= g(f_\theta(x_{t_{a_i}}, t_{a_i}), t_{a_i}, t_{a_{i+1}}) \\
&= \bigcup_{j=0}^{a_{i+1}-a_i-1} g(f_\theta(x_{t_{a_i+j}}, t_{a_i+j}), t_{a_i+j}, t_{a_i+j+1}) \\
&= \bigcup_{j=0}^{a_{i+1}-a_i-1} \mathcal{I}_{t_{a_i+j} \to t_{a_i+j+1}} \\
&\forall i \in \{0, \ldots, M-1\},
\end{aligned}
\tag{7}
$$

then the sampling path $\mathbf{p} = [\mathcal{I}_{t_{a_0} \to t_{a_1}}, \ldots, \mathcal{I}_{t_{a_{M-1}} \to t_{a_M}}]$ is defined as a feasible path. And consequently, we define $\mathcal{P}(f_\theta, g, \mathbf{T})$, which consists of all possible feasible paths, as the feasible space. There exists an optimal path $\mathbf{p}^\star = [\mathcal{I}_{t_{a_0^\star} \to t_{a_1^\star}}, \ldots, \mathcal{I}_{t_{a_{M-1}^\star} \to t_{a_M^\star}}]$ such that $|\mathbf{p}^\star| \leq |\mathbf{p}| \leq |\mathbf{p}_{oracle}|, \forall \mathbf{p} \in \mathcal{P}(f_\theta, g, \mathbf{T})$, i.e. *among all feasible paths, the optimal path requires the fewest sampling steps, while the oracle path requires the most sampling steps.*

Definition 1 and 2 indicate the great potential of DLLMs for lossless parallel decoding. However, except for the oracle path $\mathbf{p}_{oracle}$, all of the other feasible paths are agnostic to us. We don't know which steps can be merged into one step. Empirically setting a fixed number of transferred tokens or a fixed confidence threshold for parallel decoding is not guaranteed to get a feasible path that generates the same sequence as sampling via the oracle path, and the experimental results from both Wang et al. [17] and us demonstrate that threshold-based parallel decoding is very likely to result in a non-negligible performance degradation.

Theorem 1 (Verification-based Feasible Path Search) *In the feasible space $\mathcal{P}(f_\theta, g, \mathbf{T})$, suppose we have a verifier function h defined as*

$$h(t_n, d) = \begin{cases} 1, & \text{if } d = 1 \\ \min\{i | \mathcal{I}_{t_n \to t_{n+i+1}} \neq \mathcal{I}_{t_n \to t_{n+i}} \cup \mathcal{I}_{t_{n+i} \to t_{n+i+1}}, i = 1, \ldots, d-1\} & \text{if } d \geq 2 \end{cases} \tag{8}$$

where t_n is a timestep and d is a positive integer. Then we can find a feasible path $\mathbf{p} = [\mathcal{I}_{t_{a_0} \to t_{a_1}}, \ldots, \mathcal{I}_{t_{a_{M-1}} \to t_{a_M}}]$, where $a_0 = 0$, $a_M = N$, and $a_i = h(t_{a_{i-1}}, d), i \in \{1, \ldots, M-1\}$.

Lemma 1 *If we have $d \geq \|\mathbf{p}^\star\|_\infty$ where $\mathbf{p}^\star = [\mathcal{I}_{t_{a_0^\star} \to t_{a_1^\star}}, \ldots, \mathcal{I}_{t_{a_{M-1}^\star} \to t_{a_M^\star}}]$ is the optimal path, and $\|\mathbf{p}^\star\|_\infty = \max\{a_{i+1}^\star - a_i^\star | i = 0, \ldots, |\mathbf{p}^\star| - 1\}$, then $a_i^\star = h(t_{a_{i-1}^\star}, d), i \in \{1, \ldots, M-1\}$.*

Proof Sketch The proof of this Theorem 1 is quite intuitive. Starting from any t_n, if $d = 1$, we find just the oracle path, which is also a feasible path; if $d \geq 2$, then from Equation 8 we have $\mathcal{I}_{t_n \to t_{n+i+1}} = \mathcal{I}_{t_n \to t_{n+i}} \cup \mathcal{I}_{t_{n+i} \to t_{n+i+1}} \forall i \in \{1, \ldots, h(t_n, d) - 1\}$. Thus, we have $\mathcal{I}_{t_n \to t_{n+2}} = \mathcal{I}_{t_n \to t_{n+1}} \cup \mathcal{I}_{t_{n+1} \to t_{n+2}}$, $\mathcal{I}_{t_n \to t_{n+3}} = \mathcal{I}_{t_n \to t_{n+2}} \cup \mathcal{I}_{t_{n+2} \to t_{n+3}} = \mathcal{I}_{t_n \to t_{n+1}} \cup \mathcal{I}_{t_{n+1} \to t_{n+2}} \cup \mathcal{I}_{t_{n+2} \to t_{n+3}}$, …, and recursively, we get $\mathcal{I}_{t_n \to t_{n+h(t_n,d)}} = \bigcup_{i=1}^{h(t_n,d)-1} \mathcal{I}_{t_{n+i} \to t_{n+i+1}}$, which by Definition 2 induces a feasible path. For Lemma 1, if we have a large enough d that covers the largest number of decoded tokens, we can reproduce this optimal path by Theorem 1.

Given a verifier h satisfying Equation 8, Theorem 1 provides the guarantee of searching a lossless, shorter (or equal) sampling path, and Lemma 1 indicates the condition of finding the optimal path, *i.e.* d should be large enough to cover the size of the largest decision set in the optimal path. As $\mathbf{p}^\star$ is agnostic to us, we can find the sequence length as an upper bound such that $L \geq \|\mathbf{p}^\star\|_\infty$. As long as $d \geq L$, we are guaranteed to find the optimal path.

Based on our theoretical analysis above, we propose Free Draft-and-Verification (FreeDave), a fast sampling algorithm explicitly designed for DLLMs. Specifically, as shown in Figure 1 and Algorithm 2, FreeDave iteratively utilizes the remasking scheduler to sample draft candidates with different parallel decoding steps based on the estimated distribution predicted by the DLLM at the current step, and further verifies those candidates by the DLLM itself in the next step.

Unlike speculative decoding algorithms designed for AR LLMs[31, 39], which also involve the generation and verification of draft candidates, FreeDave does not require an external draft model and an independent stage of draft candidate generation. Instead, the additional operations of candidates drafting and verification are just the byproducts of the normal inference pipeline, and thus can be seamlessly incorporated. From Definition 2, if we can find a feasible path with M steps, then $M \leq N = |\mathbf{p}_{oracle}|$ is guaranteed. And by Theorem 1 we are guaranteed to find a feasible path. On the assumption of enough computation and memory budget, the time cost of a model f forward call for a single example and a batch of examples should be almost equivalent, and we denote it as $\tau(f)$, and consequently $M\tau(f) \leq N\tau(f)$. Since there is no external draft model, the FreeDave decoding latency is just $M\tau(f)$, which is proven to be less than or equal to the DLLM static decoding latency $N\tau(f)$.

On the other hand, for an AR LLM f generating a sequence of length L, the latency of the normal AR decoding is $L\tau(f)$. After applying speculative decoding with a draft model f_{draft}, the number of target model forward calls N_{target} is proven to be less than or equal to L, which implies $N_{target}\tau(f) \leq L\tau(f)$. However, when the acceptance rate is not high enough, the number of draft model forward calls N_{draft} can be very large, and the latency from draft model $N_{draft}\tau(f_{draft})$ will no longer be negligible. Consequently, the total speculative decoding latency $N_{target}\tau(f) + N_{draft}\tau(f_{draft})$ is not guaranteed to be less than or equal to the AR decoding latency $L\tau(f)$. Without this guarantee, it is possible that speculative decoding will instead hamper the inference efficiency.

4 Experiments

4.1 Experimental Settings

Our evaluations focus on math reasoning and code generation tasks. For math reasoning benchmarks, we choose MATH500 [32], GSM8K [33], and AIME2024 [34]. For code generation benchmarks,

Algorithm 2 DLLM FreeDave Sampling

Require: Model f_θ, remasking scheduler g, total generation length L, total sampling steps N, draft steps d, time schedule array $\mathbf{T} = \{t_0, \ldots, t_N | 1 = t_0 > t_1 > \cdots > t_{N-1} > t_N = 0\}$
Ensure: The generated sequence x.

$i \leftarrow 0$
$x_{t_0} \leftarrow \mathbf{m}^{1:L}$, $\tilde{x}_0 \leftarrow f_\theta(x_{t_0}, t_0)$ ▷ initialize sequence and its estimated distribution
while $i < N$ **do**
  $d_i \leftarrow \min(d, N - i)$ ▷ clip initial draft steps when no enough steps left
  $\mathbf{X}_{draft} \leftarrow [\,]$
  for k in $1 : d_i$ **do**
    $\mathcal{I}_{t_i \to t_{i+k}} \leftarrow g(\tilde{x}_0, t_i, t_{i+k})$ ▷ decision for each draft candidate
    $x_{draft,t_{i+k}} \leftarrow x_{t_i}$
    for each (a_j, z_j) in $\mathcal{I}_{t_i \to t_{i+k}}$ **do**
      $x^{a_j}_{draft,t_{i+k}} \leftarrow z_j$ ▷ replace masked tokens according to decision
    end for
    $\mathbf{X}_{draft}$.append($x_{draft,t_{i+k}}$)
  end for
  if $i = N - 1$ **then**
    $x_0 \leftarrow \mathbf{X}^0_{draft}$ ▷ no verification needed in last step; sampling ends
    break
  end if

  $\tilde{\mathbf{X}}_{draft,0} \leftarrow f_\theta(\mathbf{X}_{draft}, t_{i+1:i+d_i})$ ▷ batch forward in parallel
  $\mathbf{X}_{target} \leftarrow [\,]$
  for $k = 1 : d_i$ **do**
    $\mathcal{I}_{t_{i+k} \to t_{i+k+1}} \leftarrow g(\tilde{\mathbf{X}}^k_{draft,0}, t_{i+k}, t_{i+k+1})$ ▷ one more step ahead each draft candidate
    $x_{target,t_{i+k+1}} \leftarrow \mathbf{X}^k_{draft}$
    for each (a_j, z_j) in $\mathcal{I}_{t_{i+k} \to t_{i+k+1}}$ **do**
      $x^{a_j}_{target,t_{i+k+1}} \leftarrow z_j$ ▷ replace masked tokens according to decision
    end for
    $\mathbf{X}_{target}$.append($x_{target,t_{i+k+1}}$)
  end for

  $m \leftarrow 0$
  for k in $1 : d_i - 1$ **do**
    if $\mathbf{X}^{k+1}_{draft} = \mathbf{X}^k_{target}$ **then** ▷ draft candidate verification
      $m \leftarrow m + 1$
    else
      break
    end if
  end for

  $i \leftarrow i + m + 1$ ▷ jump to t_{i+m+1}
  $x_{t_i} \leftarrow \mathbf{X}^{m+1}_{draft}$, $\tilde{x}_0 \leftarrow \tilde{\mathbf{X}}^{m+1}_{draft,0}$ ▷ update sequence and its estimated distribution
end while
$x \leftarrow x_0$
return x

we choose MBPP [36] and HumanEval [35]. We evaluate different DLLMs, including Dream-7B-Instruct [16], TraDo-4B-Instruct, and TraDo-8B-Instruct [17]. All the models use semi-autoregressive generation like Arriola et al. [26] and have already enabled specialized KV Cache. We compare our method with **static decoding**, which follows the oracle path induced from a predefined time schedule, and **parallel decoding** [28, 29], which at each step unmasks all positions where the transferring target token confidence exceeds a predefined threshold. Our method is built on static decoding by default.

Following Wang et al. [17], for the Dream-7B-Instruct model, we use a temperature of 0.1, a further horizon size of 128, and a response limit of 1600. For parallel decoding, we use a block size of 4 with

a threshold of 0.95. Under static decoding, we use a block size of 32. For the TraDo models, we use a temperature of 1.0, a default block size of 4, and a response limit of 2048. For parallel decoding, we use a threshold of 0.9 and set top-k=0, and for static decoding, we use top-k = 1. We also use the same prompt templates as Wang et al. [17] for different models, respectively. All the models are evaluated on a single NVIDIA L40S GPU.

Regarding the number of draft steps d used for FreeDave, we set $d = 8$ for the TraDo models, and $d = 4$ for the Dream-7B-Instruct model. According to Theorem 1 and Lemma 1, a larger d will be more likely to find a shorter feasible path. Practically, if d is too large, although we can get a high throughput over NFEs, a model forward call on a large batch of inputs might take more time and memory and thus degrade the throughput over time, as the computation and memory budget is limited. In Section 4.4, we investigate how different d can impact GPU memory usage, inference time, and NFEs. Here, our choice of d can achieve a sweet spot between those metrics.

Table 1: Detailed comparison of performance and efficiency with static decoding, parallel decoding, and FreeDave decoding for different DLLMs on math reasoning benchmarks, including MATH500, GSM8K, and AIME2024.

Benchmark	Model	Decoding Strategy	Acc (%) ↑	Throughput over time (#tokens/s) ↑	Throughput over NFEs (#tokens) ↑
MATH500	Dream-7B-Instruct	Static	40.00	23.02	0.91
		Parallel	37.00 (-3.00)	16.73 (0.73×)	1.88 (2.07×)
		FreeDave	**40.20 (+0.20)**	**30.00 (1.30×)**	**2.63 (2.89×)**
	TraDo-4B-Instruct	Static	74.20	7.26	0.26
		Parallel	68.80 (-5.40)	**18.94 (2.61×)**	0.61 (2.35×)
		FreeDave	**76.40 (+2.20)**	16.36 (2.25×)	**0.67 (2.58×)**
	TraDo-8B-Instruct	Static	76.40	7.10	0.28
		Parallel	74.00 (-2.40)	**16.11 (2.27×)**	0.60 (2.14×)
		FreeDave	**77.60 (+1.20)**	15.99 (2.25×)	**0.66 (2.36×)**
GSM8K	Dream-7B-Instruct	Static	79.61	20.99	0.83
		Parallel	68.16 (-11.45)	16.61 (0.79×)	1.81 (2.18×)
		FreeDave	**80.21 (+0.60)**	**27.39 (1.30×)**	**2.34 (2.82×)**
	TraDo-4B-Instruct	Static	**91.58**	4.41	0.15
		Parallel	89.08 (-2.50)	9.82 (2.23×)	0.35 (2.33×)
		FreeDave	91.05 (-0.53)	**10.03 (2.27×)**	**0.39 (2.60×)**
	TraDo-8B-Instruct	Static	92.72	3.41	0.12
		Parallel	92.34 (-0.38)	6.17 (1.81×)	0.23 (1.92×)
		FreeDave	**92.80 (+0.08)**	**6.92 (2.03×)**	**0.28 (2.33×)**
AIME2024	Dream-7B-Instruct	Static	**6.67**	22.82	0.94
		Parallel	3.33 (-3.34)	16.09 (0.71×)	1.92 (2.04×)
		FreeDave	3.33 (-3.34)	**24.08 (1.06×)**	**3.55 (3.78×)**
	TraDo-4B-Instruct	Static	10.00	11.38	0.41
		Parallel	10.00	20.52 (1.80×)	0.75 (1.83×)
		FreeDave	**13.30 (+3.30)**	**26.07 (2.29×)**	**1.04 (2.54×)**
	TraDo-8B-Instruct	Static	13.33	15.39	0.51
		Parallel	10.00 (-3.33)	24.00 (1.56×)	0.86 (1.67×)
		FreeDave	**16.66 (+6.66)**	**29.62 (1.92×)**	**1.18 (2.31×)**

4.2 Evaluation on Math Reasoning Benchmarks

In this section, we compare the accuracy, throughput over time, and throughput over Number of Function Evaluations (NFEs) of three different DLLMs on three math reasoning and two code generation benchmarks. Specifically, the throughput is calculated by the number of valid generated tokens (*i.e.* after removing all special tokens like **m** and <EOS>) per second or per model forward

call. The reason why we use throughput instead of inference time or NFEs as our metric is that the valid length of the generated sequence is also critical. For some questions, the DLLM might generate longer responses with more valid tokens while taking more time and NFEs. Considering this point, we think throughput is a more reasonable measurement of inference efficiency.

Shown in Table 1, FreeDave is able to substantially improve the efficiency of DLLM inference. Specifically, for TraDo-4B-Instruct, it can boost the throughput either over time or over NFE to over $2\times$, while even slightly improving the accuracy by 2.20% on MATH500. For comparison, the parallel decoding, although able to get a similar speedup to our method, will significantly degrade the performance of the DLLM with an accuracy drop of 5.40%. For Dream-7B-Instruct, since the block size is set to 4, which requires more frequent KV Cache refreshing, parallel decoding will hamper the throughput over time, while FreeDavestill brings remarkable speedup from either the perspective of response time or NFEs and maintains the performance. Similar results can also be observed in GSM8K.

For evaluations on the much more challenging AIME2024, our method is still able to maintain the performance and bring a notable speedup. For Dream-7B-Instruct, both parallel decoding and FreeDave decoding result in the same level of accuracy, while FreeDave brings a higher speedup to the throughput over time and NFEs. For the TraDo models, in addition to decent inference acceleration, FreeDave even brings higher accuracy.

Table 2: Detailed comparison of performance and efficiency with static decoding, parallel decoding, and FreeDave decoding for different DLLMs on code generation benchmarks, including MBPP and HumanEval.

Benchmark	Model	Decoding Strategy	Acc (%) ↑	Throughput over time (#tokens/s) ↑	Throughput over NFEs (#tokens) ↑
MBPP	Dream-7B-Instruct	Static	46.20	15.49	0.62
		Parallel	37.40 (-8.80)	15.36 (0.99×)	1.70 (2.74×)
		FreeDave	**46.40 (+0.20)**	**20.32 (1.31×)**	**1.80 (2.90×)**
	TraDo-4B-Instruct	Static	**57.40**	1.63	0.06
		Parallel	49.40 (-8.00)	**4.28 (2.63×)**	0.14 (2.33×)
		FreeDave	56.60 (-0.80)	4.19 (2.57×)	**0.15 (2.50×)**
	TraDo-8B-Instruct	Static	63.20	1.80	0.07
		Parallel	57.00 (-6.20)	3.67 (2.04×)	0.13 (1.86×)
		FreeDave	**63.60 (+0.40)**	**3.86 (2.14×)**	**0.15 (2.14×)**
HumanEval	Dream-7B-Instruct	Static	54.88	17.85	0.72
		Parallel	35.37 (-19.51)	15.72 (0.88×)	1.77 (2.46×)
		FreeDave	**56.09 (+1.21)**	**24.40 (1.37×)**	**2.14 (2.97×)**
	TraDo-4B-Instruct	Static	59.76	4.33	0.17
		Parallel	57.32 (-2.44)	7.36 (1.70×)	0.26 (1.53×)
		FreeDave	**60.98 (+1.22)**	**8.74 (2.02×)**	**0.38 (2.24×)**
	TraDo-8B-Instruct	Static	**68.90**	2.69	0.12
		Parallel	65.24 (-3.66)	**4.57 (1.70×)**	0.22 (1.83×)
		FreeDave	**68.90 (+0.00)**	4.28 (1.59×)	**0.26 (2.17×)**

4.3 Evaluation on Code Generation Benchmarks

In addition to the math reasoning benchmarks, we also evaluate our method on code generation benchmarks, following the same configuration. As shown in Table 2, compared with math reasoning tasks, threshold-based parallel decoding will bring a more drastic performance drop to all DLLMs on code generation tasks. Particularly, for Dream-7B-Instruct evaluated on HumanEval, threshold-based parallel decoding will even cut the accuracy from 54.88% to 35.37%. On the other hand, FreeDave maintains the performance on each benchmark, but also brings an appreciable speedup, with throughput boosted up to $2.97\times$.

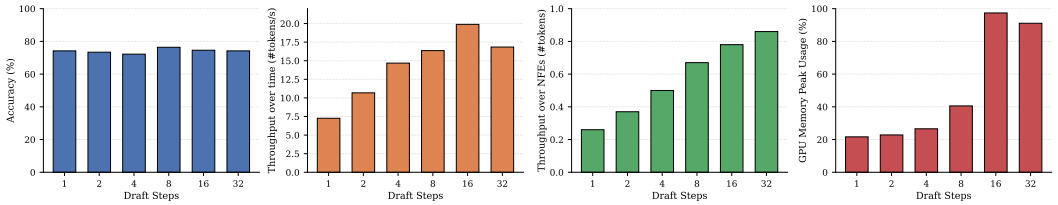

Figure 2: Evaluation of TraDo-4B-Instruct on MATH500 using different draft steps d. Specifically, when $d = 1$, the decoding strategy is static decoding. We compare accuracy, throughput over time, throughput over NFEs, and GPU memory peak usage in the inference stage.

4.4 Ablation on Draft Steps

In addition to the evaluations of our method on different benchmarks, we also investigate how to choose an appropriate draft steps d such that we can get the best trade-off between memory usage, inference time, and NFEs. As shown in Figure 2, we set $d \in \{1, 2, 4, 8, 16, 32\}$, and compare the accuracy, throughput over time, throughput over NFEs, and GPU memory peak usage in the inference of TraDo-4B-Instruct on MATH500. As d gets larger, the accuracy remains stable, which aligns with our theoretical analysis of lossless feasible path in Section 3.2. On the other hand, a larger d means a larger batch of draft candidates, which will allocate more GPU memory and occupy more streaming multiprocessors. From Lemma 1, once d is large enough to cover the maximum decision size of the optimal path, we will get the fewest NFEs, and the throughput over NFEs will converge. When the computational resources are limited and compute bound or memory bound is reached, the choice of d should be more careful , rather than indiscriminately defaulting to a larger value.

Note that the block size of TraDo-4B-Instruct is set to 4 by default. In order to use a larger d that exceeds the predefined block size, we explicitly design in our implementation. Specifically, we allow to pre-draft and pre-verify positions in future blocks, but still take all involved blocks in a left-to-right priority, instead of just merging the current block and future blocks into a larger block.

5 Discussion

In this paper, we propose FreeDave, aiming to perform a lossless parallel decoding for DLLMs. By both theoretical analysis and empirical evaluations on math reasoning and code generation benchmarks, our method is proven to achieve lossless parallel decoding, substantially accelerating the inference while maintaining the generation quality. Compared with the threshold-based parallel decoding, FreeDave is able to bring more speedup, but also overcome the challenge of performance degradation at the same time.

It is worth noting that, although the theoretical foundations of our method are built on static decoding, from our observations on the TraDo models, FreeDave can still be compatible with threshold-based parallel decoding, further boosting the throughput while maintaining the performance at the same level. However, as threshold-based parallel decoding usually leads to non-negligible performance degradation compared with static decoding, and we aim to perform lossless parallel decoding, we did not discuss details about this combination in our paper.

There are still some limitations remaining in our current version. Firstly, when we use a very large number of draft steps, a model forward call on a batch of inputs will take a longer time, as the assumption of unlimited computation and memory budget is not practical. A simple solution is Tensor Parallelism (TP) of Data Parallelism (DP) on multiple GPUs, but this will also bring an extra communication time cost. Secondly, from our theoretical analysis, FreeDave should be able to fully reproduce the sequences generated by static decoding. However, from our observations, it is not always the case, although the performance does remain at the same level on different benchmarks. Sometimes, we found FreeDave generated a sequence with pretty much the same semantics and logic, but some words or expressions were slightly different. We think it might be caused by the non-batch-invariant kernels in GPUs, like what is discussed in He and Thinking Machines [45]. We leave those problems for future exploration.

References

[1] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[2] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[3] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[4] OpenAI. GPT-5 is here , 2025. URL `https://openai.com/gpt-5/`.

[5] Llama Team, AI @ Meta. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation, 2025. URL `https://ai.meta.com/blog/llama-4-multimodal-intelligence/`.

[6] Anthropic. Claude Code, 2025. URL `https://claude.com/product/claude-code`.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[8] Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*, pages 37–42, 2023.

[9] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jian-Guang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, Yansong Tang, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. In *The Thirteenth International Conference on Learning Representations*, 2025.

[10] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[11] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[12] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

[13] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

[14] CodeGemma Team, Ale Jakse Hartman, Andrea Hu, Christopher A. Choquette-Choo, Heri Zhao, Jane Fine, Jeffrey Hui, Jingyue Shen, Joe Kelley, Joshua Howland, Kshitij Bansal, Luke Vilnis, Mateo Wirth, Nam Nguyen, Paul Michel, Peter Choy, Pratik Joshi, Ravin Kumar, Sarmad Hashmi, Shubham Agrawal, Siqi Zuo, Tris Warkentin, and Zhitao Gong. CodeGemma: Open Code Models Based on Gemma. `https://goo.gle/codegemma`, 2024.

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

[16] 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.

[17] Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models. *arXiv preprint arXiv:2509.06949*, 2025.

[18] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[19] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023.

[20] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. In *International Conference on Learning Representations*, 2021.

[21] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

[22] Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

[23] Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

[24] Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

[25] Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

[26] Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

[27] Jinjie Ni, Qian Liu, Chao Du, Longxu Dou, Hang Yan, Zili Wang, Tianyu Pang, and Michael Qizhe Shieh. Training optimal large diffusion language models. *arXiv preprint arXiv:2510.03280*, 2025.

[28] 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. *arXiv preprint arXiv:2505.22618*, 2025.

[29] Runpeng Yu, Xinyin Ma, and Xinchao Wang. Dimple: Discrete diffusion multimodal large language model with parallel decoding. *arXiv preprint arXiv:2505.16990*, 2025.

[30] Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025.

[31] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[32] 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. *arXiv preprint arXiv:2103.03874*, 2021.

[33] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[34] Mathematical Association of America, American Mathematics Competitions. American Invitational Mathematics Examination (AIME) 2024: AIME I and AIME II, 2024. URL `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`. Competition problems used as an evaluation dataset; original problems by MAA AMC.

[35] 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.

[36] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[37] Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. In *The Thirteenth International Conference on Learning Representations*, 2025.

[38] Lingzhe Zhang, Liancheng Fang, Chiming Duan, Minghua He, Leyi Pan, Pei Xiao, Shiyu Huang, Yunpeng Zhai, Xuming Hu, Philip S Yu, et al. A survey on parallel text generation: From parallel decoding to diffusion language models. *arXiv preprint arXiv:2508.08712*, 2025.

[39] 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.

[40] Benjamin Frederick Spector and Christopher Re. Accelerating LLM inference with staged speculative decoding. In *Workshop on Efficient Systems for Foundation Models @ ICML2023*, 2023. URL `https://openreview.net/forum?id=RKHF3VYjLK`.

[41] 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)*, pages 11263–11282, 2024.

[42] 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. *arXiv preprint arXiv:2401.10774*, 2024.

[43] Jacob K Christopher, Brian R. Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative diffusion decoding: Accelerating language generation through diffusion. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 12042–12059, 2025.

[44] Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025.

[45] Horace He and Thinking Machines. Defeating nondeterminism in llm inference. Thinking Machines Data Science Blog, sep 2025. URL `https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/`.

A Example of Static Decoding and FreeDave Decoding

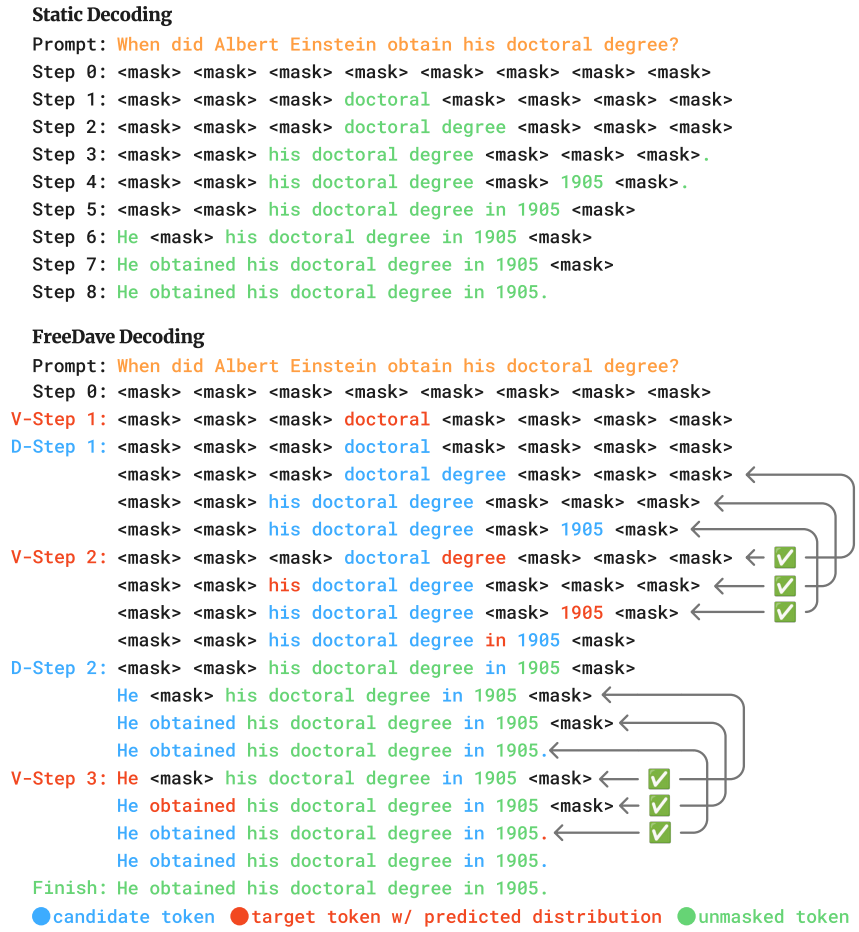

Figure 3: An example of DLLM static decoding and FreeDave decoding. In this example, at each step, static decoding is set to unmask one masked position where the predicted token has the highest confidence. For FreeDave decoding, V-Step takes as input all the candidate sequences in parallel and outputs target sequences (one more decoding step over each candidate sequence) and their estimated distributions, while D-Step samples from the estimated distribution of the maximum matched target sequence.