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

Shutong Wu
Department of Computer Sciences
University of Wisconsin–Madison
swu494@wisc.edu

Jiawei Zhang
Department of Computer Sciences
University of Wisconsin–Madison
jzhang2924@wisc.edu

Abstract

Diffusion Large Language Models (dLLMs) have emerged as a compelling paradigm for language modeling, offering the unique potential for highly efficient inference via multi-token parallel decoding. However, unlocking this potential remains challenging. High-quality generation typically defaults to one-token-per-step static decoding, while existing parallel algorithms rely on heuristic confidence thresholds, introducing a fragile trade-off between decoding efficiency and generation quality. To overcome this, we introduce Free Draft-and-Verification (FreeDave), a novel training-free and model-free fast decoding algorithm. FreeDave capitalizes on the inherent properties of dLLMs by treating future token predictions as "free drafts" and rigorously self-verifying them in subsequent forward passes. Theoretically, FreeDave searches for a shorter decoding path that is guaranteed to reproduce the exact sequence generated by static decoding, achieving algorithmically lossless acceleration. With a sufficiently large draft budget and a verification frontier that does not retreat, it can use the fewest possible model forward passes among feasible paths. Extensive evaluations on math reasoning and code generation benchmarks demonstrate that FreeDave can boost the number of generated tokens per model forward (TPF) by up to 5.48× and tokens per second (TPS) by up to 3.94×, while robustly maintaining generation quality and avoiding the task-specific performance degradation faced by heuristic methods. Code is available at https://github.com/cychomatica/FreeDave.

1 Introduction

The advent of Large Language Models (LLMs) [1–6] built on large transformers [7] and autoregressive (AR) next-token prediction has marked a revolutionary milestone on the way to artificial general intelligence. In recent years, LLMs have made significant strides in math reasoning [8–10], which requires deep logical and symbolic understanding. Similarly, in the domain of software engineering, LLMs have shown promising achievements on code generation [6, 11–14], offering powerful tools to automate development tasks and improve development productivity. Recently, Diffusion Large Language Models (dLLMs) [15–17] 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 [18–20], diffusion models have been adapted to the discrete domain, especially language modeling [21–26]. Due to their bidirectional attention mechanism, dLLMs are more capable of capturing the connection of context, showing their unique advantages in challenges like the famous "reversal curse" [15] or learning under data-constrained scenarios [27]. Moreover, since dLLMs are trained to simultaneously estimate the marginal distributions for all tokens in a sequence, they possess an inherent capability for efficient multi-token parallel decoding.

However, despite their great potential, the practical application of dLLMs is hampered by the challenge that the high generation quality often requires the number of decoding steps equal to the

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

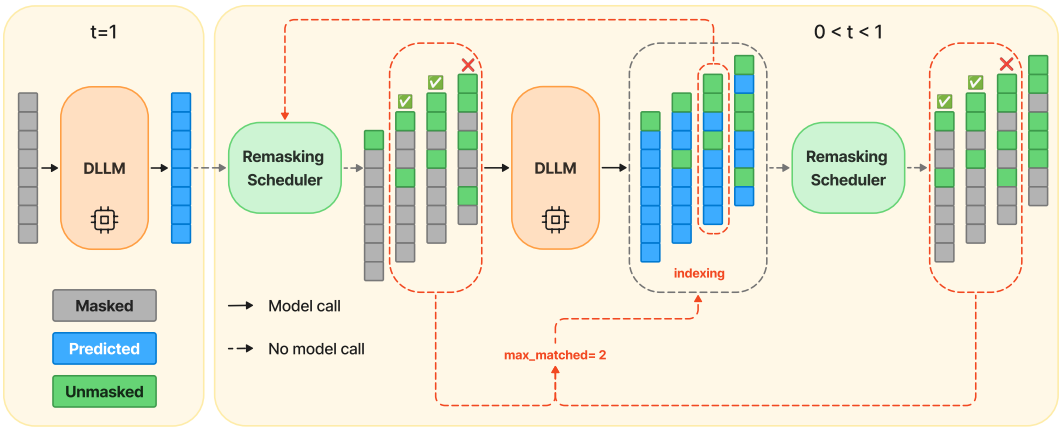

Figure 1: The overview of FreeDave decoding. At the current step, the dLLM outputs the estimated distribution of the sequence, from which the remasking scheduler constructs multiple draft candidates. At the subsequent step, the same dLLM processes these candidates in parallel, verifying their validity via sequence matching. Then the resulting output distributions are reused to sample the next set of drafts. Because the logits needed for future drafts are naturally produced as byproducts of the verification step, FreeDave creates a continuous, self-verifying parallel decoding loop that requires no auxiliary models or extra forward passes.

sequence length, which performs a one-token-per-step decoding (denoted as static decoding). Existing parallel decoding algorithms [28, 29] attempt to bypass this challenge by decoding multiple tokens with confidence higher than a pre-defined threshold. However, such heuristic-based thresholding inherently introduces a fragile trade-off between efficiency and quality. As our evaluations reveal, threshold-based methods are prone to overconfidence and can cause noticeable performance degradation and exhibit inconsistency, especially in logic-intensive tasks like math reasoning and code generation. To guarantee the performance, a high threshold should be used, which can be too conservative and lead to insufficient acceleration. Thus, fully unleashing the great potential of parallel decoding with a rigorous guarantee of generation quality remains a critical challenge for dLLMs.

To overcome this challenge, we introduce **Free D**raft-**a**nd-**Ve**rification (**FreeDave**), a novel fast decoding algorithm for dLLMs that achieves lossless parallel decoding. Taking advantage of the inherent capability of parallel decoding of dLLMs, and inspired by the idea of draft-and-verification from speculative decoding designed for autoregressive LLMs [30], we reveal that:

Your dLLM is secretly a self-verifiable parallel decoder with almost no extra cost.

Unlike autoregressive models that require an auxiliary draft model or extra network heads to achieve multi-token parallel decoding via speculative decoding, a dLLM natively estimates the marginal distribution for all tokens in the sequence at every single forward step. FreeDave capitalizes on this unique property: we treat these native future predictions as "free drafts" and verify them using the dLLM itself in subsequent standard forward passes. The generation and verification of these candidates are essentially zero-cost byproducts of the normal decoding process. Consequently, FreeDave completely eliminates the need for auxiliary draft models or separate draft-and-verify stages, while seamlessly avoiding the quality-efficiency trade-off that plagues existing parallel algorithms.

In this paper, our contributions are summarized as follows:

- We introduce FreeDave, a training-free and model-free fast decoding algorithm tailored for dLLMs that substantially compresses the total number of decoding steps and accelerates inference without sacrificing generation quality.
- Theoretically, we provide a guarantee that FreeDave achieves algorithmically lossless decoding. We prove that FreeDave flawlessly reproduces the exact sequence generated by one-token-per-step static decoding, and can use the fewest possible model forward passes among feasible paths when the verification frontier does not retreat as decoding progresses and the draft budget is sufficiently large.

- Empirically, we extensively evaluate FreeDave across various dLLMs on diverse benchmarks. Compared to static decoding, FreeDave can boost both the number of generated tokens per model forward (TPF) and tokens per second (TPS) up to over $2\times$, while robustly maintaining task performance where previous threshold-based methods can fail.
- Furthermore, FreeDave also serves as a versatile framework capable of pushing heuristic approaches (*e.g.*, threshold-based parallel decoding) to their efficiency limits with preserved generation quality. With a threshold-based scheduler incorporated, compared to static decoding, FreeDave can boost TPF up to $5.48\times$ and TPS up to $3.94\times$, while maintaining task performance.

2 Related Work

Diffusion Large Language Models (dLLMs) offer a compelling alternative to autoregressive (AR) models. Built upon discrete absorbing diffusion [21] and Masked Language Modeling [22, 23], scaled DLLMs [15, 24, 16, 17] have closed the performance gap with AR baselines. Their bidirectional attention overcomes AR limitations like the "reversal curse" [31] and improves data efficiency [32, 27]. However, this bidirectionality fundamentally complicates standard KV caching, creating a severe bottleneck in inference speed.

To mitigate this bottleneck, specialized caching strategies have been proposed, such as semi-autoregressive Block Diffusion [26], feature-redundant dLLM-Cache [33], and approximate caching in Fast-dLLM [28]. Beyond caching, parallel decoding methods aim to accelerate generation by simultaneously unmasking multiple tokens. Fast-dLLM [28] and Dimple [29] rely on heuristically predefined confidence thresholds to accept tokens. However, these heuristic-based methods suffer from a fragile efficiency-quality trade-off, often leading to suboptimal decoding efficiency, or non-negligible performance degradation compared to static decoding [17]. Attempts to adaptively mix dLLM and AR distributions [34] still struggles to eliminate this trade-off completely.

Some recent works also introduce the "draft-and-verify" concept, which originates from speculative decoding [30, 35–38], to the context of dLLMs, but they tackle fundamentally different research problems. For instance, TiDAR [39] trains a hybrid model that switches between a diffusion mode for drafting and an AR mode for verification. DFlash [40], on the other hand, trains a dLLM specifically to act as a parallel-decoding draft model to accelerate a target AR model under the traditional speculative decoding framework. However, our work focuses exclusively on accelerating pre-trained dLLMs and introduces a fully plug-and-play, training-free, and model-free algorithm, elegantly treating the pre-trained dLLM as its own drafter and verifier to unlock lossless parallel decoding.

3 Methodology

3.1 Preliminaries

Like diffusion models on continuous spaces, dLLMs model the transition from a data distribution to a known prior noise distribution on discrete spaces by a forward noising process, and generate data from noise by the induced reverse denoising process. 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, 15–17]. For the absorbing diffusion, the forward process progressively replaces each unmasked token in the original sequence x_0 independently with a special mask token $\mathbf{m} \in V$ with an increasing probability *w.r.t.* a time level t evolving from 0 to 1. Once a token is masked, it will remain masked till $t = 1$. Eventually, at $t = 1$, all the tokens in the sequence will be $\mathbf{m}$. This forward process with the absorbing state $\mathbf{m}$ can be formulated as

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

where $x_t \in V^L$ is the transferred sequence at the time level t, $x_t^i \in V$ represents the i-th token of x_t, $\alpha_t \in \mathbb{R}$ is the noising schedule monotonically decreasing *w.r.t.* $t \in [0, 1]$ and satisfies $a_0 = 1, a_1 = 0$, $\mathbf{e}_j \in \{0, 1\}^{|V|}$ is the one-hot vector of interger j over the vocabulary, and Cat represents the categorical distribution over the vocabulary V.

The reverse process is induced to progressively recover the original sequence at $t = 0$ from an all-mask sequence at $t = 1$. Specifically, the reverse process transfer 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 $0 \leq s < t \leq 1$) by the product of the marginal distributions of each token $q(x_s|x_t, x_0) = \prod_{i=1}^{L} q(x_s^i|x_t^i, x_0^i)$, where the marginal distribution of the i-th token can be formulated as

$$q(x_s^i|x_t^i, x_0^i) = \begin{cases} \text{Cat}(x_s^i; \mathbf{e}_{x_t^i}) & \text{if } x_t^i \neq \mathbf{m} \\ \text{Cat}(x_s^i; \frac{(1-\alpha_s)\mathbf{e}_{\mathbf{m}} + (\alpha_s - \alpha_t)\mathbf{e}_{x_0^i}}{1-\alpha_t}) & \text{if } x_t^i = \mathbf{m} \end{cases} \tag{2}$$

and once a token is unmasked, it will remain unchanged till $t = 0$. The challenge is that x_0 is agnostic at $t > 0$. As a solution, a parameterized model $f_\theta(\cdot, \cdot) : V^L \times \mathbb{R} \to \mathbb{R}^{L \times |V|}$ is trained to estimate the marginal distributions of each token at $t = 0$ by $f_\theta^i(x_t, t)$. Given the sequence x_t at time level t, the the marginal distribution of the i-th token can be calculated by

$$p_\theta(x_s^i|x_t^i) = \begin{cases} \text{Cat}(x_s^i; \mathbf{e}_{x_t^i}) & \text{if } x_t^i \neq \mathbf{m} \\ \text{Cat}(x_s^i; \frac{(1-\alpha_s)\mathbf{e}_{\mathbf{m}} + (\alpha_s - \alpha_t) f_\theta^i(x_t, t)}{1-\alpha_t}) & \text{if } x_t^i = \mathbf{m} \end{cases} \tag{3}$$

such that we can predict the sequence at $s < t$ by $p_\theta(x_s|x_t) = \prod_{i=1}^{L} p_\theta(x_s^i|x_t^i)$. Empirically, 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 $\mathbf{m}$ tokens to be transferred to unmasked tokens at each time level. The remasking scheduler g takes as input a source time level t, a target time level s, and the sequence x_t, 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 $\mathcal{I}_{t \to s}$, which is defined as $\mathcal{I}_{t \to s} = \{(a_1, z_1), \ldots, (a_n, z_n)\}$, 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.

Generally, for a high generation quality, the total number of decoding 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 one-token-per-step greedy decoding strategy as static decoding. Unless specified, in this paper, we focus on the static remasking scheduler g_{static}. Details of the static decoding algorithm can be found in Algorithm 2.

3.2 Decoding Algorithm: Free Draft-and-Verification

Definition 1 (Decoding Path and Oracle Path) *For the reverse process of dLLM f_θ, a decoding 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 timestep 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 $\mathcal{I}_{t_i \to t_{i+1}} = g(x_{t_i}, f_\theta(x_{t_i}, t_i), t_i, t_{i+1})$ 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*

$$\begin{aligned} \mathcal{I}_{t_{a_i} \to t_{a_{i+1}}} &= g(x_{t_{a_i}}, f_\theta(x_{t_{a_i}}, t_{a_i}), t_{a_i}, t_{a_{i+1}}) \\ &= \bigcup_{j=0}^{a_{i+1}-a_i-1} g(x_{t_{a_i+j}}, f_\theta(x_{t_{a_i+j}}, t_{a_i+j}), t_{a_i+j}, t_{a_i+j+1}) \\ &= \bigcup_{j=0}^{a_{i+1}-a_i-1} \mathcal{I}_{t_{a_i+j} \to t_{a_i+j+1}}, \ \forall i \in \{0, \ldots, M-1\}, \end{aligned} \tag{4}$$

then the decoding path $\mathbf{p} = [\mathcal{I}_{t_{a_0} \to t_{a_1}}, \ldots, \mathcal{I}_{t_{a_{M-1}} \to t_{a_M}}]$ reproduces the oracle sequence. For our theoretical analysis, we additionally require each merged interval to admit a consecutive one-step verification chain, as specified in Definition 3 in Appendix C.1. A path satisfying both conditions is defined as a feasible path, and $\mathcal{P}(f_\theta, g, \mathbf{T})$ denotes this ordered one-step-verifiable 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* $|\mathbf{p}^\star| \le |\mathbf{p}| \le |\mathbf{p}_{oracle}|, \forall \mathbf{p} \in \mathcal{P}(f_\theta, g, \mathbf{T})$, i.e. *among all feasible paths, the optimal path requires the fewest decoding steps, while the oracle path requires the most decoding steps.*

Crucially, any feasible path $p \in \mathcal{P}(f_\theta, g, T)$ structurally guarantees the reproduction of the exact same output sequence as the oracle path $\mathbf{p}_{\text{oracle}}$. Thus, we formally define decoding along any feasible path as lossless decoding. Definition 1 and 2 indicate the great potential of dLLMs for lossless parallel decoding. However, except for the oracle path $\mathbf{p}_{\text{oracle}}$, all of the other lossless 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 heuristic confidence threshold for parallel decoding is not guaranteed to get a feasible path that generates the same sequence as decoding via the oracle path, and the experimental results from both Wang et al. [17] and us demonstrate that threshold-based parallel decoding is not always robust and can result in a non-negligible performance degradation.

Following Appendix C.1, let $D_{n,r}$ be the nested draft merging the next r oracle transitions from a trusted state x_{t_n}, and let $T_{n,r}$ be its one-step target computed in the next batched forward. The first draft $D_{n,1}$ is the one-step oracle successor. For $0 \le n < j \le N$, we write $F(n, j) = 1$ when $T_{n,r} = D_{n,r+1}$ for every $r = 1, \ldots, j - n - 1$. Thus, verification accepts only a consecutive prefix and stops at the first mismatch; a later match beyond that mismatch does not certify a feasible merge.

Theorem 1 (Verification-based Feasible Path Search) *In the ordered one-step-verifiable space* $\mathcal{P}(f_\theta, g, \mathbf{T})$*, let* d *be a positive integer search budget and* $d_n = \min(d, N - n)$*. Define the verifier as the length of the longest consecutively verified prefix:*

$$h(n, d) = \max \{\ell \in \{1, \ldots, d_n\} \mid T_{n,r} = D_{n,r+1},\ \forall r = 1, \ldots, \ell - 1\}. \tag{5}$$

For $\ell = 1$*, the set of verification indices* $\{1, \ldots, \ell - 1\}$ *is empty, so the condition is automatically satisfied. This corresponds to accepting the first draft* $D_{n,1}$*, which is already the one-step oracle successor and requires no target matching. Thus,* $\ell = 1$ *is always admissible, giving* $1 \le h(n, d) \le d_n$*. Starting from* $a_0 = 0$ *and applying* $a_{i+1} = a_i + h(a_i, d)$ *until* $a_M = N$ *yields a feasible path* $\mathbf{p} = [\mathcal{I}_{t_{a_0} \to t_{a_1}}, \ldots, \mathcal{I}_{t_{a_{M-1}} \to t_{a_M}}]$ *that reproduces the oracle output.*

Let $R(n) = \max\{j : F(n, j) = 1\}$ be the farthest verifiable endpoint. For the following optimality result, we assume Verification-Frontier Non-Retreat (Assumption 1) holds: $F(i, j) \Rightarrow F(k, j)$ for $i < k < j$, or equivalently $R(i + 1) \ge R(i)$. Details can be found in Appendix C.1.

Lemma 1 (Step Optimality) *Under Assumption 1, the greedy longest-merge path* $\mathbf{p}^\star$*, defined by* $a_0^\star = 0$ *and* $a_{i+1}^\star = R(a_i^\star)$ *until* $a_M^\star = N$*, has the fewest decoding steps among all paths in the ordered one-step-verifiable space* $\mathcal{P}(f_\theta, g, \mathbf{T})$*. If* $d \ge \|\mathbf{p}^\star\|_\infty = \max\{a_{i+1}^\star - a_i^\star \mid i = 0, \ldots, M-1\}$*, then the verifier in Theorem 1 recovers this optimal path, with* $a_{i+1}^\star = a_i^\star + h(a_i^\star, d)$ *for* $i = 0, \ldots, M - 1$*.*

Proof Sketch For Theorem 1, $D_{n,1} = x_{t_{n+1}}$. If $D_{n,r} = x_{t_{n+r}}$, its one-step target is the next oracle state, so a successful check $T_{n,r} = D_{n,r+1}$ gives $D_{n,r+1} = x_{t_{n+r+1}}$. Induction over the consecutive checks in Eq. (5) therefore yields

$$\mathcal{I}_{t_n \to t_{n+h(n,d)}} = \bigcup_{r=0}^{h(n,d)-1} \mathcal{I}_{t_{n+r} \to t_{n+r+1}},$$

which induces a feasible path by Definition 2. When $d = 1$, this reduces to the oracle path.

For Lemma 1, compare the untruncated greedy path with any feasible path $0 = b_0 < \cdots < b_K = N$. Assume $a_r^\star \ge b_r$, which holds at $r = 0$. If $a_r^\star < b_{r+1}$, Assumption 1 and $F(b_r, b_{r+1}) = 1$ give $F(a_r^\star, b_{r+1}) = 1$, so greedy reaches at least b_{r+1} on its next step. Otherwise, it has already reached or passed that endpoint. Induction gives $a_r^\star \ge b_r$ until greedy terminates, proving that it reaches N in at most K steps. A budget $d \ge \|\mathbf{p}^\star\|_\infty$ leaves every greedy merge untruncated and hence recovers this optimal path. The complete proof is provided in Appendix C.2.

Remark Under the conditions of Lemma 1, a draft budget covering the largest number of oracle transitions merged along the untruncated greedy optimal path suffices to recover that path. In

particular, $d \geq N \geq \|\mathbf{p}^\star\|_\infty$ provides a sufficient condition. Detailed analysis is provided in Appendix C.

Based on our theoretical analysis above, we propose Free Draft-and-Verification (FreeDave), a fast decoding algorithm that searches for the longest consecutively mergable decoding steps, with algorithmic losslessness guaranteed by Theorem 1 and step optimality under the conditions of Lemma 1. Specifically, as detailed in Algorithm 1, FreeDave achieves this by instantiating the theoretical verifier $h(n, d)$ through a parallel sequence matching mechanism (*i.e.*, Line 27–33). At each iteration, FreeDave iteratively utilizes the remasking scheduler to sample a batch of draft candidates with varying unmasking tokens based on the current estimated distribution. At the next step, these candidates are fed into the dLLM in parallel. The subsequent sequential matching process essentially acts as the verifier $h(n, d)$ that identifies the longest consecutive match between the drafted states and the one-step-ahead target states, mathematically satisfying Eq. (5) and ensuring the accepted path remains strictly in the feasible space (*i.e.*, lossless).

It is worth noting that while FreeDave shares the concept of "draft-and-verify" with speculative decoding [30, 35–38], their operational paradigms are fundamentally different. Speculative decoding relies on an auxiliary draft model (or specific architectural modifications like auxiliary heads), separating the draft and verification stages into distinct forward passes. In contrast, FreeDave introduces zero extra forward passes compared to static decoding. Since a dLLM naturally computes the marginal distribution for all tokens simultaneously, the outputs required to verify the current drafts inherently contain the distributions needed to sample the next group of drafts. In FreeDave, drafts are strictly "free" byproducts decoded and verified concurrently during the standard forward pass. This single-model, unified-stage approach not only unlocks higher decoding efficiency but also entirely sidesteps the notorious draft-target alignment challenge faced by traditional speculative decoding.

3.3 Practical Extensions for Block Diffusion

While Section 3.2 establishes the theoretical foundation of FreeDave, deploying dLLMs in practice often relies on block diffusion [26] to enable KV caching via inter-block autoregressive and intra-block diffusion decoding. To maximize practical utility, we further introduce two critical extensions.

Block Tree Attention for Memory Efficiency. A naive implementation of Algorithm 1 expands draft candidates along the batch dimension, incurring substantial memory overhead. Inspired by Cai et al. [38], we compactly pack all draft candidates into a single sequence and employ a block-level tree attention topology (illustrated in Figure 6). This structure prevents redundant KV cache allocation for shared prefixes across different draft candidates and reduce the memory footprint during parallel verification.

Eager Mode for Cross-Block Acceleration. A fundamental limitation arises when directly applying FreeDave to small block sizes: as the intra-block decoding approaches the block boundary, the number of remaining masked positions naturally becomes smaller than the draft budget d. This forces d to be truncated, inevitably bottlenecking parallel efficiency. To circumvent this, we introduce Eager Mode (illustrated in Figure 7). Rather than strictly confining drafts within the current block, Eager Mode permits pre-drafting and pre-verifying tokens across future blocks while maintaining a strict left-to-right block acceptance priority. This not only eliminates boundary-induced truncations but also explicitly enables scaling the draft budget d beyond the block size itself. Crucially, as demonstrated in Appendix F.2, Eager Mode is structurally equivalent to a static decoding variant with delayed cache updates, thereby strictly preserving the generation quality while unlocking significantly higher decoding efficiency.

3.4 Integrating Threshold-Based Parallel Decoding into FreeDave

To further push the efficiency limits, FreeDave can be integrated with threshold-based parallel decoding. In this setting, FreeDave inherits the generation quality of threshold-based parallel decoding while further compressing the decoding steps. Specifically, at each step, among tokens excluding those already selected by the threshold-based scheduler, FreeDave selects the top $d-1$ most confident tokens and combines them with those threshold-selected tokens to construct d draft candidates. And at the next step, we perform a new threshold-based step on each draft to get

Algorithm 1 dLLM FreeDave Decoding

Require: Model f_θ, remasking scheduler g, total generation length L, total decoding 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**
  $\mathbf{X}_{\text{draft}} \leftarrow [\,]$, $d_i \leftarrow \min(d, N - i)$
  for k in $1 : d_i$ **do**
    $\mathcal{I}_{t_i \to t_{i+k}} \leftarrow g(x_{t_i}, \tilde{x}_0, t_i, t_{i+k})$
    $x_{\text{draft}} \leftarrow x_{t_i}$
    for each (a_j, z_j) in $\mathcal{I}_{t_i \to t_{i+k}}$ **do**
      $x_{\text{draft}}^{a_j} \leftarrow z_j$ ▷ replace masked tokens according to decision
    end for
    $\mathbf{X}_{\text{draft}}$.append($x_{\text{draft}}$)
  end for
  if $i = N - 1$ **then**
    $x_0 \leftarrow \mathbf{X}_{\text{draft}}[1]$
    break
  end if
  $\tilde{\mathbf{X}}_{\text{draft},0} \leftarrow f_\theta(\mathbf{X}_{\text{draft}}, t_{i+1:i+d_i})$ ▷ parallel verification forward of drafts
  $\mathbf{X}_{\text{target}} \leftarrow [\,]$
  for $k = 1 : d_i$ **do**
    $\mathcal{I}_{t_{i+k} \to t_{i+k+1}} \leftarrow g(\mathbf{X}_{\text{draft}}[k], \tilde{\mathbf{X}}_{\text{draft},0}[k], t_{i+k}, t_{i+k+1})$ ▷ one more step ahead
    $x_{\text{target}} \leftarrow \mathbf{X}_{\text{draft}}[k]$
    for each (a_j, z_j) in $\mathcal{I}_{t_{i+k} \to t_{i+k+1}}$ **do**
      $x_{\text{target}}^{a_j} \leftarrow z_j$
    end for
    $\mathbf{X}_{\text{target}}$.append($x_{\text{target}}$)
  end for
  $m \leftarrow 0$
  for k in $1 : d_i - 1$ **do** ▷ instantiation of verifier $h(n, d)$ from Theorem 1
    if $\mathbf{X}_{\text{draft}}[k+1] = \mathbf{X}_{\text{target}}[k]$ **then**
      $m \leftarrow m + 1$
    else
      break
    end if
  end for
  $i \leftarrow i + m + 1$, $x_{t_i} \leftarrow \mathbf{X}_{\text{draft}}[m+1]$, $\tilde{x}_0 \leftarrow \tilde{\mathbf{X}}_{\text{draft},0}[m+1]$
end while
$x \leftarrow x_0$
return x

target sequences. Then, each target sequence $\mathbf{X}_{\text{target}}[k]$ is compared with drafts $\mathbf{X}_{\text{draft}}[k+1:d]$ for verification. We detail the Threshold FreeDave algorithm in Appendix B.

4 Experiments

4.1 Experimental Settings

Our evaluations focus on math reasoning and code generation tasks. For math reasoning benchmarks, we choose MATH500 [41] and GSM8K [42]. For code generation benchmarks, we choose MBPP [43] and HumanEval [44]. In Appendix D.5, we also provide additional evaluations on Q&A benchmarks, including GPQA[45] and TruthfulQA[46]. We evaluate FreeDave on full-attention dLLMs including LLaDA-8B-Instruct [15] and Dream-7B-Instruct [16], and block-attention dLLMs including TraDo-4B-Instruct and TraDo-8B-Instruct [17]. We compare our method with **static decoding**, which follows the oracle path induced from a predefined time schedule, and **threshold-based parallel decoding** [28, 29], which at each step unmasks all positions where the transferring target token confidence exceeds a predefined threshold.

All the models use block-diffusion generation [26] and KV Cache. Specifically, we apply the dual cache [28] to full-attention dLLMs and the prefix cache to block-attention dLLMs. We use a

maximum generation length of 1024 for all the decoding algorithms, and set the block size to 32 for full-attention models and 4 for block-attention models. Besides, all the decoding algorithms enable an early exit mechanism, where the generation ends if the current block consists of all EOS tokens. Regarding the number of draft steps d used for FreeDave, we set $d = 8$ for the block-attention models, and $d = 4$ for the full-attention models. We enable Eager Mode on the block-attention models to unlock higher efficiency under a small block size.

4.2 Evaluation on Benchmarks

In this section, we compare the accuracy, tokens generated per forward (**TPF**), and tokens generated per second (**TPS**) of different dLLMs on math reasoning and code generation benchmarks. Specifically, TPF and TPS are calculated as the number of generated tokens divided by the Number of Function Evaluations (**NFE**) and response time, respectively. We do not directly use NFE or response time as our measurement of efficiency, as the generated sequences might have varied lengths due to the early exit mechanism.

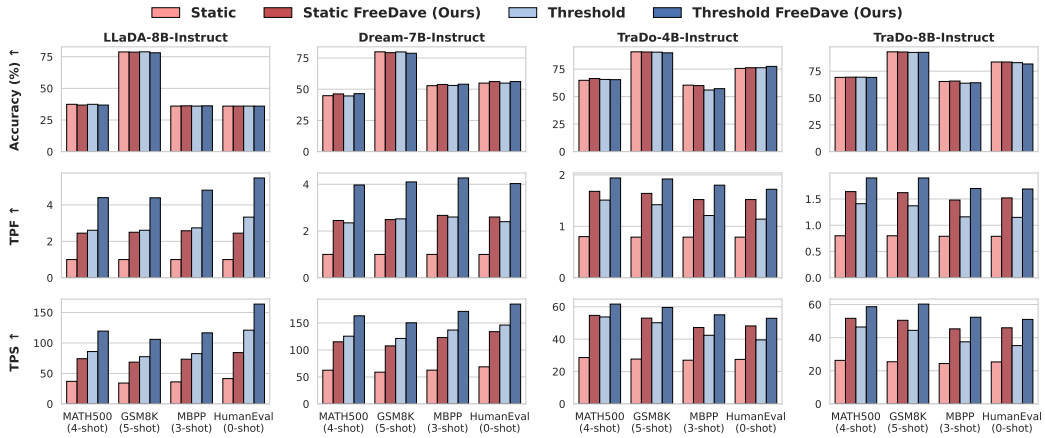

Figure 2: Performance and efficiency comparison of dLLMs with different decoding strategies on different benchmarks. More detailed results can be found in Appendix D.

Lossless Acceleration via FreeDave. As illustrated in Figure 2 and detailed in Tables2–5, FreeDave demonstrates remarkable inference speedups across all evaluated dLLMs and benchmarks. For block-attention models like TraDo-4B/8B, FreeDave significantly boosts both TPS and TPF (up to over 2x), while algorithmically anchoring the accuracy to the static baseline. The minor empirical fluctuations (within $\pm 1.6\%$) observed are strictly attributed to system-level numerical non-determinism, and we rigorously validate it under a deterministic debug mode, resulting in a 100% exact match rate (see Section 4.3).

Robustness of FreeDave. While threshold-based parallel decoding [28, 29] can sometimes achieve comparable speedups without severe accuracy loss, our evaluations reveal its inherent fragility as a heuristic approach. Because it relies on a predefined confidence scalar rather than rigorous verification, its performance is highly sensitive to the specific model and task. For instance, on logic-intensive tasks like code generation (MBPP), threshold decoding causes a noticeable 4.40% accuracy drop for TraDo-4B-Instruct. This inconsistency highlights the fundamental dilemma of thresholding: one must meticulously tune the hyperparameter for different tasks and models to avoid arbitrary degradation. Meanwhile, by shifting from a "guess-and-commit" heuristic to a "draft-and-verify" paradigm, FreeDave provides a structurally robust solution that universally applies to various dLLM architectures without the risk of task-specific performance degradation.

Pushing the Limits: FreeDave with Threshold. From Figure 2, it can be observed that FreeDave built on static decoding is already comparable and even faster than the strong baseline of threshold-based parallel decoding, with a better quality robustness. As introduced in Section 3.4 and detailed in Appendix B, FreeDave can be further integrated with a threshold-based scheduler to

unlock the extreme decoding efficiency. For example, as shown in Figure 2 and Table 2, compared to static decoding, Threshold FreeDave achieves up to 5.48 TPF (a $5.48\times$ speedup) and 163.75 TPS (a $3.94\times$ speedup) on LLaDA-8B-Instruct, while maintaining the benchmark performance. This demonstrates that our algorithm not only provides a lossless and efficient alternative to static decoding but also serves as a versatile framework capable of pushing heuristic approaches to their efficiency limits with preserved generation quality.

Model	MATH500		HumanEval	
	R_{sequence}	R_{token}	R_{sequence}	R_{token}
LLaDA-8B-Instruct	500/500	312047/312047	164/164	32977/32977
Dream-7B-Instruct	500/500	178489/178489	164/164	13136/13136
TraDo-4B-Instruct	500/500	290262/290262	164/164	24041/24041
TraDo-8B-Instruct	500/500	296300/296300	164/164	23938/23938

Table 1: On MATH500 and HumanEval, we evaluate the sequence-level R_{sequence} and token-level match rate R_{token} between static decoding and FreeDave decoding under determinism debug mode.

4.3 Disentangling Algorithmic Losslessness from System Numerical Non-determinism

By Theorem 1, FreeDave is theoretically guaranteed to traverse a feasible path that reproduces the exact sequence generated via the oracle path of static decoding. However, as observed in Section 4.2, the empirical accuracy between static and FreeDave decoding is not perfectly identical, exhibiting minor fluctuations (*e.g.*, within $\pm 1.6\%$). We rigorously attribute this empirical variance to system-level numerical non-determinism, primarily originating from floating-point non-associativity across different batch sizes and non-batch-invariant GPU kernels [47]. To isolate our algorithmic design from these system-level numerical noises, we conduct a study under a strict "determinism debug mode", where we set the dtype to float32 for a higher numerical precision and replace the batch forward with the sequential single-batch forward for FreeDave draft verification to ensure the same GPU kernels (see Appendix D.3 for details).

As we can observe, all the models achieve 100% sequence-level and token-level match rates, which proves that **our claim of losslessness is rigorous** and our code implementation of FreeDave decoding aligns with our theory without any logical errors, and **the empirical mismatch is attributed to the system-level numerical non-determinism**. It is worth noting that we primarily focus on the decoding algorithm in this paper, and resolving the non-batch-invariance of GPU kernels is another research problem for future exploration, as we mention in Section 5.

4.4 Ablation Study on Draft Steps

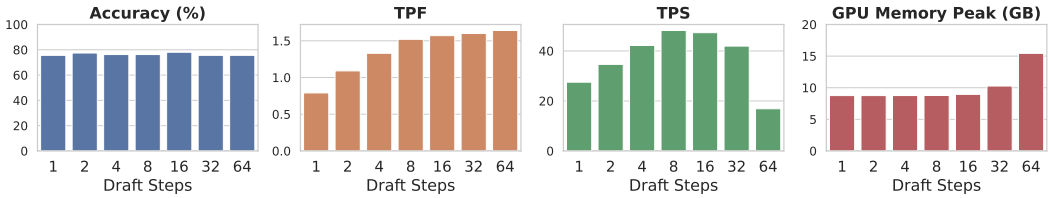

Figure 3: Evaluation of TraDo-4B-Instruct on HumanEval with different draft steps d. Specifically, when $d = 1$, the decoding algorithm falls back to static decoding.

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 fewer decoding steps and a higher TPF, a model forward will take as input a larger tensor, which might take more computation and memory and thus degrade the TPS. Thus, in addition to our evaluations on benchmarks, we also investigate the impact of different draft steps d such that we can get the best trade-off between memory overhead and inference efficiency. As shown in Figure 3, we set $d \in \{1, 2, 4, 8, 16, 32, 64\}$, and compare the accuracy, TPS, TPF, and GPU memory peak usage during the inference stage of TraDo-4B-Instruct on HumanEval. As d gets larger, the accuracy remains stable, which aligns with our theoretical

analysis of losslessness in Section 3.2. On the other hand, a larger d means a larger group of draft candidates, which can search shorter paths but will allocate more GPU memory and computation. From Lemma 1, once d is large enough to cover the maximum decision size of the optimal path, we will get the fewest NFE, and from our ablation study, the TPF shows a clear convergence tendency, which perfectly aligns with our theoretical expectation. However, an indiscriminately large d gets a marginal TPF gain but amplifies the GPU memory footprint (e.g., reaching ~ 15GB at $d = 64$) and prolongs the single-step forward latency, eventually causing a degradation in the wall-clock TPS. Therefore, the selection of d embodies a critical system-level trade-off between theoretical path compression and physical hardware constraints. Empirically, we identify $d = 8$ as a universal "sweet spot" for block-attention models, getting an optimal balance that maximizes TPS while incurring a negligible ~ 20MiB memory overhead compared to static decoding.

5 Discussion

We propose FreeDave, a novel algorithm performing lossless parallel decoding for dLLMs. By both theoretical analysis and empirical evaluations on math reasoning and code generation benchmarks, our method is proven to bring remarkable inference speedup while maintaining the generation quality, overcoming the challenge of performance degradation faced by threshold-based parallel decoding.

There are also some future directions remaining. For example, in this paper, we only focus on the offline scenario due to our limited computational resources, and extending our method to online serving with multiple prompt queries on multiple GPUs will be an interesting and critical problem. Besides, the non-batch-invariant GPU kernels bring numerical fluctuations, and resolving this challenge is also an important AI Infra research problem. We leave those problems for future exploration.

Acknowledgments and Disclosure of Funding

In this work, Shutong Wu was supported by the Office of the Vice Chancellor for Research and Graduate Education at the University of Wisconsin–Madison with funding from the Wisconsin Alumni Research Foundation and NSF Award DMS-2023239. Jiawei Zhang was supported by the Office of the Vice Chancellor for Research and Graduate Education at the University of Wisconsin–Madison with funding from the Wisconsin Alumni Research Foundation.

The authors declare no competing interests.

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] 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.

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

[36] 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`.

[37] 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.

[38] 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.

[39] Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. Tidar: Think in diffusion, talk in autoregression. *arXiv preprint arXiv:2511.08923*, 2025.

[40] Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL `https://openreview.net/forum?id=Ti67584b98`.

[46] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 3214–3252, 2022.

[47] 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/`.

[48] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL `https://zenodo.org/records/12608602`.

[49] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

A More Algorithm Details

A.1 Static Decoding

For static decoding, the remasking scheduler gives a decision set by

$$\begin{aligned}\mathcal{I}_{t\to t-\Delta} &= g_{\text{static}}(x_t, f_\theta(x_t,t), t, t-\Delta) \\ &= \left\{ (a,z) \,\middle|\, a = \arg\max_{i, x_t^i \neq \mathbf{m}} \left(\max_j f_\theta^{i,j}(x_t,t) \right), z = \arg\max_j f_\theta^{a,j}(x_t,t) \right\},\end{aligned} \tag{6}$$

where $\Delta = 1/L$ is the step size. We denote this one-token-per-step greedy decoding strategy as static decoding. Unless specified, in this paper, we focus on the static remasking scheduler g_{static}.

Algorithm 2 dLLM Static Decoding

Require: Model f_θ, remasking scheduler g_{static}, total generation length L, total decoding 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**
  $\tilde{x}_0 \leftarrow f_\theta(x_{t_i}, t_i)$
  $\mathcal{I}_{t_i \to t_{i+1}} \leftarrow g_{\text{static}}(x_{t_i}, \tilde{x}_0, t_i, t_{i+1})$
  $x_{t_{i+1}} \leftarrow x_{t_i}$
  for each (a_j, z_j) in $\mathcal{I}_{t_i \to t_{i+1}}$ **do**
    $x_{t_{i+1}}^{a_j} \leftarrow z_j$
  end for
end for
$x \leftarrow x_0$ **return** x

Algorithm 3 dLLM Threshold-Based Decoding

Require: Model f_θ, dynamic scheduler $g_{\text{top-k}}$, threshold τ, max length L
Ensure: The generated sequence x.

$x \leftarrow \mathbf{m}^{1:L}$
while $\sum_{l=0}^{L} \mathbb{1}(x^l = \mathbf{m}) > 0$ **do**
  $\tilde{x}_0 \leftarrow f_\theta(x_{t_i}, t_i)$
  $K_\tau = \min\left\{1, \sum_{i, x^i \neq \mathbf{m}} \mathbb{1}(\max_j f_\theta^{i,j}(x) \geq \tau)\right\}$
  $\mathcal{I} \leftarrow g_{\text{top-k}}(x, \tilde{x}_0, K_\tau)$
  for each (a_j, z_j) in $\mathcal{I}$ **do**
    $x^{a_j} \leftarrow z_j$
  end for
end whilereturn x

A.2 Threshold-Based Parallel Decoding

While static decoding relies on a predefined, fixed time schedule, threshold-based parallel decoding[28, 29] utilizes a dynamic, data-dependent schedule. At each step, it simultaneously unmasks all tokens whose predicted confidence exceeds a predefined threshold τ.

Because the number of tokens to unmask is dynamically determined, the exact time t for future steps is agnostic beforehand. To formalize this algorithm, we leverage a unique property of absorbing discrete diffusion that the time level t is inherently encoded in the state x itself via the mask ratio. Therefore, we can simplify the model notation from $f_\theta(x_t, t)$ to $f_\theta(x)$, safely omitting the explicit time input. Accordingly, we define this dynamic remasking scheduler that returns a dynamically

sized decision set as

$$g_{\text{top-k}}(x, f_\theta(x), K_\tau) = \left\{ (a, z) \,\middle|\, a \in \underset{i, x_t^i \neq \mathbf{m}}{\text{Top-}K_\tau} \left(\max_j f_\theta^{i,j}(x) \right), z = \arg\max_j f_\theta^{a,j}(x) \right\} \tag{7}$$

$$K_\tau = \min \left\{ 1, \sum_{i, x^i \neq \mathbf{m}} \mathbb{1}(\max_j f_\theta^{i,j}(x) \geq \tau) \right\} \tag{8}$$

Here, to ensure decoding progression even when model confidence is generally low, if no token exceeds τ, the scheduler falls back to unmasking the single token with the highest confidence. The standard threshold-based parallel decoding process is detailed in Algorithm 3.

Algorithm 4 dLLM Threshold FreeDave Decoding

Require: Model f_θ, dynamic scheduler $g_{\text{top-k}}$, threshold τ, max length L, draft steps d.
Ensure: The generated sequence x.

$x \leftarrow \mathbf{m}^{1:L}, \tilde{x}_0 \leftarrow f_\theta(x)$ ▷ initialize sequence and its estimated distribution
while $\sum_{l=0}^{L} \mathbb{1}(x^l = \mathbf{m}) > 0$ **do**
  $\mathbf{X}_{\text{draft}} \leftarrow [\,], d_{\text{current}} \leftarrow \min(d, \sum_{l=0}^{L} \mathbb{1}(x^l = \mathbf{m}))$
  $K_\tau = \min \left\{ 1, \sum_{i, x^i \neq \mathbf{m}} \mathbb{1}(\max_j f_\theta^{i,j}(x) \geq \tau) \right\}$
  for k in $1 : d_{\text{current}}$ **do**
    $\mathcal{I} \leftarrow g_{\text{top-k}}(x, \tilde{x}_0, K_\tau + k - 1)$ ▷ $k-1$ more draft tokens in addition to K_τ
    $x_{\text{draft}} \leftarrow x$
    for each (a_j, z_j) in $\mathcal{I}$ **do**
      $x_{\text{draft}}^{a_j} \leftarrow z_j$ ▷ replace masked tokens according to decision
    end for
    $\mathbf{X}_{\text{draft}}$.append$(x_{\text{draft}})$
  end for
  $\tilde{\mathbf{X}}_{\text{draft},0} \leftarrow f_\theta(\mathbf{X}_{\text{draft}})$ ▷ all drafts forward in parallel
  $\mathbf{X}_{\text{target}} \leftarrow [\,]$
  for $k = 1 : d_i$ **do**
    $K_\tau = \min \left\{ 1, \sum_{i, x^i \neq \mathbf{m}} \mathbb{1}(\max_j f_\theta^{i,j}(x) \geq \tau) \right\}$
    $\mathcal{I} \leftarrow g_{\text{top-k}}(\mathbf{X}_{\text{draft}}[k], \tilde{\mathbf{X}}_{\text{draft},0}[k], K_\tau)$ ▷ one more threshold-based step ahead
    $x_{\text{target}} \leftarrow \mathbf{X}_{\text{draft}}[k]$
    for each (a_j, z_j) in $\mathcal{I}$ **do**
      $x_{\text{target}}^{a_j} \leftarrow z_j$
    end for
    $\mathbf{X}_{\text{target}}$.append$(x_{\text{target}})$
  end for
  $m \leftarrow 0$
  while $m < d_{\text{current}} - 1$ **do**
    $m_{\text{ref}} \leftarrow m$
    for $k \in \{m+1, \ldots, d_{\text{current}}\}$ **do**
      if $\mathbf{U}(\mathbf{X}_{\text{draft}}[k]) \subseteq \mathbf{U}(\mathbf{X}_{\text{target}}[m_{\text{ref}}])$ **then**
        $m \leftarrow k$
      end if
    end for
    if $m = m_{\text{ref}}$ **then** ▷ if no more future draft matched, ends verification
      break
    end if
  end while
  $x \leftarrow \mathbf{X}_{\text{draft}}[m+1], \tilde{x}_0 \leftarrow \tilde{\mathbf{X}}_{\text{draft},0}[m+1]$
end while
$x \leftarrow x_0$ **return** x

B FreeDave Built on Threshold-Based Parallel Decoding

As discussed in Section 3.4, FreeDave is orthogonal to the choice of the remasking scheduler and can be seamlessly incorporated with threshold-based parallel decoding to further boost inference efficiency. Building upon the notation and the dynamic remasking scheduler $g_{\text{top-k}}(x, \tilde{x}_0, K_\tau)$ established in Appendix A.2, the Threshold FreeDave algorithm is detailed in Algorithm 4. The core intuition of integrating FreeDave with threshold-based decoding lies in resolving the fundamental conflict between dynamic step sizes and parallel drafting. Because the threshold-based scheduler unmask an unpredictable number of tokens based on real-time confidence, traditional exact-match verification becomes fundamentally incompatible, leading to plummeting acceptance rates.

Algorithm 4 overcomes this through a paradigm shift from strict matching to subset verification. Instead of predicting along fixed time intervals, the algorithm concurrently generates a spectrum of progressive drafts. Crucially, it replaces strict state equality with a relaxed yet rigorous subset criterion. For notation convenience, let $\mathbf{U}(x) = \{i \mid x^i \neq \mathbf{m}, i \in \{1, \ldots, L\}\}$ denote the set of unmasked token positions in state x. During verification, as long as the tokens unmasked in a draft form a strict subset of the tokens that the threshold heuristic would naturally unmask at the next step (*i.e.*, $\mathbf{U}(\mathbf{X}_{\text{draft}}[k]) \subseteq \mathbf{U}(\mathbf{X}_{\text{target}}[m_{\text{ref}}])$), the draft is deemed logically sound.

By aggressively searching for and accepting the matched draft with the most unmasked tokens, this elegant subset matching flawlessly absorbs the uncertainty of dynamic thresholds. It guarantees that any accepted draft token at one step would have inevitably been selected by the threshold-based scheduler at the next step, thereby achieving extreme parallel acceleration without sacrificing the original generation quality of threshold-based parallel decoding.

C Detailed Proof of Lemma 1

C.1 Specifications

Our theoretical setting in Section 3.2 contains two structural specifications: (i) the oracle path is an ordered sequence of transitions along the descending time axis, and feasible merges must preserve this order through a consecutive one-step verification chain; and (ii) the verification frontier is non-retreating along this ordered trajectory.

Definition 3 (Ordered One-Step-Verifiable Space) *Let*

$$x_{t_0} \to x_{t_1} \to \cdots \to x_{t_N} \tag{9}$$

be the ordered oracle trajectory. From a trusted state x_{t_i}, let $D_{i,r}$ denote the nested draft merging the next r transitions and $T_{i,r}$ its one-step target from the next batched forward. We define $F(i, j) = 1$ iff

$$T_{i,r} = D_{i,r+1}, \quad \forall r = 1, \ldots, j - i - 1, \tag{10}$$

i.e., *$x_{t_i} \to \cdots \to x_{t_j}$ forms a consecutive verification chain. A feasible path is an ordered sequence $0 = a_0 < a_1 < \cdots < a_M = N$ with $F(a_r, a_{r+1}) = 1$ for every r.*

This ordered specification excludes the infeasible post-mismatch reconvergence: certifying a rejected draft as a later oracle state requires additional sequential forwards from the trusted branch, which have already generated that state and therefore cannot reduce already conducted decoding steps.

Based on F, we further denote the farthest one-step-verifiable endpoint and the intrinsic longest merge from x_{t_i} as

$$R(i) = \max\{j : F(i, j) = 1\}, \qquad m(i) = R(i) - i. \tag{11}$$

Assumption 1 (Verification-Frontier Non-Retreat (VFNR))

$$F(i, j) \Rightarrow F(k, j), \quad i < k < j. \tag{VFNR}$$

Namely, if an oracle interval is verifiable from an earlier state, its remaining successor remains verifiable from any later oracle state within that interval. Equivalently, it can also be interpreted as $R(i + 1) \geq R(i)$ or $m(i + 1) \geq m(i) - 1$ for $\forall i$. Intuitively, for $i < k < j$, the later oracle state x_{t_k} retains all the unmasked tokens of x_{t_i} and unmasks additional ones, thereby providing strictly more

conditioning information for predicting the remaining masked tokens. Hence, if x_{t_j} can already be verified when conditioned on the less informative state x_{t_i}, conditioning on the more informative state x_{t_k} should allow the verification to reach at least x_{t_j}, *i.e.*, the verification frontier should not retreat as more tokens are unmasked.

C.2 Proof of Lemma 1

Under these two specifications, let the greedy longest-merge path be $a_0^\star = 0$, $a_{r+1}^\star = R(a_r^\star)$. Consider any feasible path $0 = b_0 < b_1 < \cdots < b_K = N$. We prove Lemma 1 by mathematical induction that

$$a_r^\star \geq b_r, \quad \forall r. \tag{12}$$

Suppose $a_r^\star \geq b_r$. If $a_r^\star \geq b_{r+1}$, then $a_{r+1}^\star \geq b_{r+1}$. Otherwise, $b_r \leq a_r^\star < b_{r+1}$. Since $F(b_r, b_{r+1}) = 1$, VFNR implies $F(a_r^\star, b_{r+1}) = 1$. Because greedy selects the farthest feasible endpoint, we have $a_{r+1}^\star = R(a_r^\star) \geq b_{r+1}$. Thus, $a_r^\star \geq b_r$ implies $a_{r+1}^\star \geq b_{r+1}$, and from $a_0^\star = b_0$ we inductively prove $a_r^\star \geq b_r, \forall r$. Thus, whenever any feasible path reaches N in K transitions, greedy reaches N in at most K transitions. Hence the greedy longest-merge path is the shortest path in the ordered one-step-verifiable space, and therefore matches the hindsight dynamic programming optimum.

For finite d, if $d \geq \max_r(a_{r+1}^\star - a_r^\star)$, no longest merge is truncated, and induction from $a_0 = 0$ yields the optimal path. Thus, $\|\mathbf{p}^\star\|_\infty$ is a sufficient lower bound for d.

C.3 Empirical Validity of the VFNR Assumption

To examine how well Assumption 1 holds in practice, we evaluate Dream-7B-Instruct on 50 MATH500 trajectories. Along each oracle path, we regenerate the drafts at every oracle state and measure the intrinsic longest merge $m(i)$. Across 13902 adjacent state pairs, the condition $m(i+1) \geq m(i) - 1$ holds for 92.78% of them, indicating that frontier non-retreat is the dominant behavior. To assess the impact of the remaining violations, we further construct the complete empirical feasible DAG and compare the greedy longest merge with the hindsight dynamic programming optimum. The greedy longest merge takes 82.30 decoding steps on average, which is only 3.18% more than the 79.76 steps of the handsight dynamic programming optimum. Therefore, VFNR guarantees the optimality of FreeDave when satisfied, while its occasional violations in practice have only a limited impact on the number of decoding steps.

D More Experimental Details

D.1 Experimental Settings

Following Wang et al. [17], for the TraDo models, the input prompt is right-padded with mask token **m** such that the block size B divides it, and the total number of generation blocks is $\lceil \frac{(|\text{prompt}| \bmod B) + \text{max_gen_length}}{B} \rceil$. And for LLaDA-8B-Instruct and Dream-7B-Instruct, the first block starts right after the input prompt without padding. For the attention mask and KV Cache, LLaDA-8B-Instruct and Dream-7B-Instruct use full-sequence bidirectional attention and dual cache [28], and TraDo-4B-Instruct and TraDo-8B-Instruct use block attention (*i.e.*, intra-block bidirectional and inter-block causal) and prefix cache.

For the decoding hyperparameters, we use a temperature of 0, and following Wang et al. [17], we use a threshold of 0.95 for LLaDA-8B-Instruct and Dream-7B-Instruct, and 0.9 for the TraDo models. Besides, an early exit mechanism is applied to all the models. Specifically, when the current block is fully decoded, if all the tokens are $\langle|\textbf{endoftext}|\rangle$, the generation stops, and the current sequence is returned. Regarding the attention mask, the TraDo models apply the block-causal mask, which is inter-block causal and intra-block bidirectional, and LLaDA-8B-Instruct and Dream-7B-Instruct apply the full bidirectional mask.

For the inference framework, we build our inference pipeline on plain PyTorch code. We did not use popular engines like vLLM and SGLang, but every experiment run is guaranteed to be under the same test environment. For the evaluation, we incorporate our decoding algorithm into lm-eval-harness [48], which is a commonly used evaluation toolkit. All the models are evaluated on a single NVIDIA H200 GPU. Our codebase is publicly available at `https://github.com/cychomatica/FreeDave`.

D.2 Detailed Results of Benchmark Evaluations

In Table 2–5, we provide the detailed numbers of benchmark evaluations. For math reasoning benchmarks, we choose MATH500 [41] and GSM8K [42]. For code generation benchmarks, we choose MBPP [43] and HumanEval [44]. It is worth noting that the TPF of static decoding for TraDo models is around 0.8 instead of 1 due to a separate model forward for cache refresh after every block is fully decoded. Since the block size is 4, there will be 5 model forwards every 4 decoded tokens.

Benchmark	Decoding Strategy	Acc (%) ↑	TPF ↑	TPS ↑
MATH500 (4-shot)	Static	37.40	1.00	37.05
	Static FreeDave (Ours)	36.80 (-0.60)	2.45 (2.45×)	74.19 (2.00×)
	Threshold	37.40 (+0.00)	2.61 (2.61×)	85.89 (2.32×)
	Threshold FreeDave (Ours)	36.80 (-0.60)	4.40 (4.40×)	119.47 (3.22×)
GSM8K (5-shot)	Static	78.77	1.00	34.23
	Static FreeDave (Ours)	78.62 (-0.15)	2.50 (2.50×)	68.52 (2.00×)
	Threshold	78.85 (+0.08)	2.61 (2.61×)	77.47 (2.26×)
	Threshold FreeDave (Ours)	78.09 (-0.68)	4.39 (4.39×)	105.94 (3.09×)
MBPP (3-shot)	Static	36.00	1.00	36.16
	Static FreeDave (Ours)	36.20 (+0.20)	2.58 (2.58×)	73.34 (2.03×)
	Threshold	36.00 (+0.00)	2.74 (2.74×)	82.48 (2.28×)
	Threshold FreeDave (Ours)	36.20 (+0.20)	4.81 (4.81×)	116.60 (3.22×)
HumanEval (0-shot)	Static	35.98	1.00	41.58
	Static FreeDave (Ours)	35.98 (+0.00)	2.45 (2.45×)	84.18 (2.02×)
	Threshold	35.98 (+0.00)	3.33 (3.33×)	120.97 (2.91×)
	Threshold FreeDave (Ours)	35.98 (+0.00)	5.48 (5.48×)	163.75 (3.94×)

Table 2: LLaDA-8B-Instruct results on math reasoning and code generation benchmarks.

Benchmark	Decoding Strategy	Acc (%) ↑	TPF ↑	TPS ↑
MATH500 (4-shot)	Static	44.80	1.00	62.51
	Static FreeDave (Ours)	46.20 (+1.40)	2.45 (2.45×)	115.16 (1.84×)
	Threshold	44.60 (-0.20)	2.35 (2.35×)	125.59 (2.01×)
	Threshold FreeDave (Ours)	46.40 (+1.60)	3.97 (3.97×)	163.34 (2.61×)
GSM8K (5-shot)	Static	79.91	1.00	58.86
	Static FreeDave (Ours)	79.23 (-0.68)	2.49 (2.49×)	107.58 (1.83×)
	Threshold	79.83 (-0.08)	2.52 (2.52×)	121.40 (2.06×)
	Threshold FreeDave (Ours)	78.77 (-1.14)	4.10 (4.10×)	150.35 (2.55×)
MBPP (3-shot)	Static	52.80	1.00	62.66
	Static FreeDave (Ours)	53.80 (+1.00)	2.67 (2.67×)	123.06 (1.96×)
	Threshold	53.00 (+0.20)	2.60 (2.60×)	136.96 (2.19×)
	Threshold FreeDave (Ours)	54.00 (+1.20)	4.27 (4.27×)	171.56 (2.74×)
HumanEval (0-shot)	Static	54.88	1.00	68.74
	Static FreeDave (Ours)	56.10 (+1.22)	2.60 (2.60×)	133.95 (1.95×)
	Threshold	54.88 (+0.00)	2.40 (2.40×)	146.19 (2.13×)
	Threshold FreeDave (Ours)	56.10 (+1.22)	4.03 (4.03×)	185.08 (2.69×)

Table 3: Dream-v0-Instruct-7B results on math reasoning and code generation benchmarks.

D.3 Details of Non-determinism Attribution Experiments

We rigorously attribute this empirical variance to system-level numerical non-determinism, primarily originating from floating-point non-associativity across different batch sizes and non-batch-invariant GPU kernels [47]. To isolate our algorithmic design from these hardware-level execution noises, we conduct an ablation study under a strict "determinism debug mode" with the following factors controlled:

Benchmark	Decoding Strategy	Acc (%) ↑	TPF ↑	TPS ↑
MATH500 (4-shot)	Static	64.80	0.80	28.63
	Static FreeDave (Ours)	66.40 (+1.60)	1.68 (2.10×)	54.69 (1.91×)
	Threshold	65.60 (+0.80)	1.51 (1.89×)	53.71 (1.88×)
	Threshold FreeDave (Ours)	65.40 (+0.60)	1.94 (2.42×)	61.65 (2.15×)
GSM8K (5-shot)	Static	90.67	0.79	27.69
	Static FreeDave (Ours)	90.52 (-0.15)	1.64 (2.08×)	52.97 (1.91×)
	Threshold	90.37 (-0.30)	1.42 (1.80×)	50.14 (1.81×)
	Threshold FreeDave (Ours)	89.76 (-0.91)	1.92 (2.43×)	59.57 (2.15×)
MBPP (3-shot)	Static	60.40	0.79	27.01
	Static FreeDave (Ours)	60.00 (-0.40)	1.52 (1.92×)	47.17 (1.75×)
	Threshold	56.00 (-4.40)	1.21 (1.53×)	42.39 (1.57×)
	Threshold FreeDave (Ours)	57.20 (-3.20)	1.80 (2.28×)	55.02 (2.04×)
HumanEval (0-shot)	Static	75.61	0.79	27.50
	Static FreeDave (Ours)	76.22 (+0.61)	1.52 (1.92×)	48.17 (1.75×)
	Threshold	76.22 (+0.61)	1.14 (1.44×)	39.55 (1.44×)
	Threshold FreeDave (Ours)	77.44 (+1.83)	1.72 (2.18×)	52.88 (1.92×)

Table 4: TraDo-4B-Instruct results on math reasoning and code generation benchmarks.

Benchmark	Decoding Strategy	Acc (%) ↑	TPF ↑	TPS ↑
MATH500 (4-shot)	Static	69.20	0.80	26.29
	Static FreeDave (Ours)	69.40 (+0.20)	1.64 (2.05×)	51.68 (1.97×)
	Threshold	69.40 (+0.20)	1.41 (1.76×)	46.43 (1.77×)
	Threshold FreeDave (Ours)	69.20 (+0.00)	1.90 (2.37×)	58.67 (2.23×)
GSM8K (5-shot)	Static	93.03	0.80	25.45
	Static FreeDave (Ours)	92.87 (-0.16)	1.62 (2.02×)	50.47 (1.98×)
	Threshold	92.49 (-0.54)	1.37 (1.71×)	44.41 (1.74×)
	Threshold FreeDave (Ours)	92.57 (-0.46)	1.90 (2.37×)	60.25 (2.37×)
MBPP (3-shot)	Static	65.40	0.79	24.38
	Static FreeDave (Ours)	65.80 (+0.40)	1.48 (1.87×)	45.30 (1.86×)
	Threshold	63.80 (-1.60)	1.16 (1.47×)	37.48 (1.54×)
	Threshold FreeDave (Ours)	64.20 (-1.20)	1.70 (2.15×)	52.31 (2.15×)
HumanEval (0-shot)	Static	83.54	0.79	25.35
	Static FreeDave (Ours)	83.54 (+0.00)	1.52 (1.92×)	45.94 (1.81×)
	Threshold	82.93 (-0.61)	1.15 (1.46×)	35.20 (1.39×)
	Threshold FreeDave (Ours)	81.71 (-1.83)	1.69 (2.14×)	51.01 (2.01×)

Table 5: TraDo-8B-Instruct results on math reasoning and code generation benchmarks.

- Set the data type of model parameters and tensors to float32 for a higher numerical precision.
- Replace the forward operation of concatenated FreeDave draft branches with an individual forward operation for each draft branch, such that the model computes tensors with the same size and the same GPU kernels are used by both static decoding and FreeDave decoding.

Under the determinism debug mode, for different models on HumanEval, we calculated the sequence-level and token-level exact match rate. Specifically, the sequence-level match rate is defined as

$$R_{\text{sequence}}(\mathbf{X}, \mathbf{Y}) = \frac{\sum_{i=1}^{N} \mathbb{1}(\mathbf{X}_i = \mathbf{Y}_i)}{N}, \tag{13}$$

and the token-level match rate is defined as

$$R_{\text{token}}(\mathbf{X}, \mathbf{Y}) = \frac{\sum_{i=1}^{N} |\text{LongestSharedPrefix}(\mathbf{X}_i, \mathbf{Y}_i)|}{\sum_{i=1}^{N} \max\left(|\mathbf{X}_i|, |\mathbf{Y}_i|\right)}, \tag{14}$$

where $\mathbf{X}_i$ and $\mathbf{Y}_i$ are the outputs static and FreeDave decoding for the i-th input, respectively, and N is the total number of inputs. The all-100% match rate results are shown in Table 1.

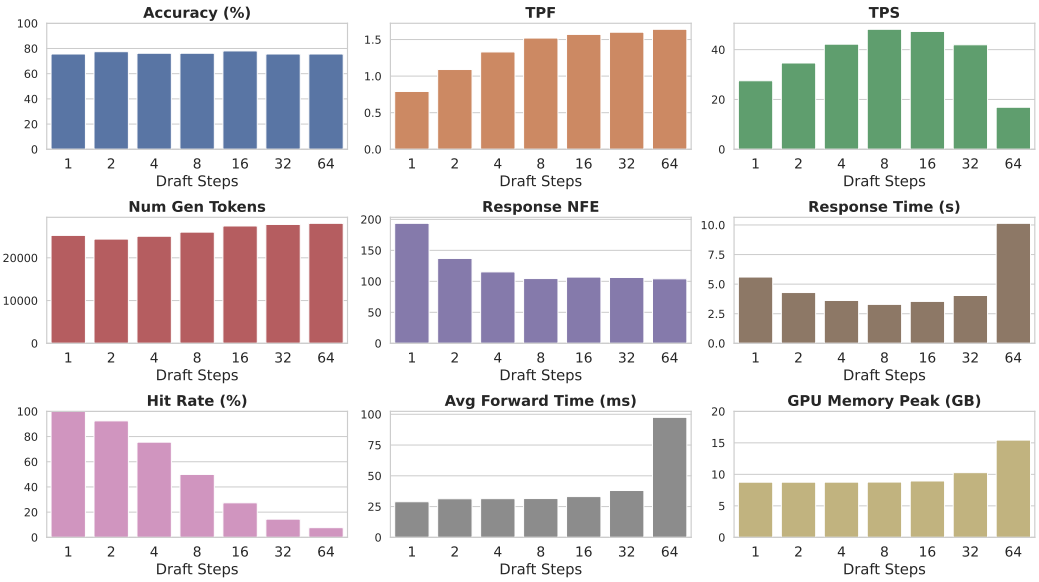

Figure 4: Ablation details

D.4 Ablation Details

In addition to our evaluations on benchmarks, we also investigate the impact of different draft steps d such that we can get the best trade-off between memory overhead and inference efficiency. As shown in Figure 4, we set $d \in \{1, 2, 4, 8, 16, 32, 64\}$, and compare the accuracy, TPS, TPF, number of generated tokens, response NFE, response time, hit rate, average forward pass time, and GPU memory peak usage during the inference stage of TraDo-4B-Instruct on HumanEval. Specifically, hit rate (*i.e.*, acceptance rate) is calculated by the number of accepted tokens divided by the number of draft tokens.

D.5 Additional Evaluations on Q&A Benchmarks

In addition to math reasoning and code generation, we also apply our method to different models and compare with other baselines on Q&A benchmarks, including GPQA[45] and TruthfulQA[46]. Specifically, for TruthfulQA, we use `rougeL_acc` as the accuracy metric. The detailed results are shown in Table 6.

E Memory Cost Analysis

E.1 Theoretical Memory Complexity

Let b denote the inference batch size, B the block size, d the number of draft steps, and L the total sequence length (*i.e.*, prompt plus generated tokens). Compared to static decoding, which evaluates the B tokens of the current block in each forward pass, FreeDave evaluates the current block together with the $d-1$ additional draft branches, *i.e.*, $(d-1)B$ additional tokens. Under FlashAttention [49], which avoids materializing the $L \times L$ attention matrix and reduces the attention memory complexity from quadratic to linear in the sequence length, the peak memory costs of static decoding and FreeDave can be expressed as

$$M_{\text{Static}} = M_\theta + M_0 + \mathcal{O}\left(bL\, m_{\text{tok}}\right), \tag{15}$$

$$M_{\text{FreeDave}} = M_\theta + M_0 + \mathcal{O}\left(b\left(L + (d-1)B\right) m_{\text{tok}}\right), \tag{16}$$

where M_θ denotes the memory for loading the model weights, M_0 the runtime overheads (*e.g.*, CUDA context and allocator workspace), and m_{tok} the working-memory footprint per token (*e.g.*, KV cache and activations). Consequently, the additional memory introduced by FreeDave is

$$\Delta M = M_{\text{FreeDave}} - M_{\text{Static}} = \mathcal{O}\left(b(d-1)B\, m_{\text{tok}}\right). \tag{17}$$

Table 6: Different models on Q&A benchmarks.

Benchmark	Model	Decoding Strategy	Acc (%) ↑	TPF ↑	TPS ↑
GPQA (0-shot)	LLaDA-8B-Instruct	Static	23.44	1.00	43.01
		Static FreeDave (Ours)	21.43 (-2.01)	2.37 (2.37×)	84.50 (1.96×)
		Threshold	23.21 (-0.23)	2.13 (2.13×)	85.16 (1.98×)
		Threshold FreeDave (Ours)	21.65 (-1.79)	3.67 (3.67×)	123.06 (2.86×)
	Dream-7B-Instruct	Static	27.23	1.00	69.53
		Static FreeDave (Ours)	26.79 (-0.44)	2.44 (2.44×)	130.34 (1.87×)
		Threshold	27.23 (+0.00)	2.29 (2.29×)	144.85 (2.08×)
		Threshold FreeDave (Ours)	26.79 (-0.44)	3.87 (3.87×)	188.48 (2.71×)
	TraDo-4B-Instruct	Static	19.42	0.80	25.92
		Static FreeDave (Ours)	21.43 (+2.01)	1.53 (1.91×)	47.89 (1.85×)
		Threshold	21.88 (+2.46)	1.22 (1.52×)	39.41 (1.52×)
		Threshold FreeDave (Ours)	21.88 (+2.46)	1.70 (2.12×)	52.68 (2.03×)
	TraDo-8B-Instruct	Static	34.82	0.80	25.98
		Static FreeDave (Ours)	34.82 (+0.00)	1.44 (1.80×)	44.48 (1.71×)
		Threshold	35.04 (+0.22)	1.09 (1.36×)	35.22 (1.36×)
		Threshold FreeDave (Ours)	35.04 (+0.22)	1.57 (1.96×)	48.17 (1.85×)
TruthfulQA (0-shot)	LLaDA-8B-Instruct	Static	38.56	1.00	42.76
		Static FreeDave (Ours)	38.31 (-0.25)	2.38 (2.38×)	83.91 (1.96×)
		Threshold	38.56 (+0.00)	1.96 (1.96×)	78.13 (1.83×)
		Threshold FreeDave (Ours)	38.43 (-0.13)	3.52 (3.52×)	115.00 (2.69×)
	Dream-7B-Instruct	Static	52.63	1.00	71.46
		Static FreeDave (Ours)	52.88 (+0.25)	2.87 (2.87×)	147.87 (2.07×)
		Threshold	52.51 (-0.12)	2.21 (2.21×)	143.13 (2.00×)
		Threshold FreeDave (Ours)	52.88 (+0.25)	4.29 (4.29×)	202.56 (2.83×)
	TraDo-4B-Instruct	Static	49.57	0.78	24.61
		Static FreeDave (Ours)	49.57 (+0.00)	1.41 (1.81×)	42.69 (1.73×)
		Threshold	48.71 (-0.86)	0.95 (1.22×)	29.88 (1.21×)
		Threshold FreeDave (Ours)	49.08 (-0.49)	1.52 (1.95×)	45.97 (1.87×)
	TraDo-8B-Instruct	Static	55.69	0.78	24.75
		Static FreeDave (Ours)	55.20 (-0.49)	1.42 (1.82×)	41.93 (1.69×)
		Threshold	55.69 (+0.00)	0.95 (1.22×)	29.42 (1.19×)
		Threshold FreeDave (Ours)	55.81 (+0.12)	1.54 (1.97×)	45.12 (1.82×)

Thus, the overhead of FreeDave scales linearly with the batch size b, the block size B, and the number of draft steps d, which is consistent with the growth of peak memory *w.r.t.* d observed in our ablation study in Figure 4. Importantly, when b, B, and d are fixed, ΔM is independent of the sequence length L, and the relative overhead $\Delta M/(M_{\text{Static}} - M_\theta - M_0) = \mathcal{O}\left((d-1)B/L\right)$ diminishes as L grows.

E.2 Empirical Validation with Different Generation Lengths

Empirically, we evaluate Dream-7B-Instruct on MATH500 with different generation lengths in $\{1024, 2048, 4096\}$, using the same block size $B = 32$ and draft steps $d = 4$ as in Section 4.1. Specifically, the memory for loading the model weights under bf16 is $M_\theta = 14569$ MiB. The detailed results are shown in Table 7, and the peak allocated memory is further visualized in Figure 5.

As shown in Figure 5, the peak memory of all the decoding strategies grows linearly with the generation length, which matches the $\mathcal{O}(bL\, m_{\text{tok}})$ term in Eq. (15) and Eq. (16). Meanwhile, the additional memory of FreeDave over its base decoding strategy remains nearly constant ($\sim$30 MiB) across all the generation lengths, which is consistent with Eq. (17), where ΔM depends only on b, B, and d rather than L.

Table 7: Results of Dream-7B-Instruct on MATH500 under different generation lengths.

Generation Length	Decoding Strategy	Acc (%) ↑	TPF ↑	TPS ↑	GPU Peak Memory (MiB)	
					Allocated	Reserved
1024	Static	44.80	1.00	62.51	16245	16746
	Static FreeDave (Ours)	46.20	2.45	115.16	16276	16752
	Threshold	44.60	2.35	125.59	16247	16746
	Threshold FreeDave (Ours)	46.40	3.97	163.34	16277	16752
2048	Static	44.80	1.00	50.21	16929	17624
	Static FreeDave (Ours)	45.80	2.40	87.61	16958	17630
	Threshold	44.80	2.53	99.84	16929	17624
	Threshold FreeDave (Ours)	45.40	4.16	123.87	16958	17630
4096	Static	44.60	1.00	35.86	18337	19552
	Static FreeDave (Ours)	44.60	2.36	57.93	18367	19558
	Threshold	44.60	3.12	72.10	18337	19552
	Threshold FreeDave (Ours)	44.20	5.03	83.76	18367	19558

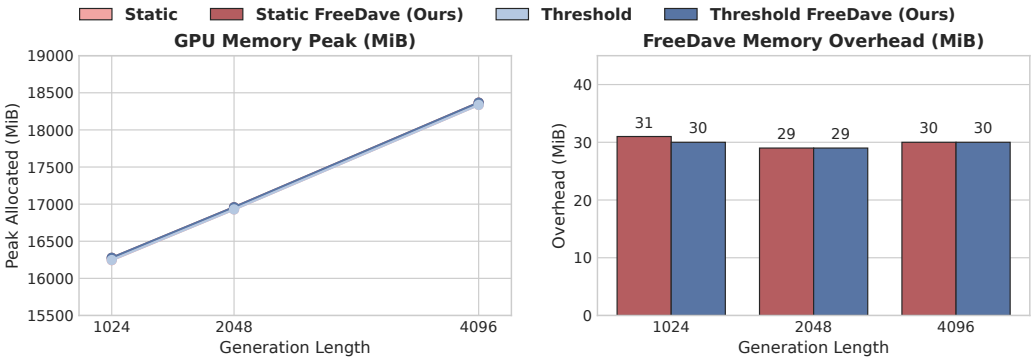

Figure 5: GPU peak memory and FreeDave memory overhead for Dream-7B-Instruct on MATH500 under different generation lengths.

F Details of Block Diffusion Extensions

F.1 Block Tree Attention Details

For block diffusion, instead of expanding draft candidates along the batch dimension, which incurs substantial memory overhead, we compactly pack all draft candidates into a single sequence and employ a block-level tree attention topology, as illustrated in Figure 6. This structure prevents redundant KV cache allocation for shared prefixes across different draft candidates and reduces the memory footprint during parallel verification.

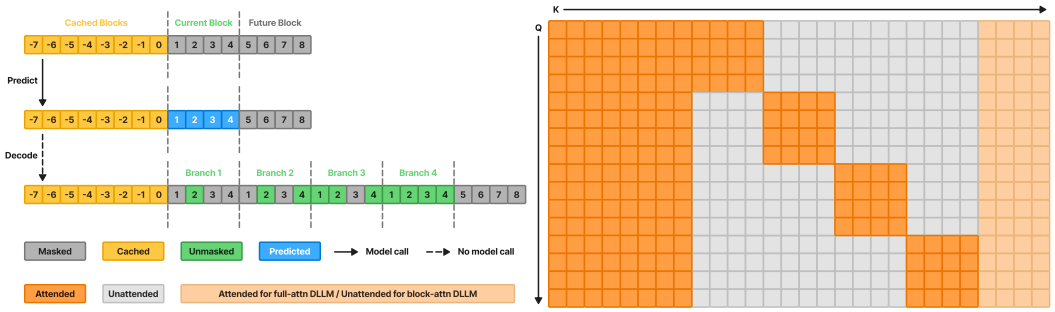

Figure 6: FreeDave with block tree attention tailored for block diffusion.

F.2 Eager Mode Details and the Equivalence to Static Decoding Variant

Eager Mode permits pre-drafting and pre-verifying tokens across future blocks while maintaining a strict left-to-right acceptance priority. This not only eliminates boundary-induced truncations but also explicitly enables scaling the draft budget d beyond the block size itself. More importantly, as demonstrated in Figure 7, Eager Mode is structurally equivalent to a static decoding variant with delayed cache updates, thereby strictly preserving the generation quality while unlocking significantly higher decoding efficiency. In the example of Figure 7, for standard static decoding (upper left), the tokens x^5, x^7, and x^8 are generated using cached KV of $x^{-7:4}$. When the cache update is delayed (upper right), x^5, x^7, and x^8 are generated using cached KV of $x^{-7:0}$ and computed KV of $x^{1:4}$, which aligns with FreeDave with Eager Mode enabled. The only difference is the cached and computed KV of $x^{1:4}$, where the latter should be more informative and at least as good as the former.

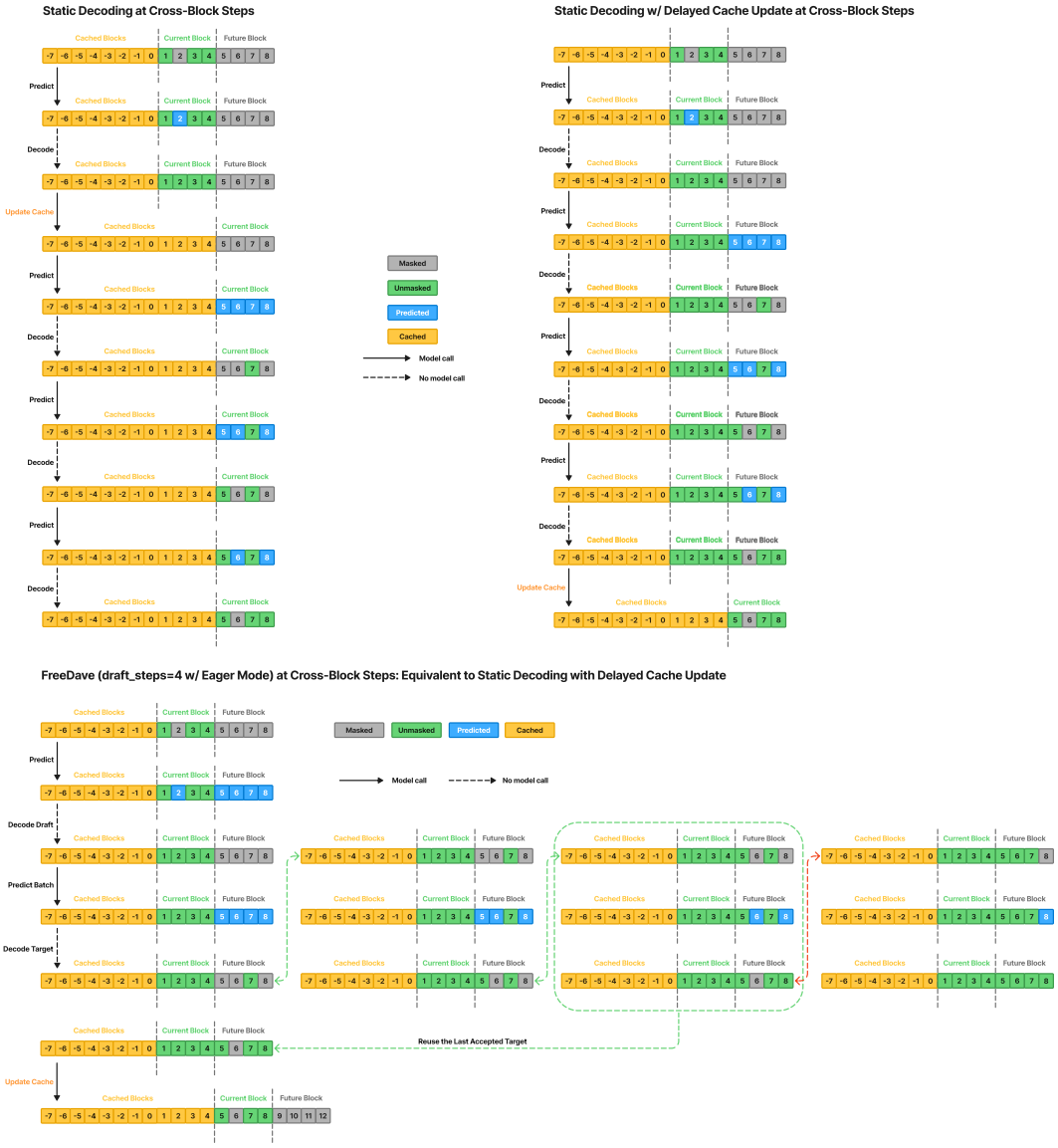

Figure 7: The equivalence between FreeDave Eager Mode and the variant of static decoding with delayed cache update.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our contributions are clearly summarized in the last paragraph of Section 1.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We provide them in Section 3.2 and Appendix C.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have open-sourced our codebase at `https://github.com/cychomatica/FreeDave` and provided detailed README files and code comments to ensure reproducibility.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We use open-sourced benchmarks and have open-sourced our codebase at `https://github.com/cychomatica/FreeDave`.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: This work does not involve training. We provide detailed descriptions of all the evaluation configurations in Section 4 and Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We use a sampling temperature of 0 for all the methods. The experiments are algorithmically deterministic and reproducible. Error bars are unnecessary in our setting.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We have provided sufficient information for reproducibility in Section 4.1 and Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We propose a fast decoding algorithm. There are no specific ethical issues involved.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: We propose a fast decoding algorithm. There are no specific societal impacts involved.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have properly credited all assets used in this paper, which can be verified in our references.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We have open-sourced our codebase at `https://github.com/cychomatica/FreeDave` and provided detailed README files and code comments to ensure reproducibility.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.