Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-26)
Survey Track

Accelerating Masked Diffusion Large Language Models: A Survey of Efficient Inference Techniques

Daehoon Gwak[1], **Minhyung Lee**[2], **Junwoo Park**[1], **Jaegul Choo**[1]
[1]KAIST AI
[2]Yonsei University
daehoon.gwak@kaist.ac.kr, jchoo@kaist.ac.kr

Abstract

Diffusion large language models (dLLMs) offer a theoretical advantage in parallel generation over standard autoregressive models. However, parallel generation alone does not guarantee practical speedups. Realizing this efficiency requires specialized inference mechanisms, such as diffusion-aware caching and reuse. Consequently, as inference efficiency becomes a prerequisite for practical deployment, recent research has actively explored acceleration techniques across algorithms, architectures, and systems. However, rigorous comparisons remain difficult, as end-to-end latency stems from intricate trade-offs between algorithmic, architectural, and system-level factors that are often conflated in existing benchmarks. In this survey, we introduce a unified latency decomposition framework for dLLMs to disentangle these factors and analyze their impact on inference speed in real deployments. Guided by this framework, we categorize acceleration techniques along three axes covering algorithmic innovations, architectural and system optimizations, and inference-time scaling. Finally, we provide guidelines for reproducible benchmarking and highlight open challenges for realizing the full potential of parallel generation.

1 Introduction

Autoregressive (AR) models remain the dominant paradigm for large language models. However, their inference cost scales linearly with the generated length. Specifically, producing L tokens requires at least L dependent decoding iterations. This sequential dependency is a fundamental barrier for latency-sensitive applications and high-throughput serving, motivating extensive research on parallel generation. Recently, Diffusion-based large language models (dLLMs) [Nie *et al.*, 2025; Ye *et al.*, 2025b; Inception Labs *et al.*, 2025] have emerged as a promising alternative. They replace strict left-to-right decoding with iterative refinement, enabling parallel updates across multiple token positions [Austin *et al.*, 2021; Lou *et al.*, 2024; Sahoo *et al.*, 2024].

However, parallel generation alone does not guarantee practical speedups. Realizing this efficiency requires specialized inference mechanisms, such as diffusion-aware caching and reuse, which introduce their own complexities. Consequently, as inference efficiency becomes a prerequisite for practical deployment, recent research has actively explored acceleration techniques across algorithms, architectures, and systems. However, rigorous comparisons remain difficult, as end-to-end latency stems from intricate trade-offs between these factors. For instance, an algorithmic method might reduce refinement steps but increase per-step policy overhead, while a system optimization might improve throughput at the cost of memory pressure. These algorithmic, architectural, and system-level impacts are often conflated in existing benchmarks, making it challenging to isolate the true sources of performance gains in real-world deployments.

To address these challenges, this survey focuses specifically on inference-time efficiency in diffusion-based language generation. While recent surveys provide broad overviews of diffusion language modeling [Li *et al.*, 2025e] or parallel generation mechanisms [Zhang *et al.*, 2025a], our work synthesizes the literature through a distinct, deployment-oriented efficiency lens. Specifically, we aim to clarify: (i) the determinants of end-to-end latency, throughput, and memory in realistic workloads; (ii) the inherent trade-offs between quality, speed, and memory across different acceleration techniques; and (iii) the best practices for reproducible benchmarking to ensure meaningful comparisons.

This survey makes three primary contributions:

- We introduce a unified latency decomposition framework tailored to dLLMs, providing a rigorous basis for analyzing inference efficiency beyond nominal step counts.
- We propose a structured taxonomy that categorizes acceleration techniques into algorithmic, architectural, and system-level optimizations, explicitly mapping them to the terms in our efficiency framework.
- We establish best practices for reproducible benchmarking, covering latency, throughput, and memory measurement, and identify key open challenges at the intersection of algorithms and systems.

The remainder of this survey is organized as follows. We begin by reviewing the fundamentals of discrete diffusion decoding (Section 2) and presenting our latency decomposition framework (Section 3). We then survey acceleration methods, grouped into algorithmic innovations (Section 4), architectural

Family	Primary levers (Eq. 1)	Key idea	Section
Algorithmic efficiency			
Schedules & policies	$T\downarrow$, $C_{\text{policy}}\uparrow$	Fewer steps via confidence/dilated schedules and selective updates; overhead depends on implementation.	Section 4.1
Decoding algorithm	$T\downarrow$ or $G_t\downarrow$, $C_{\text{policy}}\uparrow$	Larger parallel progress or draft-and-verify; gains hinge on acceptance and orchestration cost.	Section 4.2
Distillation / consistency	$T\downarrow$	Few-step samplers shift cost to training; quality/calibration can become sensitive.	Section 4.3
Architectural & systems efficiency			
Architecture & numerics	$C_{\text{fwd}}\downarrow$, Mem$\downarrow$	Reduce per-forward cost via sparse/structured decoding or quantization; depends on kernel maturity.	Section 5.1
Caching & reuse	$C_{\text{fwd}}\downarrow$, $C_{\text{sys}}\downarrow$, Mem$\uparrow$	Reuse KV/activations across refinement steps; explicit memory–speed trade-off.	Section 5.2
System-level serving	$C_{\text{sys}}\downarrow$	Kernel/orchestration/microbatching improvements matter most for batch-1 and deployment regimes.	Section 5.3
Inference-time scaling			
Guidance & search	$G_t\uparrow$, $C_{\text{policy}}\uparrow$, Mem$\uparrow$	Multi-pass decoding improves quality/constraints but increases total evaluations and memory pressure.	Section 6

Table 1: Compact taxonomy of inference-efficiency techniques for diffusion LLMs, organized by the latency decomposition in Eq. 1. Each family is summarized by the primary terms it tends to affect (arrows indicate the typical direction at a matched quality target), and the last column points to the section where we discuss representative methods, trade-offs, and evaluation considerations in detail.

and system optimizations (Section 5), and inference-time scaling strategies (Section 6). Finally, we provide a practitioner's guide for interpreting efficiency claims and mapping objectives to the levers in Eq. (1) (Section 7), and discuss future research directions (Section 8).

2 Background: Discrete Diffusion Decoding for Language

This section reviews essential mechanics and notation of discrete diffusion models needed for our *inference-time efficiency* discussion. We focus on the common *masked* discrete diffusion setting, since it induces an explicit refinement loop with repeated Transformer evaluations which is the primary bottleneck addressed by recent acceleration methods.

Setup and notation. Let $\mathcal{V}$ be a vocabulary and let $\mathbf{x} = (x_1, \ldots, x_L) \in \mathcal{V}^L$ denote a length-L generated span, typically conditioned on a prompt or context $\mathbf{c}$. When discussing inference cost, we distinguish the generated length L from the total length processed by the model per evaluation, L_{in} (e.g., prompt length plus the generated span). Discrete diffusion introduces a sequence of states $\mathbf{x}_T, \mathbf{x}_{T-1}, \ldots, \mathbf{x}_0$, where larger t indicates heavier corruption; $\mathbf{x}_0$ is the final output and $\mathbf{x}_T$ is commonly an all-`[MASK]` template. We write $\mathcal{M}_t \subseteq \{1, \ldots, L\}$ for the masked positions at step t.

A standard forward corruption in this setting is *absorbing* masking: as t increases, a subset of positions is replaced by `[MASK]`, yielding a Markov chain over discrete states [Austin *et al.*, 2021]. A corruption schedule controls how the expected mask ratio evolves with t, which determines how much information remains visible at intermediate states. Given a partially corrupted sequence $\mathbf{x}_t$ and condition $\mathbf{c}$, the denoising model predicts token distributions, often for all positions even if the loss is applied only on masked positions [Sahoo *et al.*, 2024]. For our purposes, the key operational point is that inference repeatedly evaluates the model under changing corruption patterns, and each evaluation typically processes a sequence of length L_{in}.

Reverse-time decoding and update sparsity. At inference time, decoding starts from $\mathbf{x}_T$ (often all `[MASK]`) and iteratively refines toward $\mathbf{x}_0$. A generic reverse step from $\mathbf{x}_t$ to $\mathbf{x}_{t-1}$ consists of: (1) **Predict:** run the model on $\mathbf{x}_t$ and $\mathbf{c}$ to obtain per-position token distributions; (2) **Select:** choose an update set $\mathcal{U}_t \subseteq \mathcal{M}_t$ (which masked positions to fill or revise); (3) **Commit:** sample or take $\arg\max$ on $\mathcal{U}_t$ to form $\mathbf{x}_{t-1}$, optionally allowing remasking or other corrections. When $|\mathcal{U}_t|$ is large, many tokens are updated in parallel, which is the main pathway to potential speedups over left-to-right AR decoding.

Although the model may be evaluated on the full sequence, many practical schedules and policies change only a subset of positions per step. Let $\Delta_t \subseteq \{1, \ldots, L\}$ denote the positions whose token values actually change between steps t and $t-1$ (often $\Delta_t \subseteq \mathcal{U}_t$). When $|\Delta_t| \ll L$, the trajectory exhibits *update sparsity*, a property exploited by both adaptive scheduling/policy choices and diffusion-aware reuse mechanisms surveyed later. However, quantifying the actual speedups from these techniques requires disentangling multiple interacting factors such as step count, per-step compute, and systems overhead, which motivates the latency decomposition framework we introduce next.

3 Latency Decomposition and Measuring Inference Efficiency

Efficiency claims for diffusion-based LLM inference are easy to misinterpret. In masked dLLMs, the refinement step count T is a coarse proxy. However, wall-clock latency depends on how many *full-sequence* model evaluations are executed, how expensive each evaluation is under the chosen hardware/precision/length/batch regime, and how much overhead

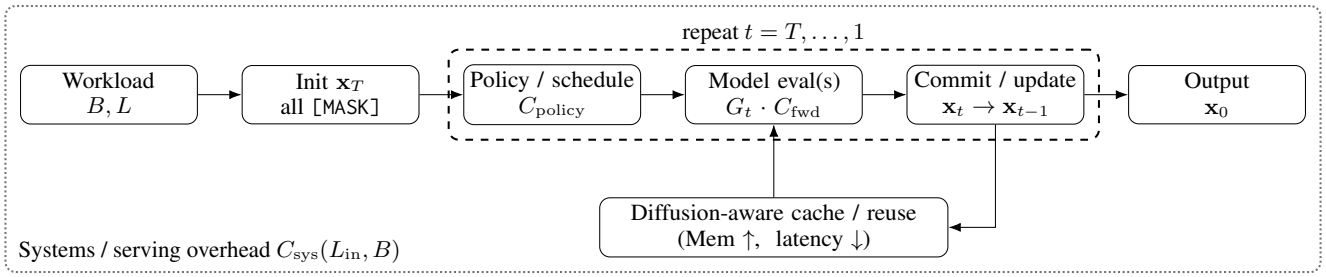

Figure 1: End-to-end inference schematic for masked/discrete diffusion decoding and the latency terms in Eq. 1. The reverse-time refinement loop (repeat $t = T, \ldots, 1$) alternates lightweight policy/schedule decisions (C_{policy}) and G_t model evaluations (each with cost C_{fwd}, dependent on the processed length L_{in} and batch size B) to update $\mathbf{x}_t \to \mathbf{x}_{t-1}$. Diffusion-aware caching/reuse can reduce redundant per-step computation or orchestration at the cost of higher memory, while systems/serving overhead $C_{\text{sys}}(L_{\text{in}}, B)$ applies to the entire request.

arises from scheduling policies and serving systems. This section introduces a compact decomposition that we use as a unifying lens throughout the survey.

Latency decomposition. Consider conditional generation under prompt/context $\mathbf{c}$ with batch size B and generated span length L. Let L_{in} denote the total sequence length processed per model evaluation, typically prompt length plus the generated span. Most masked/discrete decoders run an iterative loop for T refinement steps, and each step typically performs at least one full-sequence model evaluation. Moreover, inference-time mechanisms such as guidance or deliberate compute scaling can require multiple evaluations per nominal step [Schiff *et al.*, 2025; Dang *et al.*, 2025]. We decompose end-to-end latency as:

$$\text{Latency}(L_{\text{in}}, B) \approx \sum_{t=1}^{T} \Big(G_t \cdot C_{\text{fwd}}(L_{\text{in}}, B) + C_{\text{policy}}(t; L) \Big) + C_{\text{sys}}(L_{\text{in}}, B), \tag{1}$$

where $C_{\text{fwd}}(L_{\text{in}}, B)$ is the cost of one model forward pass; $G_t \geq 1$ is the number of forward passes executed at step t (capturing multi-pass decoding); $C_{\text{policy}}(t; L)$ captures schedule/policy overhead (e.g., update-set selection, remasking, stopping); and $C_{\text{sys}}(L_{\text{in}}, B)$ captures systems overhead such as kernel dispatch, memory movement, cache management, and framework orchestration, which can be substantial in interactive and microbatching regimes [Ma *et al.*, 2025b; Fan *et al.*, 2025]. We also define the total number of model evaluations as $N_{\text{fwd}} \triangleq \sum_{t=1}^{T} G_t$.

Equation (1) clarifies that "reducing T" is only one lever. Speedups can come from reducing T (fewer refinement iterations), reducing N_{fwd} (less multi-pass compute), lowering C_{fwd} (architectural/numerical improvements), lowering C_{policy} (lighter scheduling and fewer synchronizations), or lowering C_{sys} (serving and orchestration optimizations). Table 1 summarizes how major technique families map to these terms, and Figure 1 shows where each cost arises in the decoding pipeline.

Common pitfalls include treating T as a speed proxy without accounting for N_{fwd}, using ambiguous "tokens/sec" definitions, and timing without a consistent end-to-end protocol (warmup, synchronization, and clear inclusion/exclusion of host overhead). In the remainder of the survey, we organize acceleration methods by which terms in Eq. (1) they primarily target, making trade-offs explicit.

4 Algorithmic Efficiency: Reducing T and N_{fwd}

Algorithmic acceleration methods primarily reduce inference latency by decreasing the refinement iterations T and/or the total number of model evaluations $N_{\text{fwd}} = \sum_{t=1}^{T} G_t$ in Eq. 1. They modify the *sampling trajectory*—the schedule, the update policy, or the decoding algorithm—rather than the model kernels or serving stack. In practice, realized gains depend not only on T but also on policy overhead C_{policy} and synchronization costs (Section 3).

4.1 Advanced Schedules & Policies

Schedules control the corruption level across steps, while policies decide which positions to update and when to stop. Both determine how much parallel progress each step makes, and thus how small T can be at a fixed quality target.

Non-uniform, dilated, and jump schedules

Reducing T can be achieved by reallocating steps non-uniformly along the reverse-time trajectory, e.g., skipping/dilating portions while maintaining stable refinement [Luxembourg *et al.*, 2025; Amin *et al.*, 2025]. Related work studies faster discrete diffusion solvers or higher-order updates that approximate longer trajectories with fewer evaluations [Chen *et al.*, 2024; Ren *et al.*, 2025]. For meaningful comparisons, the mapping from "nominal steps" to actual evaluation count (including multi-stage updates) should be explicit, since it affects both T and N_{fwd}.

Confidence/entropy-driven unmasking and early exit

Adaptive policies use uncertainty signals (probability, entropy, margin) to choose update sets $\mathcal{U}_t$ and concentrate computation on difficult positions [Mohamed *et al.*, 2025; Ben-Hamu *et al.*, 2025]. Training-free variable-length denoising or early stopping further reduces the *average* step count by terminating when confidence criteria are met [Li *et al.*, 2025c]. These methods can introduce nontrivial C_{policy} (scoring/sorting/thresholding) and implementation-sensitive overheads; thus, reporting should include N_{fwd} and step-count distributions (e.g., percentiles), not only mean T.

Learned unmasking policies
Learned policies predict which positions to update (or when to stop) via supervised learning or policy-gradient/RL objectives [Jazbec *et al.*, 2025; Wang *et al.*, 2025a]. They can improve the quality–latency frontier, but should be reported with (i) the policy feature set, (ii) any extra model calls, and (iii) an ablation isolating policy overhead from gains due to fewer evaluations.

4.2 Speculative & Parallel Decoding

Decoding-algorithm changes can reduce the *effective* expensive computation per sequence by enabling larger jumps in sequence space or amortizing strong-model calls via verification. In Eq. 1, these methods target $T \downarrow$ and/or reduce expensive evaluations, but the net gain depends on acceptance and orchestration cost.

Draft-and-verify (speculative) diffusion decoding
Speculative diffusion decoding produces candidate updates with a cheaper mechanism and verifies/corrects them with a stronger denoiser [Christopher *et al.*, 2025; Li *et al.*, 2025a; Gao *et al.*, 2025]. Speedups depend on acceptance rate and the relative cost of draft vs verification, so compute accounting should separate draft and verify evaluations and report acceptance statistics alongside end-to-end latency.

Block / set decoding and structured parallel updates
Block/set decoding updates structured groups of tokens per iteration, effectively interpolating between AR and fully parallel refinement [Gat *et al.*, 2025; Arriola *et al.*, 2025a]. These methods can reduce iterations at matched quality, but may increase per-step decision complexity; evaluation should clarify how the structure changes Δ_t and whether full-length L_{in} reprocessing is required each step.

Hybrid diffusion–autoregressive decoding
Hybrid decoders combine diffusion refinement with autoregressive components, e.g., diffusion for global proposals and AR for final emission [Li *et al.*, 2025b; Liu *et al.*, 2025]. Because different modules may dominate runtime under different regimes, results should state which module dominates compute and how L_{in} differs across modules.

4.3 Distillation & Consistency

Distillation-based methods reduce inference cost by shifting work to training, aiming to shrink T toward a small constant without increasing G_t.

Step / sampler distillation
Progressive and learnable sampler distillation compress multi-step sampling into fewer steps [Salimans and Ho, 2022; Fu *et al.*, 2025; Hayakawa *et al.*, 2025]. The main benefit is direct $T \downarrow$, while common caveats include sensitivity in calibration/controllability as trajectories become shorter.

Consistency-style objectives
Consistency objectives enable few-step (or direct) mapping from noisy templates to clean samples [Song *et al.*, 2023; Kim *et al.*, 2025]. They target the dominant $T \cdot C_{\text{fwd}}$ term with minimal inference-time logic, but should be evaluated across step budgets to show graceful quality degradation as T shrinks.

Self-distillation through time and unrolled generation
Self-distillation across steps and unrolled training encourage faster convergence in fewer iterations [Savinov *et al.*, 2022; Deschenaux and Gulcehre, 2025]. From an efficiency lens, the key is robustness across workloads and decoding configurations, not only the smallest achievable T.

5 Architectural & Systems Efficiency: Reducing C_{fwd} and C_{sys}

Even with the same refinement iterations T and evaluation count N_{fwd}, end-to-end latency in Eq. 1 can vary substantially across implementations and deployment regimes. This section focuses on lowering the per-evaluation cost $C_{\text{fwd}}(L_{\text{in}}, B)$ and the request-level overhead $C_{\text{sys}}(L_{\text{in}}, B)$. A key difference from autoregressive serving is that masked diffusion decoding repeatedly evaluates a (typically) full-sequence Transformer under gradually changing token patterns, which creates both opportunities (reuse across similar steps) and challenges (cache invalidation, orchestration overhead, memory pressure).

5.1 Dynamic Compute & Sparsity

Architectural and numerical optimizations primarily target C_{fwd} (and sometimes peak memory), aiming to make each refinement evaluation cheaper. Unlike Section 4, these methods do not necessarily reduce T, but can yield large gains when N_{fwd} remains nontrivial.

Sparse / structured computation for diffusion decoding
Because diffusion decoding revisits similar intermediate states across steps, several works investigate sparsity patterns tailored to diffusion LMs, e.g., sparse attention mechanisms to reduce attention cost and memory footprint [Wang *et al.*, 2025c]. The practical benefit depends strongly on kernel maturity and how sparsity interacts with batching and sequence length; thus, wall-clock latency (not only FLOPs) should be reported under the intended runtime stack.

Lightweight denoisers and decoding architectures
Another route is to reduce C_{fwd} via architectural choices that maintain refinement behavior with lower per-step cost, such as convolutional decoding or alternative decoder structures for diffusion language modeling [Seo *et al.*, 2025; Arriola *et al.*, 2025b]. In practice, such designs should be evaluated together with the decoding configuration (steps/policy), since architectural changes can shift where the bottleneck lies (compute vs memory bandwidth vs orchestration).

Numerics and quantization
Quantization and reduced-precision execution can lower both compute and memory cost per evaluation, but require calibration that matches the diffusion decoding distribution across timesteps and masking patterns [Xu and Yang, 2025; Zhang *et al.*, 2025b]. For efficiency claims, it is important to report the precision mode, any accuracy recovery tricks, and whether the timed measurement includes dequantization/casting overheads inside the forward path.

5.2 Diffusion-Aware Caching (KV & Activation)

Caching and reuse aim to exploit redundancy across refinement steps, reducing effective C_{fwd} and sometimes C_{sys} at the cost of higher memory. Unlike autoregressive decoding, diffusion decoding revises token values across the sequence; thus, naive KV caching can become stale and requires diffusion-specific refresh and eviction strategies.

KV caching across refinement steps

A growing line of work adapts KV caching to diffusion LMs by selectively reusing attention states across steps while accounting for token revisions [Ma *et al.*, 2025a; Wu *et al.*, 2025; Hu *et al.*, 2025; Nguyen-Tri *et al.*, 2025]. These approaches can reduce repeated attention computation when consecutive states are similar, but they introduce a clear memory–speed trade-off and can shift the bottleneck to memory bandwidth and cache management.

Selective refresh and cache eviction

Because only a subset of token values often changes per step (Section 2), caching can be made more effective by refreshing only where updates occur and evicting cache entries predicted to be unhelpful [Huang *et al.*, 2025a; Song *et al.*, 2025; Jiang *et al.*, 2025; Bu *et al.*, 2025]. A key evaluation point is whether caching remains beneficial under realistic serving regimes (microbatching, long-context prompts, and concurrent requests), where memory pressure and fragmentation can dominate.

Reporting implications

For diffusion-aware caching, a minimal report should include: (i) peak GPU memory with and without caching, (ii) wall-clock latency and throughput under matched L_{in} and concurrency, and (iii) quality at the exact caching configuration. Because caching can also change C_{sys} (extra bookkeeping, synchronization, or memory movement), timing should follow an end-to-end protocol (Section 3) rather than isolated kernel timings.

5.3 System-Level Optimization

System-level optimizations target $C_{\text{sys}}(L_{\text{in}}, B)$ by reducing orchestration overheads and improving utilization across the repeated refinement loop. These effects are often most visible for batch-1 latency and interactive settings, where dispatch and synchronization overheads can become comparable to compute.

Diffusion-aware inference frameworks

Dedicated inference frameworks for diffusion LMs aim to streamline the refinement loop, manage caching, and reduce per-step overheads that do not scale with model FLOPs [Ma *et al.*, 2025b]. From an efficiency perspective, the systems should be evaluated across batch sizes and concurrency levels to show when C_{sys} dominates versus when compute dominates.

Production serving and memory dynamics

Production-oriented work highlights that diffusion decoding can create distinct memory and orchestration challenges due to repeated full-sequence evaluations, cache growth, and step-wise control flow [Fan *et al.*, 2025]. Accordingly, system claims should report not only average latency but also tail latency (p95) and memory headroom under realistic request mixes.

6 Inference-Time Scaling: The G_t Factor

Many recent improvements in diffusion-based language generation come from *inference-time scaling*: deliberately spending more computation at test time to improve quality, controllability, or constraint satisfaction. In Eq. 1, these methods typically increase the per-step evaluation multiplier G_t (hence N_{fwd}), and often also increase C_{policy} and memory footprint due to maintaining additional trajectories, scores, or caches. As a result, step counts alone can be especially misleading in this regime: two decoders with the same T can differ substantially in wall-clock latency and peak memory depending on the scaling strategy.

6.1 Guidance as Multi-Pass Decoding

Guidance mechanisms improve generation quality or enforce preferences by altering the reverse-time update using additional signals, often requiring multiple model evaluations per refinement step. In masked/discrete diffusion, simple guidance variants can be implemented in a way that resembles classifier-free guidance (or related conditioning tricks), which naturally introduces a multi-pass structure [Schiff *et al.*, 2025]. Recent work also explores adaptive guidance rules that modulate the guidance strength or update pattern based on uncertainty, which can improve the quality–compute trade-off but makes G_t and C_{policy} configuration-dependent [Li *et al.*, 2025d; Ye *et al.*, 2025a]. Remasking-based scaling can be viewed similarly: additional refinement passes (or repeated mask-and-refine cycles) increase effective compute beyond the nominal step count [Wang *et al.*, 2025b].

From an efficiency standpoint, guidance should be reported by explicitly stating: (i) how many forwards are executed per step (the resulting G_t schedule), (ii) whether evaluations are performed on the full length L_{in}, and (iii) the exact operating point (guidance strength / masking thresholds) at which quality is measured. Without this, comparisons that only match T can significantly under- or over-estimate runtime.

6.2 Search and Trajectory-Level Scaling

A second scaling family treats diffusion decoding as an implicit search problem over refinement trajectories. Instead of committing to a single reverse chain, these methods explore multiple candidates and select or resample trajectories based on scores, rewards, or constraints. This often yields strong gains in alignment or constraint satisfaction, but increases compute roughly with the breadth/particles/expansions used, directly inflating N_{fwd}.

Particle / SMC-style scaling

Particle-based methods (e.g., Particle Gibbs sampling) scale computation by maintaining and resampling a set of candidate trajectories, using the additional compute to better explore the posterior over discrete sequences [Dang *et al.*, 2025]. In Eq. 1, the dominant effect is $G_t \uparrow$ (multiple evaluations per step and/or per particle), with additional policy overhead from resampling and scoring.

Tree search and constrained inference

Tree-search formulations explicitly branch on promising refinements and evaluate candidates with search policies, e.g., MCTS layered on top of diffusion decoding [Huang *et al.*, 2025b]. Related work studies constrained decoding objectives for discrete diffusion, where the scaling budget is spent on ensuring constraints or optimizing utility under constraints [Cardei *et al.*, 2025; Suresh *et al.*, 2025]. Diffusion tree sampling provides another scalable framework for inference-time alignment by expanding candidate refinement paths [Jain *et al.*, 2025]. These approaches can be effective when constraints are hard to satisfy with a single-pass decoder, but their efficiency is highly regime-dependent: branching increases memory pressure (multiple partial states and caches) and can interact strongly with serving-level overheads (Section 5).

7 Practitioner's Guide

This section translates the surveyed techniques into a practical workflow grounded in Eq. (1). We do *not* introduce a new benchmark nor report new measurements; instead, we summarize diffusion-specific knobs that determine inference-time compute and explain how to interpret efficiency claims beyond nominal step counts.

Core principle. In diffusion-based decoding, the refinement budget T alone is not a reliable proxy for speed. End-to-end latency depends on the total number of model evaluations $N_{\text{fwd}} = \sum_{t=1}^{T} G_t$, the per-evaluation cost, and overhead from schedule/policy logic and iterative control flow (Eq. (1)). Accordingly, when comparing methods, it is essential to make the effective compute budget explicit (via T and G_t) and to compare at matched quality operating points.

7.1 A Workflow Grounded in Eq. (1)

We recommend the following workflow when selecting and evaluating acceleration methods.

Step 1: Fix the workload and objective. Specify prompt length and generated length (or their distributions), and define the processed length per model evaluation L_{in}. Clarify the objective: latency-sensitive generation, throughput-oriented serving, or quality/constraint-critical decoding.

Step 2: Make the compute budget explicit in diffusion terms. Report the refinement budget T together with the per-step evaluation multiplier G_t (hence $N_{\text{fwd}} = \sum_t G_t$). If the method introduces multi-pass evaluation (e.g., guidance, verification, particles, or branching), state how G_t changes and what the effective evaluation count becomes.

Step 3: Choose levers that target the dominant term(s). Use Eq. (1) as a diagnostic lens: methods in Section 4 primarily reduce iterations (T) and/or evaluations (N_{fwd}); methods in Section 5 reduce per-evaluation cost via numerics, structured compute, and reuse/caching across steps; methods in Section 6 often increase G_t (thus N_{fwd}) to improve quality or satisfy constraints. In practice, the most meaningful comparison is to isolate which term(s) changed and what trade-offs were introduced.

Step 4: Compare on a small matched-quality frontier. When a method exposes a compute–quality knob (steps, early-exit thresholds, guidance strength, number of particles/branches), reporting a few operating points makes the trade-off explicit and avoids comparisons at mismatched operating points.

Quick reference. Table 2 provides a compact map from common goals (latency, throughput, and quality-critical decoding) to the corresponding levers in Eq. (1) and the survey sections where representative methods are discussed.

Bottleneck / Goal	Recommended Approach	Sec.
Latency-Sensitive	**Minimize T & C_{sys}**	
(Batch ≈ 1, Chat)	· Advanced Schedules	§4.1
	· System/Kernel Opt.	§5.3
Throughput-Bound	**Minimize C_{fwd} (Mem trade-off)**	
(Serving, Offline)	· Diffusion-aware Cache	§5.2
	· Sparsity / Quantization	§5.1
Quality-Critical	**Scale G_t (Multi-pass)**	
(Reasoning, Math)	· Guidance / Re-ranking	§6
	· Verify w/ cheap Draft	§4.2

Table 2: Decision matrix mapping deployment scenarios to primary efficiency factors in Eq. 1. We identify the dominant bottleneck for each case, including latency, throughput, and quality, and recommend corresponding acceleration families discussed in this survey.

7.2 Diffusion-Specific Notes for Interpreting Efficiency Claims

Most ambiguity in efficiency comparisons for diffusion decoding comes from under-specifying the refinement trajectory and the number of model evaluations. The following diffusion-specific items are often sufficient to make comparisons interpretable without prescribing a particular measurement protocol:

- **Trajectory and stopping:** T, the schedule type, and any early-exit criteria.
- **Evaluation accounting:** G_t (or its description) and $N_{\text{fwd}} = \sum_t G_t$, especially when guidance/search/verification is used.
- **Speculative or hybrid pipelines:** evaluation counts *per module* (draft vs verify; diffusion vs AR) and acceptance/rejection statistics when applicable.
- **Update sparsity (recommended):** a compact statistic such as the average fraction of changed positions $|\Delta_t|/L$, since many reuse/caching mechanisms implicitly rely on sparsity across refinement steps (Section 2).

7.3 Common Pitfalls for Interpreting Speedups

Finally, we highlight recurring sources of ambiguity that can make "speedup" claims hard to interpret:

- **T reported without G_t:** identical step counts can correspond to very different compute budgets when multi-pass decoding is used.

- **Multi-pass logic hidden in a single iteration:** methods that pack multiple evaluations into one nominal step should make the effective N_{fwd} explicit.
- **Comparisons at mismatched operating points:** when a method trades compute for quality (or vice versa), reporting a small frontier is often more informative than a single tuned point.

Taken together, making $(T,\, G_t,\, N_{\text{fwd}})$ explicit and tying claimed gains back to Eq. (1) is typically sufficient to keep diffusion inference efficiency comparisons robust and interpretable.

8 Conclusion & Open Challenges

Diffusion-based large language models (dLLMs) offer a fundamentally different inference interface from autoregressive decoding: generation proceeds by iterative refinement with parallel updates over multiple token positions. This creates a genuine opportunity for faster-than-AR decoding, but practical speedups require *inference mechanisms* that turn parallel updates into fewer effective model evaluations, cheaper evaluations, and lower iteration overhead.

This survey focused on that inference-efficiency question. We organized the acceleration landscape into three complementary axes. First, *algorithmic* techniques modify the refinement trajectory—schedules, update policies, decoding algorithms, and distillation—to reduce the iteration budget T and/or the effective evaluation count N_{fwd} (Section 4). Second, *architectural and systems* techniques reduce the per-evaluation cost and amortize redundant computation across steps via numerics, structured compute, and diffusion-aware reuse (Section 5). Third, *inference-time scaling* methods deliberately increase compute (often via multi-pass evaluation) to improve quality or satisfy constraints, making the compute–quality trade-off explicit (Section 6).

A unifying lesson across these threads is that efficiency gains in dLLM inference are rarely attributable to a single knob. Step reduction is most effective when it does not silently increase multi-pass computation (G_t), policy overhead, or sensitivity to decoding hyperparameters. Reuse and caching can yield substantial wall-clock gains when consecutive states are similar, but they introduce explicit memory trade-offs and require diffusion-specific refresh/eviction logic. Finally, scaling strategies can be powerful in quality-critical settings, but their benefits are meaningful only when the added evaluation budget is accounted for in the same currency as the base decoder (Eq. (1)). To keep these trade-offs interpretable, we distilled a lightweight practitioner workflow and diffusion-specific disclosure items in Section 7.

Open challenges. We highlight several directions that, in our view, will most strongly shape the next phase of progress in accelerating diffusion-based language inference:

- **Few-step decoding without brittle quality or control.** Distillation and consistency-style approaches can shrink T dramatically, but robustness across prompts, lengths, and controllability settings remains uneven. An open problem is to achieve few-step generation that degrades gracefully with step budgets and remains well-calibrated under diverse decoding policies (Section 4.3).
- **Predictable acceleration from adaptive schedules and policies.** Adaptive unmasking and early-exit policies can reduce average compute, yet their benefits can be eroded by policy overhead or instability across workloads. Designing low-overhead policies with predictable speed–quality behavior (and clear failure modes) is central to making trajectory-level acceleration reliable (Section 4.1).
- **Diffusion-aware reuse with correctness guarantees under token revisions.** Reuse mechanisms must remain valid when tokens are revised across steps. Better criteria for cache validity, selective refresh, and error control—especially under long contexts and memory constraints—are needed to make reuse both fast and dependable (Section 5.2).
- **Co-design of parallel updates and kernel-friendly execution.** Many decoders exhibit update sparsity ($|\Delta_t| \ll L$), yet most implementations still execute near full-sequence computation. Bridging diffusion-specific structure to hardware- and compiler-friendly execution (e.g., structured sparsity and efficient partial updates) is key to converting theoretical parallelism into consistent wall-clock gains (Section 5.1).
- **Compute-adaptive scaling for reasoning and constraints.** Guidance and search-based methods can improve quality, but they increase G_t and can interact strongly with the base trajectory. A promising direction is compute-adaptive scaling that allocates additional evaluations only when needed, while keeping the compute budget and its effect on quality transparent (Section 6).

Limitations of this survey. This survey focuses on *masked/discrete* diffusion language models, where the refinement loop involves explicit token-level updates. Continuous-embedding approaches (e.g., Diffusion-LM [Li *et al.*, 2022] and its variants) operate in a fundamentally different latent space and involve distinct efficiency trade-offs that warrant separate treatment.

Although this survey focuses on language generation, diffusion-style iterative refinement is also relevant to structured sequence domains such as time-series imputation and forecasting [Alcaraz and Strodthoff, 2023]. Recent work further applies large language diffusion models to time-series forecasting [Pei *et al.*, 2025].

Also, the field is evolving rapidly: many techniques surveyed here were published or released within the past year, and standardized benchmarks for dLLM inference efficiency remain lacking. As a result, direct comparisons across papers are often confounded by differences in model scale, evaluation protocol, and hardware configuration. While we have attempted to organize the literature through a unified framework (Eq. 1), we caution that reported speedups should be interpreted with attention to the specific experimental setup.

Finally, we restrict our attention to *inference-time* efficiency. Training efficiency (*e.g.,* data efficiency and adaptation from pretrained AR models) is an important but orthogonal concern that we do not address in depth.

Acknowledgments

This work was supported by the Institute for Information & communications Technology Planning & Evaluation (IITP) grants funded by the Korean government (MSIT) (RS-2019-II190075, Artificial Intelligence Graduate School Program (KAIST); RS-2025-02304967, AI Star Fellowship (KAIST); and RS-2024-00396828, Development of AI-based Low-Power 5G-A O-DU/O-CU; contribution rate: 33.3%).

References

[Alcaraz and Strodthoff, 2023] Juan Miguel Lopez Alcaraz and Nils Strodthoff. Diffusion-based time series imputation and forecasting with structured state space models. *TMLR*, 2023.

[Amin *et al.*, 2025] Alan N. Amin, Nate Gruver, and Andrew Gordon Wilson. Why Masking Diffusion Works: Condition on the Jump Schedule for Improved Discrete Diffusion. In *NeurIPS*, 2025.

[Arriola *et al.*, 2025a] Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, et al. Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models. In *ICLR*, 2025.

[Arriola *et al.*, 2025b] Marianne Arriola, Yair Schiff, Hao Phung, Aaron Gokaslan, and Volodymyr Kuleshov. Encoder-Decoder Diffusion Language Models for Efficient Training and Inference. In *NeurIPS*, 2025.

[Austin *et al.*, 2021] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured Denoising Diffusion Models in Discrete State-Spaces. In *NeurIPS*, 2021.

[Ben-Hamu *et al.*, 2025] Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated Sampling from Masked Diffusion Models via Entropy Bounded Unmasking. In *NeurIPS*, 2025.

[Bu *et al.*, 2025] Jiazi Bu, Pengyang Ling, Yujie Zhou, et al. DiCache: Let Diffusion Model Determine Its Own Cache, 2025. arXiv.

[Cardei *et al.*, 2025] Michael Cardei, Jacob K. Christopher, Thomas Hartvigsen, Kailkhura, et al. Constrained Discrete Diffusion. In *NeurIPS*, 2025.

[Chen *et al.*, 2024] Zixiang Chen, Huizhuo Yuan, Yongqian Li, Yiwen Kou, Junkai Zhang, and Quanquan Gu. Fast Sampling via Discrete Non-Markov Diffusion Models with Predetermined Transition Time. In *NeurIPS*, 2024.

[Christopher *et al.*, 2025] Jacob K. Christopher, Brian R. Bartoldson, Tal Ben-Nun, et al. Speculative Diffusion Decoding: Accelerating Language Generation through Diffusion. In *NAACL*, 2025.

[Dang *et al.*, 2025] Meihua Dang, Jiaqi Han, Minkai Xu, Kai Xu, Akash Srivastava, and Stefano Ermon. Inference-Time Scaling of Diffusion Language Models with Particle Gibbs Sampling, 2025. arXiv.

[Deschenaux and Gulcehre, 2025] Justin Deschenaux and Caglar Gulcehre. Beyond Autoregression: Fast LLMs via Self-Distillation Through Time. In *ICLR*, 2025.

[Fan *et al.*, 2025] Jiakun Fan, Yanglin Zhang, Xiangchen Li, and Dimitrios S. Nikolopoulos. Taming the Memory Footprint Crisis: System Design for Production Diffusion LLM Serving, 2025. arXiv.

[Fu *et al.*, 2025] Feiyang Fu, Tongxian Guo, and Zhaoqiang Liu. Learnable Sampler Distillation for Discrete Diffusion Models. In *NeurIPS*, 2025.

[Gao *et al.*, 2025] Yifeng Gao, Ziang Ji, Yuxuan Wang, Biqing Qi, Hanlin Xu, and Linfeng Zhang. Self Speculative Decoding for Diffusion Large Language Models, 2025. arXiv.

[Gat *et al.*, 2025] Itai Gat, Heli Ben-Hamu, Marton Havasi, Daniel Haziza, Jeremy Reizenstein, Gabriel Synnaeve, David Lopez-Paz, Brian Karrer, and Yaron Lipman. Set Block Decoding is a Language Model Inference Accelerator, 2025. arXiv.

[Hayakawa *et al.*, 2025] Satoshi Hayakawa, Yuhta Takida, Masaaki Imaizumi, Hiromi Wakaki, and Yuki Mitsufuji. Distillation of Discrete Diffusion through Dimensional Correlations. In *ICML*, 2025.

[Hu *et al.*, 2025] Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed S. Abdelfattah, Jae-sun Seo, Zhiru Zhang, and Udit Gupta. FlashDLM: Accelerating Diffusion Language Model Inference via Efficient KV Caching and Guided Diffusion, 2025. arXiv.

[Huang *et al.*, 2025a] Jianuo Huang, Yaojie Zhang, Yicun Yang, Benhao Huang, Biqing Qi, Dongrui Liu, and Linfeng Zhang. Mask Tokens as Prophet: Fine-Grained Cache Eviction for Efficient dLLM Inference, 2025. arXiv.

[Huang *et al.*, 2025b] Zheng Huang, Kiran Ramnath, et al. Diffusion Language Model Inference with Monte Carlo Tree Search, 2025. arXiv.

[Inception Labs *et al.*, 2025] Inception Labs, Samar Khanna, Siddhant Kharbanda, et al. Mercury: Ultra-Fast Language Models Based on Diffusion, 2025. arXiv.

[Jain *et al.*, 2025] Vineet Jain, Kusha Sareen, Mohammad Pedramfar, and Siamak Ravanbakhsh. Diffusion Tree Sampling: Scalable inference-time alignment of diffusion models. In *NeurIPS*, 2025.

[Jazbec *et al.*, 2025] Metod Jazbec, Theo X. Olausson, et al. Learning Unmasking Policies for Diffusion Language Models, 2025. arXiv.

[Jiang *et al.*, 2025] Yuchu Jiang, Yue Cai, Xiangzhong Luo, Jiale Fu, Jiarui Wang, Chonghan Liu, and Xu Yang. d ^2 Cache: Accelerating Diffusion-Based LLMs via Dual Adaptive Caching, 2025. arXiv.

[Kim *et al.*, 2025] Minseo Kim, Chenfeng Xu, Coleman Hooper, Harman Singh, Ben Athiwaratkun, Ce Zhang, Kurt Keutzer, and Amir Gholami. CDLM: Consistency Diffusion Language Models For Faster Sampling, 2025. arXiv.

[Li *et al.*, 2022] Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-LM Improves Controllable Text Generation. In *NeurIPS*, 2022.

[Li *et al.*, 2025a] Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. DiffuSpec: Unlocking Diffusion Language Models for Speculative Decoding, 2025. arXiv.

[Li *et al.*, 2025b] Jia-Nan Li, Jian Guan, Wei Wu, and Chongxuan Li. ReFusion: A Diffusion Large Language Model with Parallel Autoregressive Decoding, 2025. arXiv.

[Li *et al.*, 2025c] Jinsong Li, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Jiaqi Wang, and Dahua Lin. Beyond Fixed: Training-Free Variable-Length Denoising for Diffusion Large Language Models, 2025. arXiv.

[Li *et al.*, 2025d] Pengxiang Li, Shilin Yan, et al. Adaptive Classifier-Free Guidance via Dynamic Low-Confidence Masking. In *NeurIPS*, 2025.

[Li *et al.*, 2025e] Tianyi Li, Mingda Chen, Bowei Guo, and Zhiqiang Shen. A Survey on Diffusion Language Models, 2025. arXiv.

[Liu *et al.*, 2025] 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, 2025. arXiv.

[Lou *et al.*, 2024] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete Diffusion Modeling by Estimating the Ratios of the Data Distribution. In *ICML*, 2024.

[Luxembourg *et al.*, 2025] Omer Luxembourg, Haim Permuter, et al. Plan for Speed: Dilated Scheduling for Masked Diffusion Language Models, 2025. arXiv.

[Ma *et al.*, 2025a] Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dKV-Cache: The Cache for Diffusion Language Models. In *NeurIPS*, 2025.

[Ma *et al.*, 2025b] Yuxin Ma, Lun Du, Lanning Wei, et al. dInfer: An Efficient Inference Framework for Diffusion Language Models, 2025. arXiv.

[Mohamed *et al.*, 2025] Amr Mohamed, Yang Zhang, Michalis Vazirgiannis, and Guokan Shang. Fast-Decoding Diffusion Language Models via Progress-Aware Confidence Schedules, 2025. arXiv.

[Nguyen-Tri *et al.*, 2025] Quan Nguyen-Tri, Mukul Ranjan, and Zhiqiang Shen. Attention Is All You Need for KV Cache in Diffusion LLMs, 2025. arXiv.

[Nie *et al.*, 2025] Shen Nie, Fengqi Zhu, Zebin You, et al. Large Language Diffusion Models. In *NeurIPS*, 2025.

[Pei *et al.*, 2025] Yuhang Pei, Tao Ren, Yifan Wang, Zhipeng Sun, Wei Ju, Chong Chen, Xian-Sheng Hua, and Xiao Luo. LEAF: Large language diffusion model for time series forecasting. In *EMNLP*, 2025.

[Ren *et al.*, 2025] Yinuo Ren, Haoxuan Chen, et al. Fast Solvers for Discrete Diffusion Models: Theory and Applications of High-Order Algorithms. In *NeurIPS*, 2025.

[Sahoo *et al.*, 2024] Subham Sekhar Sahoo, Marianne Arriola, et al. Simple and Effective Masked Diffusion Language Models. In *NeurIPS*, 2024.

[Salimans and Ho, 2022] Tim Salimans and Jonathan Ho. Progressive Distillation for Fast Sampling of Diffusion Models. In *ICLR*, 2022.

[Savinov *et al.*, 2022] Nikolay Savinov, Junyoung Chung, Mikolaj Binkowski, Erich Elsen, and Aaron van den Oord. Step-unrolled Denoising Autoencoders for Text Generation. In *ICLR*, 2022.

[Schiff *et al.*, 2025] Yair Schiff, Subham Sekhar Sahoo, et al. Simple Guidance Mechanisms for Discrete Diffusion Models. In *ICLR*, 2025.

[Seo *et al.*, 2025] Yeongbin Seo, Dongha Lee, Jaehyung Kim, and Jinyoung Yeo. Fast and Fluent Diffusion Language Models via Convolutional Decoding and Rejective Fine-tuning. In *NeurIPS*, 2025.

[Song *et al.*, 2023] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency Models. In *ICML*, 2023.

[Song *et al.*, 2025] Yuerong Song, Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. Sparse-dLLM: Accelerating Diffusion LLMs with Dynamic Cache Eviction, 2025. arXiv.

[Suresh *et al.*, 2025] Tarun Suresh, Debangshu Banerjee, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. DINGO: Constrained Inference for Diffusion LLMs. In *NeurIPS*, 2025.

[Wang *et al.*, 2025a] Chenyu Wang, Paria Rashidinejad, DiJia Su, et al. SPG: Sandwiched Policy Gradient for Masked Diffusion Language Models, 2025. arXiv:2510.09541.

[Wang *et al.*, 2025b] Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking Discrete Diffusion Models with Inference-Time Scaling. In *NeurIPS*, 2025.

[Wang *et al.*, 2025c] Zeqing Wang, Gongfan Fang, Xinyin Ma, Xingyi Yang, and Xinchao Wang. SparseD: Sparse Attention for Diffusion Language Models, 2025. arXiv.

[Wu *et al.*, 2025] Chengyue Wu, Hao Zhang, et al. Fast-dLLM: Training-free Acceleration of Diffusion LLM by Enabling KV Cache and Parallel Decoding, 2025. arXiv.

[Xu and Yang, 2025] Chen Xu and Dawei Yang. DLLMQuant: Quantizing Diffusion-based Large Language Models, 2025. arXiv.

[Ye *et al.*, 2025a] He Ye, Rojas Kevin, and Tao Molei. What Exactly Does Guidance Do in Masked Discrete Diffusion Models, 2025. arXiv:2506.10971.

[Ye *et al.*, 2025b] Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion Large Language Models, 2025. arXiv.

[Zhang *et al.*, 2025a] Lingzhe Zhang, Liancheng Fang, et al. A Survey on Parallel Text Generation: From Parallel Decoding to Diffusion Language Models, 2025. arXiv.

[Zhang *et al.*, 2025b] Tianao Zhang, Zhiteng Li, Xianglong Yan, Haotong Qin, Yong Guo, and Yulun Zhang. QuantdLLM: Post-Training Extreme Low-Bit Quantization for Diffusion Large Language Models, 2025. arXiv.