Article

Not peer-reviewed version

A Comparative Survey of Inference Acceleration for DLLMs against AR-LLMs: No Free Lunch

Haoyun Jiang , Junqi He , Muyi Wang , Fanqin Zeng , Feng Hong , Geng Yu , Pengyi Chen , Yushi Ye , Yuting Cao , Yicheng Fu , Ziyi Tang , Haolin Li , Yuchen Xiong , Zhiyong Chen , Xiaofeng Cao , Xiangtao Li , Bo Han , Ya Zhang , Yanfeng Wang , Jiangchao Yao *

Posted Date: 12 May 2026

doi: 10.20944/preprints202605.0776.v1

Keywords: diffusion LLMs; autoregressive LLMs; inference acceleration; parallel decoding; comparative analysis

 doi:10.20944/preprints202605.0776.v1

Article

A Comparative Survey of Inference Acceleration for DLLMs Against AR-LLMs: No Free Lunch

Haoyun Jiang [1], Junqi He [1], Muyi Wang [1], Fanqin Zeng [1], Feng Hong [1], Geng Yu [1], Pengyi Chen [1], Yushi Ye [1], Yuting Cao [1], Yicheng Fu [1], Ziyi Tang [1], Haolin Li [2], Yuchen Xiong [1], Zhiyong Chen [1], Xiaofeng Cao [3], Xiangtao Li [4], Bo Han [5,6], Ya Zhang [1], Yanfeng Wang [1] and Jiangchao Yao [1,*]

[1] Shanghai Jiao Tong University, China
[2] Fudan University, China
[3] Tongji University, China
[4] Jilin University, China
[5] Hong Kong Baptist University, Hong Kong, China
[6] RIKEN, Japan
* Correspondence: sunarker@sjtu.edu.cn

Abstract

Autoregressive large language models (AR-LLMs) have achieved remarkable success, but their inherently sequential decoding process remains a fundamental bottleneck for efficient inference. Diffusion large language models (DLLMs), with bidirectional modeling and parallel token generation, offer a promising alternative to break this token-by-token limitation. Yet despite rapid progress, the practical inference efficiency of current DLLMs remains unclear. From a verification perspective, this survey establishes a systematic taxonomy of existing acceleration methods, benchmarks representative techniques under a unified experimental setting, and further evaluates strong strategy combinations to quantify the gap between mainstream DLLM inference methods and state-of-the-art AR baselines. Specially, the overall analysis highlights that the parallel decoding efficiency of DLLMs still remains a significant lag compared to the decoding efficiency of AR-LLMs under inference acceleration. We provide an in-depth experimental analysis about the underlying trade-offs among generation quality, latency, and system compatibility, and build up a standard evaluation bench open to the community. Remaining bottlenecks are also summarized, together with future directions for more practical and competitive DLLM inference. Code is available at https://github.com/haoyun-jiang/DLLM-AccelEval.

Keywords: diffusion LLMs; autoregressive LLMs; inference acceleration; parallel decoding; comparative analysis

1. Introduction

The emergence and rapid advancement of Large Language Models (LLMs) mark a new peak in the development of deep learning [1]. Contemporary LLMs, such as GPT, Claude, Gemini, and Qwen [2–5], have evolved into versatile general-purpose foundation models, exhibiting strong capabilities across a wide spectrum of tasks, including writing, programming, and domain-specific reasoning [6,7]. As these models are increasingly deployed in real-world applications, inference efficiency has become a central concern for practical and cost-effective usage [8].

Mainstream LLMs predominantly rely on the autoregressive (AR) decoding paradigm [9]. Specially, each newly generated token is conditioned on the entire preceding context, enabling thorough token-level interaction that supports linguistic coherence and logical consistency. However, this token-by-token generation dependency also incurs substantial computational overhead and limits decoding flexibility [10]. As the demand in real-world applications continues to grow, these characteristics increasingly hinder hardware utilization efficiency and pose significant challenges for economical deployment.

 doi:10.20944/preprints202605.0776.v1

Extensive research has sought to optimize the inference pipeline of AR-LLMs and alleviate the efficiency bottlenecks. Representative approaches include quantization [11,12], pruning [13,14], efficient memory management [15], and speculative decoding [16,17], which improve inference efficiency from different perspectives. However, these methods still operate within the autoregressive paradigm and therefore do not fundamentally remove the dependency of token-by-token generation. Consequently, the potential for further acceleration within the AR framework remains inherently limited.

This has motivated research along two different directions. Within the autoregressive framework, speculative decoding improves inference efficiency by introducing internal parallelism through a draft-then-verify process, while still operating under the causal generation paradigm [23]. Recent work has also explored alternative generation paradigms with stronger native parallelism. One representative design is the family of discrete diffusion large language models (DLLMs), which formulate text generation as a discrete diffusion process [24,25]. By leveraging bidirectional attention, DLLMs enable parallel and controllable token generation, offering a fundamentally different route beyond token-by-token decoding.

Despite their native parallelism, DLLMs suffer from unique challenges in practice. A central tension lies in the trade-off between generation quality and inference speed, as output quality is closely related to the number of denoising steps [19,26]. Besides, many optimizations developed for AR models, such as KV cache management and causal sparse attention, are not directly compatible with DLLMs due to their bidirectional and non-causal computation pattern [27,28]. These challenges raise a practical question: *What level of inference efficiency can current DLLM acceleration methods actually achieve, and how do they compare with well-optimized AR baselines?*

While several recent studies [52,74] claim that DLLMs can surpass AR baselines under certain acceleration settings, the supporting evidence remains incomplete, as comparisons do not always include stronger AR baselines with speculative decoding or evaluation in long-context scenarios. As a result, the efficiency frontier of current DLLM inference remains unclear. To answer this question more comprehensively, we establish a unified evaluation framework and systematically compare strong combined DLLM acceleration pipelines against optimized AR baselines. This also distinguishes our survey from existing works: as shown in Table 1, except method overview and taxonomy, we additionally provide a modular evaluation and an explicit DLLM–AR efficiency comparison. In summary, our contributions are as follows:

- We establish a systematic taxonomy of DLLM inference acceleration methods, organizing the literature into four primary research directions. Based on this taxonomy, we review representative methods in a unified manner and analyze their key ideas, strengths, and limitations, thereby providing a structured and comprehensive view of the current acceleration landscape for DLLMs.
- We quantify the efficiency gap between state-of-the-art DLLM inference and AR-LLM baselines under a unified experimental framework. Specifically, we conduct a modular evaluation of representative acceleration techniques and further assess strong combinations of compatible strategies for a rigorous comparison between DLLM and AR-LLM inference. The comparison clearly reveals a remaining gap in the DLLM inference ecosystem relative to that of AR-LLMs. We release the code to the community to support future research and fair benchmarking.
- We summarize the major remaining bottlenecks in current DLLM inference and highlight directions for future acceleration study. By integrating insights from the literature with our experimental findings, we identify key factors that continue to hinder practical DLLM deployment and provide a roadmap toward more efficient DLLM systems.

 doi:10.20944/preprints202605.0776.v1

Table 1. Comparison of our survey with related works. (✓: Fully covered, ✔: Partially covered, ✗: Not covered)

Survey	Acceleration Taxonomy	Unified Modular Evaluation	DLLM–AR Gap	Future Directions
Yu et al.[18]	✔	✗	✗	✗
Li et al.[19]	✔	✗	✗	✓
Tseng et al.[20]	✔	✗	✗	✓
Lin et al.[21]	✓	✗	✗	✓
Kyaw et al.[22]	✓	✗	✗	✓
Ours	✓	✓	✓	✓

2. Literature Review

In this section, we review DLLM inference acceleration methods through the taxonomy in Figure 1. Existing approaches can be grouped into four directions: KV management, sparse computation, decoding methods, and speculative decoding. These categories reflect different perspectives on efficient DLLM inference, including memory reuse, sparse execution, decoding strategy design, and hybrid acceleration. The following subsections review the methods in each direction and analyze their strengths, limitations, and practical trade-offs.

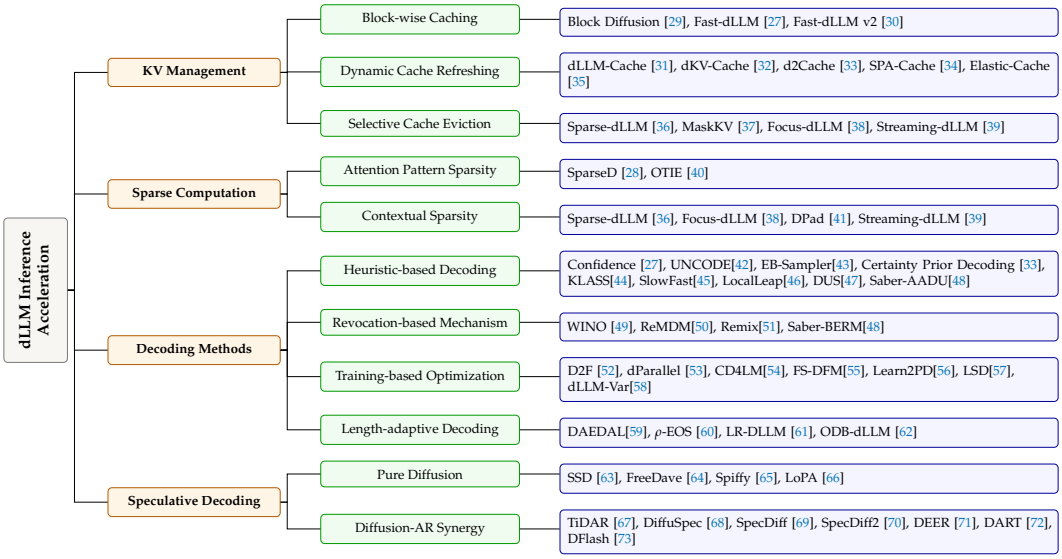

Figure 1. A taxonomy of DLLM inference acceleration and some representative acceleration methods in the corresponding directions.

2.1. KV Management

KV caching originates as an optimization for Transformer computation, where self-attention repeatedly queries an ever-growing prefix during decoding [75]. By storing previously computed keys and values, KV caching effectively avoids redundant recomputation over past tokens and turns inference into an incremental procedure [76,77] (as shown in Figure 2(a)). In DLLMs, however, iterative denoising and bidirectional attention challenge this "compute-once, reuse-forever" assumption, since token representations dynamically evolve across refinement steps. As a result, KV management in DLLMs must jointly address cache validity and cache footprint. As illustrated in Figure 2(b), existing methods can be roughly categorized into three aspects: *block-wise caching*, which partitions decoding into stable blocks for structured reuse; *dynamic cache refreshing*, which selectively recomputes stale entries; and *KV cache eviction*, which retains only high-utility entries under limited memory budgets.

 doi:10.20944/preprints202605.0776.v1

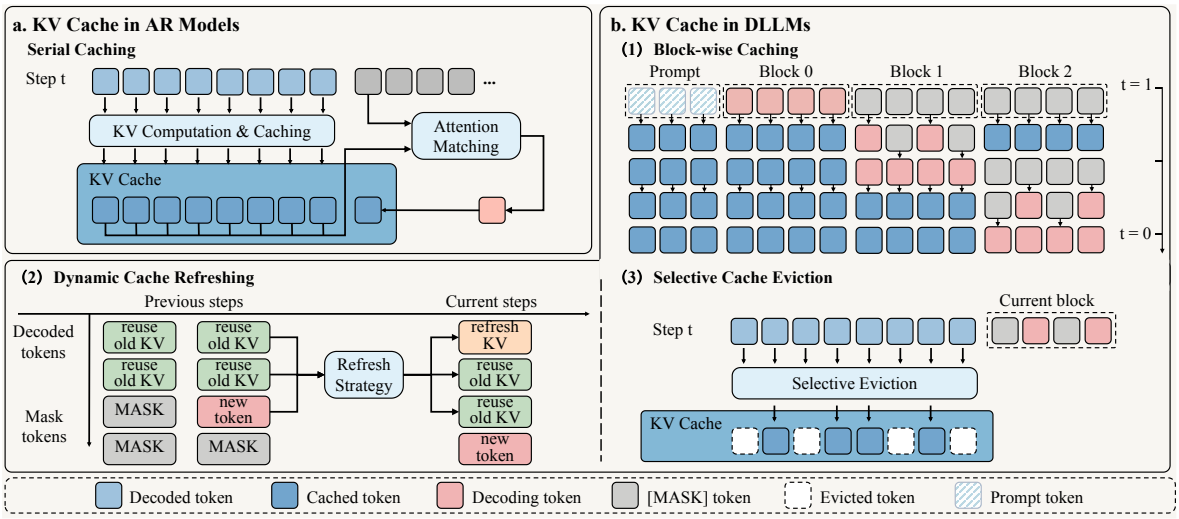

Figure 2. Comparison of KV cache mechanisms in AR models and DLLMs. **(a)** Serial caching in AR models, where newly generated tokens are cached sequentially and the KV cache grows incrementally with exact reuse of past states. **(b)** KV management in DLLMs, including (1) block-wise caching, which introduces reusable block boundaries for structured cache reuse, (2) dynamic cache refreshing, which selectively recomputes stale KV states across denoising steps, and (3) selective cache eviction, which improves memory efficiency by retaining only high-utility cache entries.

2.1.1. Block-Wise KV Cache

Block-wise caching strategies (Figure 2(b)(1)) make KV reuse feasible by treating each decoding block as a relatively stable unit, sharing cache states within the block to match parallel generation patterns. Block Diffusion [29] introduces a semi-autoregressive framework that generates sequences block by block, freezing and reusing KV states once a block is finalized. Extending this to fully bidirectional DLLMs, Fast-dLLM [27] proposes an approximate KV cache that exploits the high similarity of activations across adjacent steps, offering Prefix and Dual Cache variants for efficient context reuse. To mitigate the resulting training-inference gap, Fast-dLLM v2 [30] incorporates this lossy reuse pattern directly into training via complementary attention masks and a hierarchical cache design.

2.1.2. Dynamic Cache Refreshing

Instead of relying on rigid block boundaries, dynamic cache refreshing (Figure 2(b)(2)) selectively recomputes stale entries to maintain validity as token representations evolve. dLLM-Cache [31] adopts a region-aware strategy, applying long-interval caching for quasi-static prompts and similarity-guided, short-interval updates for dynamic response tokens. Similarly, dKV-Cache [32] leverages previous-step masking patterns to determine safe reuse, utilizing configurable intervals to balance speed and quality. Pushing towards finer granularity, d2Cache [33] performs token-level recomputation for update-critical tokens. SPA-Cache [34] further allocates refresh budgets adaptively across layers using a low-dimensional proxy, while Elastic-Cache [35] jointly optimizes the timing and location of refreshes to minimize unnecessary recomputation.

2.1.3. KV Cache Eviction

KV cache eviction focuses on capacity management by retaining only the most useful entries under constrained memory budgets (Figure 2(b)(3)). Sparse-dLLM [36] estimates utility along the *temporal* dimension: it exploits the cross-step stability of attention patterns in bidirectional diffusion to preserve pivotal tokens that remain consistently salient throughout the iterative denoising process. In contrast, MaskKV [37] targets *spatial* budget allocation. It uses a mask-query-guided scoring mechanism to assess prompt-token utility, dynamically redistributing the limited cache budget across specific transformer layers and attention heads. Meanwhile, context-selection methods such as Focus-

 doi:10.20944/preprints202605.0776.v1

dLLM [38] and Streaming-dLLM [39] move beyond discrete token scoring toward *regional* bounding, reducing global storage by isolating continuous informative subregions or persistent suffix structures.

Design spirit: Because bidirectional attention makes KV reuse inherently lossy in DLLMs, effective KV management must balance direct reuse and selective recomputation to accelerate inference while maintaining quality.

Discussion. The transition from AR to DLLM KV management fundamentally shifts cache reuse from an exact mechanism to an approximate one. Because token representations continually evolve under bidirectional attention and iterative denoising, cached states can quickly become stale. The core challenge lies in navigating the trade-off between aggressive reuse (which improves efficiency but accumulates errors) and frequent recomputation (which preserves fidelity but erodes acceleration). Existing methods address this by enlarging reusable regions through structured schedules, selectively correcting stale states, or controlling memory costs via eviction. Ultimately, while increasingly fine-grained inference heuristics improve this balance, aligning the model with lossy caching patterns during training presents a cleaner and more scalable path to reducing reuse-induced errors.

2.2. Sparse Computation

Sparse computation improves inference efficiency by reducing redundant attention and memory access. In autoregressive (AR) models, sparse attention is mainly motivated by the quadratic cost of long-context prefilling and the growing overhead of KV cache access during decoding. Existing AR sparsification methods typically identify salient tokens or structured sparse patterns to restrict computation to the most relevant context [78–80] (as shown in Figure 3(a)). In DLLMs, sparse computation must be designed differently because bidirectional attention leads to attention patterns distinct from causal masking, while iterative denoising requires considering not only spatial sparsity with a step but also its consistency across the denoising process. As illustrated in Figure 3(b), existing DLLM sparse computation methods can be broadly organized into two categories: *attention pattern sparsity*, which exploits structured sparsity in attention maps across denoising steps, and *contextual sparsity*, which selectively retains the most informative context tokens to reduce redundant computation.

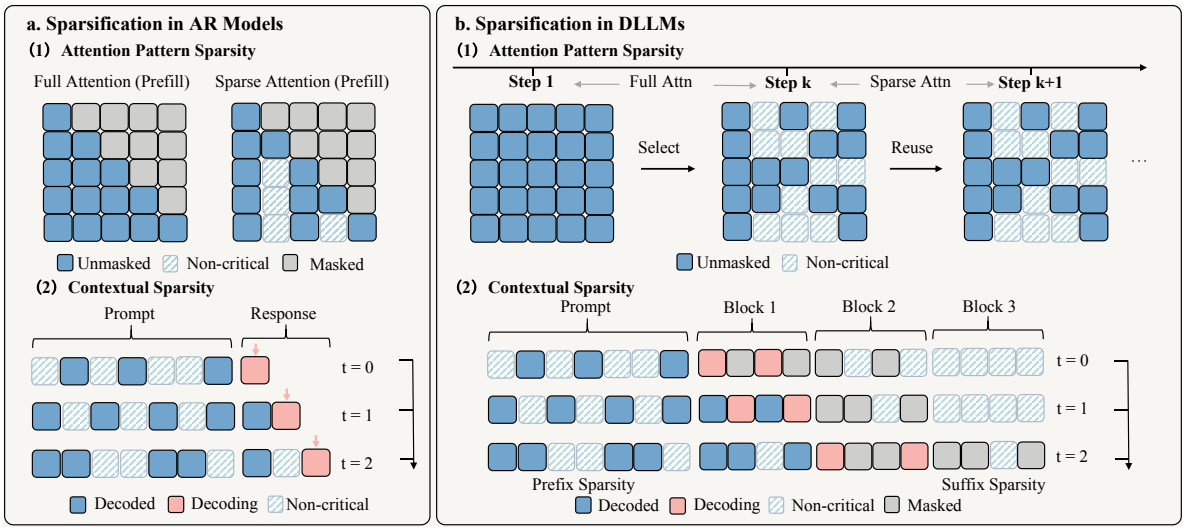

Figure 3. Comparison of sparse computation strategies in AR Models and DLLMs. **(a)** AR models minimize the computation through selection of critical tokens during both prefill and decoding stages under the attention layers (Attention Pattern Sparsity) or the input (Contextual Sparsity). **(b)** The sparsification of DLLM exploits spatio-temporal redundancy: (1) attention patterns transition from full to sparse at specific steps, after which the sparse mask is frozen and reused; (2) contextual sparsity involves the selection of pivotal prefix and suffix tokens across spatial blocks and denoising timesteps to enhance efficiency.

 doi:10.20944/preprints202605.0776.v1

2.2.1. Attention Pattern Sparsity

Attention pattern sparsity accelerates DLLM inference by identifying and reusing sparse attention structures that remain stable across denoising steps (as shown in Figure 3(b)(1)), thereby reducing repeated dense attention computation throughout the diffusion trajectory. SparseD [28] shows that although attention patterns vary across heads, they remain similar within each head over denoising steps, making reuse across steps feasible. It also finds that early denoising steps are more sensitive to attention perturbation, so it applies full attention early and switches to sparse attention later, reusing head-specific sparse patterns across subsequent steps. OTIE [40] further improves this line by addressing the moving-sink phenomenon in DLLMs. It observes that shifting attention sinks across denoising steps can hurt inference robustness, and introduces a position-stable sink token via attention mask to stabilize attention allocation.

2.2.2. Contextual Sparsity

Contextual sparsity accelerates DLLM inference by pruning redundant context during diffusion decoding (as shown in Figure 3(b)(2)). Sparse-dLLM [36] and Focus-dLLM [38] both leverage the temporal stability of token saliency to identify critical prefix and suffix context. Sparse-dLLM combines this idea with block-wise decoding and caching to retain pivotal entries for each block, while Focus-dLLM performs dynamic step-wise selection by using past-step confidence to predict the current unmasked regions and applying sink-aware pruning to the resulting active context. Another line of work targets redundant suffix context. DPad [41] observes that most suffix tokens behave like a low-entropy scratchpad and therefore retains only a small set of nearby and informative suffix tokens through sliding-window restriction and distance-decay dropout. Streaming-dLLM [39] builds on a similar intuition, further combining attenuation-guided suffix pruning with cache reuse and dynamic confidence-aware decoding for additional speedup.

> **Design spirit:** In DLLMs, iterative denoising makes sparsity inherently a cross-step problem, so effective sparse computation should exploit cross-step consistency in attention and context to identify reusable sparse patterns that reduce computation while preserving quality.

Discussion. Sparse computation is an important direction for DLLM inference, especially in long-context scenarios where redundant attention and contextual processing become increasingly costly. Compared with AR models, sparsity in DLLMs is inherently a cross-step problem, since bidirectional attention and iterative denoising continuously reshape token interactions, making isolated per-step sparsification insufficient. Effective DLLM sparsification therefore needs to capture cross-step consistency in both attention structures and contextual importance, so that structured and reusable sparse patterns can be identified throughout decoding, with future designs likely benefiting more from the intrinsic regularities of the denoising process than from rigid step-wise heuristics. Moreover, because fully recomputing dense interactions at every step is prohibitively expensive, combining sparsification with cache-based reuse may offer a more practical and scalable direction for future DLLM inference.

2.3. Decoding Methods

As shown in Figure 4(a), autoregressive (AR) models generate text in a left-to-right manner, where each token depends on all previously generated ones. This ensures strong sequential consistency but inherently limits inference parallelism [9,81]. In contrast, diffusion large language models (DLLMs) fundamentally break from this strictly serial paradigm by enabling parallel token updates through bidirectional attention [24,25,82]. However, naively decoding many tokens simultaneously can amplify uncertainty and error propagation, leading to unstable or degraded generation quality. To balance parallelism and quality, existing DLLM decoding methods in Figure 4(b) can be broadly categorized into four groups: *heuristic-based decoding*, which uses inference-time signals such as confidence to regulate parallel updates; *revocation-based mechanisms*, which revise or re-mask low-quality drafts during decoding; *training-based optimization*, which better aligns training objectives with parallel

 doi:10.20944/preprints202605.0776.v1

inference; and *length-adaptive decoding*, which dynamically adjusts the generation span to overcome the fixed-length constraint in DLLMs.

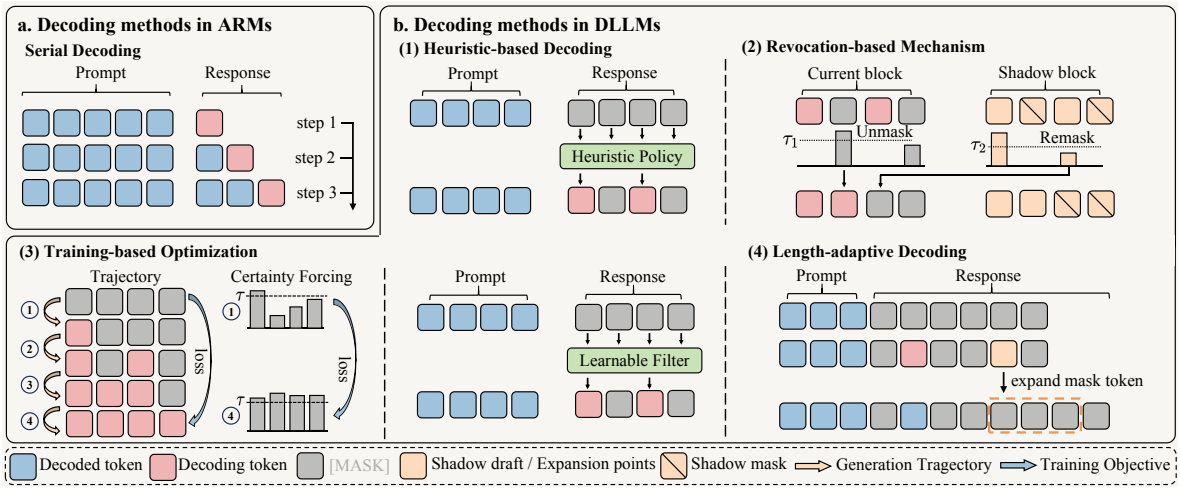

Figure 4. Comparison of decoding paradigms in AR models and DLLMs. (a) Traditional AR models follow a strictly serial decoding process where tokens are generated one-by-one in a left-to-right causal order; (b) DLLMs leverage bidirectional attention to move beyond sequential generation through four primary strategies: (1) heuristic-based decoding employs statistical signals like confidence thresholds to identify and unmask multiple "safe" tokens in parallel; (2) revocation-based mechanisms introduce a "draft-and-verify" capability, allowing the model to aggressively propose candidates and subsequently correct low-quality drafts to prevent error propagation; (3) training-based optimization, which aligns training with efficient parallel decoding trajectories and replaces heuristic policies with learnable filters; and (4) length-adaptive decoding, which adjusts the generation length during decoding to reduce redundant computation.

2.3.1. Heuristic-Based Decoding

Heuristic-based decoding accelerates DLLM inference without modifying model parameters by exploiting inference-time signals such as confidence, entropy, and prediction stability (Figure 4(b)(1)). Fast-dLLM [27] parallelizes tokens whose confidence exceeds a fixed threshold, while UNCODE [42] refines this idea with position-aware weighting and frequency-calibrated confidence to avoid over-selecting trivial tokens. EB-Sampler [43] replaces fixed thresholds with an entropy-bounded criterion that adaptively determines the parallel decoding budget based on cumulative uncertainty. Other methods exploit structural regularities to design schedules: DUS [47] groups non-adjacent positions to reduce joint uncertainty, LocalLeap [46] uses reliable local anchors to guide neighborhood decoding, and Saber-AADU [48] gradually increases parallelism as structural context becomes clearer in code generation. Several approaches also leverage temporal stability during decoding. KLASS [44] uses inter-step KL divergence to identify low-risk tokens, while SlowFast [45] and Certainty Prior Decoding [33] exploit phase-wise decoding dynamics to enable faster yet more stable generation.

2.3.2. Revocation-Based Mechanisms

A key limitation of heuristic-based decoding is that accepted tokens are typically irreversible, allowing early errors to propagate. Revocation-based methods address this by re-masking or revising previously decoded tokens (Figure 4(b)(2)). ReMDM [50] introduces a tunable remasking probability into masked discrete diffusion, enabling inference-time scaling and iterative error correction without retraining. WINO adopts a draft-and-verify scheme that generates many tokens in parallel and then re-masks low-quality drafts using bidirectional context. ReMix [51] further extends revocation to continuous-state refinement, resetting unstable positions to mask tokens through a rejection rule. For code generation, Saber-BERM [48] implements revocation as lightweight backtracking, rolling back tokens when confidence drops or logical inconsistencies arise.

 doi:10.20944/preprints202605.0776.v1

2.3.3. Training-Based Optimization

Unlike heuristic and revocation-based approaches that operate only at inference time, training-based methods improve parallel decoding more fundamentally by aligning training objectives with parallel inference (Figure 4(b)(3)). D2F [52] converts a standard DLLM into a block-wise AR-diffusion hybrid and explicitly trains it for more effective parallel generation. FS-DFM [55] combines consistency-aware training across different step budgets with Runge-Kutta solvers to reduce sampling steps. dParallel [53] accelerates certainty formation on masked tokens through certainty-forcing distillation, while LSD/LSD+ [57] aligns student trajectories with strong teachers and learns non-uniform time schedules. Consistency Distillation [54] further enables direct jumps from intermediate noisy states to cleaner ones for aggressive step compression. Beyond step reduction, dLLM-Var [58] supports native variable-length generation by explicitly modeling EOS, and Learn2PD [56] trains a lightweight MLP to decide which masked tokens should be generated immediately and which should remain masked, thereby reducing premature predictions and enabling earlier termination.

2.3.4. Length-Adaptive Decoding

This line addresses the fixed-length limitation of DLLMs by dynamically adjusting the generation span (Figure 4(b)(4)). DAEDAL [59] proposes a two-stage training-free strategy that combines initial length adjustment with iterative mask insertion, allowing the span to expand when the current length is insufficient. ODB-dLLM [62] further integrates adaptive length prediction into cache refresh, reducing prefill overhead and redundant computation from overestimated response lengths. ρ-EOS [60] enables bidirectional length control within a unified denoising process by tracking implicit EOS density and deciding whether the masked space should expand or contract. Distinctly, LR-DLLM [61] treats length as an explicit inference-time variable and introduces a length-regularized confidence criterion for more reliable candidate-length comparison, enabling training-free probing and bidirectional refinement.

Design spirit: Parallel decoding in DLLMs weakens token dependencies and makes error propagation more severe. Achieving a strong quality-speed trade-off in DLLMs ultimately depends on effectively reducing parallelism errors and controlling their propagation.

Discussion. Decoding methods play a central role in translating the theoretical parallelism of DLLMs into practical inference gains. Unlike AR decoding, DLLMs parallelize token updates under weakened inter-token dependency, making error accumulation harder to avoid. Although many decoding strategies have been proposed from different angles, the DLLM inference ecosystem is still far from mature, and current decoding designs remain highly fragmented. Many of them are developed largely in isolation, with limited consideration of their compatibility with other inference components such as KV caching and sparse computation, which in turn restricts their generalizability and system-level effectiveness. Future progress will likely require first establishing a unified and stable inference paradigm, upon which new decoding strategies can be more systematically integrated and more tightly aligned with training for better overall coordination.

2.4. Speculative Decoding

Speculative decoding has been highly successful in autoregressive (AR) models through a draft-then-verify paradigm that reduces expensive target-model calls [83] (Figure 5(a)). However, this framework does not directly generalize to DLLMs, since diffusion decoding relies on iterative parallel refinement rather than fixed left-to-right generation, requiring new designs for both drafting and verification. However, the core idea of speculation remains effective for DLLM acceleration. As shown in Figure 5(b)(c), existing methods can be broadly grouped into two categories: *pure diffusion self-speculation*, where a single DLLM handles drafting and verification across different denoising steps, and *diffusion-AR synergy speculation*, where drafting follows DLLM-style block diffusion while verification remains AR-style.

 doi:10.20944/preprints202605.0776.v1

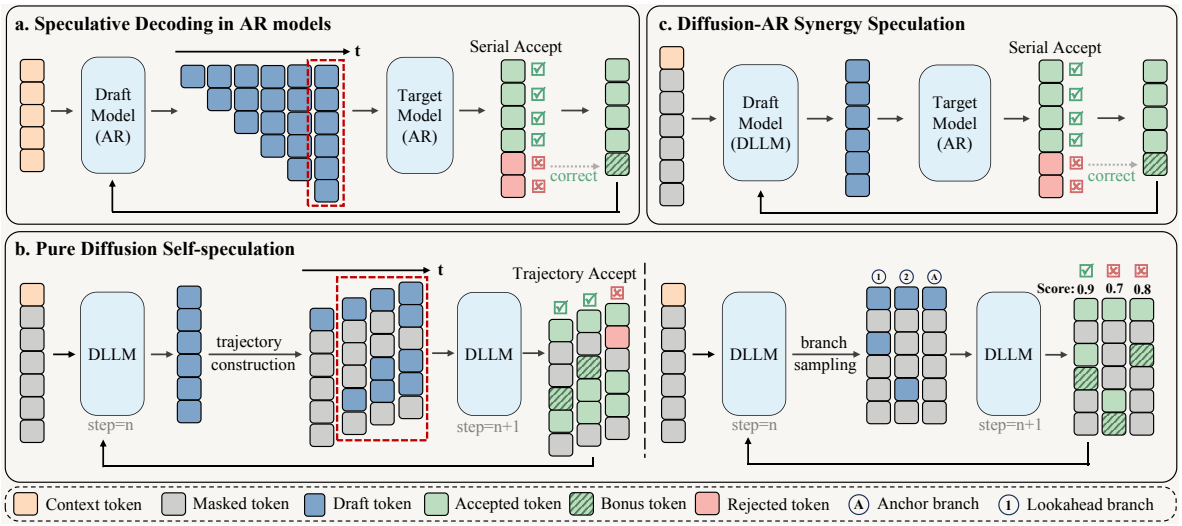

Figure 5. Comparison of speculative decoding paradigms across AR models and DLLMs. (a) A lightweight AR drafter generates tokens serially, which are then verified in parallel and accepted sequentially by an AR target model; (b) the DLLM performs self-speculation by constructing multiple candidate trajectories or sampling lookahead branches at one denoising step, followed by trajectory verification or best-branch selection at the next step; (c) a highly parallel DLLM replaces the serial drafter to accelerate proposal generation, while the AR target model still performs causal verification and sequential acceptance.

2.4.1. Pure Diffusion Self-Speculation

The first category, pure diffusion self-speculation (Figure 5(b)), accelerates DLLMs by exploiting their own iterative denoising process, where earlier steps draft speculative futures and later steps verify them. Unlike AR-style speculation, this setting is inherently non-sequential, so the central challenge is how to organize future trajectories or lookahead branches for efficient verification. SSD [63] follows a linear draft-and-verify chain, using self-drafting and hierarchical verification to accept multiple drafted tokens within one forward pass. FreeDAVE [64] constructs multiple candidate trajectories by sampling states with different numbers of unmasked tokens and verifies them at the next denoising step. Spiffy [65] extends this trajectory-based drafting into an offline-calibrated directed draft graph for parallel verification. In contrast, LoPA [66] adopts a lookahead-branch formulation, sampling an anchor branch together with multiple lookahead branches to favor future parallelization. Together, these works show that even without a fixed decoding order, speculation remains feasible in DLLMs through well-designed draft trajectories and verification schemes.

2.4.2. Diffusion-AR Synergy

Another line of work, diffusion-AR synergy (Figure 5(c)), uses the block-wise generation capability of DLLMs to accelerate AR decoding. Since AR speculative decoding still depends on sequential drafting, recent methods explore hybrid designs that combine DLLM-style drafting with AR-style verification. TiDAR [67] unifies diffusion drafting and AR verification within a single framework using structured attention masks. DiffuSpec [68], SpecDiff [69], and SpecDiff-2 [70] employ diffusion models as AR drafters, with SpecDiff-2 further improving draft–target alignment. DEER [71], DART [72], and DFlash [73] further strengthen this paradigm with lightweight DLLM drafters and improved alignment strategies, demonstrating strong practical competitiveness over AR-based drafters. Although these methods primarily target AR acceleration rather than native DLLM inference, they highlight the potential of DLLMs as effective collaborators in hybrid inference frameworks.

Design spirit: The uncertainty of DLLM decoding paths makes traditional speculative verification difficult, so the design core of DLLM speculative decoding is to rapidly predict future trajectories and bypass redundant refinement steps for higher parallel efficiency.

 doi:10.20944/preprints202605.0776.v1

Discussion. The success of speculative decoding in AR models naturally motivates its extension to DLLMs. However, diffusion decoding paths in DLLMs are often sample-adaptive and stochastic, making traditional AR-style verification hard to apply. As a result, speculative decoding in DLLMs shifts from precise token-level verification to predicting future generation trajectories in advance and then refining them locally, thereby reducing redundant denoising steps. Under this view, self-speculation becomes a particularly natural design choice for DLLMs, and existing methods have accordingly explored different ways of constructing such draft trajectories. At the same time, such path variability makes it difficult to utilize more sophisticated tree-based speculation mechanisms developed for AR speculation. Future progress will likely depend on more fine-grained trajectory or tree-based speculation strategies, as well as stronger collaboration mechanisms.

3. Benchmark

Our benchmarking framework follows a phased approach to ensure a rigorous comparison between DLLM and AR-LLM paradigms. We begin by conducting a modular evaluation of representative techniques across the key research axes identified in our taxonomy. This initial phase serves to assess the individual performance gains and mutual compatibility of various methods, allowing us to identify the optimal components for the overall integration. Following this modular assessment, we synthesize these top-performing components into synergistic DLLM configurations to conduct a head-to-head comparison against state-of-the-art AR models, thereby quantifying the empirical efficiency gap of two paradigms.

3.1. Representative Acceleration Selection in Four Directions

Building on our literature review, we empirically evaluate competitive methods to isolate their individual efficiency gains. To ensure a rigorous and fair comparison, we select candidate methods based on three criteria: (i) preserving generation quality, (ii) delivering meaningful inference speedup, and (iii) integrating easily with other techniques. Following these principles, we benchmark open-source methods on LLaDA-8B-Instruct [24] and Dream-v0-7B-Instruct [25] over GSM8K [84], HumanEval [85], IFEval [86], and GPQA [87], with detailed benchmark settings provided in Appendix A.1. We adopt a standardized block-wise decoding setup with block size 32, and fix the maximum generation length to 256 for GSM8K/GPQA and 512 for HumanEval/IFEval. For fairness, we tune each method to keep accuracy degradation within, or as close as possible to, a 1% threshold relative to the vanilla baseline, following the selection protocol in Appendix A.2. This setup enables a direct comparison of their raw speedup potential under nearly identical quality constraints.

3.1.1. KV Management

Following above criteria, we evaluate representative open-source, reproducible KV management methods, with results in Figure 6. Here, and in later figures, green denotes accuracy degradation within the strict 1% threshold, yellow indicates marginal acceptance, and red marks substantial loss beyond the acceptable range. To ensure a reliable inference pipeline, we prioritize methods that minimize red-zone failures while still delivering meaningful speedup.

Among block-wise caching methods, Prefix Cache is the most dependable. Across datasets on both LLaDA and Dream, it consistently avoids red-zone failures, keeps most results in the green zone, and delivers strong, stable throughput gains. By contrast, although Dual Cache occasionally achieves higher peak speedups, it loses accuracy in more than half of the tested cases, suggesting that its more aggressive reuse of evolving token states is too lossy for general deployment.

Dynamic refreshing methods are even more task-sensitive, especially on reasoning-heavy benchmarks. While d2Cache sometimes provides higher speedups, it repeatedly fails on logic-intensive tasks such as HumanEval and GPQA. In comparison, dKV-Cache is markedly more stable, meeting the accuracy requirement across all tests while still providing substantial acceleration. It also consistently outperforms finer-grained alternatives such as SPA-Cache in raw speedup, making it our preferred dynamic refreshing method. Finally, KV eviction methods like Sparse-dLLM are excluded from the

 doi:10.20944/preprints202605.0776.v1

final configuration, as discarding pivotal tokens often causes severe accuracy degradation in DLLMs. Overall, *Prefix Cache* and *dKV-Cache* provide the best balance between stability and speed, and thus serve as the most suitable KV-management components for our integrated DLLM inference pipeline.

3.1.2. Sparse Computation

To evaluate sparse computation methods, we choose SparseD as the representative of attention-pattern sparsity and DPad as the representative of contextual sparsity. As shown in Figure 7, SparseD achieves a better overall balance between accuracy and speed across most benchmarks, whereas DPad more often falls into the red zone due to its aggressive suffix dropping. Although SparseD yields only moderate gains under our current setting and is not uniformly stable across all tasks, sparse attention remains particularly promising for long-context scenarios, where attention increasingly dominates inference cost and the induced sparsity patterns are often more favorable for preserving accuracy. Given this superiority, we retain *SparseD* as the sparse computation component in our subsequent integration.

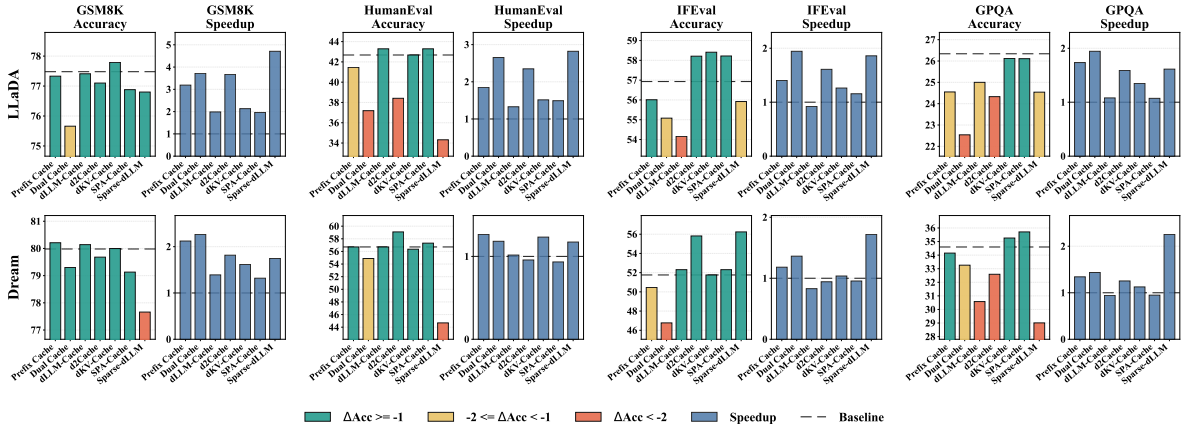

Figure 6. Performance and speed comparison of various KV caching methods across four benchmarks. Green indicates accuracy degradation within the 1% threshold, yellow indicates marginal acceptance, and red indicates unacceptable accuracy degradation. The same color convention is used in subsequent figures.

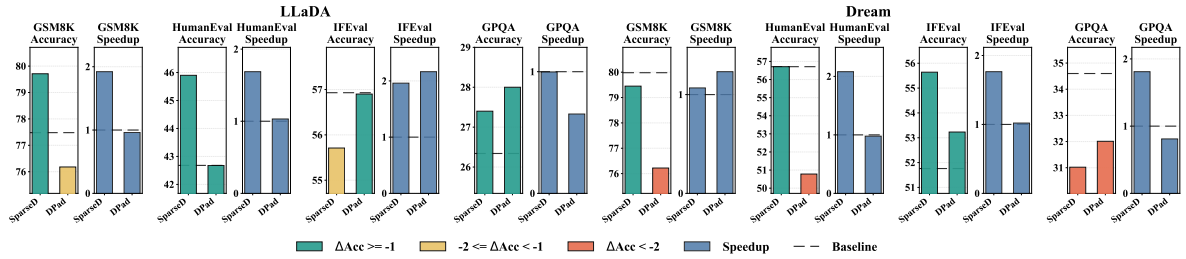

Figure 7. Performance and speed comparison of various sparsification methods across four benchmarks. The same color convention as in Figure 6 applies.

3.1.3. Decoding Methods

Our evaluation of decoding strategies first considers heuristic methods, including the basic top-k rule and more advanced policies such as Confidence thresholding, EB-Sampler, KLASS, and UNCODE. Since UNCODE is designed mainly to improve generation precision rather than direct acceleration, we evaluate it together with EB-Sampler, following its original setup. As shown in Figure 4, Confidence thresholding emerges as one of the fastest methods overall, maintaining "green" accuracy in most cases. KLASS also delivers strong fidelity, staying in the green zone across all benchmarks while still providing stable acceleration. By contrast, the basic top-k rule causes severe accuracy degradation, while EB-Sampler and UNCODE can reach high peak speedups in specific cases but are less stable overall.

 doi:10.20944/preprints202605.0776.v1

For revocation-based decoding, we evaluate WINO as a representative method. Its draft-and-verify design can outperform standard confidence thresholding in speed, but the more aggressive drafting also introduces noticeable accuracy loss in some cases. For training-based optimization, we evaluate D2F, dParallel, and Learn2PD. Among them, dParallel shows the strongest overall profile, achieving a clear speed advantage over the vanilla confidence-thresholding baseline while maintaining green-level accuracy on most datasets. In contrast, D2F and Learn2PD are less stable and exhibit visible accuracy drops on several benchmarks. Length-adaptive decoding is represented by DAEDAL, which we combine with confidence thresholding in our parallel decoding setting. Empirically, it improves accuracy in a few cases, such as IFEval and GPQA on LLaDA, while remaining broadly comparable to confidence thresholding overall. However, as dynamic length adjustment is not well compatible with cache-based acceleration and other optimizations, we exclude it from our final integrated pipeline.

Overall, because heuristic methods are highly compatible and require only changes to the sampling process, we retain *Confidence thresholding* and *KLASS* as versatile decoding components. We also include the *dParallel*-refined model as a strong model-level component in our optimized pipeline. Together, these three methods serve as the core decoding blocks for balancing inference speed and generation quality.

3.1.4. Speculative Decoding

We evaluate FreeDave and LoPA as representative DLLM speculative decoding methods, corresponding to trajectory-based speculation and lookahead-branch speculation, respectively. To isolate the effect of speculation itself, we do not combine them with cache acceleration; for FreeDave, we adopt a flexible confidence threshold to enable dynamic drafting. As shown in Figure 8, LoPA achieves more aggressive acceleration but often suffers substantial quality degradation, with accuracy dropping by more than three points on most tasks. By contrast, FreeDave offers a more stable speed–quality trade-off, preserving accuracy under careful threshold control while still delivering consistent speedup. However, under our setting, it still does not outperform strong heuristic parallel decoding methods such as confidence thresholding or KLASS in overall speed-accuracy balance. We therefore retain the simpler heuristic decoding components as the default choice in our subsequent integrated pipeline.

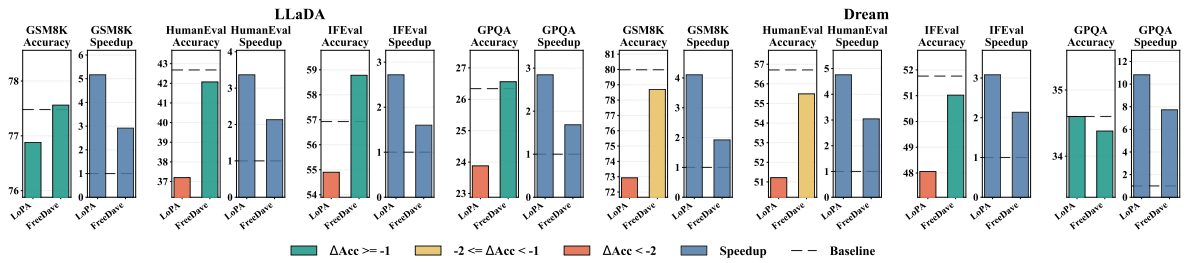

Figure 8. Performance and speed comparison of various speculative decoding methods across four benchmarks. The same color convention as in Figure 6 applies.

 doi:10.20944/preprints202605.0776.v1

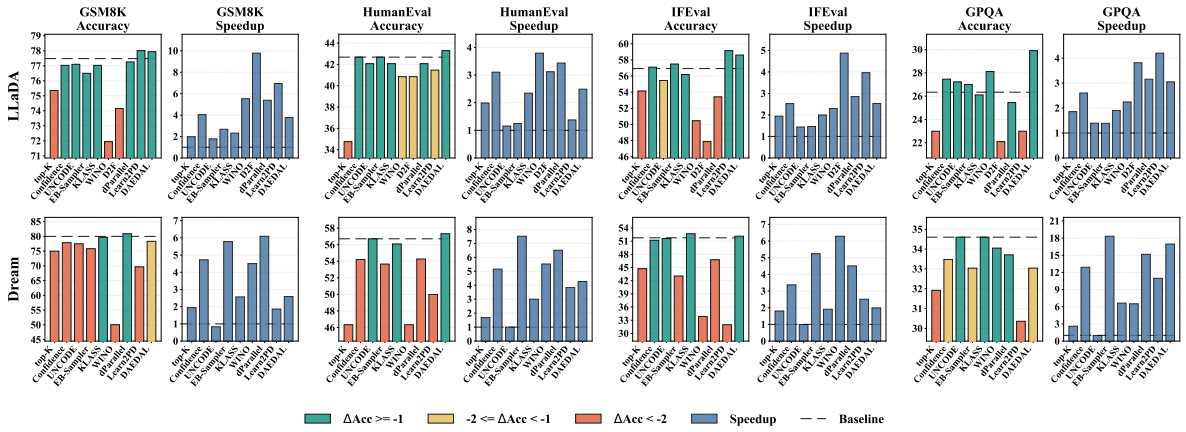

Figure 9. Performance and speed comparison of various decoding methods across four benchmarks. The same color convention as in Figure 6 applies.

3.2. DLLM v.s. AR Under Both State-of-the-Art Acceleration

Building on the component-level analysis above, we next compare optimized DLLM inference systems with state-of-the-art autoregressive (AR) baselines. We construct synergistic DLLM pipelines by integrating the selected components: Prefix Cache, dKV-Cache, SparseD, Confidence thresholding, KLASS, and dParallel. We also include *Fast-dLLM v2* [30] as a representative integrated DLLM system, since it already combines cache optimization, training adaptation, and parallel decoding within a unified design. To ensure a strictly fair and engineering-agnostic comparison, all models are implemented and evaluated in a unified codebase. We report both end-to-end throughput and step-level efficiency, with detailed definitions and measurement protocols provided in Appendix A.3.

For a more comprehensive evaluation, we extend the benchmark suite with MATH-500 [88] and MBPP [89], which feature greater task complexity and longer generations, and with LongBench-v1 [90] to assess long-context performance and efficiency. As strong AR baselines, we benchmark architecturally aligned models from the LLaMA [91] and Qwen [92] families under both vanilla decoding and EAGLE-3 [17], a state-of-the-art lossless speculative decoding framework that provides a strong reference for AR inference.

As current DLLMs require the generation length to be specified in advance, we calibrate an appropriate length for each benchmark before large-scale evaluation. Figure 10 shows these benchmark-specific calibration results, with the selected setting marked by star. This protocol exposes the strongest achievable performance of each DLLM under benchmark-appropriate inference budgets while avoiding bias from a single fixed generation-length setting. For Fast-dLLM v2, we follow the official setup and fix the generation length at 2048. Additional settings and final configurations are summarized in Appendix A.2.

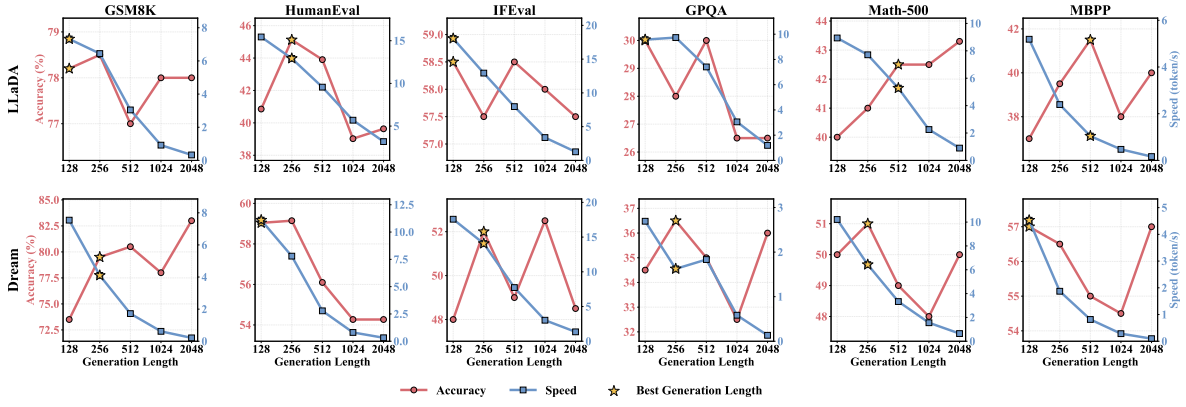

Figure 10. Performance and speed comparison of various decoding generation lengths across six benchmarks.

 doi:10.20944/preprints202605.0776.v1

As reported in Table 2, when accuracy degradation is kept limited, combining mutually compatible DLLM acceleration components yields clear cumulative speedups over the vanilla DLLM baseline. At the same time, the strongest DLLM configurations (grey zone) generally lie between vanilla AR decoding and EAGLE-3 in throughput: they are often able to approach, and in a few cases surpass, vanilla AR speed, but still remain consistently below the speculative AR frontier. A similar pattern is observed in long-context evaluation. As shown in Table 3, even the fastest DLLM configuration, *Prefix Cache + Confidence*, still fails to match the throughput of vanilla AR baselines on LongBench.

Main Results: Under general-generation settings and near-lossless accuracy constraints, optimized DLLM systems can already approach and in some cases exceed the throughput of vanilla AR decoding. However, they still remain below state-of-the-art speculative AR inference, and this gap persists in long-context scenarios.

To further examine the accuracy-speed trade-off beyond a single operating point, we plot the operating curves of *Prefix Cache + Confidence* on LLaDA and *Fast-dLLM v2* across multiple decoding thresholds. These smooth curves highlight an inherent property of DLLM decoding: its effective parallelism can be adjusted through the acceptance threshold, trading accuracy for higher throughput. This also enables comparison in the full accuracy-speed plane, where methods closer to the upper-right Pareto frontier achieve a more favorable trade-off. As shown in Figure 11, pushing DLLM decoding toward higher parallelism can sometimes make it faster than AR + EAGLE-3, but usually only under aggressive thresholds that incur substantial accuracy loss. Consequently, although DLLM curves may extend further right in some cases, AR + EAGLE-3 generally stays closer to the Pareto frontier on most benchmarks, indicating that optimized AR inference still holds a stronger overall efficiency advantage under current setting.

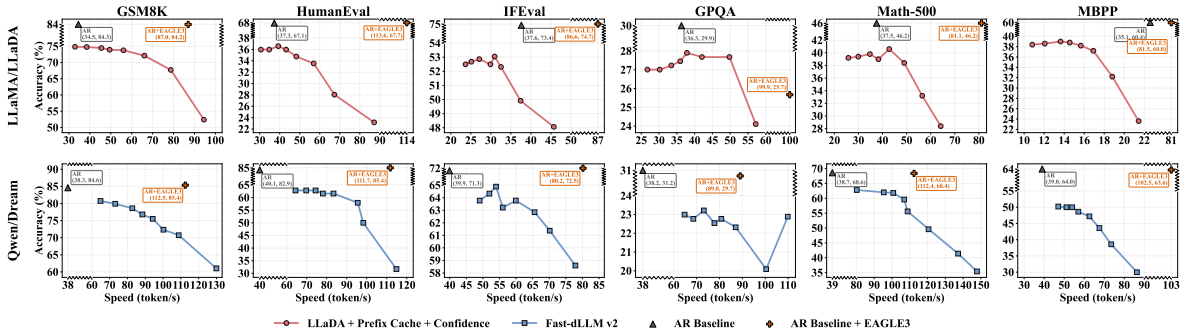

Figure 11. Accuracy–speed trade-off comparison between optimized DLLM inference systems and autoregressive baselines across six benchmarks. For each benchmark, we evaluate *LLaDA + Prefix Cache + Confidence* and *Fast-dLLM v2* under eight decoding thresholds ranging from 0.50 to 0.98, and compare their operating curves with the corresponding vanilla AR baseline and AR baseline equipped with EAGLE-3. Along each curve, moving from left to right corresponds to decreasing the decoding threshold.

Beyond efficiency, generation quality remains another key concern for DLLMs. Across most evaluated benchmarks, AR models still achieve stronger overall performance, suggesting that practical DLLM deployment requires further progress not only in speed, but also in generation quality and reliability.

 doi:10.20944/preprints202605.0776.v1

Table 2. Performance and speed comparison. **AR** (Autoregressive) and **DLLM** variants are evaluated across six benchmarks. **Acc.** denotes task accuracy, and **Speed** denotes end-to-end inference throughput, measured in tokens per second from input to completion.

Family	Method	GSM8K		HumanEval		IFEval		GPQA		Math-500		MBPP		Avg.	
		Acc.	Speed	Acc.	Speed	Acc.	Speed	Acc.	Speed	Acc.	Speed	Acc.	Speed	Acc.	Speed
Llama-3 / LLaDA	LLaMA-3.1-8B-Instruct	84.31	34.54	67.07	37.29	73.38	37.62	29.91	36.30	46.20	37.53	60.40	35.07	60.21	36.39
	+ EAGLE-3	84.23	86.97	67.68	113.60	74.68	86.59	25.67	99.87	46.20	81.10	60.00	81.49	59.74	91.60
	LLaDA-8B-Instruct	77.10	7.53	45.12	12.96	55.45	17.80	28.12	10.30	39.40	5.43	40.20	2.20	47.56	9.37
	+ Prefix Cache + Confidence	74.45	45.53	36.59	38.65	53.05	30.94	27.68	49.80	40.60	42.79	39.00	14.56	45.23	37.05
	+ Prefix Cache + KLASS	73.92	31.26	36.59	26.19	52.50	21.14	28.79	26.55	38.80	28.25	39.20	10.01	44.97	23.90
	+ Prefix Cache + dParallel	74.91	56.59	35.98	31.87	47.32	31.19	30.80	47.71	37.00	36.95	38.00	12.72	44.00	36.17
	+ dKV-Cache + Confidence	75.13	45.00	40.24	34.65	54.90	29.68	29.02	39.58	39.20	30.27	39.40	12.51	46.31	31.95
	+ d KVCache + KLASS	73.84	31.95	37.20	27.91	51.57	21.78	28.57	26.37	36.00	22.08	38.60	10.09	44.30	23.36
	+ dKV-Cache + dParallel	75.44	47.82	40.85	26.55	49.35	29.47	32.81	43.86	37.20	20.48	40.80	10.29	46.07	29.75
	+ SparseD + Confidence	76.80	20.60	45.12	20.31	54.53	18.59	28.79	39.79	39.00	7.32	41.40	9.50	47.61	19.35
	+ SparseD + KLASS	76.42	12.03	45.73	12.34	54.34	10.84	27.68	21.88	40.40	7.80	41.40	7.10	47.66	12.00
	+ SparseD + dParallel	75.89	29.92	44.51	17.24	50.28	26.95	23.66	37.83	36.40	18.91	40.20	9.51	45.16	23.39
Qwen-2.5 / Dream	Qwen-2.5-7B-Instruct	84.61	38.33	82.93	40.13	71.35	39.93	31.25	38.20	68.60	38.69	64.00	39.02	67.12	39.05
	+ EAGLE-3	85.37	112.49	85.37	111.73	72.46	80.17	29.67	89.05	68.40	112.43	63.60	102.52	67.48	101.40
	Dream-v0-7B-Instruct	79.83	4.29	58.54	11.36	52.68	13.60	34.38	1.96	48.00	6.60	56.00	4.43	54.91	7.04
	+ Prefix Cache + Confidence	78.85	31.49	54.27	32.82	49.54	23.19	33.93	20.51	46.00	23.11	53.80	25.67	52.73	26.13
	+ Prefix Cache + KLASS	78.09	16.00	57.93	20.04	48.43	14.85	33.93	11.03	44.40	10.72	51.20	14.78	52.33	14.57
	+ Prefix Cache + dParallel	79.15	47.08	51.83	48.46	44.92	32.22	33.04	22.24	44.20	41.27	53.20	25.44	51.06	36.12
	+ dKV-Cache + Confidence	78.39	24.34	54.27	30.58	51.02	21.75	35.04	20.33	48.00	26.55	55.20	22.35	53.65	24.32
	+ dKV-Cache + KLASS	76.35	15.15	53.66	18.38	46.21	14.38	33.26	10.12	41.40	15.40	49.60	14.33	50.08	14.63
	+ dKV-Cache + dParallel	79.45	40.42	54.27	31.54	47.13	23.50	32.59	20.79	45.60	37.40	51.60	25.68	51.77	29.89
	+ SparseD + Confidence	78.32	17.34	57.32	32.22	48.61	19.87	33.48	22.19	48.00	11.49	55.80	18.24	53.59	20.22
	+ SparseD + KLASS	77.86	8.75	57.32	16.94	48.06	11.88	34.82	12.04	48.00	8.72	55.20	9.24	53.54	11.26
	+ SparseD + dParallel	79.00	27.89	51.83	54.01	45.10	29.02	32.14	25.71	44.80	18.71	52.80	16.02	50.95	28.56
	Fast-dLLM v2	78.62	82.67	62.80	74.41	64.88	53.95	23.21	73.17	61.80	100.73	50.00	53.95	56.88	73.15

doi:10.20944/preprints202605.0776.v1

Table 3. Performance evaluation on **LongBench-v1**. Accuracy scores are reported for each sub-task, with **Avg.** denoting overall accuracy and **TPS** denoting decode-stage throughput in generated tokens per second.

Method	Single-Doc. QA		Multi-Doc. QA			Summarization		Few-shot Learning		Synthetic		Code		Avg.	TPS
	Qasper	MFQA-en	HotpotQA	2Wiki	MuSiQue	Gov	QMSum	TREC	TriviaQA	LSHT	PRe	LCC	RB-P		
LLaMA-3.1-8B-Instruct	44.65	56.15	56.11	42.75	31.32	35.54	25.15	72.50	90.21	45.00	99.50	63.76	60.77	55.65	41.86
UltraLLaDA	11.78	26.06	30.55	24.45	21.79	31.95	21.37	79.00	91.86	42.00	89.63	67.72	48.53	45.13	1.32
+ Prefix Cache + Confidence	11.59	30.01	39.47	30.43	28.61	29.91	21.52	73.00	89.19	39.00	89.25	66.68	45.19	45.68	27.67
+ dKV-Cache + Confidence	11.54	27.46	34.63	24.07	23.73	31.41	21.96	73.00	89.19	39.00	84.74	67.46	46.28	44.19	18.09
+ SparseD + Confidence	11.61	25.33	30.32	23.24	20.79	31.35	21.83	75.25	90.45	40.00	91.80	66.96	47.88	44.37	7.67
Qwen-2.5-7B-Instruct	43.75	52.34	58.33	48.21	31.41	31.21	23.50	72.00	87.21	38.00	100.00	61.39	68.48	58.59	46.46
Dream-v0-7B-Instruct	34.56	42.37	43.84	39.58	24.99	23.41	18.89	72.50	90.04	16.00	31.00	63.64	60.83	43.20	0.81
+ Prefix Cache + Confidence	31.72	42.08	44.10	40.56	23.07	22.06	19.18	71.25	85.85	16.00	46.50	62.74	60.52	43.51	23.21
+ dKV-Cache + Confidence	30.58	41.24	43.37	38.64	22.07	23.05	18.73	72.00	87.57	15.50	40.50	62.80	60.03	42.77	21.20
+ Prefix Cache + dParallel	31.60	40.86	42.75	42.78	19.99	23.58	19.26	70.50	86.07	12.75	42.00	55.78	58.24	42.01	24.15
+ SparseD + Confidence	33.55	41.39	44.25	39.65	25.07	21.97	10.23	72.50	89.37	16.00	31.00	63.63	56.38	41.92	4.62
Fast-dLLM v2	37.16	49.21	41.64	30.32	15.92	32.98	24.38	66.50	73.87	42.50	83.50	40.83	52.64	45.50	32.15

 doi:10.20944/preprints202605.0776.v1

3.3. Analysis

The empirical results above suggest that, although recent acceleration techniques have substantially improved DLLM inference, current DLLM systems still do not consistently surpass well-optimized AR systems under rigorous comparison. In this section, we distill several key insights from our empirical study and discuss their implications in greater depth, with the aim of clarifying the major bottlenecks of current DLLM inference and informing future research directions.

Insight 1: DLLM inference modules are highly composition-sensitive, so future acceleration methods should also take the design for system-level composability into consideration rather than only optimized in isolation.

Table 2 shows that, although the preceding experiments select relatively accuracy-preserving techniques for combination, their joint use does not always preserve accuracy across benchmarks. This suggests that inference modules in DLLMs interact in non-trivial ways, so independently optimized components may not compose into an optimal pipeline. In particular, approximation errors introduced by one module can be amplified once combined with other acceleration strategies, leading to larger-than-expected accuracy degradation. This observation suggests that future DLLM inference methods should be designed together with the target inference pipeline, such as the intended cache mechanism or parallel decoding strategy, rather than optimized as standalone modules.

Insight 2: The remaining gap between DLLMs and AR models in general generation settings mainly stems from limited effective parallelism under accuracy constraints and substantially heavier decoding computation.

To better understand this gap from a more fine-grained efficiency perspective, Table 4 breaks inference latency into the average latency of full-query steps (L_{full}), partial-query steps (L_{part}), the ratio of full-query steps (r_{full}), the average latency per forward step (L_{step}), and the average number of effectively accepted tokens per forward step (TPF). A more detailed explanation of these efficiency measures is provided in Appendix A.3. The results show that, although current DLLMs achieve non-trivial parallelism, their effective generation progress still generally lags behind strong AR speculative decoding methods such as EAGLE-3 under comparable accuracy constraints. A key reason is the persistent quality-parallelism trade-off: finalizing more tokens in parallel weakens dependency modeling among newly generated tokens and often causes larger accuracy degradation, forcing practical parallel decoding to remain conservative. In addition, because most current DLLMs require a conservatively overestimated generation length in advance, part of the computation is wasted on decoding tokens that do not contribute meaningful content, which further limits the achievable TPF.

 doi:10.20944/preprints202605.0776.v1

Table 4. Latency breakdown of AR and DLLM inference on GSM8K and HumanEval. L_{full}, L_{part}, and L_{step} denote the average latency of full-query, partial-query, and all forward steps, respectively. r_{full} is the full-query step ratio, **TPF** is the average accepted tokens per forward step, and **Speed** is end-to-end throughput in tokens per second.

Method	GSM8K						HumanEval					
	L_{full}	L_{part}	r_{full}	L_{step}	TPF	Speed	L_{full}	L_{part}	r_{full}	L_{step}	TPF	Speed
LLaMA-3.1-8B-Instruct	93.98	28.77	0.01	29.48	1.00	33.55	49.48	28.90	0.01	29.03	1.00	34.20
+ EAGLE-3	132.68	45.15	0.05	49.44	4.72	90.41	55.56	44.90	0.03	45.17	5.52	117.62
LLaDA-8B-Instruct / UltraLLaDA	129.12	/	1.00	129.12	1.00	7.48	73.38	/	1.00	73.38	1.00	13.03
+ Prefix Cache + Confidence	132.31	48.71	0.08	55.70	2.56	44.57	77.10	48.26	0.07	50.36	2.10	40.50
+ dKV-Cache + Confidence	136.85	47.18	0.29	73.49	3.38	45.04	79.92	47.43	0.28	56.48	2.06	35.64
+ SparseD + Confidence	133.50	/	1.00	133.50	2.71	20.81	107.86	/	1.00	107.86	1.87	20.51
Qwen-2.5-7B-Instruct	92.76	26.69	0.01	27.05	1.00	36.78	45.28	26.62	0.02	26.85	1.00	36.68
+ EAGLE-3	107.62	37.88	0.03	39.61	4.65	114.15	56.06	37.71	0.04	38.19	4.39	110.12
Dream-v0-7B-Instruct	127.54	/	1.00	127.54	1.00	4.27	53.20	/	1.00	53.20	1.00	11.28
+ Prefix Cache + Confidence	133.44	63.49	0.15	74.06	2.54	31.31	72.44	55.56	0.12	57.33	1.91	30.27
+ dKV-Cache + Confidence	139.45	63.38	0.33	88.35	2.35	24.44	76.94	53.61	0.31	60.79	1.91	28.80
+ Prefix Cache + dParallel	128.90	59.65	0.24	75.89	4.21	48.39	66.46	50.16	0.20	53.12	2.91	47.37
+ dKV-Cache + dParallel	132.98	56.47	0.38	85.17	4.02	41.33	73.80	50.76	0.33	58.15	1.97	30.36
+ SparseD + Confidence	144.67	/	1.00	144.67	2.67	17.29	57.97	/	1.00	57.97	1.90	31.95
Fast-dLLM v2	108.62	27.50	0.01	29.00	2.40	82.67	43.61	26.51	0.01	27.26	2.07	74.41

More importantly, DLLMs also incur substantially higher system costs during decoding. Unlike AR decoding, where a typical step uses a single-token query, partial-query steps in DLLMs usually attend over the entire uncached decoding region, leading to a much larger query span and thus a higher L_{part} than AR baselines. Full-query steps are also generally more expensive, as DLLMs must model not only prompt-side context but also interactions within the decoding region, increasing L_{full}. In addition, because cache reuse in current mainstream DLLMs is often lossy, inference still requires refresh-like or full-query computation to control accumulated errors, which further raises r_{full}. Overall, the remaining DLLM-AR gap is best understood as the combined effect of limited effective parallelism and heavier per-step computation. This also suggests that future practical DLLM systems may benefit more from cache-compatible semi-autoregressive designs. For example, Fast-dLLM v2 [30] adopts a training–inference aligned semi-autoregressive decoding scheme with block-level caching, avoiding refresh-style recomputation and restricting the query span to the current block, thereby reducing the gap in L_{part}, L_{full}, and r_{full} relative to AR baselines.

Insight 3: Long-context settings expose a sharper efficiency bottleneck for DLLM inference, as the recomputation overhead of the current KV cache and the sparse mechanisms grows with the context length.

Long-context settings make the efficiency bottlenecks of DLLM inference even more apparent. Existing cache-based methods cannot cleanly reuse intermediate states and therefore require periodic recomputation during iterative denoising. As context length grows, the cost of this recomputation rises quickly, reducing the practical benefit of cache acceleration. Sparse methods face a similar issue. Although approaches such as SparseD can lower attention cost in isolation, they remain insufficiently compatible with current cache designs, leaving attention recomputation substantial in practice. Together, these factors offset part of the theoretical parallelism advantage of DLLMs and make long-context inference especially hard to accelerate. More effective cache reuse and more adaptable sparse computation will therefore be essential for making DLLMs competitive in long-context settings.

4. Future Research Directions

Based on the above evaluation, we argue that future research on DLLM inference acceleration should move beyond isolated gains under customized settings and instead target more general and practically effective directions. As shown in Figure 12, these directions can be organized into three pillars: algorithm design, unified recipes, and practical scenarios. Together, they provide a more stable path for translating the theoretical parallelism of DLLMs into real-world efficiency gains.

 doi:10.20944/preprints202605.0776.v1

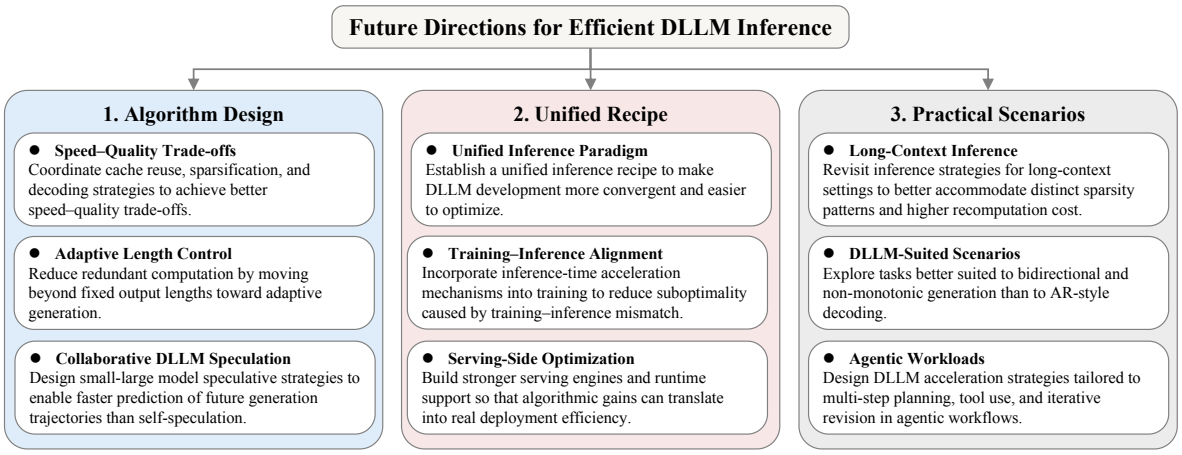

Figure 12. Overview of future directions for efficient DLLM inference, organized around three pillars: algorithm design, unified recipe, and practical scenarios.

4.1. Algorithm Design

4.1.1. Speed–Quality Trade-offs

A central challenge in DLLM inference is improving efficiency without sacrificing generation quality. In practice, aggressive parallel decoding and approximate cache reuse can both degrade fidelity, suggesting that future progress depends not only on stronger individual techniques but also on a clearer understanding of how cache reuse, sparsification, and decoding interact. Rather than optimizing components separately, a more promising direction is to co-design them for better speed–quality trade-offs.

4.1.2. Adaptive Length Control

Another practical bottleneck is length control in variable-length generation. Many current DLLMs still rely on fixed output lengths or rigid denoising schedules, which often undermine their efficiency advantages: overestimating length wastes computation, while underestimating it harms completeness or quality. Existing remedies, such as early stopping and suffix pruning, remain local fixes rather than full solutions. Looking ahead, more adaptive schemes, including block-wise semi-autoregressive designs that reduce reliance on fixed lengths while preserving parallel decoding, may offer a more practical path forward.

4.1.3. Collaborative DLLM Speculation

Another promising direction is collaborative speculative inference for DLLMs. While small-large model cooperation has been highly successful in autoregressive models, similar gains have yet to be fully realized for DLLMs. A key challenge is that DLLM decoding lacks the same sequential dependency structure, making standard draft-verify pipelines difficult to transfer. As smaller DLLMs and more standardized training recipes emerge, future work should explore collaborative speculation where lightweight models help larger ones predict generation more efficiently than current self-speculative designs.

4.2. Unified Recipe

4.2.1. Unified Inference Paradigm

Beyond improving individual modules, DLLMs still lack a stable and widely adopted inference recipe. Existing methods explore diverse design choices, but without a common paradigm, they remain difficult to combine, compare, or optimize systematically at the engine level. This fragmentation also weakens alignment with model architecture and training objectives. Future work should therefore move toward a unified inference paradigm that makes DLLM development more convergent and easier to optimize.

 doi:10.20944/preprints202605.0776.v1

4.2.2. Training-Inference Alignment

A closely related issue is the mismatch between DLLM training and accelerated inference. Many current optimizations are applied post hoc to models not explicitly trained for aggressive parallel decoding, approximate cache reuse, or specialized sampling, which limits both effectiveness and stability. A more principled direction is to incorporate these acceleration mechanisms into training, so that the inference maintains sufficient alignment with training.

4.2.3. Serving-Side Optimization

Beyond algorithm design, serving-side optimization is critical for translating algorithmic gains into deployment efficiency. Most current DLLM acceleration methods are still evaluated mainly at the algorithmic level, while the serving systems needed to support them remain underdeveloped. As inference recipes become more standardized, future work should increasingly focus on stronger serving stacks and inference engines that can fully realize the practical benefits of accelerated DLLM inference.

4.3. Practical Scenarios

4.3.1. Long-Context Inference

These challenges become even more severe in long-context settings. Due to bidirectional attention and global refinement, each denoising step may involve all tokens in the sequence, making its cost closer to an autoregressive prefill pass and increasing overall expense as context grows. Future work should therefore not simply extrapolate short-context acceleration strategies to long-context DLLMs. Instead, it should revisit inference design to better exploit their distinct sparsity patterns and higher recomputation cost.

4.3.2. DLLM-Suited Scenarios

More generally, the long-term value of DLLMs should not be judged solely by whether they reproduce autoregressive generation more efficiently. Their bidirectional and non-monotonic nature may make them inherently better suited to tasks that benefit from global revision rather than strict left-to-right generation, such as constrained generation, infilling-style reasoning, program repair, and multi-span rewriting. Exploring such scenarios may clarify where DLLMs are not only competitive with AR models, but also fundamentally better matched to the tasks.

4.3.3. Agentic Workloads

Another emerging direction is DLLM-based agents. Agentic workloads involve multi-step planning, tool use, intermediate revision, and iterative interaction with the environment, creating efficiency demands beyond one-pass text generation. In these settings, the parallelism of DLLMs may accelerate each generation step, but overall efficiency will also depend on how planning and revision are organized. Future work should therefore study acceleration for DLLM agents not only at the token level, but also at the levels of planning, action scheduling, and iterative refinement.

5. Conclusion

In this survey, we presented a comparative study of DLLM inference acceleration against AR-LLMs from the perspective of practical efficiency. We organized existing methods into four major directions (KV management, sparse computation, decoding methods, and speculative decoding) and examined them within a unified, modular evaluation framework. Our analysis shows that, although recent advances have substantially improved the practicality of DLLM inference, a clear gap still remains between current DLLM systems and highly optimized AR baselines. This gap stems from multiple factors, including unstable speed-quality trade-offs, immature cache reuse and adaptive generation mechanisms, and limited serving-side support. Overall, our findings suggest that DLLM inference acceleration is far from a free lunch, and that future progress will likely depend on more fundamental breakthrough in algorithm design, better recipe for unified inference paradigms, stronger

 doi:10.20944/preprints202605.0776.v1

training-inference alignment, and more mature deployment infrastructures, as well as more broad scenarios including long-context cases, suited tasks for DLLMs and their agentic workloads.

Appendix A Benchmark Details

This appendix complements the benchmark analysis by providing the experimental protocol required to reproduce our comparisons. We clarify the task variants, evaluation metrics, decoding constraints, hyperparameter selection rules, integrated configurations, and efficiency measurements used throughout the benchmark.

Appendix A.1 Tasks and Evaluation Settings

All benchmark results are evaluated using `lm-evaluation-harness`. Table A2 summarizes the benchmark tasks, metric/filter definitions, and decoding temperatures used in the main experiments. Unless otherwise specified, all methods adopt deterministic decoding with temperature 0. For LongBench-v1, we follow the task-specific evaluation protocol in `lm-evaluation-harness`. When a DLLM method requires a pre-specified output length, we set its default total generation length to the corresponding maximum generation tokens in Table A3.

Table A1. Task-specific confidence thresholds for integrated DLLM configurations. τ denotes the confidence threshold for token commitment. Thresholds are listed in the order of GSM8K, HumanEval, IFEval, GPQA, MATH-500, MBPP, and LongBench.

Method	Cache Setting	LLaDA Series Decoding Setting	Dream Series Decoding Setting
Prefix Cache + Confidence	block size = 32	$\tau = \{0.90, 0.90, 0.80, 0.60, 0.80, 0.90, 0.40\}$	$\tau = \{0.95, 0.98, 0.98, 0.90, 0.95, 0.95, 0.70\}$
Prefix Cache + KLASS	block size = 32	$\tau = \{0.70, 0.90, 0.80, 0.60, 0.80, 0.70, -\}$	$\tau = \{0.75, 0.75, 0.75, 0.75, 0.90, 0.95, -\}$
Prefix Cache + dParallel	block size = 32	$\tau = \{0.92, 0.98, 0.90, 0.75, 0.95, 0.95, -\}$	$\tau = \{0.90, 0.92, 0.95, 0.92, 0.95, 0.98, 0.80\}$
dKV-Cache + Confidence	refresh interval = 4	$\tau = \{0.80, 0.92, 0.75, 0.75, 0.90, 0.90, 0.40\}$	$\tau = \{0.98, 0.95, 0.92, 0.90, 0.92, 0.95, 0.70\}$
dKV-Cache + KLASS	refresh interval = 4	$\tau = \{0.70, 0.80, 0.60, 0.60, 0.80, 0.70, -\}$	$\tau = \{0.95, 0.90, 0.95, 0.75, 0.95, 0.75, -\}$
dKV-Cache + dParallel	refresh interval = 4	$\tau = \{0.92, 0.98, 0.90, 0.75, 0.98, 0.98, -\}$	$\tau = \{0.90, 0.98, 0.98, 0.92, 0.92, 0.95, -\}$
SparseD + Confidence	–	$\tau = \{0.90, 0.95, 0.90, 0.75, 0.98, 0.90, 0.50\}$	$\tau = \{0.92, 0.98, 0.98, 0.92, 0.98, 0.95, 0.90\}$
SparseD + KLASS	–	$\tau = \{0.70, 0.90, 0.80, 0.60, 0.60, 0.60, -\}$	$\tau = \{0.80, 0.75, 0.75, 0.75, 0.80, 0.90, -\}$
SparseD + dParallel	–	$\tau = \{0.92, 0.98, 0.90, 0.90, 0.95, 0.95, -\}$	$\tau = \{0.90, 0.92, 0.92, 0.92, 0.98, 0.98, -\}$
Fast-dLLM v2	block size = 32, small block size = 8	–	$\tau = \{0.90, 0.90, 0.90, 0.90, 0.90, 0.90, 0.70\}$

Table A2. Benchmark tasks and evaluation settings. Task IDs and metric/filter names follow `lm-evaluation-harness`.

Benchmark	Task ID	Metric / Filter	Temp.
GSM8K	gsm8k	exact_match, flexible-extract	0.0
HumanEval	humaneval_instruct	pass@1, create_test	0.0
IFEval	ifeval	prompt_level_strict_acc, none	0.0
GPQA	gpqa_main_generative_n_shot	exact_match, flexible-extract	0.0
MATH-500	math-500	math_verify, none	0.0
MBPP	mbpp_instruct	pass_at_1, extract_code	0.0
LongBench-v1	longbench	task-specific	0.0

Appendix A.2 Implementation and Hyperparameter Selection

We conduct the benchmark on 8 NVIDIA A100 GPUs within a unified evaluation codebase. In all inference runs, we use batch size 1 per GPU and report the corresponding end-to-end speed for fair comparison. Unless otherwise specified, DLLM decoding adopts a block size of 32, while unreported hyperparameters follow the default settings of the corresponding official implementations.

For acceleration methods involving a decoding threshold, including the modular selection experiments in Sec. III-A and the integrated configurations in Sec. III-B of the main paper, we use separate search ranges for general and long-context benchmarks. For the general benchmarks, thresholds are

 doi:10.20944/preprints202605.0776.v1

searched from 0.50 to 0.95 with a step size of 0.05, with an additional values of 0.92 and 0.98. For LongBench, thresholds are searched from 0.30 to 0.90 with a step size of 0.10. We choose the fastest setting whose accuracy degradation is no larger than 1% relative to the vanilla baseline; if no setting satisfies this constraint, we instead choose the setting with the highest accuracy. This rule provides a consistent accuracy-constrained protocol for comparing speedup across different acceleration methods.

Table A1 reports the final settings used for the integrated DLLM configurations of the main paper. The confidence threshold τ is the main tuned hyperparameter, as it directly controls token commitment during parallel decoding and is selected separately for each task. Other auxiliary parameters follow the default or recommended configurations of the corresponding official implementations after lightweight validation. KLASS uses a history length of 2, with the KL-divergence threshold set to 0.01 for the LLaDA series and 0.001 for the Dream series. SparseD uses a skip ratio of 0.2, sparse block size 32, and sparse ratio 0.5 on the general benchmarks. On long-context benchmarks, the skip ratio remains 0.2, while the sparse block size and sparse ratio are set to 128 and 0.3, respectively. For the AR baseline, EAGLE-3 uses depth 5, top-k 10, and draft-token budget 60 for the general benchmarks, and depth 4, top-k 1, and draft-token budget 6 for LongBench.

Appendix A.3 Efficiency Metrics and Latency Analysis

Table A3. LongBench-v1 evaluation settings. "Input Len." denotes the average tokenized input length measured with LLaMA-3.1-8B, and "Gen. Len." denotes the maximum evaluation length, also used as the default DLLM output length.

Task	Metric	Input Len.	Gen. Len.	Samples
Qasper	F1	5123	128	200
MultifieldQA_en	F1	6976	64	150
HotpotQA	F1	12889	32	200
2WikiMQA	F1	7203	32	200
Musique	F1	15652	32	200
GovReport	Rouge-L	10310	512	200
QMsum	Rouge-L	13952	512	200
TREC	Accuracy	6786	64	200
TriviaQA	F1	11816	32	200
LSHT	Accuracy	18349	64	200
PassageRetrieval_en	Accuracy	12557	32	200
RepoBench-P	Edit Sim	10803	64	500
Lcc	Edit Sim	3173	64	500

We adopt scenario-specific speed metrics and report end-to-end throughput for general benchmarks:

$$\text{Speed} = \frac{N_{\text{out}}}{T_{\text{total}}}, \tag{A1}$$

where N_{out} is the number of valid output tokens and T_{total} is the total wall-clock time from input processing to generation completion. For these benchmarks, we use this metric because DLLM inference involves additional computation over the full decoding region, which can make its prompt-side and cache-initialization costs differ from those of AR decoding. End-to-end throughput therefore offers a more comprehensive measure of inference efficiency than decoding-only speed.

 doi:10.20944/preprints202605.0776.v1

For long-context benchmarks, we instead report decoding throughput to better isolate generation-stage efficiency:

$$\text{Decode TPS} = \frac{N_{\text{out}}}{T_{\text{decode}}}, \tag{A2}$$

where T_{decode} denotes the wall-clock time spent in the decoding stage. LongBench tasks typically contain long inputs but relatively short outputs, so prefill latency accounts for a large portion of the total cost. Under this condition, variations in output length across models or decoding strategies can disproportionately affect end-to-end throughput, making it less reflective of generation-stage efficiency. Since the prefill cost is comparable across AR and DLLM models for the same long input, Decode TPS better isolates decoding efficiency and provides a clearer comparison in long-context scenarios.

We provide a detailed explanation of the latency-related quantities reported in Table 4. A full-query step denotes a forward step whose query covers the full active region. It includes the initial prefill computation and cache refresh computations, such as the first step of each new block in Prefix Cache and the fixed-interval refresh steps in dKV-Cache. For sparse-computation methods such as SparseD, which are not directly combined with cache reuse in our setting, every forward step is counted as a full-query step because the query is still computed over the full region. A partial-query step denotes a cache-enabled normal decoding step whose query only covers the uncached or currently active decoding region.

Based on this categorization, we denote the average latencies of full-query and partial-query steps by L_{full} and L_{part}, respectively. Their comparison reflects how much cache reuse reduces the cost of individual forward steps and reveals differences across decoding paradigms. The full-query ratio r_{full} measures the fraction of full-query steps among all forward steps, indicating how frequently a method invokes expensive refresh or full-region computation. We also report L_{step}, the average latency per forward step, and TPF, the average number of effectively accepted tokens per forward step, to characterize the step-level cost and effective decoding parallelism. For vanilla AR decoding, TPF is fixed to 1, while for speculative AR decoding, it corresponds to the average accepted length per verification step. Together, these quantities help identify where additional overhead arises when different methods realize parallel decoding, and whether the speedup mainly comes from lower per-step cost, fewer full-query computations, or larger effective token progress per step.

References

1. Huang, J.; Chang, K.C.C. Towards reasoning in large language models: A survey. In Proceedings of the Findings of the association for computational linguistics: ACL 2023, 2023, pp. 1049–1065.
2. Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F.L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* **2023**.
3. Bai, Y.; Kadavath, S.; Kundu, S.; Askell, A.; Kernion, J.; Jones, A.; Chen, A.; Goldie, A.; Mirhoseini, A.; McKinnon, C.; et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073* **2022**.
4. Comanici, G.; Bieber, E.; Schaekermann, M.; Pasupat, I.; Sachdeva, N.; Dhillon, I.; Blistein, M.; Ram, O.; Zhang, D.; Rosen, E.; 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**.
5. Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* **2025**.
6. Hui, B.; Yang, J.; Cui, Z.; Yang, J.; Liu, D.; Zhang, L.; Liu, T.; Zhang, J.; Yu, B.; Lu, K.; et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186* **2024**.
7. Bubeck, S.; Chandrasekaran, V.; Eldan, R.; Gehrke, J.; Horvitz, E.; Kamar, E.; Lee, P.; Lee, Y.T.; Li, Y.; Lundberg, S.; et al. Paper Review:'Sparks of Artificial General Intelligence: Early experiments with GPT-4' **2023**.
8. Zhou, Z.; Ning, X.; Hong, K.; Fu, T.; Xu, J.; Li, S.; Lou, Y.; Wang, L.; Yuan, Z.; Li, X.; et al. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294* **2024**.
9. Zhao, W.X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. A survey of large language models. *arXiv preprint arXiv:2303.18223* **2023**.

 doi:10.20944/preprints202605.0776.v1

10. Xia, H.; Yang, Z.; Dong, Q.; Wang, P.; Li, Y.; Ge, T.; Liu, T.; Li, W.; Sui, Z. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *Findings of the Association for Computational Linguistics: ACL 2024* **2024**, pp. 7655–7671.
11. Frantar, E.; Ashkboos, S.; Hoefler, T.; Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323* **2022**.
12. Lin, J.; Tang, J.; Tang, H.; Yang, S.; Chen, W.M.; Wang, W.C.; Xiao, G.; Dang, X.; Gan, C.; Han, S. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems* **2024**, *6*, 87–100.
13. Frantar, E.; Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. In Proceedings of the International conference on machine learning. PMLR, 2023, pp. 10323–10337.
14. Sun, M.; Liu, Z.; Bair, A.; Kolter, J.Z. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695* **2023**.
15. Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C.H.; Gonzalez, J.; Zhang, H.; Stoica, I. Efficient memory management for large language model serving with pagedattention. In Proceedings of the Proceedings of the 29th symposium on operating systems principles, 2023, pp. 611–626.
16. Leviathan, Y.; Kalman, M.; Matias, Y. Fast inference from transformers via speculative decoding. In Proceedings of the International Conference on Machine Learning. PMLR, 2023, pp. 19274–19286.
17. Li, Y.; Wei, F.; Zhang, C.; Zhang, H. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840* **2025**.
18. Yu, R.; Li, Q.; Wang, X. Discrete Diffusion in Large Language and Multimodal Models: A Survey. *arXiv preprint arXiv:2506.13759* **2025**.
19. Li, T.; Chen, M.; Guo, B.; Shen, Z. A survey on diffusion language models. *arXiv preprint arXiv:2508.10875* **2025**.
20. Tseng, C.Y.; Zhang, D.; Song, J.; Bi, Z. Diffusion-based Large Language Models Survey. *Authorea Preprints* **2025**.
21. Lin, H.; Jia, X.; Liu, S.; Xia, S.; Huang, W.; Xu, H.; Li, J.; Xiao, Y.; Xing, X.; Guo, Z.; et al. Efficient Diffusion Language Models: A Comprehensive Survey. *Authorea Preprints* **2026**.
22. Kyaw, K.M.; Chen, W.; Chan, J.; Kwok, J. Accelerating Inference in Diffusion Large Language Models: A Survey. *Authorea Preprints* **2026**.
23. Yan, M.; Agarwal, S.; Venkataraman, S. Decoding speculative decoding. In Proceedings of the Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), 2025, pp. 6460–6473.
24. Nie, S.; Zhu, F.; You, Z.; Zhang, X.; Ou, J.; Hu, J.; Zhou, J.; Lin, Y.; Wen, J.R.; Li, C. Large language diffusion models. *arXiv preprint arXiv:2502.09992* **2025**.
25. Ye, J.; Xie, Z.; Zheng, L.; Gao, J.; Wu, Z.; Jiang, X.; Li, Z.; Kong, L. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487* **2025**.
26. Lou, A.; Meng, C.; Ermon, S. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834* **2023**.
27. Wu, C.; Zhang, H.; Xue, S.; Liu, Z.; Diao, S.; Zhu, L.; Luo, P.; Han, S.; Xie, E. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618* **2025**.
28. Wang, Z.; Fang, G.; Ma, X.; Yang, X.; Wang, X. SparseD: Sparse Attention for Diffusion Language Models. *arXiv preprint arXiv:2509.24014* **2025**.
29. Arriola, M.; Gokaslan, A.; Chiu, J.T.; Yang, Z.; Qi, Z.; Han, J.; Sahoo, S.S.; Kuleshov, V. Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models. In Proceedings of the The Thirteenth International Conference on Learning Representations.
30. Wu, C.; Zhang, H.; Xue, S.; Diao, S.; Fu, Y.; Liu, Z.; Molchanov, P.; Luo, P.; Han, S.; Xie, E. Fast-dllm v2: Efficient block-diffusion llm. *arXiv preprint arXiv:2509.26328* **2025**.
31. Liu, Z.; Yang, Y.; Zhang, Y.; Chen, J.; Zou, C.; Wei, Q.; Wang, S.; Zhang, L. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295* **2025**.
32. Ma, X.; Yu, R.; Fang, G.; Wang, X. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781* **2025**.
33. Jiang, Y.; Cai, Y.; Luo, X.; Fu, J.; Wang, J.; Liu, C.; Yang, X. d^2Cache: Accelerating Diffusion-Based LLMs via Dual Adaptive Caching. *arXiv preprint arXiv:2509.23094* **2025**.

 doi:10.20944/preprints202605.0776.v1

34. Sun, W.; Tu, R.C.; Ding, Y.; Jin, Z.; Liao, J.; Jing, Y.; Tao, D. SPA-Cache: Singular Proxies for Adaptive Caching in Diffusion Language Models, 2026, [arXiv:cs.LG/2602.02544]. arXiv:2602.02544v1.
35. Nguyen-Tri, Q.; Ranjan, M.; Shen, Z. Attention is all you need for kv cache in diffusion llms. *arXiv preprint arXiv:2510.14973* **2025**.
36. Song, Y.; Liu, X.; Li, R.; Liu, Z.; Huang, Z.; Guo, Q.; He, Z.; Qiu, X. Sparse-dllm: Accelerating diffusion llms with dynamic cache eviction. *arXiv preprint arXiv:2508.02558* **2025**.
37. Huang, J.; Zhang, Y.; Yang, Y.; Huang, B.; Qi, B.; Liu, D.; Zhang, L. Mask tokens as prophet: Fine-grained cache eviction for efficient dllm inference. *arXiv preprint arXiv:2510.09309* **2025**.
38. Long, L.; Huang, Y.; Bai, S.; Gong, R.; Zhang, J.; Zhou, A.; Yang, J. Focus-dLLM: Accelerating Long-Context Diffusion LLM Inference via Confidence-Guided Context Focusing. *arXiv preprint arXiv:2602.02159* **2026**.
39. Xiao, Z.; Hao, Z.; Guo, J.; Luo, Y.; Liu, J.; Xu, J.; Hu, H. Streaming-dLLM: Accelerating Diffusion LLMs via Suffix Pruning and Dynamic Decoding. *arXiv preprint arXiv:2601.17917* **2026**.
40. Zhang, Z.; Xie, Z.; Zhong, L.; Liu, H.; Cao, S. One Token Is Enough: Improving Diffusion Language Models with a Sink Token. *arXiv preprint arXiv:2601.19657* **2026**.
41. Chen, X.; Huang, S.; Guo, C.; Wei, C.; He, Y.; Zhang, J.; Li, H.; Chen, Y. DPad: Efficient Diffusion Language Models with Suffix Dropout. *arXiv preprint arXiv:2508.14148* **2025**.
42. Huang, P.; Liu, S.; Liu, Z.; Yan, Y.; Wang, S.; Chen, Z.; Xiao, T. Pc-sampler: Position-aware calibration of decoding bias in masked diffusion models. *arXiv preprint arXiv:2508.13021* **2025**.
43. Ben-Hamu, H.; Gat, I.; Severo, D.; Nolte, N.; Karrer, B. Accelerated Sampling from Masked Diffusion Models via Entropy Bounded Unmasking. *arXiv preprint arXiv:2505.24857* **2025**.
44. Kim, S.H.; Hong, S.; Jung, H.; Park, Y.; Yun, S.Y. KLASS: KL-Guided Fast Inference in Masked Diffusion Models. *arXiv preprint arXiv:2511.05664* **2025**.
45. Wei, Q.; Zhang, Y.; Liu, Z.; Liu, D.; Zhang, L. Accelerating Diffusion Large Language Models with SlowFast: The Three Golden Principles. *arXiv preprint arXiv:2506.10848* **2025**.
46. Kong, F.; Zhang, J.; Liu, Y.; Wu, Z.; Tian, Y.; Zhou, G.; et al. Accelerating diffusion llm inference via local determinism propagation. *arXiv preprint arXiv:2510.07081* **2025**.
47. Luxembourg, O.; Permuter, H.; Nachmani, E. Plan for speed: Dilated scheduling for masked diffusion language models. *arXiv preprint arXiv:2506.19037* **2025**.
48. Dong, Y.; Ma, Z.; Jiang, X.; Fan, Z.; Qian, J.; Li, Y.; Xiao, J.; Jin, Z.; Cao, R.; Li, B.; et al. Saber: An efficient sampling with adaptive acceleration and backtracking enhanced remasking for diffusion language model. *arXiv preprint arXiv:2510.18165* **2025**.
49. Hong, F.; Yu, G.; Ye, Y.; Huang, H.; Zheng, H.; Zhang, Y.; Wang, Y.; Yao, J. Wide-in, narrow-out: Revokable decoding for efficient and effective dllms. *arXiv preprint arXiv:2507.18578* **2025**.
50. Wang, G.; Schiff, Y.; Sahoo, S.S.; Kuleshov, V. Remasking discrete diffusion models with inference-time scaling. *arXiv preprint arXiv:2503.00307* **2025**.
51. Ye, Y.; Hong, F.; Zheng, H.; Chen, X.; Chen, Z.; Wang, Y.; Yao, J. Rejection Mixing: Fast Semantic Propagation of Mask Tokens for Efficient DLLM Inference. *arXiv preprint arXiv:2602.22868* **2026**.
52. Wang, X.; Xu, C.; Jin, Y.; Jin, J.; Zhang, H.; Deng, Z. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192* **2025**.
53. Chen, Z.; Fang, G.; Ma, X.; Yu, R.; Wang, X. dparallel: Learnable parallel decoding for dllms. *arXiv preprint arXiv:2509.26488* **2025**.
54. Liang, Y.; Wang, Z.; Chen, H.; Sun, X.; Wu, J.; Yu, X.; Liu, J.; Barsoum, E.; Liu, Z.; Jha, N.K. CD4LM: Consistency Distillation and aDaptive Decoding for Diffusion Language Models. *arXiv preprint arXiv:2601.02236* **2026**.
55. Monsefi, A.K.; Bhendawade, N.; Ciosici, M.R.; Culver, D.; Zhang, Y.; Belousova, I. Fs-dfm: Fast and accurate long text generation with few-step diffusion language models. *arXiv preprint arXiv:2509.20624* **2025**.
56. Bao, W.; Chen, Z.; Xu, D.; Shang, Y. Learning to parallel: Accelerating diffusion large language models via learnable parallel decoding. *arXiv preprint arXiv:2509.25188* **2025**.
57. Fu, F.; Guo, T.; Liu, Z. Learnable sampler distillation for discrete diffusion models. *arXiv preprint arXiv:2509.19962* **2025**.
58. Yang, Y.; Wang, C.; Wang, S.; Wen, Z.; Qi, B.; Xu, H.; Zhang, L. Diffusion llm with native variable generation lengths: Let [eos] lead the way. *arXiv preprint arXiv:2510.24605* **2025**.
59. Li, J.; Dong, X.; Zang, Y.; Cao, Y.; Wang, J.; Lin, D. Beyond fixed: Training-free variable-length denoising for diffusion large language models. *arXiv preprint arXiv:2508.00819* **2025**.

 doi:10.20944/preprints202605.0776.v1

60. Yang, J.; Jiang, Y.; Shao, J. ρ-EOS: Training-free Bidirectional Variable-Length Control for Masked Diffusion LLMs. *arXiv preprint arXiv:2601.22527* **2026**.
61. Cheng, Z.; Jia, R.; Li, J.; Yang, G.W.; Guo, M.H.; Hu, S.M. Improving Variable-Length Generation in Diffusion Language Models via Length Regularization. *arXiv preprint arXiv:2602.07546* **2026**.
62. Wei, L.; Chen, W.; Tang, P.; Guo, X.; Ye, L.; Wang, R.; Li, M. Orchestrating Dual-Boundaries: An Arithmetic Intensity Inspired Acceleration Framework for Diffusion Language Models. *arXiv preprint arXiv:2511.21759* **2025**.
63. Gao, Y.; Ji, Z.; Wang, Y.; Qi, B.; Xu, H.; Zhang, L. Self Speculative Decoding for Diffusion Large Language Models. *CoRR* **2025**, *abs/2510.04147*.
64. Wu, S.; Zhang, J. Free Draft-and-Verification: Toward Lossless Parallel Decoding for Diffusion Large Language Models. *CoRR* **2025**, *abs/2510.00294*.
65. Agrawal, S.; Garrepalli, R.; Goel, R.; Lee, M.; Lott, C.; Porikli, F. Spiffy: Multiplying Diffusion LLM Acceleration via Lossless Speculative Decoding. *CoRR* **2025**, *abs/2509.18085*.
66. Xu, C.; Jin, Y.; Li, J.; Tu, Y.; Long, G.; Tu, D.; Song, M.; Si, H.; Hou, T.; Yan, J.; et al. Lopa: Scaling dllm inference via lookahead parallel decoding. *arXiv preprint arXiv:2512.16229* **2025**.
67. Liu, J.; Dong, X.; Ye, Z.; Mehta, R.; Fu, Y.; Singh, V.; Kautz, J.; Zhang, C.; Molchanov, P. TiDAR: Think in Diffusion, Talk in Autoregression, 2025, [arXiv:cs.CL/2511.08923].
68. Li, G.; Fu, Z.; Fang, M.; Zhao, Q.; Tang, M.; Yuan, C.; Wang, J. Diffuspec: Unlocking diffusion language models for speculative decoding. *arXiv preprint arXiv:2510.02358* **2025**.
69. Christopher, J.K.; Bartoldson, B.R.; Ben-Nun, T.; Cardei, M.; Kailkhura, B.; Fioretto, F. Speculative Diffusion Decoding: Accelerating Language Generation through Diffusion, 2025, [arXiv:cs.CL/2408.05636].
70. Sandler, J.; Christopher, J.K.; Hartvigsen, T.; Fioretto, F. SpecDiff-2: Scaling Diffusion Drafter Alignment For Faster Speculative Decoding, 2025, [arXiv:cs.CL/2511.00606].
71. Cheng, Z.; Yang, G.W.; Li, J.; Deng, Z.; Guo, M.H.; Hu, S.M. DEER: Draft with Diffusion, Verify with Autoregressive Models, 2025, [arXiv:cs.LG/2512.15176].
72. Liu, F.; Li, X.; Zhao, K.; Gao, Y.; Zhou, Z.; Zhang, Z.; Wang, Z.; Dou, W.; Zhong, S.; Tian, C. DART: Diffusion-Inspired Speculative Decoding for Fast LLM Inference, 2026, [arXiv:cs.CL/2601.19278].
73. Chen, J.; Liang, Y.; Liu, Z. DFlash: Block Diffusion for Flash Speculative Decoding, 2026, [arXiv:cs.CL/2602.06036].
74. Ma, Y.; Du, L.; Wei, L.; Chen, K.; Xu, Q.; Wang, K.; Feng, G.; Lu, G.; Liu, L.; Qi, X.; et al. dinfer: An efficient inference framework for diffusion language models. *arXiv preprint arXiv:2510.08666* **2025**.
75. Shazeer, N. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150* **2019**.
76. Zhang, Z.; Sheng, Y.; Zhou, T.; Chen, T.; Zheng, L.; Cai, R.; Song, Z.; Tian, Y.; Ré, C.; Barrett, C.; et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems* **2023**, *36*, 34661–34710.
77. Li, H.; Li, Y.; Tian, A.; Tang, T.; Xu, Z.; Chen, X.; Hu, N.; Dong, W.; Li, Q.; Chen, L. A survey on large language model acceleration based on kv cache management. *arXiv preprint arXiv:2412.19442* **2024**.
78. Jiang, H.; Li, Y.; Zhang, C.; Wu, Q.; Luo, X.; Ahn, S.; Han, Z.; Abdi, A.H.; Li, D.; Lin, C.Y.; et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems* **2024**, *37*, 52481–52515.
79. Lai, X.; Lu, J.; Luo, Y.; Ma, Y.; Zhou, X. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. *arXiv preprint arXiv:2502.20766* **2025**.
80. Tang, J.; Zhao, Y.; Zhu, K.; Xiao, G.; Kasikci, B.; Han, S. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774* **2024**.
81. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Advances in neural information processing systems* **2020**, *33*, 1877–1901.
82. Gong, S.; Li, M.; Feng, J.; Wu, Z.; Kong, L. Diffuseq: Sequence to sequence text generation with diffusion models. *arXiv preprint arXiv:2210.08933* **2022**.
83. Leviathan, Y.; Kalman, M.; Matias, Y. Fast Inference from Transformers via Speculative Decoding. In Proceedings of the International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA; Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; Scarlett, J., Eds. PMLR, 2023, Vol. 202, *Proceedings of Machine Learning Research*, pp. 19274–19286.
84. Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168* **2021**.

 doi:10.20944/preprints202605.0776.v1

85. Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H.P.D.O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* **2021**.
86. Zhou, J.; Lu, T.; Mishra, S.; Brahma, S.; Basu, S.; Luan, Y.; Zhou, D.; Hou, L. Instruction-Following Evaluation for Large Language Models. *arXiv preprint arXiv:2311.07911* **2023**.
87. Rein, D.; Hou, B.L.; Stickland, A.C.; Petty, J.; Pang, R.Y.; Dirani, J.; Michael, J.; Bowman, S.R. Gpqa: A graduate-level google-proof q&a benchmark. In Proceedings of the First conference on language modeling, 2024.
88. Lightman, H.; Kosaraju, V.; Burda, Y.; Edwards, H.; Baker, B.; Lee, T.; Leike, J.; Schulman, J.; Sutskever, I.; Cobbe, K. Let's verify step by step. In Proceedings of the The Twelfth International Conference on Learning Representations, 2023.
89. Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* **2021**.
90. Bai, Y.; Lv, X.; Zhang, J.; Lyu, H.; Tang, J.; Huang, Z.; Du, Z.; Liu, X.; Zeng, A.; Hou, L.; et al. Longbench: A bilingual, multitask benchmark for long context understanding. In Proceedings of the Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers), 2024, pp. 3119–3137.
91. Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783* **2024**.
92. Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; et al. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115* **2024**.