Hardware Characterization of Diffusion vs. Autoregressive Language Model Inference: Compute vs. Memory Bottlenecks

Zhenxing Fan, Kevin Skadron
Department of Computer Science, University of Virginia
{fjy3ws, skadron}@virginia.edu

Abstract

Autoregressive (AR) language models dominate today's LLM deployments. Masked diffusion language models (MDLMs) have recently achieved quality comparable to similarly scaled AR models, yet their inference behavior on modern GPUs remains largely uncharacterized. We present a hardware-level characterization of representative diffusion language models and show that, although MDLMs and AR models share similar Transformer building blocks, diffusion inference exhibits fundamentally different bottlenecks at the hardware level, breaking the assumptions underlying modern AR serving architectures and systems.

First, we find that each denoising forward pass is compute-bound and operates near the GPU roofline, shifting the bottleneck from HBM bandwidth in AR decode to tensor-core throughput. Second, under standard (uncached) implementation, diffusion inference is activation-dominated rather than KV-cache-dominated. Third, block diffusion interpolates between AR and full-sequence diffusion behavior. **In particular, block size emerges as a hardware operating parameter that allows block diffusion models to be tuned to match the available system resources.** *Lastly, several AR-inherited optimizations exhibit qualitatively different behavior under diffusion workloads. Building on these observations, we analyze the transferability of the modern AR optimization stack to diffusion workloads and identify emerging hardware and system optimization directions for diffusion-oriented inference accelerators.*

1. Introduction

Autoregressive (AR) language models dominate today's LLM deployments, and modern serving systems have largely been built around two core properties of AR inference: token-by-token decoding is memory-bandwidth-bound, and the KV cache is the dominant source of dynamic memory growth and bandwidth pressure. Masked diffusion language models (MDLMs) generate text through a fundamentally different mechanism. Instead of emitting one token at a time conditioned on a left context, an MDLM starts with a fully masked sequence and iteratively denoises it using bidirectional full-sequence prediction. After a fixed number of denoising steps, the sequence becomes fully unmasked and is returned as the output.

This non-autoregressive formulation leads to properties absent in AR models. First, because each position conditions on the full sequence context instead of only its left prefix,

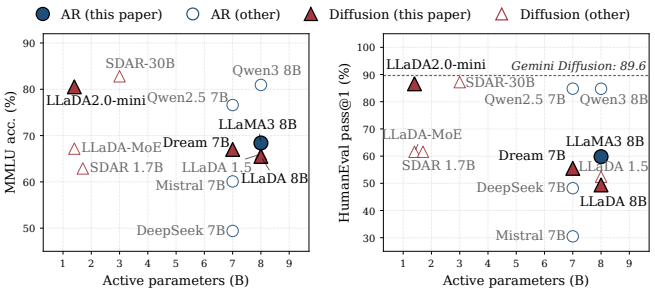

Figure 1: Generation quality of representative MDLMs and AR models at comparable scale (active parameters). Reported accuracies are from prior work [9, 11, 38, 47, 63, 64, 69]. The dashed line marks Gemini Diffusion's reported pass@1 as a closed-source reference [12].

MDLMs mitigate pathologies such as the reversal curse and improve performance on inherently non-causal tasks such as code generation [15, 61]. Second, iterative full-sequence refinement also enables finer-grained controllability and exposes quality–latency trade-off [3, 38, 64]. Historically, these advantages were accompanied by weaker generation quality; however, recent MDLMs have narrowed this gap rapidly and now achieve performance competitive with similarly scaled AR baselines on benchmarks such as MMLU [14] and HumanEval [7], as shown in Fig. 1.

Recent work on MDLMs has focused primarily on algorithmic optimizations [17, 32, 35, 58], analytical/roofline characterization [22], or throughput evaluation [46]. Yet, the underlying GPU execution behavior of diffusion models, such as per-kernel composition, fine-grained roofline characteristics, memory-hierarchy interaction, and scaling behavior, has not been systematically measured. To address this, we present a hardware-level characterization of representative models spanning the major emerging paradigms, including autoregressive, full-sequence diffusion, and block diffusion with both GQA-based [2] and Mixture-of-Experts (MoE)-based [49] models. Using kernel-level and system-level GPU profiling, we show that although diffusion and AR models share the same Transformer building blocks, diffusion inference exhibits fundamentally different hardware bottlenecks and scaling behavior.

We make four main findings. First, unlike AR decode, which is memory-bandwidth-bound, a single forward pass of full-sequence diffusion transitions from memory-bound to compute-bound as sequence length grows and approaches

the GPU tensor-core roofline at practical sequence lengths, making per-step computation of pure diffusion closely resemble AR *prefill*. Second, full-sequence diffusion replaces the KV-cache-dominated memory growth of autoregressive models with activation-dominated memory growth, where activations (intermediate tensors produced during the forward pass) scale linearly with sequence length and batch size. When caching approaches are applied, the magnitude of such activation dominance depends on specific caching scheme. Third, block diffusion exposes a continuum between AR-style and full-diffusion execution behaviors, with block size serving as a systems-level parameter rather than a purely algorithmic choice. **Varying block size shifts compute throughput, cache locality, and multi-GPU communication cost, enabling block-diffusion deployments to better match an accelerator's compute capability, on-chip SRAM capacity, and memory bandwidth.** Finally, AR-inherited architectural choices can behave non-intuitively under diffusion. For example, GQA, originally introduced to reduce memory bandwidth pressure and latency in AR inference, provides little attention-speedup benefit under full-sequence diffusion; instead, its primary benefit shifts to reducing activation-memory footprint.

Building on these findings, we show that diffusion workloads fundamentally alter the optimization priorities of LLM inference serving stacks and accelerator design, rendering AR-centric optimizations, including KV-cache management infrastructure, request-level batching, and even weight-only quantization substantially less effective or counterproductive. Instead, diffusion inference shifts architectural pressure toward high-throughput dense compute, large on-chip SRAM capacity, activation-aware memory management, and aggressive elementwise-kernel fusion, motivating a distinct hardware–software co-design direction for diffusion-oriented inference accelerators.

In summary, this paper makes the following contributions:

- We present a systematic hardware-level characterization of MDLM inference, covering full-sequence diffusion, block diffusion, and MoE-based variants alongside an autoregressive baseline, quantitatively identify the sequence length crossover, kernel composition, and memory behavior shifts that distinguish diffusion from AR inference.
- We show that block size acts as a system-level operating parameter that interpolates between AR-style and full-sequence diffusion behavior, shaping both hardware efficiency and compatibility with optimizations in AR-oriented serving systems.
- We provide a transferability analysis of the modern AR serving optimization stack to diffusion workloads and derive design guidelines for diffusion-oriented serving systems.

2. Background

Modern LLMs can be broadly categorized into two generative paradigms: autoregressive (AR) models, which form the basis of the vast majority of deployed LLMs today, and masked diffusion language models, a recently scaled alternative that iteratively refines a fully masked sequence through multiple denoising steps. This section describes the inference procedures of both paradigms.

2.1. Autoregressive Inference

An AR model factorizes the joint probability of a sequence into left-conditional distributions, where each token is predicted from all preceding tokens. Inference is therefore inherently sequential, where the model is invoked once per generated token, with each decoding step conditioned on the accumulated context. Two properties largely determine the execution characteristics of AR inference. First, attention is causal, where each token attends only to itself and preceding tokens, producing a lower-triangular attention mask. Second, this causality enables KV caching. Because the key/value tensors of previously generated tokens remain unchanged as new tokens are appended, they can be computed once and reused across subsequent decoding steps, reducing the per-token attention compute complexity from $O(n^2)$ to $O(n)$ with respect to sequence length.

These properties create two distinct execution phases. Prefill processes the entire prompt in a single forward pass and is compute-bound at moderate sequence lengths. In contrast, decode is memory-bandwidth-bound, where each step streams the full model weights and the growing KV cache from HBM while performing relatively little computation. End-to-end latency therefore scales linearly with output length as compute throughput is underutilized.

2.2. Masked Diffusion Language Model Inference

Diffusion language models encompass a range of approaches, including continuous-space models [28, 54], discrete variants such as absorbing-state [4] and uniform-state models [16], score-based approaches [34, 36], and hybrid autoregressive-diffusion schemes [3, 13, 66]. We focus on the masked diffusion models [48, 51] (MDMs), which have demonstrated generation quality comparable to similarly scaled autoregressive models [5, 6, 9, 27, 38, 64].

Rather than factorizing generation over token positions in a fixed order, a masked diffusion language model (MDLM) defines a forward corruption process that progressively replaces tokens in a clean sequence with a special mask token (denoted as [MASK]) until all positions are masked. A learned reverse process then reconstructs the sequence by iteratively predicting the masked tokens. The model is trained to predict all masked positions simultaneously from bidirectional (potentially partially masked) context. To generate a response of length L conditioned on a prompt, the MDLM inference begins with a sequence in which all L positions in response are occupied by [MASK], where L is a user- or system-specified hyperparameter, with the prompt held fixed.[1] The reverse process is then discretized

1. If the natural response is shorter than L, an EOS is produced somewhere in the middle, and post-EOS positions are discarded at output time, but those positions still participate in every forward pass.

TABLE 1: Summary of evaluated models.

	LLaMA 3-8B	LLaDA-8B	Dream-7B	LLaDA2.1-mini
Type	AR	Diffusion	Diffusion	Block diffusion
Total params	8B	8B	7B	16B
Active params	8B	8B	7B	∼1.4B
Architecture	Dense	Dense	Dense	MoE
Base config	–	LLaMA3†	Qwen2.5	Ling-mini
Attn pattern	Causal	Bi-dir.	Bi-dir.	Block-causal + bi-dir.
Attn. mech.	GQA	MHA	GQA	GQA
KV cache	Full	None‡	None	None
Parallelism	1 token/step	Full seq.	Full seq.	Block

†LLaDA-8B replaces GQA with MHA [56] and reduces FFN dimension to maintain a comparable parameter count [38]. ‡Optimized inference frameworks support approximate KV caching; we analyze its impact in Section 5.4.

into T sampling steps; at each step, the model performs one forward pass over the full prompt-response sequence and predicts a distribution over the vocabulary for every masked position in parallel, a subset of which is committed (unmasked) while the remainder are fed back as [MASK] into the next step. After T steps, the sequence is fully unmasked and returned as output. The mask predictor is a bidirectional Transformer with full (non-causal) attention over the entire sequence with no left-to-right constraint.

Block diffusion is a structural variant that partitions the response into segments of size L_B (block size) and generates them sequentially. Within each block, masked positions are denoised in parallel over multiple diffusion steps, while across blocks, generation proceeds left-to-right similar to AR models. Therefore, the attention pattern within the active block is bidirectional, while attention from the active block back to committed blocks is causal (one-directional). Block diffusion thus partially restores KV-cache reusability for committed blocks, recovering an optimization unavailable in pure MDLMs. The relationship between block size and generation quality is non-monotonic and shaped by several competing factors. Larger L_B increases bidirectional context within each denoising step, but also increases gradient variance during training, reduces supervision density, and requires denoising from more heavily masked inputs, which degrades quality [3, 6, 52, 58].

3. Experimental Setup and Methodology

3.1. Workload Selection

We study four representative models that collectively span the major architectural and inference paradigms emerging across AR and diffusion language modeling (Table 1). All four sit in the 7B–16B parameter range, enabling consistent hardware characterization while preserving diversity in attention structure, caching behavior, and execution pattern. In Section 6.2, we additionally profile SDAR-30B-A3B [9], a larger block-diffusion model to test the impact of block size under tensor parallelism beyond the single-GPU setting.

LLaMA 3-8B [37] serves as our AR baseline, a standard decoder-only Transformer with causal masking, GQA, and full KV caching, representative of the dominant deployment paradigm for contemporary LLM inference. **LLaDA 8B** [38] is an 8B-parameter masked diffusion model trained from scratch. Its depth, hidden dimension, and attention head count match LLaMA 3-8B, while each denoising step recomputes attention over the full sequence with no KV reuse, making it our primary representative of the pure diffusion inference. **Dream 7B** [64] is a full-sequence diffusion model derived from the Qwen2.5-7B [47] AR backbone by replacing causal attention with bidirectional attention and fine-tuning under a masked diffusion objective. Unlike LLaDA, Dream retains the GQA configuration of its AR base. The Dream/LLaDA pair allows us to compare two full-sequence MDLMs that differ in attention head organization and inherited AR architectural choices independent of the diffusion algorithm. **LLaDA2.1-mini** [5] is a larger 16B-parameter MoE block diffusion model extended from LLaDA2.0 [6] lineage, built on the Ling-mini [55] backbone. The inclusion of LLaDA2.1-mini allows us to study how block diffusion (block size) and MoE sparsity jointly shape diffusion inference behavior.

3.2. Profiling Setup

We profile all models on a single NVIDIA H100 94 GB GPU [40] in BF16 precision, except for the multi-GPU tensor-parallel study in Section 6.2, which uses NVLink-connected B200 GPUs [39]. We use two complementary profiling paths to capture kernel-level behavior and system-level resource consumption. **Kernel-level profiling** uses NVIDIA Nsight Compute (NCU) [44] to capture per-kernel arithmetic intensity, SM throughput, DRAM bandwidth utilization, and cache hit rates. For diffusion models, we profile a single denoising forward pass. For LLaMA 3-8B, we profile prefill and decode separately, with decode measurements taken after KV-cache prefilling to the targeted sequence length to reflect realistic decode behavior. **System-level measurement** uses PyTorch's allocator-level memory accounting APIs to track HBM allocation across forward passes, and wall-clock timing with explicit CUDA synchronization for latency and throughput. All measurements include warmup iterations to avoid initialization and cold-cache effects.

NCU profile data is post-processed in two stages. First, kernels are grouped into logical categories: GEMM (cuBLAS/cuBLASLt-dispatched [41], used for projections and MLP), Attention (FlashAttention forward and its associated combine kernels [10]), Sampling (top-k and related sorting/indexing for committing denoised tokens), and Elementwise (RMSNorm, SiLU, residual adds, elementwise multiply, gather/index, concatenation). Then cumulative metrics (e.g., cycles, duration) are summed within each category, while rate-based metrics (e.g., throughput, utilization) are aggregated using duration-weighted averages.

4. Compute Characterization

This section establishes the first structural shift in diffusion inference relative to AR: compute characteristic.

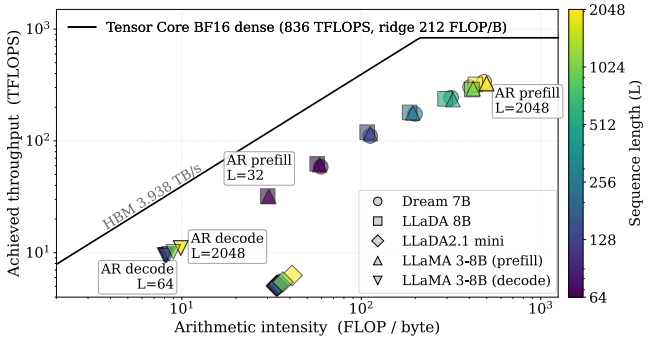

Figure 2: Model roofline across sequence lengths (L = input length + output length) on H100.

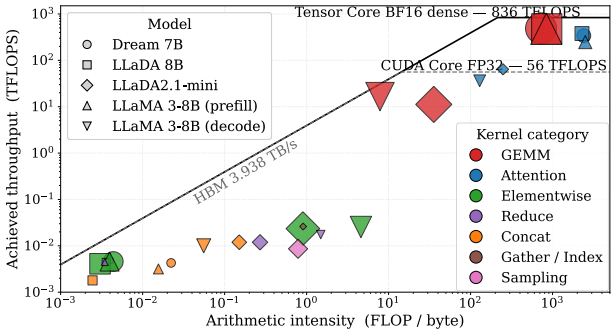

Figure 3: Per-kernel roofline on H100 for input and output length 1024. Marker size is proportional to total runtime.

4.1. Arithmetic Intensity and Roofline Analysis

Fig. 2 places each workload's model-level operating point of one forward pass on the H100 roofline, where achieved throughput (y-axis) is computed as the time-weighted average across all CUDA kernels: $(\sum_i FLOPs)/(\sum_i duration)$, and arithmetic intensity (AI) (x-axis) as $(\sum_i FLOPs)/(\sum_i DRAM\ bytes)$. Fig. 3 further decomposes one forward pass into kernel categories.

A single diffusion forward pass behaves as a compute-bound, prefill-like workload, with tensor-core throughput as the main bottleneck at long sequence length, instead of HBM bandwidth. As shown in Fig. 2, both AR prefill and pure diffusion workloads transit smoothly from memory-bound to compute-bound with increasing sequence length, reaching the roofline knee at 300–370 TFLOPS (AI $\approx$ 400–500 FLOP/byte) at $L = 2048$, though still below the peak theoretical roofline for reasons discussed later. AR decode, in contrast, remains bandwidth-limited regardless of sequence length, at AI 1–10 FLOP/byte and achieves around 10 TFLOPS. This shift is a direct consequence of the bidirectional, parallel nature of the denoising step. As sequence length grows, every additional token contributes to larger GEMM shapes in the linear and attention layers, such that both arithmetic intensity and achieved FLOPs rise roughly linearly until saturating near the throughput ceiling. Although the crossover L depends on model and hardware configuration, full-sequence diffusion avoids the sharp prefill/decode separation that AR systems must schedule around.

The arithmetic intensity of Block diffusion (LLaDA2.1-mini in Fig. 2) sits between these two regimes since each of its forward passes denoises only tokens within the active block while attending over the full prefix, making its per-step structure resemble a wider version of AR decode. Two effects pull its operating point further away from the roofline ceiling. First, restricting parallelism to a single block lowers achievable arithmetic intensity for the same reason AR decode is bandwidth-bound (at block rather than single-token granularity). This places block diffusion at a tunable interpolation point between AR and full diffusion. Second, its sparse MoE backbone yields smaller per-expert GEMMs that further reduce achievable FLOPs.

The gap to peak BF16 throughput is driven by the non-GEMM kernels, not by inefficiency in GEMM or attention. Fig. 3 shows that GEMM and attention kernels operate near their respective ceilings across all diffusion models, but a substantial fraction of runtime is consumed by low-AI elementwise operations, normalization, and masking-related operations that achieve orders-of-magnitude lower throughput against the FP32 CUDA-core peak, which pulls the model-level average down from the roofline ceiling.

4.2. Kernel-Level Execution Breakdown

To quantify which operations dominate execution time and where the largest deviations from AR inference occur, we decompose execution time into kernel categories. Fig. 4 shows per-kernel latency breakdown for a single forward pass of three diffusion models alongside LLaMA 3-8B in both prefill and decode phases at $L = 2048$ tokens, across eight kernel categories plus a small MISC bucket; absolute single-pass step time is annotated to the right of each bar.

Across all five workloads, execution is dominated by the same primary Transformer components: linear layer (GEMM) accounts for 41–61% of time, and attention contributes 6–15%, with the remainder distributed across memory-bound elementwise operations. This reinforces the roofline result. LLaDA 8B exhibits a kernel composition similar to LLaMA 3 prefill (57% GEMM, 10% attention versus 61% and 6%), whereas AR decode contains a slightly larger non-GEMM fraction driven primarily by KV-cache (copy) operations. We observe that the absolute execution time of a single LLaDA forward pass takes 108 ms, roughly 7$\times$ longer than a single LLaMA decode step, and must be repeated for every denoising iteration. However, the end-to-end execution time also depends on the total number of denoising steps T.

Reducing T increases throughput but may degrade generation quality. This quality–speed tradeoff is already characterized by the original LLaDA [38] and Dream [64] work, which sweep T against downstream accuracy and throughput, and find the comparison to be task-dependent and non-monotonic (e.g., LLaDA matches LLaMA3-8B's

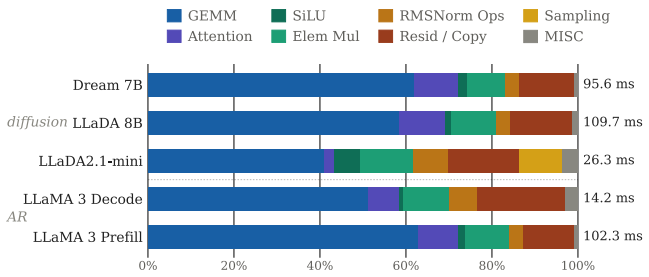

Figure 4: Single-pass kernel latency breakdown

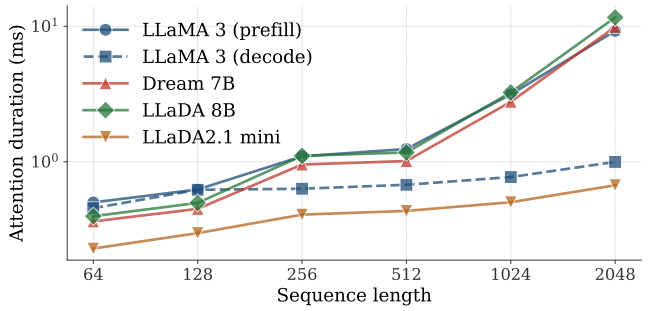

Figure 5: Attention latency scaling across Sequence Length

GSM8K/Math accuracy at 1.5–1.8× its throughput; on MBPP, however, LLaDA trails LLaMA3-8B's quality even at its slowest, most accurate setting). In contrast to this prior work, we treat T as fixed and instead isolate the per-step hardware mechanism.

Block diffusion exhibits a hybrid kernel composition. Relative to full-sequence diffusion, LLaDA2.1-mini shows a smaller GEMM fraction (38%) and larger Sampling (11.5%, top-k token commitment performed at the end of each block) and Resid/Copy (17%, materializing block outputs and concatenating them into the growing sequence) contributions, whereas full-sequence diffusion incurs neither overhead. **Takeaway: A single diffusion forward pass exhibits GEMM-dominant kernel latency composition similar to that of AR prefill at every denoising step, while non-GEMM kernels become a first-order bottleneck for block diffusion.**

4.3. Attention Scaling Analysis

Since attention is one of the primary sources of compute-behavior divergence between full-sequence diffusion and AR models, driven by their different masking structures, we study attention-kernel latency scaling separately. Fig. 5 plots attention-kernel latency per denoising (or decode) step across sequence lengths from L=64 to L=2048.

Diffusion attention scales quadratically while AR decode scales linearly. At shorter sequence lengths (< 256), all curves flatten substantially due to kernel launch overheads and FlashAttention tile quantization effects [42]. From 1024 to 2048 tokens, attention time increases by 3.6× for LLaDA 8B and Dream 7B and 3.5× for LLaMA prefill, approaching the expected 4× quadratic $O(L^2)$ scaling of full self-attention. In contrast, LLaMA decode increases by only 1.29×, consistent with a single-query decode kernel operating over a linearly growing KV cache with complexity of $O(L)$. LLaDA2.1-mini stays similarly flat via block diffusion. Notably, at L=2048, its per-step attention cost is around 17× lower than LLaDA 8B and Dream 7B, approaching LLaMA 3 decode. This confirms that block diffusion closes much of the attention-cost gap to KV-cached AR inference by parallel decoding only L_B tokens, while full diffusion gains the benefit of bidirectional context understanding at the cost of full-sequence computation. Its additional 1.5× speedup over LLaMA 3 comes from fewer attention layers, inherited from its AR backbone.

AR-inherited attention configurations can lose their effectiveness under diffusion. Dream 7B is consistently around 1.14× faster than LLaDA 8B across all sequence lengths due to having fewer Transformer layers (28 vs. 32). However, their attention latencies per invocation match closely (353 μs vs 363 μs), meaning Dream's inherited GQA provides no extra benefit, since GQA was originally introduced to reduce KV-cache bandwidth pressure during memory-bound AR decode [2]. Under full-sequence bidirectional attention, however, execution is compute-bound over a dense $L \times L$ score matrix (DRAM bandwidth utilization below 5% as shown in Section 5.2). Therefore, fewer KV heads does not reduce the FLOPs of the dominant $QK^\top$ or PV computations. **Takeaway: Full-sequence diffusion pays AR-prefill attention cost at every denoising step, and AR-inherited attention configurations such as GQA may lose their benefit.**

5. Memory Characterization

Although a single diffusion step exhibits kernel composition similar to AR prefill, the cumulative memory behavior over the entire inference process differs fundamentally. This section establishes the second structural shift: diffusion inference replaces KV-dominated memory growth with activation-dominated memory consumption under the original (uncached) implementation; whether this persists once diffusion-specific caching is introduced depends on the caching scheme (Section 5.4).

5.1. Memory Footprint Across Forward Passes

Fig. 6 shows memory consumption during the generation of 1024 output tokens. LLaMA 3-8B exhibits the canonical AR near-linear increase driven by KV-cache growth. The step near index 512 might reflect PyTorch's allocator promoting the cache to a larger contiguous arena once the previous reservation is exhausted. LLaDA 8B and Dream 7B remain nearly constant throughout generation, because bidirectional attention prevents exact KV-cache reuse across denoising steps. Consequently, **memory consumption of full diffusion depends only on sequence length (Section 5.3), not generation steps.**

LLaDA2.1-mini exhibits a staircase-shaped profile due to both its implementation and block diffusion architecture.

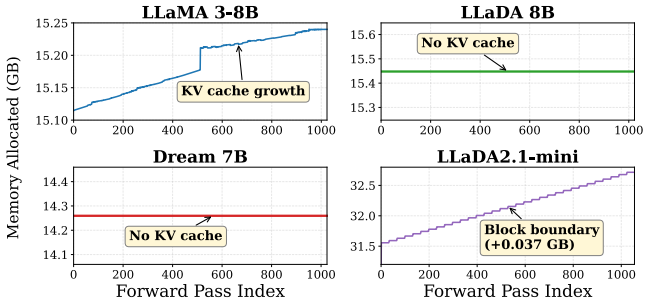

Figure 6: HBM capacity over inference steps. Although the y-axis variation appears modest at batch size (B) 1, these memory fluctuations scale linearly with B, whereas the model weight footprint remains constant; consequently, their impact becomes more significant at larger batch sizes.

Its block-causal attention mask lets tokens in the current block attend bidirectionally while attending causally to prior blocks, which in principle supports exact KV-cache reuse for committed blocks. However, its plain Hugging Face implementation recomputes them from scratch every step. This causes the token length used by the LM head to increase by L_B after each block. As shown in Fig. 6, flat regions reflect LLaDA-style recomputation within the active block, interleaved with discrete 0.037 GB jumps every 32–33 forward passes. To estimate the impact of enabling KV caching, we next analytically consider an optimized implementation. An implementation that exploits the block-causal structure by caching the KV states of committed blocks and recomputing only the current block would reduce the context-dependent memory footprint to $2 \times N_{layer} \times H_{kv} \times d \times T$ bytes (T: committed tokens), yielding an approximately 22× reduction for LLaDA2.1-mini.

5.2. Cache and Bandwidth Behavior

The roofline position in Section 4.1 does not reveal the bottleneck within the memory hierarchy or the off-chip memory bandwidth pressure. Fig. 7 therefore examines per-kernel L1, L2 hit rate and HBM bandwidth utilization by kernel category at $L = 2048$.

For GEMM and attention, full-sequence diffusion closely mirrors AR prefill, where GEMMs in LLaDA-8B and Dream-7B achieve 77% L2 hit rates and only 15–19% HBM bandwidth utilization, reflecting compute-bound execution with substantial weight reuse, in contrast to AR decode's ($< 10\%$) L2 reuse, which drives HBM bandwidth utilization above 57%. The same holds for the attention kernel at L1. AR prefill, LLaDA, and Dream sustain 22–24% L1 hit rates via intra-SM K/V reuse of FlashAttention, whereas LLaMA 3 decode and LLaDA2.1-mini achieve only 1% as KV tensors outgrow L1 capacity.

Elementwise kernels become major bandwidth consumers in diffusion inference. This represents a clear cache-hierarchy shift relative to AR decode, where elementwise operations consume negligible HBM bandwidth (0.1%) since single-token activations stay L1/L2-resident. In

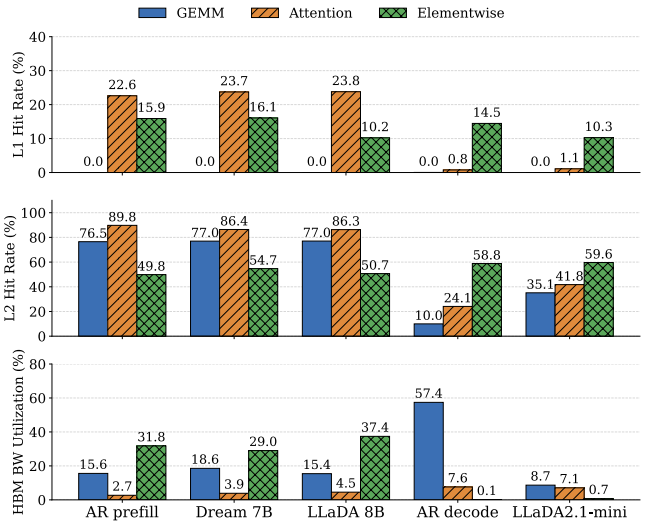

Figure 7: Per-kernel L1 cache hit rate (top), L2 cache hit rate (middle), and HBM bandwidth utilization (bottom).

LLaDA 8B and Dream 7B, however, elementwise kernels reach 29–37% HBM bandwidth utilization, higher than even GEMM kernels, as RMSNorm, residual adds, elementwise multiply, activations, and masking operations stream full-sequence tensor working sets exceeding L2 capacity. Therefore, **Operator Fusion** across these kernels becomes substantially more valuable under diffusion than AR decode.

LLaDA2.1-mini exhibits a behavior consistent with previous analysis. Its 35% GEMM L2 hit rate lies between full diffusion and AR-decode, since the 32-token denoising window ($L_B = 32$ for this experiment) is sufficient to partially amortize weight reuse, yet too short to sustain the strong cache locality observed in full-sequence diffusion. Operating on only 32 tokens each step also avoids the full-sequence activation streaming, thus elementwise HBM utilization drops below 1%, similar to AR decode. Its MoE architecture further suppresses GEMM HBM bandwidth pressure (lowest overall HBM utilization, 8.67%), since sparse expert activation reduces the effective GEMM working set. MoE is thus a particularly natural fit for block-diffusion inference. **Takeaway: Full-sequence diffusion preserves prefill-like locality in GEMM and attention. Elementwise kernels shift from being L2-resident in AR decode to being major HBM-bandwidth consumers.**

5.3. Peak Memory and Batch Scaling

To gain a finer-grained breakdown beyond the per-step cost, Fig. 8 decomposes peak HBM footprint during the whole inference process into model weights, KV cache, and activations, as L varies from 64 to 8192 tokens at batch size 1 (Dream terminates at $L = 2048$, its max context [64]).

Diffusion inverts the memory composition of AR inference. LLaMA shows the classic AR pattern where weights dominate at short L, KV cache grows linearly to roughly 1 GB at $L = 8192$, and activations remain negligible.

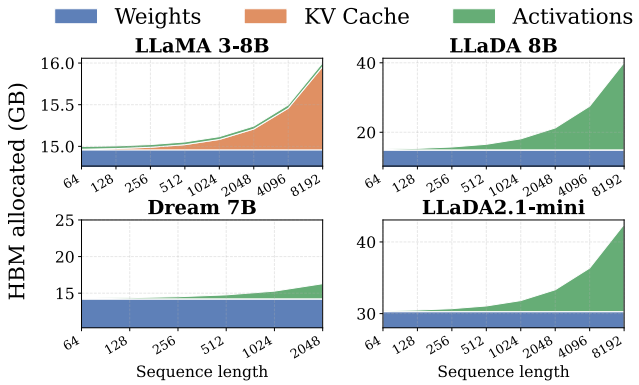

Figure 8: Peak memory breakdown across sequence lengths.

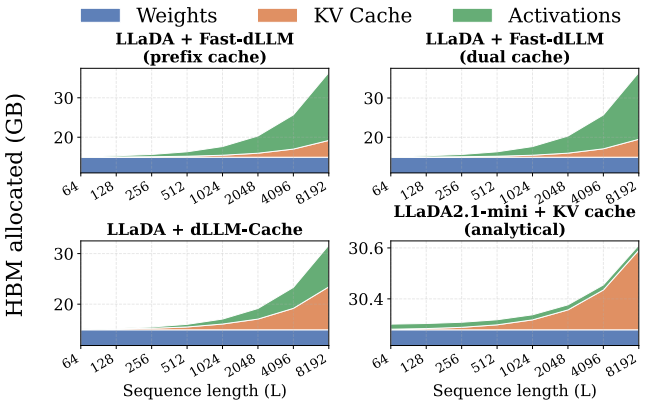

Figure 9: Memory composition under various Diffusion-Specific KV Caching techniques.

The diffusion models reverse this composition. For LLaDA, Dream, and LLaDA2.1-mini, KV-cache usage remains zero across the entire sweep, while activation memory scales linearly with L and quickly dominates HBM capacity. At $L = 8192$, LLaDA's activations alone reach 25 GB, exceeding the model's weight footprint and pushing total HBM usage above 40 GB. This shift also reduces the headroom for batching, since activation memory now scales with B × L rather than B alone in AR decode.

Dream 7B exhibits the same qualitative scaling as LLaDA 8B but uses approximately 3× less activation memory per token (1.03 vs. 3.15 MB/token at $L = 8192$), driven mainly by its inherited 7:1 GQA configuration (2.67× fewer K/V projection activations than LLaDA's symmetric 32:32 MHA). A shallower Transformer stack (28 vs. 32 layers) contributes an additional 1.14× reduction, compounding to the observed 3.05× gap. Thus, GQA remains valuable but for a different reason than its AR motivation. Its benefit transfers from KV-cache reduction to activation-memory reduction. **Takeaway: Under the standard (uncached) implementation characterized above, the variable component of peak memory in diffusion inference is activation memory rather than KV cache; AR-inherited optimizations such as GQA remain beneficial by reducing the activation footprint.**

5.4. Memory Composition under Diffusion-Specific KV Caching Mechanisms

The analysis above characterizes memory under the standard HuggingFace implementation; we next examine whether this composition persists once diffusion-specific caching is introduced.

MDLMs lack a single standardized caching strategy analogous to AR's exact KV cache. Recent methods instead either reuse the K/V tensors of previously committed positions, as in Fast-dLLM's block-wise approximate KV cache [59], or a broader set of intermediate features, as in dLLM-Cache's layer-wise feature cache [32]. For LLaDA 8B, we profile Fast-dLLM's prefix- and dual-cache modes and dLLM-Cache's feature cache, using the same HBM decomposition (Fig. 9). For block-diffusion such as LLaDA2.1-mini, its block-causal pattern natively supports cross-block KV reuse. However, no public caching implementation is currently available, so we retain an analytical decomposition, with KV cache footprint (bytes) derived from $M_{\text{KV}} = 2LN_{\text{layer}}H_{\text{kv}}d_{\text{head}}$ and activation memory from

$$M_{\text{act}} = B \times L_B \left(4V + 2\left[3H + N_h d + 2N_{\text{kv}} d + 2d^*_{\text{FFN}}\right]\right) \quad (1)$$

where B is the batch size, L_B the active block size, H the hidden dimension, N_h/N_{kv} the number of attention/KV heads, and d the attention head dimension. $4V$ is the FP32 logits buffer, and the remaining terms are BF16 intermediate activations from hidden states, attention projections, and FFN layers. The effective FFN intermediate dimension is

$$d^*_{\text{FFN}} = \max\left(d_{\text{dense}},\ (N_e + N_s) \times d_e\right) \quad (2)$$

where d_{dense} is the dense FFN intermediate dimension and N_e, N_s, d_e are the routed experts per token, shared experts, and per-expert intermediate dimension.

Fig. 9 shows that under Fast-dLLM, activation memory remains 3.7–5.5× larger than the KV cache for all L. Under dLLM-Cache, the cache footprint is comparable to or slightly larger than activations, since dLLM-Cache caches not only K/V tensors but also per-layer attention outputs and MLP outputs. At $L = 8192$, peak HBM usage drops from 40.1 GB in the uncached baseline to 36.4–36.5 GB under Fast-dLLM and 31.7 GB under dLLM-Cache, despite dLLM-Cache's cache (8.49 GB) being nearly twice as large as Fast-dLLM's (4.3–4.6 GB). This larger reduction correlates with the activation footprint eliminated: dLLM-Cache reduces activation memory to 8.25 GB, roughly half of Fast-dLLM's (17 GB), and this relationship holds across all evaluated L. LLaDA2.1-mini's analytical decomposition shows that with KV cache, it exhibits a hybrid pattern similar to AR, where KV cache scales linearly with L and activations stay constant with sequence length but scale near linearly with block size L_B. The KV cache also contributes substantially to total peak memory footprint reduction by reducing the activation memory.

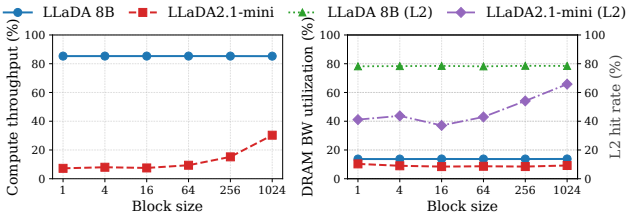

Figure 10: Effect of block size on compute throughput, DRAM bandwidth utilization, and L2 hit rate.

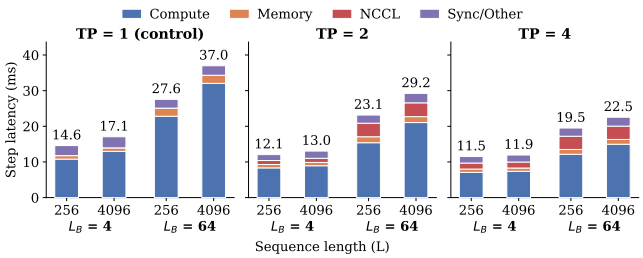

Figure 11: Communication scaling in tensor-parallel SDAR-30B inference across block size (L_B) and sequence length. Values above bars denote total step time.

Takeaway: whether diffusion-model inference remains activation-dominated after caching depends on the caching techniques and the types of tensors cached. KV-only caching largely preserves activation dominance, whereas caching broader features can make the cache comparable to or larger than the activations. However, reductions in peak memory consistently correlate with the amount of activation recomputation eliminated.

6. Block Size as a Hardware Parameter

Section 4 and Section 5 showed that block diffusion consistently exhibits behavior intermediate between AR decode and full-sequence diffusion. This section analyzes the effect of block size on hardware characteristics.

6.1. Block Size on Compute Characteristics

As shown in Fig. 10, compute-throughput rises monotonically from 7% at block size 1 to 30% at block size 1024, with L2 hit rate showing a similar trend, confirming that larger blocks shift execution more compute-bound by increasing intra-block parallelism. As a control, we also evaluate LLaDA 8B. Although it exposes a block-size hyperparameter, it remains architecturally a pure masked-diffusion model; its "block diffusion" corresponds to semi-autoregressive remasking layered on top of a full-sequence forward pass [38]. Consequently, its per-step compute and memory traffic are effectively independent of block size and remain nearly constant at approximately 85% and 14%, respectively. LLaDA2.1-mini also achieves substantially lower compute and DRAM utilization than dense LLaDA 8B due to its MoE architecture, where only a subset of experts is active per token. This sparsity reduces both tensor-core occupancy and sustained memory traffic. Therefore, DRAM throughput remains flat as block size increases since larger blocks improve GEMM efficiency within each active expert, increasing compute throughput, while expert weights are reused across more tokens (higher L2 hit rate), and therefore do not proportionally increase memory traffic.

Taken together with Sections 4 and 5, these results show that block size shapes compute intensity, bandwidth pressure, and activation scaling, extending its role beyond a purely algorithmic hyperparameter governing the balance between intra-block bidirectional modeling and inter-block autoregressive generation [3]. First, smaller L_B produces AR-like behavior where bandwidth-oriented optimizations remain effective, whereas larger L_B approaches full-sequence diffusion behavior dominated by compute throughput and activation-memory capacity. Second, MoE architectures appear particularly compatible with block diffusion, suggesting an additional architectural design space in which intermediate block sizes may better match sparse expert execution than either AR decode or full-sequence diffusion alone. Third, since block size and MoE structure jointly shape compute intensity, they also affect batching efficiency. Unlike full-sequence diffusion, where batching is ineffective due to saturated tensor-core throughput (Section 7.1), block diffusion with MoE may again benefit from request-level batching. Overall, **block size emerges as a system-level operating parameter that allows the block diffusion model to adapt its compute and memory behavior to the throughput, cache capacity, and bandwidth characteristics of the target accelerator, a flexibility unavailable to either pure AR or full-sequence diffusion alone.**

6.2. Tensor-Parallel under Different Block Sizes

The characterization above focuses on a single-GPU setting. We next examine whether block size remains an effective hardware tuning parameter when a model is distributed across multiple GPUs. We profile SDAR-30B-A3B [9], the largest model in our study, using tensor parallelism (TP) on NVLink-connected B200 GPUs, and compare TP=1, 2, and 4 for block lengths of 4 and 64 over sequence lengths of 256 and 4096 at batch size 16. Fig. 11 reports the per-step runtime breakdown into compute, memory, NVIDIA Collective Communications Library (NCCL) [43], and synchronization/other overhead.

NCCL communication time remains nearly constant across L for a fixed L_B, yet increases by 2.2–3.4× when L_B grows from 4 to 64, since the communication volume of all-reduced activation scales with the number of simultaneously denoised tokens ($\propto L_B$) and is independent of sequence length. Moving from TP=1 to TP=2 and TP=4 provides speedup in every configuration, but the effect of block size depends on TP degree. Larger blocks expose more computation to be parallelized. At the same time, the added

TABLE 2: STRUCTURAL VS. CONFIGURATION-DEPENDENT FINDINGS.

Finding	Category
Full-sequence diffusion resembles AR prefill	Structural
Activation- vs. KV-cache memory shift	Structural
Block diffusion spans the continuum between AR and full diffusion	Structural
Quadratic vs. linear attention scaling (full diffusion vs. AR decode)	Structural
Elementwise kernels runtime increase	Structural
Roofline crossover point/achieved TFLOPS	Config.-dep.
Magnitude of GQA's (lost) benefit	Config.-dep.
Magnitude of activation dominance after applying caching techniques	Config.-dep.
Magnitude of TP speedup vs. block size	Config.-dep.

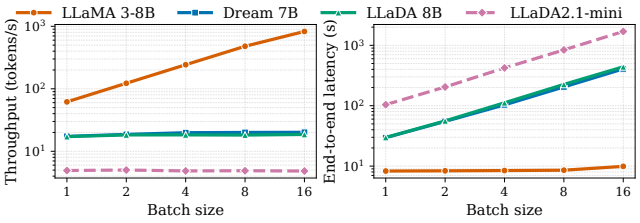

Figure 12: Throughput and latency scaling at L=1024.

communication cost relative to TP=1 (at $L_B = 64$) is nearly identical for TP=2 and TP=4 (3.8 and 3.7 ms), whereas the additional compute reduction is considerably larger for TP=4 (13 ms reduction vs. 8.5 ms for TP=2). Consequently, TP=4's speedup over TP=1 increases with block size (1.33× to 1.49×), whereas TP=2's remains essentially flat (1.25× to 1.20×). Thus, larger block sizes improve the scaling efficiency of higher TP degrees.

7. System Implications and Guidelines

The workload shifts characterized in Sections 4 to 6 reshape the optimization landscape for diffusion language model serving; this section translates those measurements into practical implications for serving-system design. We first distinguish findings that generalize across MDLM architectures and hardware platforms from those whose quantitative behavior depends on particular models, caching schemes, or hardware configurations. Table 2 summarizes this distinction. The remainder of this section derives guidelines primarily from the structural findings.

7.1. Transferability of AR Serving Optimizations

Modern AR serving systems such as vLLM [23], SGLang [68], and TensorRT-LLM [45] derive significant throughput from request-level batching mechanisms, including continuous batching [65], request multiplexing [1], and KV-cache-aware scheduling [23]. These mechanisms are effective precisely because AR decode is bandwidth-bound at small batch sizes, leaving substantial compute

TABLE 3: TRANSFERABILITY OF AR SERVING OPTIMIZATIONS TO DIFFUSION LMS.

Optimization	AR	Pure Diff.	Block Diff.
KV-cache opts	✓	*Partial*	*Partial*
Speculative decoding	✓	✗	✗
Static batching	*Partial*	✗	*Partial*
Continuous batching	✓	✗	*Partial*
Activation quant.	✓	✓	✓
Kernel fusion	✓	✓	✓
GQA	✓	✓	✓
MoE	✓	✓	✓

headroom to amortize across requests. For full-sequence diffusion however, Fig. 12 shows that when increasing batch size from 1 to 16, Dream and LLaDA exhibit near-linear latency scaling (13.8× and 14.8×, respectively) but only marginal throughput improvement (1.16× and 1.08×). And both models exceed 94 GB HBM capacity beyond $B = 16$, whereas LLaMA 3-8B continues achieving performance benefits for scaling batch size beyond $B = 16$.[2]

Apart from batching, speculative decoding [25] is effective in AR inference because it increases compute throughput by verifying multiple draft tokens per pass. Diffusion largely lacks this opportunity since each denoising step already processes an entire sequence/block in parallel.

The throughput-scaling axis exploited by modern AR serving systems therefore largely disappears under full-sequence diffusion. Instead, throughput improvements must come from reducing per-request work, either fewer denoising steps or lower per-step computation. Table 3 maps major optimizations in modern AR serving stacks onto diffusion paradigms based on the workload characteristics identified above. KV-centric optimizations [23, 33, 50] may lose their effectiveness under full-sequence diffusion, while block diffusion partially restores their applicability for committed blocks, albeit at a coarser granularity and a different attention pattern than AR decode. In contrast, optimizations such as kernel fusion [8, 19], GQA, and MoE, transfer directly to diffusion workloads, and techniques targeting activation-memory reduction, such as activation quantization [60], may also remain effective.

Continuous batching remains applicable to block diffusion but with reduced effectiveness relative to AR serving. In AR inference, requests can be inserted or evicted at token granularity, allowing schedulers to continuously refill freed slots and maintain high GPU utilization. Block diffusion instead generates an entire block through multiple joint denoising steps that cannot be interrupted mid-block. Request replacement is therefore only possible at block boundaries, leaving freed slots idle for up to an entire block's worth of computation. Consequently, the efficiency of continuous batching degrades as block size increases.

2. LLaDA2.1-mini's released implementation does not support batched generation; we therefore report sequential B=1 measurements (dashed line in Fig. 12) for completeness.

7.2. Emerging Optimization Targets

Activation-memory reduction. Per-step activations emerge as the dominant variable-memory cost under full-sequence diffusion, contributing to memory bandwidth and limiting per-GPU concurrency. Since each denoising step resembles a long-context prefill, techniques developed for it transfer naturally, including sequence-parallel attention [20, 31] and chunked execution [62]. Architectures that reduce K/Q/V projection size, such as aggressive GQA ratios, also remain well aligned with diffusion workloads even though their original AR motivation no longer applies.
Elementwise-kernel fusion. Since elementwise kernels become the largest DRAM-bandwidth consumers under full-sequence diffusion (Section 5.2), operator fusion across the elementwise kernels becomes substantially more valuable.

7.3. Hardware Architecture Implications

The workload shifts identified above also reveal mismatches between current GPU design and diffusion workloads, highlighting hardware–software co-design opportunities for future inference accelerators.

GEMM-shape implications. In full-sequence diffusion, the GEMM shape becomes large and square $L \times H \times H$, unlike the tall-skinny, bandwidth-bound GEMMs in AR decode ($1 \times H \times H$, GEMV-like). Weight-only quantization [30] thus provides zero or negative benefit since weights are reused sufficiently per fetch that bandwidth is no longer the bottleneck, while dequantization arithmetic can instead fall on the compute-critical path. In contrast, architectural features that provide higher-throughput tensor-core datapath, such as native low-precision support (e.g., FP4 on B200 [39]), remain well aligned with diffusion inference.

Activation-dominated memory hierarchy. Elementwise kernel working sets exceed L2 capacity under full-sequence diffusion, contributing significantly to HBM bandwidth consumption and per-step latency. Diffusion workloads would benefit from accelerators with substantially larger on-chip SRAM (e.g., Cerebras WSE-class designs [29]), as full diffusion activation memory footprints are largely predictable ahead of execution and remain constant during inference, unlike the dynamically growing KV cache of AR that must be managed during runtime.

Block size as a hardware–software co-design parameter. Prior work shows that the relationship between block size and generation quality is non-monotonic and shaped by several competing factors [3, 6, 52, 58]. Optimal block size is an open question investigated in the emerging block-diffusion work, while architecturally, the block-causal structure improves compatibility with existing AR-oriented serving infrastructure [6]. Our results show that varying L_B simultaneously shifts compute utilization, HBM bandwidth pressure, and multi-GPU communication cost, causing execution to move between AR-like and full-diffusion-like behavior. Consequently, block size exposes a three-way tradeoff among hardware efficiency, compatibility with AR serving systems, and generation quality, and creates a deployment-time co-design opportunity.

AR-Inherited Design Under Diffusion. Architectural choices inherited from AR models can behave differently under diffusion workloads (e.g., GQA and MoE), which indicates that, as diffusion LMs reuse AR backbones, inherited architecture design should be re-evaluated.

8. Related work

Hardware Characterization of Diffusion Language Models. Kim et al. [22] present a runtime/roofline analysis comparing LLaMA 3 and LLaDA, establishing that diffusion inference exhibits higher arithmetic intensity than AR decode. Peng et al. [46] provide a throughput evaluation and examine efficiency-evaluation practices in the diffusion LM literature, both at the model or throughput level. This paper provides a systematic kernel-level and memory-hierarchy characterization of MDLM inference, covering a variety of MDLMs, and establishes our findings at the hardware level.

Step-count reduction and scheduling. A parallel line of work exploits the quality–speed trade-off exposed by denoising step count T, via adaptive parallel decoding [18], early-commit decoding [26], non-uniform step scheduling [57], and layer-/token-level skipping [24, 70].

Optimizations via approximate caching. Recent work addresses DLM inference inefficiency via approximate caching, such as Fast-dLLM [58, 59], dLLM-Cache [32] and d^2Cache [21] that exploit inter-step feature similarity to reduce redundant computation. Sparse-dLLM [53] explores dynamic cache eviction. FlashDLM [17] combines KV caching with AR-guided unmasking. Mosaic [67] reduces activations through mask-only logits, lazy chunking, and global memory planning.

9. Conclusions

This paper presented a hardware-level characterization of masked diffusion language model inference. We find that although MDLMs reuse the Transformer building blocks of AR models, their inference behavior differs fundamentally at the hardware level. Full-sequence diffusion shifts the bottleneck from HBM bandwidth and KV-cache growth to dense compute and activation memory, though the latter's magnitude depends on the caching scheme applied. Block diffusion spans a continuum between AR-style and full-sequence-diffusion behavior, with block size as a hardware operating parameter that allows a single model to adapt to an accelerator's compute, cache, and bandwidth characteristics. As diffusion language models continue to scale, their different compute, memory, and scheduling characteristics may require rethinking both serving systems and future accelerator design.

Acknowledgments

This work was supported in part by PRISM, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, and by NSF grant no. PPoSS-2217071. We also thank the reviewers for their helpful feedback.

References

[1] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, A. Tumanov, and R. Ramjee, "Taming throughput-latency tradeoff in LLM inference with sarathi-serve," in *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'24. USA: USENIX Association, Jul. 2024, pp. 117–134.

[2] J. Ainslie, J. Lee-Thorp, M. d. Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints," Dec. 2023, arXiv:2305.13245 [cs]. [Online]. Available: http://arxiv.org/abs/2305.13245

[3] M. Arriola, A. Gokaslan, J. T. Chiu, Z. Yang, Z. Qi, J. Han, S. S. Sahoo, and V. Kuleshov, "Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models," May 2025, arXiv:2503.09573 [cs]. [Online]. Available: http://arxiv.org/abs/2503.09573

[4] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. van den Berg, "Structured denoising diffusion models in discrete state-spaces," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, ser. NIPS '21. Red Hook, NY, USA: Curran Associates Inc., Dec. 2021, pp. 17 981–17 993.

[5] T. Bie, M. Cao, X. Cao, B. Chen, F. Chen, K. Chen, L. Du, D. Feng, H. Feng, M. Gong, Z. Gong, Y. Gu, J. Guan, K. Guan, H. He, Z. Huang, J. Jiang, Z. Jiang, Z. Lan, C. Li, J. Li, Z. Li, H. Liu, L. Liu, G. Lu, Y. Lu, Y. Ma, X. Mou, Z. Pan, K. Qiu, Y. Ren, J. Tan, Y. Tian, Z. Wang, L. Wei, T. Wu, Y. Xing, W. Ye, L. Zha, T. Zhang, X. Zhang, J. Zhao, D. Zheng, H. Zhong, W. Zhong, J. Zhou, J. Zhou, L. Zhu, M. Zhu, and Y. Zhuang, "LLaDA2.1: Speeding Up Text Diffusion via Token Editing," Feb. 2026, arXiv:2602.08676 [cs]. [Online]. Available: http://arxiv.org/abs/2602.08676

[6] T. Bie, M. Cao, K. Chen, L. Du, M. Gong, Z. Gong, Y. Gu, J. Hu, Z. Huang, Z. Lan, C. Li, C. Li, J. Li, Z. Li, H. Liu, L. Liu, G. Lu, X. Lu, Y. Ma, J. Tan, L. Wei, J.-R. Wen, Y. Xing, X. Zhang, J. Zhao, D. Zheng, J. Zhou, J. Zhou, Z. Zhou, L. Zhu, and Y. Zhuang, "LLaDA2.0: Scaling Up Diffusion Language Models to 100B," Dec. 2025, arXiv:2512.15745 [cs]. [Online]. Available: http://arxiv.org/abs/2512.15745

[7] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating Large Language Models Trained on Code," Jul. 2021, arXiv:2107.03374 [cs.LG]. [Online]. Available: http://arxiv.org/abs/2107.03374

[8] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: an automated end-to-end optimizing compiler for deep learning," in *Proceedings of the 13th USENIX conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, Oct. 2018, pp. 579–594.

[9] S. Cheng, Y. Bian, D. Liu, L. Zhang, Q. Yao, Z. Tian, W. Wang, Q. Guo, K. Chen, B. Qi, and B. Zhou, "SDAR: A Synergistic Diffusion-AutoRegression Paradigm for Scalable Sequence Generation," Oct. 2025, arXiv:2510.06303 [cs.LG]. [Online]. Available: http://arxiv.org/abs/2510.06303

[10] T. Dao, "FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning," Jul. 2023, arXiv:2307.08691 [cs.LG]. [Online]. Available: http://arxiv.org/abs/2307.08691

[11] DeepSeek-AI, "DeepSeek LLM: Scaling Open-Source Language Models with Longtermism," Jan. 2024, arXiv:2401.02954 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2401.02954

[12] Google DeepMind, "Gemini diffusion," https://deepmind.google/models/gemini-diffusion/, 2025, accessed: 2026-05-18.

[13] X. Han, S. Kumar, and Y. Tsvetkov, "SSD-LM: Semi-autoregressive Simplex-based Diffusion Language Model for Text Generation and Modular Control," Jun. 2023, arXiv:2210.17432 [cs]. [Online]. Available: http://arxiv.org/abs/2210.17432

[14] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring Massive Multitask Language Understanding," Jan. 2021, arXiv:2009.03300 [cs.CY]. [Online]. Available: http://arxiv.org/abs/2009.03300

[15] M. Hersche, S. Moor-Smith, T. Hofmann, and A. Rahimi, "Soft-Masked Diffusion Language Models," Mar. 2026, arXiv:2510.17206 [cs.LG]. [Online]. Available: http://arxiv.org/abs/2510.17206

[16] E. Hoogeboom, D. Nielsen, P. Jaini, P. Forré, and M. Welling, "Argmax flows and multinomial diffusion: learning categorical distributions," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, ser. NIPS '21. Red Hook, NY, USA: Curran Associates Inc., Dec. 2021, pp. 12 454–12 465.

[17] Z. Hu, J. Meng, Y. Akhauri, M. S. Abdelfattah, J.-s. Seo, Z. Zhang, and U. Gupta, "FlashDLM: Accelerating Diffusion Language Model Inference via Efficient KV Caching and Guided Diffusion," May 2025. [Online]. Available: https://arxiv.org/abs/2505.21467v2

[18] D. Israel, G. V. d. Broeck, and A. Grover, "Accelerating Diffusion LLMs via Adaptive Parallel Decoding," Oct. 2025, arXiv:2506.00413 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2506.00413

[19] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler, "Data Movement Is All You Need: A Case Study on Optimizing Transformers," Nov. 2021, arXiv:2007.00072 [cs.LG]. [Online]. Available: http://arxiv.org/abs/2007.00072

[20] S. A. Jacobs, M. Tanaka, C. Zhang, M. Zhang, R. Y. Aminadabi, S. L. Song, S. Rajbhandari, and Y. He, "System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models," in *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, ser. PODC '24. New York, NY, USA: Association for Computing Machinery, Jun. 2024, pp. 121–130. [Online]. Available: https://dl.acm.org/doi/10.1145/3662158.3662806

[21] Y. Jiang, Y. Cai, X. Luo, J. Fu, J. Wang, C. Liu, and X. Yang, "d^2cache: Accelerating Diffusion-Based LLMs via Dual Adaptive Caching," Feb. 2026, arXiv:2509.23094 [cs]. [Online]. Available: http://arxiv.org/abs/2509.23094

[22] M. Kim, C. Hooper, A. Tomar, C. Xu, M. Farajtabar, M. W. Mahoney, K. Keutzer, and A. Gholami, "Beyond Next-Token Prediction: A Performance Characterization of Diffusion versus Autoregressive Language Models," Oct. 2025. [Online]. Available: https://arxiv.org/abs/2510.04146v2

[23] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient Memory Management for Large Language Model Serving with PagedAttention," Sep. 2023, arXiv:2309.06180 [cs.LG]. [Online]. Available: http://arxiv.org/abs/2309.06180

[24] Y. Lee, J. Lee, S. Dan, J. Park, and J. H. Ahn, "DyLLM: Efficient Diffusion LLM Inference via Saliency-based Token Selection and Partial Attention," Mar. 2026, arXiv:2603.08026 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2603.08026

[25] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23, vol. 202. Honolulu, Hawaii, USA: JMLR.org, Jul. 2023, pp. 19 274–19 286.

[26] P. Li, Y. Zhou, D. Muhtar, L. Yin, S. Yan, L. Shen, S. Vosoughi, and S. Liu, "Diffusion Language Models Know the Answer Before Decoding," Apr. 2026, arXiv:2508.19982 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2508.19982

[27] T. Li, M. Chen, B. Guo, and Z. Shen, "A Survey on Diffusion Language Models," Dec. 2025, arXiv:2508.10875 [cs]. [Online]. Available: http://arxiv.org/abs/2508.10875

[28] X. L. Li, J. Thickstun, I. Gulrajani, P. Liang, and T. B. Hashimoto, "Diffusion-LM improves controllable text generation," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., Nov. 2022, pp. 4328–4343.

[29] S. Lie, "Wafer-scale AI: GPU impossible performance," in *2024 IEEE Hot Chips 36 Symposium (HCS)*, Cupertino, CA, USA, Aug. 2024, cerebras Systems CTO presentation. WSE-3 specifications: 44 GB on-chip SRAM, 21 PB/s on-chip memory bandwidth, 900,000 cores, 125 PetaFLOPS BF16. [Online]. Available: https://hc2024.hotchips.org/assets/program/conference/day2/72_HC2024.Cerebras.Sean.v03.final.pdf

[30] J. Lin, J. Tang, H. Tang, S. Yang, G. Xiao, and S. Han, "AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration," *GetMobile: Mobile Computing and Communications*, vol. 28, no. 4, pp. 12–17, Jan. 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3714983.3714987

[31] H. Liu, M. Zaharia, and P. Abbeel, "Ring Attention with Blockwise Transformers for Near-Infinite Context," Nov. 2023, arXiv:2310.01889 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2310.01889

[32] Z. Liu, Y. Yang, Y. Zhang, J. Chen, C. Zou, Q. Wei, S. Wang, and L. Zhang, "dLLM-Cache: Accelerating Diffusion Large Language Models with Adaptive Caching," May 2025, arXiv:2506.06295 [cs]. [Online]. Available: http://arxiv.org/abs/2506.06295

[33] Z. Liu, J. Yuan, H. Jin, S. H. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu, "KIVI: a tuning-free asymmetric 2bit quantization for KV cache," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24, vol. 235. Vienna, Austria: JMLR.org, Jul. 2024, pp. 32 332–32 344.

[34] A. Lou, C. Meng, and S. Ermon, "Discrete diffusion modeling by estimating the ratios of the data distribution," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24, vol. 235. Vienna, Austria: JMLR.org, Jul. 2024, pp. 32 819–32 848.

[35] Y. Ma, L. Du, L. Wei, K. Chen, Q. Xu, K. Wang, G. Feng, G. Lu, L. Liu, X. Qi, X. Zhang, Z. Tao, H. Feng, Z. Jiang, Y. Xu, Z. Huang, Y. Zhuang, H. Xu, J. Hu, Z. Lan, J. Zhao, J. Li, and D. Zheng, "dInfer: An Efficient Inference Framework for Diffusion Language Models," Oct. 2025, arXiv:2510.08666 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2510.08666

[36] C. Meng, K. Choi, J. Song, and S. Ermon, "Concrete score matching: generalized score matching for discrete data," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., Nov. 2022, pp. 34 532–34 545.

[37] Meta AI, "The Llama 3 Herd of Models," Nov. 2024, arXiv:2407.21783 [cs]. [Online]. Available: http://arxiv.org/abs/2407.21783

[38] S. Nie, F. Zhu, Z. You, X. Zhang, J. Ou, J. Hu, J. Zhou, Y. Lin, J.-R. Wen, and C. Li, "Large Language Diffusion Models," Oct. 2025, arXiv:2502.09992 [cs]. [Online]. Available: http://arxiv.org/abs/2502.09992

[39] NVIDIA Corp., "NVIDIA Blackwell Architecture," https://resources.nvidia.com/en-us-blackwell-architecture, 2024, accessed: 2026-05-19.

[40] NVIDIA Corporation, "NVIDIA H100 Tensor Core GPU," 2026. [Online]. Available: https://www.nvidia.com/en-us/data-center/h100/

[41] NVIDIA CUDA Team, "Nvidia cublas documentation," https://docs.nvidia.com/cuda/cublas/, 2026, accessed: 2026-05-17.

[42] NVIDIA Deep Learning Performance Team, "Matrix multiplication background user's guide," https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html, 2023, accessed: 2026-05-17.

[43] NVIDIA NCCL Team, "Nvidia collective communications library (nccl)," https://developer.nvidia.com/nccl, 2026.

[44] NVIDIA Nsight Compute Team, "NVIDIA Nsight Compute," 2026. [Online]. Available: https://developer.nvidia.com/nsight-compute

[45] NVIDIA TensorRT-LLM Team, "Tensorrt-llm," https://github.com/NVIDIA/TensorRT-LLM, 2023, accessed: 2026-05-19.

[46] H. Peng, P. Liu, Z. Dong, D. Cheng, J. Li, Y. Tang, S. Wang, and W. X. Zhao, "How Efficient Are Diffusion Language Models? A Critical Examination of Efficiency Evaluation Practices," Nov. 2025, arXiv:2510.18480 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2510.18480

[47] Qwen, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu, "Qwen2.5 Technical Report," Jan. 2025, arXiv:2412.15115 [cs]. [Online]. Available: http://arxiv.org/abs/2412.15115

[48] S. S. Sahoo, M. Arriola, Y. Schiff, A. Gokaslan, E. Marroquin, J. T. Chiu, A. Rush, and V. Kuleshov, "Simple and effective masked diffusion language models," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, ser. NIPS '24, vol. 37. Red Hook, NY, USA: Curran Associates Inc., Dec. 2024, pp. 130 136–130 184.

[49] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," Jan. 2017, arXiv:1701.06538 [cs]. [Online]. Available: http://arxiv.org/abs/1701.06538

[50] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "FlexGen: high-throughput generative inference of large language models with a single GPU," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23, vol. 202. Honolulu, Hawaii, USA: JMLR.org, Jul. 2023, pp. 31 094–31 116.

[51] J. Shi, K. Han, Z. Wang, A. Doucet, and M. K. Titsias, "Simplified and generalized masked diffusion for discrete data," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, ser. NIPS '24, vol. 37. Red Hook, NY, USA: Curran Associates Inc., Dec. 2024, pp. 103 131–103 167.

[52] Y. Shu, Y. Tian, C. Xu, Y. Wang, and H. Chen, "Deferred Commitment Decoding for Diffusion Language Models," Jan. 2026. [Online]. Available: https://arxiv.org/abs/2601.02076v2

[53] Y. Song, X. Liu, R. Li, Z. Liu, Z. Huang, Q. Guo, Z. He, and X. Qiu, "Sparse-dLLM: Accelerating Diffusion LLMs with Dynamic Cache Eviction," Nov. 2025, arXiv:2508.02558 [cs]. [Online]. Available: http://arxiv.org/abs/2508.02558

[54] R. Strudel, C. Tallec, F. Altché, Y. Du, Y. Ganin, A. Mensch, W. Grathwohl, N. Savinov, S. Dieleman, L. Sifre, and R. Leblond, "Self-conditioned Embedding Diffusion for Text Generation," Nov. 2022, arXiv:2211.04236 [cs]. [Online]. Available: http://arxiv.org/abs/2211.04236

[55] L. Team, A. Li, B. Liu, B. Hu, B. Li, B. Zeng, B. Ye, C. Tang, C. Tian, C. Huang, C. Zhang, C. Qian, C. Ju, C. Li, C. Tang, C. Fu, C. Ren, C. Wu, C. Zhang, C. Peng, D. Xu, D. Wang, D. Zhang, D. Jin, D. Zhu, D. Hu, F. Zhao, F. Wu, F. Zhu, G. Wang, H. Zhang, H. Zhao, H. Zhang, H. Wang, H. Qian, H. Yu, H. Zhang, H. Zhang, H. Luan, H. Dong, H. Li, J. Li, J. Liu, J. Zhu, J. Sha, J. Wei, J. Yang, J. Ma, J. Wu, J. Huang, J. Tian, J. Zhang, J. Sun, J. Tu, J. Liu, J. Xu, J. Zhou, J. Ou, J. Fang, K. Zhang, K. Hu, K. Shi, K. Tang, K. Chen, L. Mei, L. Liang, L. Xu, L. Zhang, L. Ju, L. Yuan, L. Zhong, L. Ma, L. Liu, L. Yu, L. Cai, M. Zhu, M. Li, M. Chen, M. Xue, M. Cai, M. Yin, P. Jiang, P. Zhao, P. Liu, Q. Zhao, Q. Cui, Q. Huang, Q. Yang, Q. Yu, S. Wei, S. Lian, S. Zheng, S. Song, S. Zhang, S. Zhang, S. Li, S. Liu, T. Guo, T. Zhao, W. Gu, W. Wu, W. Han, W. Fang, W. Wang, X. Shu, X. Shi, X. Lan, X. Zhang, X. Sun, X. Zhao, X. Lu, X. Xu, X. Wang, X. Yang, Y. Yang, Y. Xiang, Y. Li, Y. Zhang, Y. Wang, Y. Li, Y. Guo, Y. Fu, Y. Wang, Y. Yang, Y. Yu, Y. Deng, Y. Zhang, Y. Yu, Y. Zhang, Y. He, Z. Gui, Z. Huan, Z. Wang, Z. Zhu, Z. Wang, Z. Zhang, Z. Wang, Z. Zeng, Z. Liu, Z. Xuan, and Z. Tang, "Every Activation Boosted: Scaling General Reasoner to 1 Trillion Open Language Foundation," Oct. 2025. [Online]. Available: https://arxiv.org/abs/2510.22115v2

[56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., Dec. 2017, pp. 6000–6010. [Online]. Available: https://dl.acm.org/doi/10.5555/3295222.3295349

[57] Q. Wei, Y. Zhang, Z. Liu, P. Zeng, Y. Wang, B. Qi, D. Liu, and L. Zhang, "Accelerating Diffusion Large Language Models with SlowFast Sampling: The Three Golden Principles," Mar. 2026, arXiv:2506.10848 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2506.10848

[58] C. Wu, H. Zhang, S. Xue, S. Diao, Y. Fu, Z. Liu, P. Molchanov, P. Luo, S. Han, and E. Xie, "Fast-dLLM v2: Efficient Block-Diffusion LLM," Sep. 2025, arXiv:2509.26328 [cs]. [Online]. Available: http://arxiv.org/abs/2509.26328

[59] C. Wu, H. Zhang, S. Xue, Z. Liu, S. Diao, L. Zhu, P. Luo, S. Han, and E. Xie, "Fast-dLLM: Training-free Acceleration of Diffusion LLM by Enabling KV Cache and Parallel Decoding," Jul. 2025, arXiv:2505.22618 [cs]. [Online]. Available: http://arxiv.org/abs/2505.22618

[60] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: accurate and efficient post-training quantization for large language models," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23, vol. 202. Honolulu, Hawaii, USA: JMLR.org, Jul. 2023, pp. 38 087–38 099.

[61] Z. Xiong, Y. Cai, Z. Li, and Y. Wang, "Unveiling the Potential of Diffusion Large Language Model in Controllable Generation," Sep. 2025, arXiv:2507.04504 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2507.04504

[62] A. Yang, J. Yang, A. Ibrahim, X. Xie, B. Tang, G. Sizov, J. Reizenstein, J. Park, and J. Huang, "Context Parallelism for Scalable Million-Token Inference," Apr. 2025, arXiv:2411.01783 [cs.DC]. [Online]. Available: http://arxiv.org/abs/2411.01783

[63] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu, "Qwen3 Technical Report," May 2025, arXiv:2505.09388 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2505.09388

[64] J. Ye, Z. Xie, L. Zheng, J. Gao, Z. Wu, X. Jiang, Z. Li, and L. Kong, "Dream 7B: Diffusion Large Language Models," Aug. 2025, arXiv:2508.15487 [cs]. [Online]. Available: http://arxiv.org/abs/2508.15487

[65] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A Distributed Serving System for Transformer-Based Generative Models," 2022, pp. 521–538. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/yu

[66] R. Yu, Q. Li, and X. Wang, "Discrete Diffusion in Large Language and Multimodal Models: A Survey," Sep. 2025, arXiv:2506.13759 [cs]. [Online]. Available: http://arxiv.org/abs/2506.13759

[67] L. Zheng, B. Shi, Y. Hu, J. Zhang, R. Li, S. Chen, W. Li, and K. Li, "Mosaic: Unlocking Long-Context Inference for Diffusion LLMs via Global Memory Planning and Dynamic Peak Taming," Jan. 2026, arXiv:2601.06562 [cs.LG]. [Online]. Available: http://arxiv.org/abs/2601.06562

[68] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng, "SGLang: Efficient Execution of Structured Language Model Programs," Jun. 2024, arXiv:2312.07104 [cs.AI]. [Online]. Available: http://arxiv.org/abs/2312.07104

[69] F. Zhu, Z. You, Y. Xing, Z. Huang, L. Liu, Y. Zhuang, G. Lu, K. Wang, X. Wang, L. Wei, H. Guo, J. Hu, W. Ye, T. Chen, C. Li, C. Tang, H. Feng, J. Hu, J. Zhou, X. Zhang, Z. Lan, J. Zhao, D. Zheng, C. Li, J. Li, and J.-R. Wen, "LLaDA-MoE: A Sparse MoE Diffusion Language Model," Sep. 2025, arXiv:2509.24389 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2509.24389

[70] Z. Zhu, F. Ren, Z. Tan, and K. Ma, "ES-dLLM: Efficient Inference for Diffusion Large Language Models by Early-Skipping," Mar. 2026. [Online]. Available: https://arxiv.org/abs/2603.10088v1

Appendix

1. Abstract

This artifact contains the profiling scripts, post-processing pipelines, and plotting scripts used to produce all measurements in this paper. It covers kernel-level GPU profiling (NVIDIA Nsight Compute) and system-level memory profiling (PyTorch allocator). The artifact also includes the aggregated intermediate CSVs, so every figure in the paper can be regenerated from data without requiring GPU access. This artifact corresponds to the *Available* badge.

2. Artifact check-list (meta-information)

- **Program:** Python profiling scripts wrapping PyTorch model forward passes, driven either directly (PyTorch allocator) or under NVIDIA Nsight Compute (`ncu`).
- **Model:** LLaMA 3-8B, LLaDA-8B, Dream-7B, LLaDA2.1-mini, and SDAR-30B-A3B.
- **Run-time environment:** Linux, CUDA 12.8, Python 3.12.12; see `environment.yml`/`requirements.txt`.
- **Metrics:** GPU kernel duration, achieved TFLOPS, arithmetic intensity, L1/L2 cache hit rate, DRAM/HBM bandwidth utilization, HBM memory allocated, end-to-end throughput/latency.
- **Output:** CSV files and PDF/PNG figures.
- **Experiments:** One experiment class per figure; see Section below.
- **How much disk space required (approximately)?:** Fully regenerating raw Nsight Compute reports from scratch would require tens of GB of scratch space per model sweep.
- **How much time is needed to prepare workflow (approximately)?:** 15–30 minutes (environment setup, checkpoint download).
- **How much time is needed to complete experiments (approximately)?:** Regenerating all figures from the shipped aggregated CSVs: minutes. Regenerating the underlying profiling data from scratch on H100 hardware: on the order of several GPU-hours across all sweeps.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** MIT License.
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.21839997

3. Description

3.1. How to access. The artifact is archived on Zenodo at https://doi.org/10.5281/zenodo.21839997. Download and unzip the archive; it contains the full directory structure described in the artifact's `README.md`, including a figure/table-to-code mapping table.

3.2. Hardware dependencies. Single-GPU experiments (kernel-level roofline/memory profiling, block-size sweeps, end-to-end throughput) require 1× NVIDIA H100 94GB GPU. The tensor-parallel communication study (Fig. 11) requires 4× NVLink-connected NVIDIA B200 GPUs. Regenerating figures from the aggregated CSVs already included in the artifact requires no GPU.

3.3. Software dependencies. Linux, CUDA 12.8, NVIDIA Nsight Compute 2025.1.1 (`ncu`) for kernel-level profiling, and Python 3.12.12. The full dependency list is provided as both `environment.yml` (conda) and `requirements.txt` (pip). `flash-attn` is pinned to a prebuilt wheel for this specific CUDA/torch/Python ABI; substitute a matching wheel or build from source if your environment differs. The vendored `sdar_tp/JetEngine` inference engine is not pip-installable from the requirements file and must be installed separately: `pip install -e sdar_tp/JetEngine`.

3.4. Data sets. No standalone data sets are used; all measurements are collected directly from model forward passes on synthetic (randomly generated or fixed-length) input sequences of varying length, as described in the paper's Section 3 (Experimental Setup and Methodology).

3.5. Models. LLaMA 3-8B, LLaDA-8B, Dream-7B, and LLaDA2.1-mini checkpoints will be pulled from the Hugging Face Hub on first run. SDAR-30B-A3B requires downloading two separate checkpoints corresponding to its block-size-4 and block-size-64 training variants.

4. Installation

```
conda env create -f environment.yml
# or: pip install -r requirements.txt
conda activate diffusion_model
pip install -e sdar_tp/JetEngine
export HF_HOME=/path/to/your/hf/cache
```

5. Experiment workflow

1) *(GPU-free) Regenerate all figures from shipped data:* run the plotting script in each of `*_plot/` directory, each reads aggregated CSVs and produces a PDF/PNG matching the corresponding paper figure (see the figure-to-code mapping table in the artifact's `README.md`).

6. Evaluation and expected results

Running each plotting script in step 1 above should produce a PDF/PNG visually matching the corresponding numbered figure in the paper (kernel category proportions, roofline operating points, memory-composition trends, attention-latency scaling, etc.). Absolute latency/throughput numbers regenerated from scratch on different GPU hardware, driver/CUDA versions, or library versions than those listed in `environment.yml` are expected to differ in magnitude while preserving the qualitative trends.

2) *(1 GPU required) Regenerate the profiling data:* run the NCU-wrapped scripts in `nsight_profiling/` (kernel-level) and the PyTorch-allocator harnesses in `pytorch_memory/` and `pytorch_e2e/` (memory/throughput), then the post-processing scripts in `nsight_profiling/post_processing_scripts/` to regenerate the aggregated CSVs.
3) *(4 GPUs required) Tensor-parallel study:* run `sdar_tp/run_tp1.slurm` and `run_tp2.slurm` (or the `accelerate launch` commands directly), then `sdar_tp/plot_breakdown.py` to reproduce Fig. 11.

7. Experiment customization

Each profile script exposes command-line flags for sequence length, batch size, block size, and number of denoising steps, allowing the sweeps in Figs. 2–12 to be extended to other configurations or models beyond the four/five studied in the paper. The kernel categorization rules (GEMM/Attention/Elementwise/etc.) used by the post-processing scripts in `nsight_profiling/` can be edited directly in those scripts to explore alternative groupings.

8. Notes

Raw `.ncu-rep` Nsight Compute reports and raw per-invocation "sections" CSVs (the intermediate output of `ncu --import` before aggregation) are not included in this artifact, as individual reports range from tens of MB to low-GB; only the small, already-aggregated CSVs that the plotting scripts consume are shipped. These can be regenerated via `nsight_profiling/run_ncu_profile.slurm` → `run_ncu_postprocess.slurm`, or an equivalent raw directory can be pointed to via the `NCU_SECTIONS_DIR` environment variable. SLURM batch scripts included in the artifact were written for the authors' university HPC cluster; adapt the `#SBATCH` directives to your own cluster, or invoke the underlying `python` commands directly on any machine with matching GPU hardware.

9. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- https://cTuning.org/ae