Understanding and Exploiting Diagonal Attention Sparsity in Autoregressive Image Generation

Daeun Kim[1], Junwha Hong[2], Changhun Oh[1], Yoonsung Kim[1], Yoonhyeong Lee[3], and Jongse Park[1]

[1]*KAIST*
[2]*Agency for Defense Development*
[3]*Seoul National University*

Abstract

Autoregressive image generation has emerged as a paradigm for multimodal AI systems due to its compatibility with transformer-based LLM serving infrastructures. However, generating thousands of visual tokens per request makes decoding increasingly bottlenecked by KV cache accesses during attention computation. Sparse attention is particularly attractive for this workload because many visual generation applications tolerate moderate quality degradation in exchange for improved performance and efficiency. While sparse attention has been extensively explored for text-based LLM inference, it remains unclear whether its sparsity assumptions generalize effectively to autoregressive image generation.

We present the first systematic characterization of attention sparsity in autoregressive image generation across diverse workloads and representative open-source models. Our analysis reveals several distinguishing properties, including a pronounced prefill-decode asymmetry, strong attention concentration on prompt and local tokens, and a unique diagonal attention sparsity pattern arising from the spatial locality of visual tokens. Motivated by these observations, we propose a diagonal-aware sparse attention mechanism that selectively skips KV entries along the diagonal attention direction within a recent window. Implemented on top of a GPU-based serving system using FlexGen, FlashAttention-2, and custom kernels, our approach achieves up to 3.1× throughput and 1.19× latency improvements with less than 2% quality degradation compared to dense inference.

1. Introduction

Recently, multimodal AI systems are increasingly evolving beyond text generation toward native visual generation capabilities. Emerging applications such as multimodal agents [88], interactive image editing [18, 19, 54], and simulation-based world models [3] increasingly rely on generating visual outputs within the model interaction loop. *Autoregressive image generation* has become a promising paradigm since it formulates visual synthesis as next-token prediction, naturally aligning with transformer architectures and LLM serving infrastructures. However, autoregressive image generation sequentially generates thousands of visual tokens per request, leaving the decode phase bottlenecked by repeated KV cache accesses in attention computation.

Generally speaking, for computationally intensive workloads, prior works have extensively explored approximation techniques that trade modest accuracy degradation for substantial gains in performance and efficiency [5, 22, 23, 32, 33, 46, 53, 75, 90, 91]. Particularly, many visual generation workloads are known to tolerate moderate quality degradation in exchange for improved performance and efficiency [21, 56, 57, 76], suggesting that such techniques are attractive for autoregressive image generation. Practical scenarios such as draft image generation and iterative prompt exploration [2, 51] further reinforce this characteristic. Among these approaches, *sparse attention* has emerged as a promising direction for accelerating autoregressive transformers by selectively attending to only a subset of cached tokens during decoding [7, 11, 14, 15, 24, 40, 44, 55, 69, 84, 85, 92, 94].

However, it remains unclear whether sparse attention techniques originally developed for text-based LLM inference generalize effectively to autoregressive image generation. Although both workloads share the same autoregressive transformer decoding framework, visual token generation fundamentally differs from text generation due to the spatial structure and locality inherent in images. As a result, the token importance patterns exploited by existing sparse attention techniques may not directly transfer to image generation workloads, requiring a characterization-driven understanding of attention sparsity in autoregressive image generation. In particular, two important questions remain unanswered:

- **Question 1:** What sparsity structures emerge during autoregressive image generation?
- **Question 2:** How can such structures be exploited for efficient sparse attention?

To answer the first question, we systematically characterize the attention sparsity in autoregressive image generation across diverse real-world image-generation workloads and representative open-source models. Our analysis reveals several distinguishing properties that fundamentally differ from text-based LLM inference. First, autoregressive image generation exhibits a pronounced prefill-decode asymmetry, where the KV cache is overwhelmingly dominated by decode-phase visual tokens rather than prompt tokens. Second, attention concentrates on prompt tokens and local visual tokens, making sparse designs that retain

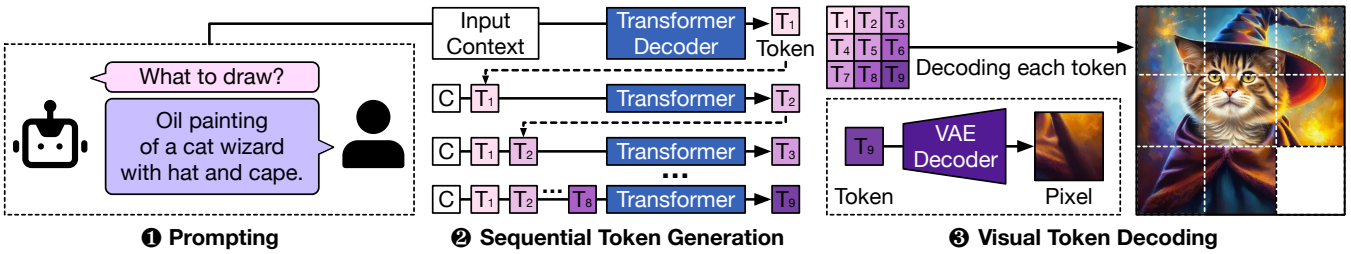

Figure 1: Overview of autoregressive image generation pipeline.

them particularly effective. Most importantly, we observe a structural *diagonal attention sparsity* pattern unique to image generation, where attention importance propagates diagonally across decoding steps due to the spatial locality inherent in visual tokens. These observations reveal that sparse attention behaviors in autoregressive image generation fundamentally differ from those of text-generation workloads, reversing several design intuitions established in prior sparse attention techniques for LLM inference.

Motivated by these observations, we propose a sparse attention mechanism that explicitly exploits the diagonal attention sparsity observed in autoregressive image generation. Unlike prior sparse attention techniques for LLM inference that primarily rely on identifying globally important tokens, our approach leverages the observation that attention importance in image generation shifts diagonally across decoding steps. Based on this insight, we introduce a diagonal-aware sparse attention policy that selectively skips KV entries along the diagonal attention direction within a recent window. Our design extends sliding-window-style sparse attention with diagonal-aware token selection, enabling more effective sparse decoding for visual token generation. Importantly, the proposed approach naturally follows from the structural properties identified in our characterization, demonstrating how image-specific sparsity behaviors can guide the design of efficient sparse attention mechanisms for autoregressive image generation.

We implement the proposed sparse attention mechanism on top of a GPU-based autoregressive image generation serving system built upon FlexGen and FlashAttention-2, integrating custom Triton kernels for diagonal-aware sparse attention. We evaluate the design across diverse open-source autoregressive image generation models and representative text-to-image benchmarks. Our evaluation shows that the proposed approach consistently achieves a superior quality-performance tradeoff compared to existing sparse attention techniques originally designed for LLM inference. In particular, diagonal-aware sparse attention preserves generation quality even under highly aggressive sparsity levels, achieving up to 3.1× throughput improvement and 1.19× latency improvement with less than 2% quality degradation compared to dense inference. These results demonstrate that autoregressive image generation exposes distinct sparsity opportunities fundamentally different from those of text-generation workloads. More broadly, our work suggests that exploiting workload-specific sparsity structures can substantially improve the efficiency of future GPU-based image generation serving systems without incurring significant quality degradation.

2. Background and Motivation

2.1. Applications of Text-to-Image Generation

Recent advances in text-to-image generation have enabled high-fidelity image synthesis from natural language prompts, fueling its adoption across a wide range of services. These services impose diverse quality requirements depending on the target application. While some applications demand high visual fidelity [25], others tolerate moderate quality in exchange for higher throughput or lower latency. For example, draft generation produces low-resolution previews that let users iterate on prompts before committing to a full-quality generation [2, 51]. Since drafts are intermediate artifacts rather than final deliverables, their fidelity matters far less than the exploration speed. As another example, safety filtering feeds generated images to a classifier that only needs category-level recognition rather than user-facing quality [12, 43]. These quality-tolerant use cases open a design space for aggressive efficiency optimizations, motivating efficient image generation paradigms.

2.2. Autoregressive Image Generation

Emergence of autoregressive image generation. Earlier image generation models [6, 50, 64] produce high-fidelity outputs. However, their architectures are not naturally compatible with existing LLM inference stacks. As a result, multimodal AI systems integrate them as standalone generation pipelines, requiring concurrent serving of heterogeneous models. To overcome this incompatibility, autoregressive (AR) image generation [10, 42, 47, 68, 71, 77] has emerged as a promising direction because it inherits the core modeling principles of LLMs and formulates visual synthesis as next-token prediction, making it naturally compatible with existing LLM inference stacks. Consequently, frontier AI platforms [19, 54, 81] increasingly adopt AR image generation as an effective integration approach.

Autoregressive image generation pipeline. AR image generation follows sequential visual token decoding, which consists of three stages, as illustrated in Fig. 1. First, the system begins with ❶ *Prompting*, which provides detailed information about the target image. Second, the system proceeds with ❷ *Sequential token generation*, which iteratively generates each visual token conditioned on the input context and previously generated visual tokens until

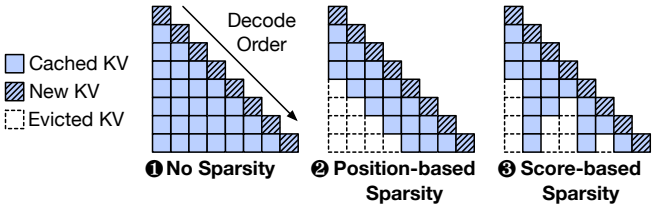

Figure 2: Categories of existing KV sparsity techniques.

the full token grid is filled. Finally, the system performs ❸ *Visual token decoding*, which converts the completed token grid into the output image using a visual decoder, often borrowed from VQ-VAE autoencoders [63, 73].

LLM-aligned decoding behavior. AR image generation follows the same decoding process as LLMs. The decoder transformer iteratively generates tokens through repeated layers of QKV projection, causal self-attention, and Feed-Forward Network. This LLM-aligned decoding structure allows AR image generation to be analyzed through the lens of LLM inference, opening the door to performance gains from decoding-phase techniques originally proposed for LLMs. To provide this background, we next review prior work on efficient LLM inference during the decoding phase, with a particular focus on KV cache-related techniques.

2.3. KV Cache Optimizations for LLM Decoding

Decode bottleneck and optimization opportunity. During LLM decoding, KV caching reuses the keys and values of previously generated tokens to avoid redundant computation. However, each decoding step still needs to retrieve the accumulated KV pairs from prior tokens, making KV access a growing overhead as the sequence length increases. As a result, decoding performance increasingly bottlenecked by the KV cache. Prior work [38, 44, 55, 85, 92] has observed that only a subset of cached key-value pairs are critical for generating the token. This observation has motivated a class of approaches that exploit sparsity in the KV cache by selecting only the relevant pairs at each step, thereby reducing the cost of attending to the KV cache.

KV cache sparsity techniques. We categorize KV cache sparsity techniques by their selection criterion for retained KV pairs, as illustrated in Fig. 2. ① *No sparsity* serves as the baseline and retains all KV pairs throughout decoding after the prefill stage. ② *Position-based sparsity* retains KV pairs according to their positions in the cache, yielding retention patterns with positional regularity. Representative methods include SlidingWindow [5], which keeps only the most recent KV pairs, and StreamingLLM [85], which augments the local window with a small number of attention-sink tokens (i.e., the first few tokens). In contrast, ③ *Score-based sparsity* selects KV pairs by importance metrics such as attention scores, producing irregular retention patterns scattered across the cache. These methods implicitly assume that a token's importance persists across subsequent queries, corresponding to vertical patterns in the attention score map. Methods in this category include H2O [92], TOVA [55], and ALISA [94], which select KV pairs using accumulated or per-step attention scores.

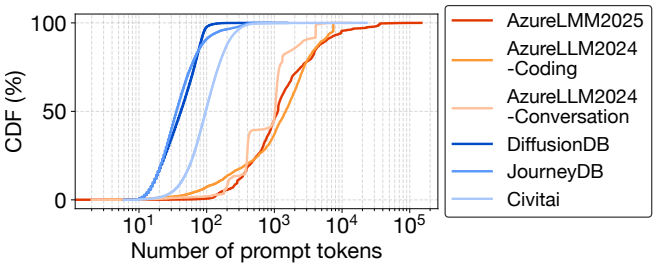

Figure 3: Prompt token length distribution in various Text-to-Image and LLM workloads.

Open questions for autoregressive image generation. Leveraging existing KV cache sparsity techniques from LLM decoding is a natural starting point for accelerating AR image generation. However, whether these techniques remain effective under visual-token generation workloads remains an open question. Unlike text decoding, where token dependencies follow sequential linguistic order, AR image generation produces visual tokens whose attention patterns reflect spatial locality and spatial structure. These structural differences may fundamentally shift the efficiency-quality tradeoff of KV cache sparsity. Accordingly, the following sections first characterize autoregressive image generation. Building on this analysis, we then revisit existing KV cache sparsity techniques and explore extension directions grounded in visual-token characteristics.

3. Characterization of Autoregressive Image Generation

In this section, we characterize autoregressive image generation for sparse KV cache design. Section 3.1 describes our methodology, Section 3.2 and 3.3 analyze sequence lengths and the resulting decoding bottlenecks. Then Section 3.4 identifies attention sparsity properties that distinguish AR image generation from text decoding.

3.1. Characterization Methodology

We design our methodology to expose workload properties that distinguish autoregressive image generation from text decoding. To this end, we evaluate representative autoregressive image generation models and LLMs using datasets that reflect their respective serving workloads. We conduct all experiments on a server-class platform equipped with an NVIDIA RTX A6000 GPU with 48GB of memory and an Intel Xeon Gold CPU.

Models. We analyze three representative open-source autoregressive image generation models: Janus-Pro-1B and 7B [10], Lumina-mGPT-7B [42] in resolution 512, and NextStep-1-14B [71]. For comparison with text decoding, we include three open-source LLMs of similar scale: Qwen3-8B [86], Llama3.1-8B [20], and Gemma3-4B [70].

Datasets. For image generation, we analyze prompts from three real-world text-to-image traces, including DiffusionDB [78], Civitai-prompts [1], and JourneyDB [67], which span Stable Diffusion deployments, Civitai posts, and

TABLE 1: OUTPUT TOKEN LENGTH OF IMAGE MODELS.

Model	Resolution	VAE latent ratio	Output length
Janus-Pro [10]	384x384	1/16	576
Llamagen [68]	512x512	1/16	1024
Lumina-mGPT [42]	512x512	1/16	1024
	1024x1024	1/16	4096
Emu3 [77]	512x512	1/8	4096
	720x720	1/8	8100
NextStep-1 [71]	512x512	1/16	1024
TokenShuffle [47]	2048x2048	1/32	4096

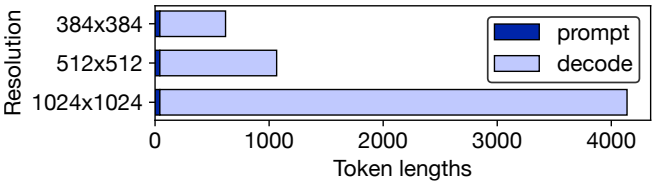

Figure 4: Breakdown of prefill and decode token lengths for image model across different target image resolutions.

Midjourney generations, respectively, to capture prompt-length distributions across diverse real-world text-to-image platforms. For text-generation workloads, we characterize text-generation workloads using three Azure inference traces [49]: AzureLMM2025, AzureLLM2024-Coding, and AzureLLM2024-Conversation, which cover multimodal, coding, and conversational serving requests, respectively.

3.2. Sequence Length Analysis

We first analyze sequence lengths to understand how prefill and decode contribute to KV cache pressure in autoregressive image generation across the workloads.

Short prompts compared to text generation. We analyze prompt token lengths and compare them against representative text generation workloads. Fig. 3 shows the cumulative distribution function (CDF) of prompt token lengths, with red lines representing LLM workloads and blue lines representing image generation workloads. Those workloads exhibit much shorter prompt lengths, with most prompts falling below 100 tokens, whereas LLM workloads have average prompt lengths exceeding 1K tokens. Compared to text generation, where long prompts place substantial pressure on the KV cache, autoregressive image generation workloads incur relatively low prefill-time KV cache pressure due to their short prompt lengths.

Long decode lengths in image generation. In contrast to their short prompts, image generation workloads involve substantially longer decode sequences, whose lengths are fixed by the output resolution and the VAE latent downsampling ratio. As summarized in Table 1, most existing models produce roughly 1K to 4K visual tokens under a 1/16 latent ratio, and higher-resolution settings can increase the decode length up to 8100 tokens. Combined with the prompt-length analysis above, this creates a pronounced prefill-decode asymmetry. Fig. 4 illustrates this asymmetry for models using a typical 1/16 VAE downsampling ratio, where the decode length exceeds the prompt length by up to 40× at 1024×1024 resolution. Consequently, visual

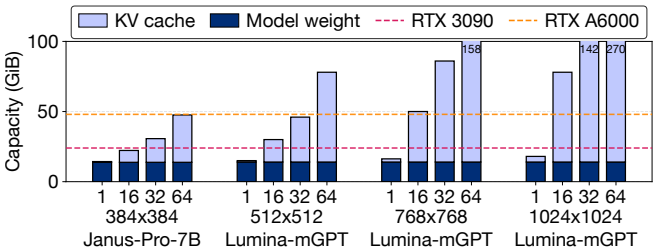

Figure 5: Memory footprint in various image generation models. x-axis represents batch size, output image resolution, and model name.

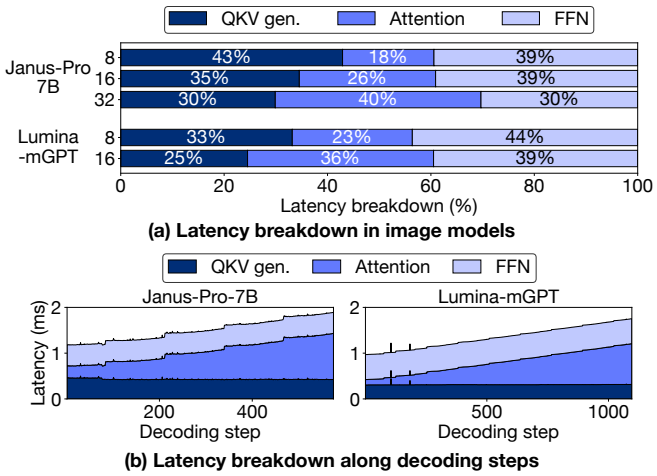

Figure 6: Transformer latency breakdown in autoregressive image models. (a) Janus-Pro-7B and Lumina-mGPT latency breakdown across batch sizes, averaged over decoding steps. (b) Per-step latency breakdown for Janus-Pro-7B at batch size 32 (left) and Lumina-mGPT at batch size 16 (right).

tokens accumulated during the decode phase dominate the KV cache in autoregressive image generation.

> **Takeaway #1**: Sparse KV design should focus on reducing accesses to the long and continuously accumulating decode KV cache, rather than the already short prefill KV cache.

3.3. Decoding Phase Bottleneck Analysis

Building on the prefill-decode asymmetry, we next analyze how long visual decode sequences drive memory and compute pressure during decoding.

KV cache capacity bottleneck in image generation. Image generation services use batched inference to generate multiple images per prompt or improve throughput across concurrent requests. From this batching perspective, we analyze the per-request memory footprint of representative autoregressive image generation models across resolutions and batch sizes. As illustrated in Fig. 5, we separate model weights from the KV cache and compare the total footprint against RTX 3090 (24GB) and RTX A6000 (48GB) memory budgets. While lower resolutions allow moderate batching for some models, higher resolutions quickly push the

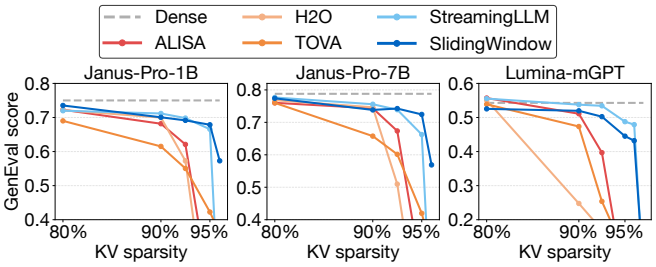

Figure 7: GenEval score of LLM baselines applied to image models.

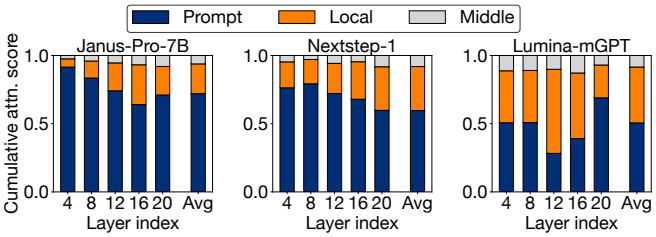

Figure 8: Cumulative attention score in prompt, local, and middle token in image models, averaged over all attention heads and decoding steps.

memory footprint beyond practical GPU capacity. These results show that KV cache capacity becomes a practical bottleneck when autoregressive image generation requires both high batching and high output resolution.

Attention computation bottleneck during decoding. Beyond KV cache capacity, we further examine how different transformer components contribute to end-to-end generation latency. We break down per-layer latency into three stages using two representative autoregressive image generation models under different batch sizes, as shown in Fig. 6(a). The results show that attention accounts for a non-negligible portion of decoding latency, reaching up to 40%. To further understand how this overhead evolves during decoding, we analyze the per-step latency breakdown in Fig. 6(b). Attention initially accounts for 22.3% and 12.1% of per-step latency for Janus-Pro-7B and Lumina-mGPT respectively, but its portion gradually increases to 53.3% and 50.8% as decoding progresses. Long visual decode sequences not only increase KV cache memory usage but also amplify attention computation, making KV access a central efficiency bottleneck.

Takeaway #2: KV cache pressures both memory capacity and attention latency, motivating sparse attention that reduces both axes simultaneously.

3.4. Attention Sparsity Analysis

We next characterize attention sparsity as an opportunity to reduce memory footprint and attention cost. We begin by revisiting existing sparse KV caching methods under AR image generation, and then analyze the attention sparsity underlying their behavior.

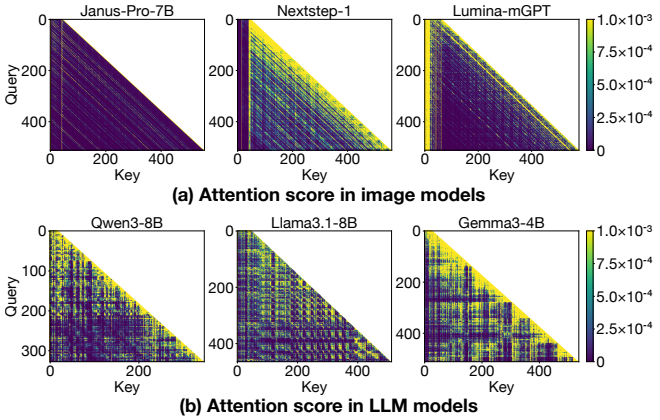

Figure 9: Attention score in (a) image models and (b) LLMs. We used attention score map of layer 10 and head 5.

Revisiting LLM sparse KV caching works. Fig. 7 compares two categories of LLM sparse KV baselines defined in Section 2.3: blue lines for position-based methods [5, 85, 91] and red/orange lines for score-based methods [55, 92, 94]. Prior LLM studies have shown that score-based selection preserves accuracy better than position-based methods because important tokens are distributed irregularly across the sequence [69, 92]. In image generation, this ordering reverses: position-based methods consistently outperform score-based methods across all sparsity levels. We show that this reversal stems from two structural properties of autoregressive image generation: attention concentrates on prompt and recent tokens, and attention sparsity follows a diagonal pattern. Together, these properties make the irregular middle-token importance patterns exploited by score-based methods in LLMs less effective for image generation workloads.

Takeaway #3: Unlike in LLM decoding, position-based sparsity outperforms score-based sparsity in image generation.

Skewed attention toward prompt and local tokens. Fig. 8 reports cumulative attention scores over three token groups: prompt tokens, local tokens, defined as the most recent 10% of decoded tokens, and middle tokens, defined as the remaining decoded tokens. Although prompt tokens occupy only a small fraction of the full sequence, as shown in Section 3.2, they capture more than half of the total attention score. Local tokens account for most of the attention mass outside the prompt region, while middle tokens contribute only a small fraction. Across the three models, middle tokens capture only 6.2%, 8.1%, and 8.5% of the total attention score on Janus-Pro, NextStep-1, and Lumina-mGPT. This skewed pattern reflects the structure of autoregressive image generation. Each visual token must remain aligned with the prompt while also preserving spatial consistency with nearby generated tokens.

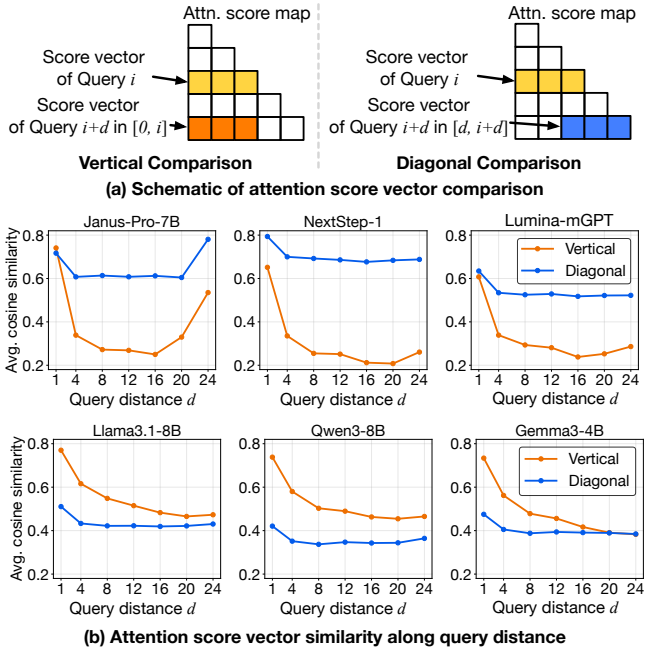

Figure 10: (a) Schematic of vertical vs. diagonal attention score vector comparison on a score map. (b) Attention score vector similarity along query distance d for image models(top) and LLMs(bottom), averaged over all layers, heads and queries.

> **Takeaway #4**: Attention in image models concentrates on prompt and local tokens, making sparse attention designs that retain them particularly effective.

Diagonal attention sparsity pattern. Fig. 9 visualizes attention scores across three image models and LLMs. Beyond the prompt region on the left, all three image models exhibit pronounced diagonal stripes spanning the decode region. In contrast, LLM attention maps show no such structure. To quantitatively confirm this diagonal structure, we measure the cosine similarity between attention score vectors in two ways, as illustrated in Fig. 10(a). Vertical similarity compares attention score vector of query i and query $i+d$ over their common key range $[0, i]$, capturing how two queries separated by query distance d attend to the same keys. Diagonal similarity compares attention score vector of query i with query $i+d$ over key range $[d, i+d]$, measuring how similar attention scores are along the diagonal direction of the score map. As shown in Fig. 10(b), diagonal similarity consistently exceeds vertical similarity in image models, contrast to LLMs. Diagonal alignment is therefore a structural characteristic specific to the image generation.

Why diagonal patterns emerge. In LLMs, attention concentrates on a few globally important tokens, producing vertical patterns in the attention map. In image generation, however, no comparably dominant tokens emerge. Due to spatial locality, the keys most relevant to each query lie in its neighborhood. As the query advances, this neighborhood shifts together with it, preserving the relative position between important query-key pairs. And this repeated structure manifests as the diagonal pattern observed in the attention score map shown in Fig. 11. This confirms that attention in image models is governed by the relative position between query and key, rather than by absolute key positions. As a result, existing score-based methods fail to capture diagonal sparsity, while conventional position-based methods exploit it only implicitly through local windows.

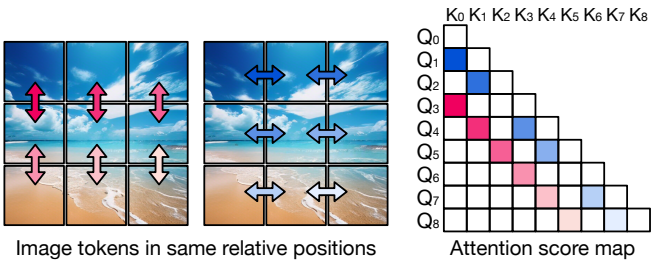

Figure 11: Relative position similarity in images projects onto diagonal patterns in the attention score map.

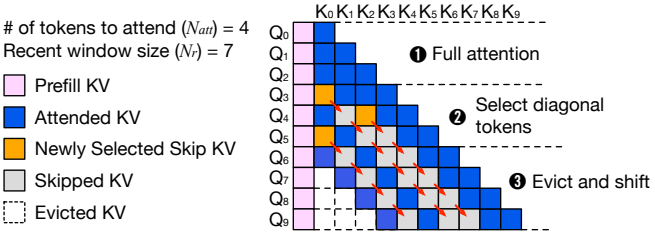

Figure 12: Diagonal-aware sparse attention algorithm.

> **Takeaway #5**: Spatial locality induces diagonal attention sparsity patterns, motivating sparse attention designs that exploit diagonal structure.

4. Diagonal-Aware Sparse Attention

Our analysis reveals a diagonal attention pattern in autoregressive image generation beyond simple prompt and local-token concentration. Existing position-based methods partially exploit this through local windows, but degrade sharply at high sparsity. We therefore propose a sparse attention scheme that explicitly exploits diagonal sparsity to further improve the quality-latency tradeoff.

Design rationale. First, decode KV dominates the cache, so sparsity must target the decode region rather than the prompt. Second, attention concentrates on prompt and local tokens while middle tokens contribute negligibly, suggesting that a recent window combined with always-attended prompt KV captures most of the attention weight. Third, attention exhibits a diagonal sparsity pattern within the decode region, suggesting that important attention regions can be tracked along diagonal directions rather than simple local windows.

Algorithm overview. Based on the design rationale, our algorithm maintains a wider recent window and computes attention with diagonal token selection. Within the recent window, we skip a growing set of positions that trace

Algorithm 1: Diagonal-Aware Sparse Attention

Input: Recent window size N_r
Input: Number of tokens to attend N_{att}
Input: Decode KV cache K, V
Input: Prompt KV cache K_p, V_p

1 $skip\_idx \leftarrow [\,]$
2 **for** each decode step t **do**
3 $window \leftarrow [\max(0, t - N_r), \ldots, t]$
4 **if** $t < N_{att} - 1$ **then**
  `// Phase 1: full attention`
5 $idx \leftarrow window$
6 $\text{out}_t \leftarrow \text{Attn}(Q_t,\ K_p + K[idx],\ V_p + V[idx])$
7 **else if** $t < N_r$ **then**
  `// Phase 2: grow diagonal skip set`
8 $skip\_idx \leftarrow [j + 1 \textbf{ for } j \in skip\_idx]$
9 $idx \leftarrow window - skip\_idx$
10 $\text{out}_t \leftarrow \text{Attn}(Q_t,\ K_p + K[idx],\ V_p + V[idx])$
11 **if** $|skip\_idx| < N_r - N_{att}$ **then**
    `// select newly skipped KV`
12 $w \leftarrow \text{softmax}(Q_t \cdot K[idx]^\top)$
13 $j^* \leftarrow \arg\min_j w[j]$
14 $skip\_idx.\text{append}(idx[j^*])$
15 **else**
  `// Phase 3: evict oldest KV and shift`
16 $K[t - N_r] \leftarrow \varnothing$
17 $V[t - N_r] \leftarrow \varnothing$
  `// Shift diagonal positions by +1`
18 $skip\_idx \leftarrow [j + 1 \textbf{ for } j \textbf{ in } skip\_idx]$
19 $idx \leftarrow window - skip\_idx$
20 $\text{out}_t \leftarrow \text{Attn}(Q_t,\ K_p + K[idx],\ V_p + V[idx])$

diagonals through the attention score map, as illustrated in Fig. 12. Once a key receives low attention from Q_t, its diagonal neighbor is likely to receive low attention from Q_{t+1}, allowing selection decisions to propagate along the diagonal without re-scoring. We maintain independent diagonal selections for each attention head.

Detailed explanation. Fig. 12 and Algorithm 1 realize these principles with two parameters: recent window size N_r and number of tokens to attend N_{att}. We illustrate with $N_r = 7$ and $N_{\text{att}} = 4$ as in Fig. 12.

❶ **Full attention (lines 4-6)** For the first $N_{\text{att}} - 1 = 3$ steps, there is no *skip_idx* to exclude in attention. Each query attends to all available decode KV and prompt KV.

❷ **Select diagonal tokens. (lines 7-14)** Once decoding step $t \geq N_{\text{att}}$, every step appends one new skip KV. At $t = 3$, we compute attention scores over $K[0{:}3]$, identify the lowest-scoring key (say K_0), and set $skip\_idx = [0]$. This newly selected entry is not skipped at the current step. In the next decoding step ($t = 4$), the skip indices shift by +1, so $skip\_idx = [1]$, tracking the diagonal successor of (Q_3, K_0). Q_4 then compute attention and score over $\{K_0, K_2, K_3, K_4\}$, select the new lowest (say K_2), and append: $skip\_idx = [1, 2]$. At $t = 5$, the skip indices shift again to $skip\_idx = [2, 3]$, continuing to trace the diagonal skip pattern. The skip set grows by one entry per decoding step until $|skip\_idx|$ becomes $N_r - N_{\text{att}} (= 3)$.

❸ **Evict and shift (lines 15-19)** Once $t \geq N_r = 7$, the window slides forward and no new keys are selected to be skipped. At $t = 7$, the oldest one K_0 is evicted from the cache. The existing skip indices in *skip_idx* shift by +1, continuing to trace their diagonals. The attended count stays fixed at $N_{\text{att}} = 4$ throughout this phase. Throughout all phases, the prompt KV is always attended in full.

Parameter selection. Both parameters N_r and N_{att} are directly determined by the target memory and compute budgets, without per-model tuning. N_r sets the KV cache footprint and is chosen as a fraction of the expected decoding length L_{dec}; we use $N_r = 0.5 \times L_{\text{dec}}$ in our experiments. N_{att} sets the per-step attention cost and is chosen to match the desired sparsity ratio. Together, (N_r, N_{att}) expose two independent knobs: N_r controls memory, N_{att} controls compute, allowing the user to navigate the memory-compute trade-off directly.

Cost analysis. The KV cache footprint is bounded by $N_r + L_p$, where L_p is the prompt length. In contrast, dense attention grows linearly with $L_{\text{dec}} + L_p$. Per-step attention cost is $O(N_{att} + L_p)$, fixed across decode steps. The selection step adds an $O(N_r)$ argmin, but it runs only for $N_r - N_{att}$ steps in phase 2 and is amortized over the remaining decoding. In evict and shift state, the algorithm reduces memory by a factor of $(L_{\text{dec}} + L_p)/(N_r + L_p)$ and per-step compute by $(L_{\text{dec}} + L_p)/(N_{att} + L_p)$ compared to dense attention.

5. Evaluation

5.1. Methodology

Implementation. We implement our diagonal-aware sparse attention on top of FlexGen [65], a high-throughput transformer inference engine, integrating FlashAttention-2 [13] for dense attention and custom Triton [72] kernels for sparse attention with diagonal selection. Our Triton kernel fuses indirect KV gather from the buffer into the attention pass itself, eliminating the intermediate gather buffer. It further fuses the argmin over recent-window logits into the attention kernel, removing a separate kernel.

Models. We evaluate on three representative AR image generation models spanning different scales and tokenizer designs. Janus-Pro [10] is evaluated at both 1B and 7B parameters with 384×384 resolution (576 visual tokens), and Lumina-mGPT [42] at 7B parameters with 512×512 resolution (1024 visual tokens). We set the classifier-free guidance weight to 5.0 for Janus and 4.0 for Lumina-mGPT as default setting in official repository.

Sparse KV baselines. We compare against dense decoding and five sparse KV baselines. *Position-based* methods retain KV pairs by positional rules: StreamingLLM [85] keeps 4 attention sink tokens together with a recent window, while SlidingWindow [5] keeps only the most recent window. For StreamingLLM, we retain the first 4 decode tokens as attention sinks, since prompt KV is always kept in full in our setting. *Score-based* methods retain KV pairs by importance

TABLE 2: GenEval and DPG-bench score.

Model	Method	GenEval					DPG-bench				
		Sparsity					Sparsity				
		80%	90%	92.50%	95%	96%	80%	90%	92.50%	95%	96%
Janus-Pro-1B	Dense	0.750	-	-	-	-	82.22	-	-	-	-
	ALISA	0.723	0.682	0.621	0.192	0.118	82.17	82.06	79.65	55.07	46.47
	H2O	0.722	0.700	0.573	0.048	0.033	82.10	81.74	72.17	39.05	36.72
	TOVA	0.690	0.615	0.550	0.422	0.361	81.79	79.78	76.08	72.04	68.14
	SlidingWindow	**0.735**	0.701	0.692	0.679	0.573	**82.86**	**82.67**	81.72	**81.86**	77.70
	StreamingLLM	0.720	0.712	0.698	0.665	0.083	82.79	82.58	82.11	81.29	44.70
	Diagonal	0.719	**0.718**	0.692	**0.688**	0.668	82.04	81.51	**82.21**	81.46	**81.01**
	Diagonal + sink4	0.716	0.710	**0.704**	0.682	**0.671**	81.89	81.39	81.63	81.02	80.75
Janus-Pro-7B	Dense	0.788	-	-	-	-	84.16	-	-	-	-
	ALISA	0.760	0.743	0.674	0.230	0.125	83.46	82.83	81.07	60.91	48.83
	H2O	0.768	0.746	0.510	0.024	0.019	83.09	83.09	73.12	37.91	34.58
	TOVA	0.760	0.658	0.602	0.420	0.307	82.80	80.13	78.15	73.30	67.25
	SlidingWindow	0.774	0.738	0.743	0.724	0.569	83.77	83.33	84.15	**83.88**	78.25
	StreamingLLM	0.777	0.756	0.739	0.662	0.071	83.42	83.18	83.27	81.09	44.33
	Diagonal	**0.792**	0.758	0.768	**0.749**	0.738	83.55	**83.64**	**83.62**	83.18	**82.67**
	Diagonal + sink4	0.785	**0.784**	**0.774**	0.743	**0.749**	**83.81**	83.35	82.58	82.28	82.00
Lumina-mGPT	Dense	0.543	-	-	-	-	75.67	-	-	-	-
	ALISA	**0.557**	0.511	0.397	0.051	0.035	**76.60**	74.25	67.41	46.24	42.12
	H2O	0.551	0.248	0.170	0.016	0.037	75.15	60.34	57.11	35.95	40.14
	TOVA	0.539	0.474	0.254	0.103	0.058	75.76	69.53	59.36	51.58	48.68
	SlidingWindow	0.525	0.520	0.502	0.445	0.432	74.88	74.13	73.17	69.35	68.46
	StreamingLLM	0.555	0.538	0.534	0.488	0.479	75.92	**75.64**	74.57	71.43	71.37
	Diagonal	0.542	0.540	0.547	0.520	0.492	75.93	75.15	75.13	**74.48**	**72.89**
	Diagonal + sink4	0.555	**0.541**	**0.548**	**0.524**	**0.504**	76.23	75.20	**75.23**	73.68	72.24

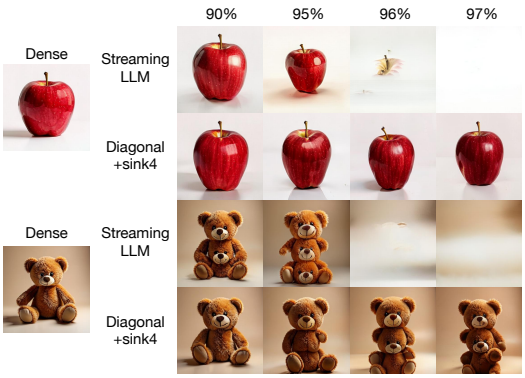

Figure 13: Qualitative result in Janus-Pro-7B at high sparsity.

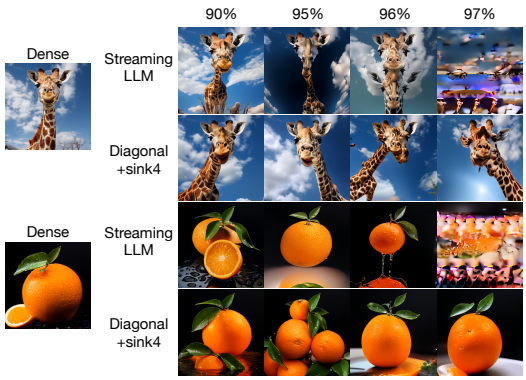

Figure 14: Qualitative result in Lumina-mGPT at high sparsity.

metrics: H2O [92] selects heavy hitters by accumulated attention scores, TOVA [55] evicts the minimum-scoring keys at each step, and ALISA [94] combines sparsity-aware selection with a recent accumulated scores. Across all methods, we sweep KV sparsity ratios of 80% to 96% relative to the dense decode KV, where the ratio denotes the fraction of decode KV not used for the attention computation.

Our methods. We refer to our two variants as *Diagonal* and *Diagonal+Sink4* throughout the evaluation. Diagonal+Sink4 retains the 4 initial decode tokens as attention sinks, following StreamingLLM, and applies diagonal selection for the remaining slots. Sparsity ratios refer to the fraction of skipped decode KVs relative to the maximum decode length, i.e., $1 - N_{att}/L_{dec}$.

Benchmarks. We evaluate generation quality on two standard text-to-image benchmarks. GenEval [16] measures object-focused prompt alignment across 553 prompts with 4 images generated per prompt, scoring attributes such as object presence, count, color, and spatial relations. DPG-Bench [30] complements GenEval with dense, compositional prompts that stress fine-grained semantic alignment.

System setup. All experiments run on a single NVIDIA RTX A6000 (48GB) with an Intel Xeon Gold CPU. We report end-to-end per-image latency and throughput (images per second). Latency and throughput are measured with a fixed prompt length of 50 tokens, the median prompt length in DiffusionDB [78]. All latency and throughput results are averaged over 3 inferences after a warm-up.

5.2. Generation Quality

Quantitative results. Table 2 reports GenEval and DPG-bench scores across three models. First, score-based baselines (ALISA, H2O, TOVA) collapse beyond 90% sparsity, retaining less than 30% of dense accuracy at 95-96%. This confirms Takeaway #3: methods tuned for irregular LLM

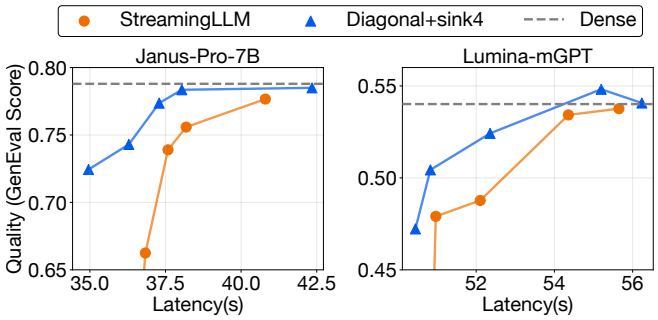

Figure 15: Quality-latency tradeoff. Janus-Pro-7B used batch size of 96, and Lumina-mGPT used batch size of 54.

patterns fail when image attention is dominated by prompt and local structure. Second, position-based baselines hold up through 90-92.5% but degrade at 95-96%. StreamingLLM shows a sharp cliff at 96% on both Janus-Pro models, losing over 90% of dense accuracy. Third, Diagonal selection holds the accuracy frontier at extreme sparsity. Across all three models, Diagonal stays within 1–6% of dense accuracy even at 96% sparsity, where the best position-based baseline drops by 10-30% and score-based baselines drop by over 80%. Position-based baselines reaches high sparsity by shrinking the recent window itself, restricting attention to a narrow local region. Our method instead observes a wider recent window and skips diagonally-aligned attention within it. Attention covers the same number of keys drawn from a broader context, which explains the consistent quality gain.

Qualitative results. Fig. 13 and Fig. 14 compare StreamingLLM and Diagonal selection against dense outputs. At 90% sparsity, both methods preserve image fidelity. At higher sparsity, StreamingLLM degrades rapidly with repetitive objects and distorted structures. Diagonal selection retains coherent shapes and prompt-aligned content up to 96-97%, which is consistent with quantitative results.

5.3. Performance

Tradeoff between latency and generation quality. Fig. 15 plots the quality-latency tradeoff for our method and StreamingLLM on Janus-Pro-7B and Lumina-mGPT. We compare against StreamingLLM as it achieves the highest quality among the baselines and incurs no selection overhead, making it the strongest reference point on both axes. The x-axis shows end-to-end latency, and the y-axis shows the GenEval score. Moving up the plot reflects improved quality and moving left reflects reduced latency, so the top-left corner represents the most favorable tradeoff. Diagonal dominates StreamingLLM across the sweep on both models. On Janus-Pro-7B at batch size 96, Diagonal+sink4 reaches dense-level GenEval score at 38s, while StreamingLLM requires 41s to reach comparable quality. On Lumina-mGPT at batch size 54, Diagonal sustains 2-3% of quality above StreamingLLM at every operating point, with the gap widening as latency tightens. Diagonal selection recovers structurally important tokens that a fixed local window would otherwise drop, pushing the quality-latency frontier beyond the baseline.

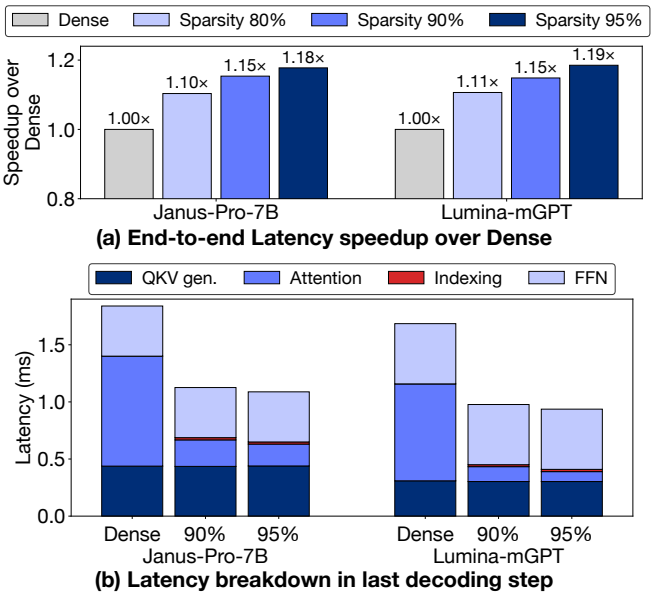

Figure 16: (a) Latency speedup over dense and (b) latency breakdown in the last decoding step. Janus-Pro-7B used batch size 32 and Lumina-mGPT used batch size 16.

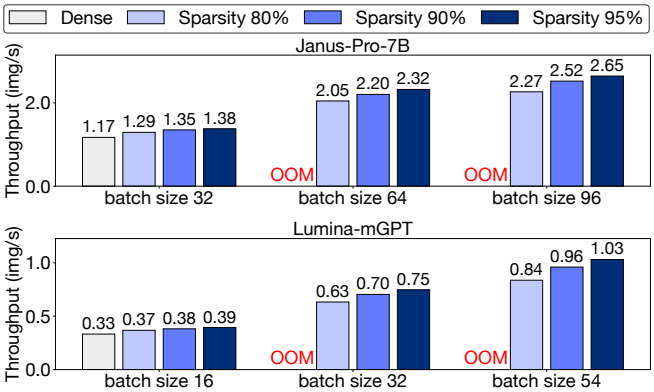

Figure 17: Throughput improvement across different batch sizes on Janus-Pro-7B and Lumina-mGPT.

Latency speedup breakdown. Fig. 16(a) shows the end-to-end latency speedup over dense model. Speedup over dense reaches 1.18× and 1.19× at 95% sparsity for Janus-Pro-7B and Lumina-mGPT, respectively. Fig. 16(b) breaks down the per-step latency at the final decoding step into QKV projection, attention, sparse indexing overhead, and FFN. At the final decoding step, attention alone speeds up by 4.15× and 5.03× over dense at 90% and 95% sparsity on Janus-Pro-7B, and by 6.52× and 9.78× on Lumina-mGPT. Indexing overhead stays small relative to the attention savings, as our custom kernel fuses indexing and attention and removed redundant buffer allocation. The attention latency shrinks with sparsity while indexing adds only a little overhead, leaving net latency well below dense.

Throughput. Fig. 17 reports per-second image throughput across batch sizes. First, the smaller KV footprint admits larger batches: dense Janus-Pro-7B runs out of memory at batch size 64 and beyond, while Diagonal scales to batch size 96. Lumina-mGPT shows the same pattern, with dense capped at batch size 16 while Diagonal reaches batch size 54. Second, within feasible batch sizes, per-image throughput improves with sparsity. On Janus-Pro-7B at batch size 32, Diagonal at 95% sparsity reaches 1.38 img/s versus 1.17 img/s dense. At the largest feasible batch size, the combined effect yields up to 2.65 img/s on Janus-Pro-7B and 1.03 img/s on Lumina-mGPT, a 2.3× and 3.1× higher throughput over the dense configuration. Overall, by keeping only the recent-window KV on the GPU, our method enables higher batch sizes, while its efficient sparse attention also raises throughput at a fixed batch size.

6. Related Works

Image generation model. Diffusion models [9, 52, 64, 79] have enabled the practical deployment of image generation systems by overcoming major limitations of earlier GAN [17, 35, 62] based approaches, including low fidelity and training instability. However, as discussed in Section 2.2, diffusion backbones such as U-Net [28, 59, 66] and DiT [4, 58, 87] remain architecturally distinct from LLMs, forcing multimodal serving frameworks to provision diffusion and LLM workloads separately, often partitioning GPU resources across different model pipelines [61, 89]. In this respect, AR image generation [10, 30, 42, 47, 48, 68, 71, 77, 80] offers greater compatibility with LLM-centric serving infrastructures. Rather than replacing diffusion, AR image generation represents an orthogonal design point, and industry systems increasingly choose between diffusion and AR models based on serving-stack requirements [19, 31, 81].

KV sparsity for LLM. The decoding phase of LLMs is memory-bandwidth-bound, and KV sparsity [38, 44, 55, 85, 92, 94] mitigates this by reducing the number of cache entries accessed at each step. Beyond the basic position- and score-based schemes covered in Section 2.3, prior work has refined KV sparsity along several axes. SnapKV [40] and NACL [11] move selection to the prefill stage to avoid per-step decisions, using a prompt-end observation window and proxy tokens. PyramidKV [7] and Ada-KV [15] distribute cache capacity unevenly across layers and heads. LESS [14] keeps a low-rank summary of dropped KV, DuoAttention [84] and CateKV [34] splits heads into retrieval and streaming. Quest [69] and OmniKV [24] go further by preserving the full cache, reducing only per-step access via per-page key bounds and inter-layer selection reuse. All of these designs target the attention patterns of text LLMs; our characterization in Section 3 shows these patterns shift in autoregressive image generation, motivating the diagonal-aware policy in Section 4.

KV quantization for LLM. KV quantization [37] addresses the same bandwidth bottleneck, lowering per-entry bit-width. KIVI [45] and KVQuant [29] enable 2-bit KV caches through per-channel key and per-token value quantization, and Atom [93] and QServe [41] support low-bit KV serving through outlier-aware quantization. KV quantization is orthogonal to our method and can be applied on top of it for further bandwidth savings.

AR image generation optimizations. Image generation specific properties have been widely exploited to accelerate vision inference [8, 36, 39, 46], and recent work extends this principle to AR image generation [26, 27, 74, 82]. ZipAR [26], PAR [74], and NAR [27] all exploit the spatial locality of image tokens to decode multiple tokens in parallel, by overlapping or reordering the token sequences. These approaches improve decoding parallelism, so they are orthogonal to our method and can be combined with it. Closer to our work, HACK [60] and ADSA [83] illustrate diagonal attention behavior of autoregressive image models. HACK classifies heads by sparsity variance and falls back to score-based selection within a local window, while ADSA selects diverse middle tokens per step. In contrast, we turn the diagonal into an explicit selection rule, and quantitatively characterize the pattern across multiple AR image generation models.

7. Conclusion

This paper presents the first characterization of attention sparsity in autoregressive image generation and shows that its sparsity behaviors fundamentally differ from those established in text-based LLM inference. Our analysis reveals that autoregressive image generation exhibits unique structural properties, including diagonal attention sparsity arising from the spatial locality of visual tokens. Building on these observations, we demonstrate that exploiting workload-specific sparsity structures enables substantial throughput and latency improvements with minimal quality degradation. As multimodal AI systems continue evolving toward agentic and physically interactive workloads, efficient visual generation becomes an increasingly important systems challenge. We believe this work represents an initial step toward a broader direction of approximation-aware image generation serving, where practical efficiency is achieved not only through hardware scaling, but also through workload-specific compromises guided by the structural properties of visual generation workloads.

Acknowledgments

This work was partly supported by the IITP(Institute of Information & Communications Technology Planning & Evaluation)-ITRC(Information Technology Research Center) grant funded by the Korea government(MSIT)(IITP-2026-RS-2020-II201795), the IITP grant funded by the Korea government(MSIT) (No.RS-2024-00459797), and IITP under the Graduate School of Artificial Intelligence Semiconductor(IITP-2026-RS-2023-00256472) grant funded by the Korea government(MSIT). The authors used Claude for code, text, and figures, but all outputs were reviewed carefully by the authors.

References

[1] AdamCodd, "Civitai 2m prompts," https://huggingface.co/datasets/AdamCodd/Civitai-2m-prompts, 2024, accessed: 2026-04-27.

[2] Adobe. (2025) Generate images quickly using Fast mode — Adobe Firefly Web Help. Accessed: 2026-05-20. [Online]. Available: https://helpx.adobe.com/firefly/web/generate-images-with-text-to-image/generate-images-using-text-prompts/use-fast-mode-for-quick-image-generations.html

[3] N. Agarwal, A. Ali, M. Bala, Y. Balaji, E. Barker, T. Cai, P. Chattopadhyay, Y. Chen, Y. Cui, Y. Ding et al., "Cosmos world foundation model platform for physical ai," arXiv preprint arXiv:2501.03575, 2025.

[4] F. Bao, S. Nie, K. Xue, Y. Cao, C. Li, H. Su, and J. Zhu, "All are worth words: A vit backbone for diffusion models," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2023, pp. 22 669–22 679.

[5] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," arXiv preprint arXiv:2004.05150, 2020.

[6] J. Betker, G. Goh, L. Jing, T. Brooks, J. Wang, L. Li, L. Ouyang, J. Zhuang, J. Lee, Y. Guo et al., "Improving image generation with better captions," Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf, vol. 2, no. 3, p. 8, 2023.

[7] Z. Cai, Y. Zhang, B. Gao, Y. Liu, Y. Li, T. Liu, K. Lu, W. Xiong, Y. Dong, J. Hu et al., "Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling," arXiv preprint arXiv:2406.02069, 2024.

[8] A. Castillo, J. Kohler, J. C. Pérez, J. P. Pérez, A. Pumarola, B. Ghanem, P. Arbeláez, and A. Thabet, "Adaptive guidance: Training-free acceleration of conditional diffusion models," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 39, no. 2, 2025, pp. 1962–1970.

[9] J. Chen, J. Yu, C. Ge, L. Yao, E. Xie, Z. Wang, J. Kwok, P. Luo, H. Lu, and Z. Li, "Pixart-alpha: Fast training of diffusion transformer for photorealistic text-to-image synthesis," in International conference on learning representations, vol. 2024, 2024, pp. 57 611–57 640.

[10] X. Chen, Z. Wu, X. Liu, Z. Pan, W. Liu, Z. Xie, X. Yu, and C. Ruan, "Janus-pro: Unified multimodal understanding and generation with data and model scaling," arXiv preprint arXiv:2501.17811, 2025.

[11] Y. Chen, G. Wang, J. Shang, S. Cui, Z. Zhang, T. Liu, S. Wang, Y. Sun, D. Yu, and H. Wu, "Nacl: A general and effective kv cache eviction framework for llm at inference time," in Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2024, pp. 7913–7926.

[12] CompVis and Stability AI. (2022) Stable Diffusion safety checker. Accessed: 2026-05-22. [Online]. Available: https://huggingface.co/CompVis/stable-diffusion-safety-checker

[13] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," in International Conference on Learning Representations, vol. 2024, 2024, pp. 35 549–35 562.

[14] H. Dong, X. Yang, Z. Zhang, Z. Wang, Y. Chi, and B. Chen, "Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference," arXiv preprint arXiv:2402.09398, 2024.

[15] Y. Feng, J. Lv, Y. Cao, X. Xie, and S. K. Zhou, "Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference," Advances in Neural Information Processing Systems, vol. 38, pp. 113 152–113 188, 2026.

[16] D. Ghosh, H. Hajishirzi, and L. Schmidt, "Geneval: An object-focused framework for evaluating text-to-image alignment," Advances in Neural Information Processing Systems, vol. 36, pp. 52 132–52 152, 2023.

[17] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," Advances in neural information processing systems, vol. 27, 2014.

[18] Google, "2.5 Flash and Native Capabilities – Audio & Image: Model Card," Dec. 2025, model card, published December 2025. [Online]. Available: https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-2-5-Flash-Model-Card.pdf

[19] ——, "Gemini 3 Pro Image: Model Card," Nov. 2025, model card, published November 2025. [Online]. Available: https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Image-Model-Card.pdf

[20] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan et al., "The llama 3 herd of models," arXiv preprint arXiv:2407.21783, 2024.

[21] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder, "Foveated 3d graphics," ACM transactions on Graphics (tOG), vol. 31, no. 6, pp. 1–10, 2012.

[22] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee et al., "Aˆ 3: Accelerating attention mechanisms in neural networks with approximation," in 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2020, pp. 328–341.

[23] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," arXiv preprint arXiv:1510.00149, 2015.

[24] J. Hao, Y. Zhu, T. Wang, J. Yu, X. Xin, B. Zheng, Z. Ren, and S. Guo, "Omnikv: Dynamic context selection for efficient long-context llms," in The Thirteenth International Conference on Learning Representations, 2025.

[25] J. Hartmann, Y. Exner, and S. Domdey, "The power of generative marketing: Can generative AI create superhuman visual marketing content?" International Journal of Research in Marketing, vol. 42, no. 1, pp. 13–31, 2025.

[26] Y. He, F. Chen, Y. He, S. He, H. Zhou, K. Zhang, and B. Zhuang, "Zipar: Parallel auto-regressive image generation through spatial locality," arXiv preprint arXiv:2412.04062, 2024.

[27] Y. He, Y. He, S. He, F. Chen, H. Zhou, K. Zhang, and B. Zhuang, "Neighboring autoregressive modeling for efficient visual generation," in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2025, pp. 19 000–19 010.

[28] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," Advances in neural information processing systems, vol. 33, pp. 6840–6851, 2020.

[29] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami, "Kvquant: Towards 10 million context length llm inference with kv cache quantization," Advances in Neural Information Processing Systems, vol. 37, pp. 1270–1303, 2024.

[30] X. Hu, R. Wang, Y. Fang, B. Fu, P. Cheng, and G. Yu, "Ella: Equip diffusion models with llm for enhanced semantic alignment," arXiv preprint arXiv:2403.05135, 2024.

[31] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford et al., "Gpt-4o system card," arXiv preprint arXiv:2410.21276, 2024.

[32] J. Hwang, D. Kim, S. Lee, Y. Kim, G. Heo, H. Kim, Y. Jeong, T. Meaza, E. Park, J. Ahn et al., "D\'ej\a vu: Efficient video-language query engine with learning-based inter-frame computation reuse," arXiv preprint arXiv:2506.14107, 2025.

[33] J. Hwang, M. Kim, D. Kim, S. Nam, Y. Kim, D. Kim, H. Sharma, and J. Park, "{CoVA}: Exploiting {Compressed-Domain} analysis to accelerate video analytics," in 2022 USENIX annual technical conference (USENIX ATC 22), 2022, pp. 707–722.

[34] H. Jiang, F. Huang, Q. Hu, M. Sun, S. Xiao, Y. Li, J. Lin, J. Yao et al., "Catekv: On sequential consistency for long-context llm inference acceleration," in Forty-second International Conference on Machine Learning, 2025.

[35] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 4401–4410.

[36] D. Kim, J. Hwang, C. Oh, and J. Park, "Mixdit: Accelerating image diffusion transformer inference with mixed-precision mx quantization," IEEE Computer Architecture Letters, vol. 24, no. 1, pp. 141–144, 2025.

[37] M. Kim, S. Hong, R. Ko, S. Choi, H. Lee, J. Kim, J.-Y. Kim, and J. Park, "Oaken: Fast and efficient llm serving with online-offline hybrid kv cache quantization," in Proceedings of the 52nd Annual International Symposium on Computer Architecture, 2025, pp. 482–497.

[38] W. Lee, J. Lee, J. Seo, and J. Sim, "{InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management," in 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), 2024, pp. 155–172.

[39] Y. Lee, B. Pak, J. Hong, and H. Kim, "Tortoise and hare guidance: Accelerating diffusion model inference with multirate integration," Advances in Neural Information Processing Systems, vol. 38, pp. 2871–2898, 2026.

[40] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen, "Snapkv: Llm knows what you are looking for before generation," Advances in Neural Information Processing Systems, vol. 37, pp. 22 947–22 970, 2024.

[41] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han, "Qserve: W4a8kv4 quantization and system co-design for efficient llm serving," Proceedings of Machine Learning and Systems, vol. 7, 2025.

[42] D. Liu, S. Zhao, L. Zhuo, W. Lin, Y. Xin, X. Li, Q. Qin, Y. Qiao, H. Li, and P. Gao, "Lumina-mgpt: Illuminate flexible photorealistic text-to-image generation with multimodal generative pretraining," arXiv preprint arXiv:2408.02657, 2024.

[43] M. Liu, S. Zhang, and C. Long, "Wukong framework for not safe for work detection in text-to-image systems," in Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1, 2026, pp. 891–902.

[44] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyrillidis, and A. Shrivastava, "Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time," Advances in Neural Information Processing Systems, vol. 36, pp. 52 342–52 364, 2023.

[45] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu, "Kivi: A tuning-free asymmetric 2bit quantization for kv cache," arXiv preprint arXiv:2402.02750, 2024.

[46] X. Ma, G. Fang, and X. Wang, "Deepcache: Accelerating diffusion models for free," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2024, pp. 15 762–15 772.

[47] X. Ma, P. Sun, H. Ma, H. Tang, C.-Y. Ma, J. Wang, K. Li, X. Dai, Y. Shi, X. Ju et al., "Token-shuffle: Towards high-resolution image generation with autoregressive models," arXiv preprint arXiv:2504.17789, 2025.

[48] Y. Ma, X. Liu, X. Chen, W. Liu, C. Wu, Z. Wu, Z. Pan, Z. Xie, H. Zhang, X. Yu et al., "Janusflow: Harmonizing autoregression and rectified flow for unified multimodal understanding and generation," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2025, pp. 7739–7751.

[49] Microsoft Azure, "Azure LLM Inference Trace 2025," https://github.com/Azure/AzurePublicDataset, 2025, accessed: 2026-04-22.

[50] Midjourney. (2022) Midjourney. https://www.midjourney.com. Accessed: 2026-05-20.

[51] ——. (2025) Draft and conversational modes. Accessed: 2026-05-20. [Online]. Available: https://docs.midjourney.com/hc/en-us/articles/35577175650957-Draft-Conversational-Modes

[52] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in International conference on machine learning. PMLR, 2021, pp. 8162–8171.

[53] C. Oh, S. Oh, J. Hwang, Y. Kim, H. Sharma, and J. Park, "Neo: Real-time on-device 3d gaussian splatting with reuse-and-update sorting acceleration," in Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, 2026, pp. 1268–1284.

[54] OpenAI. (2025) Addendum to gpt-4o system card: Native image generation. Accessed: 2026-05-20. [Online]. Available: https://cdn.openai.com/11998be9-5319-4302-bfbf-1167e093f1fb/Native_Image_Generation_System_Card.pdf

[55] M. Oren, M. Hassid, N. Yarden, Y. Adi, and R. Schwartz, "Transformers are multi-state rnns," in Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, 2024, pp. 18 724–18 741.

[56] J. Park, E. Amaro, D. Mahajan, B. Thwaites, and H. Esmaeilzadeh, "Axgames: Towards crowdsourcing quality target determination in approximate computing," in Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 623–636.

[57] A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke, and A. Lefohn, "Towards foveated rendering for gaze-tracked virtual reality," ACM Transactions On Graphics (TOG), vol. 35, no. 6, pp. 1–12, 2016.

[58] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in Proceedings of the IEEE/CVF international conference on computer vision, 2023, pp. 4195–4205.

[59] D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach, "Sdxl: Improving latent diffusion models for high-resolution image synthesis," in International Conference on Learning Representations, vol. 2024, 2024, pp. 1862–1874.

[60] Z. Qin, Y. Lv, M. Lin, H. Guo, Z. Zhang, D. Zou, and W. Lin, "Head-aware kv cache compression for efficient visual autoregressive modeling," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 40, no. 30, 2026, pp. 24 982–24 990.

[61] H. Qiu, A. Biswas, Z. Zhao, J. Mohan, A. Khare, E. Choukse, Í. Goiri, Z. Zhang, H. Shen, C. Bansal, R. Ramjee, and R. Fonseca, "Modserve: Modality-and stage-aware resource disaggregation for scalable multimodal model serving," in Proceedings of the 2025 ACM Symposium on Cloud Computing, 2025, pp. 817–830.

[62] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," arXiv preprint arXiv:1511.06434, 2015.

[63] A. Razavi, A. Van den Oord, and O. Vinyals, "Generating diverse high-fidelity images with vq-vae-2," Advances in neural information processing systems, vol. 32, 2019.

[64] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2022, pp. 10 684–10 695.

[65] 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 International Conference on Machine Learning. PMLR, 2023, pp. 31 094–31 116.

[66] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," arXiv preprint arXiv:2010.02502, 2020.

[67] K. Sun, J. Pan, Y. Ge, H. Li, H. Duan, X. Wu, R. Zhang, A. Zhou, Z. Qin, Y. Wang et al., "Journeydb: A benchmark for generative image understanding," Advances in neural information processing systems, vol. 36, pp. 49 659–49 678, 2023.

[68] P. Sun, Y. Jiang, S. Chen, S. Zhang, B. Peng, P. Luo, and Z. Yuan, "Autoregressive model beats diffusion: Llama for scalable image generation," arXiv preprint arXiv:2406.06525, 2024.

[69] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han, "Quest: Query-aware sparsity for efficient long-context llm inference," arXiv preprint arXiv:2406.10774, 2024.

[70] G. Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, R. Merhej, S. Perrin, T. Matejovicova, A. Ramé, M. Rivière, L. Rouillard, T. Mesnard, G. Cideron, J. bastien Grill, S. Ramos, E. Yvinec, M. Casbon, E. Pot, I. Penchev, G. Liu, F. Visin, K. Kenealy, L. Beyer, X. Zhai, A. Tsitsulin, R. Busa-Fekete, A. Feng, N. Sachdeva, B. Coleman, Y. Gao, B. Mustafa, I. Barr, E. Parisotto, D. Tian, M. Eyal, C. Cherry, J.-T. Peter, D. Sinopalnikov, S. Bhupatiraju, R. Agarwal, M. Kazemi, D. Malkin, R. Kumar, D. Vilar, I. Brusilovsky, J. Luo, A. Steiner, A. Friesen, A. Sharma, A. Sharma, A. M. Gilady, A. Goedeckemeyer, A. Saade, A. Feng, A. Kolesnikov, A. Bendebury, A. Abdagic, A. Vadi, A. György, A. S. Pinto, A. Das, A. Bapna, A. Miech, A. Yang, A. Paterson, A. Shenoy, A. Chakrabarti, B. Piot, B. Wu, B. Shahriari, B. Petrini, C. Chen, C. L. Lan, C. A. Choquette-Choo, C. Carey, C. Brick, D. Deutsch, D. Eisenbud, D. Cattle, D. Cheng, D. Paparas, D. S. Sreepathihalli, D. Reid, D. Tran, D. Zelle, E. Noland, E. Huizenga, E. Kharitonov, F. Liu, G. Amirkhanyan, G. Cameron, H. Hashemi, H. Klimczak-Plucińska, H. Singh, H. Mehta, H. T. Lehri, H. Hazimeh, I. Ballantyne, I. Szpektor, I. Nardini, J. Pouget-Abadie, J. Chan, J. Stanton, J. Wieting, J. Lai, J. Orbay, J. Fernandez, J. Newlan, J. yeong Ji, J. Singh, K. Black, K. Yu, K. Hui, K. Vodrahalli, K. Greff, L. Qiu, M. Valentine, M. Coelho, M. Ritter, M. Hoffman, M. Watson, M. Chaturvedi, M. Moynihan, M. Ma, N. Babar, N. Noy, N. Byrd, N. Roy, N. Momchev, N. Chauhan, N. Sachdeva, O. Bunyan, P. Botarda, P. Caron, P. K. Rubenstein, P. Culliton, P. Schmid, P. G. Sessa, P. Xu, P. Stanczyk, P. Tafti, R. Shivanna, R. Wu, R. Pan, R. Rokni, R. Willoughby, R. Vallu, R. Mullins, S. Jerome, S. Smoot, S. Girgin, S. Iqbal, S. Reddy, S. Sheth, S. Põder, S. Bhatnagar, S. R. Panyam, S. Eiger, S. Zhang, T. Liu, T. Yacovone, T. Liechty, U. Kalra, U. Evci, V. Misra, V. Roseberry, V. Feinberg, V. Kolesnikov, W. Han, W. Kwon, X. Chen, Y. Chow, Y. Zhu, Z. Wei, Z. Egyed, V. Cotruta, M. Giang, P. Kirk, A. Rao, K. Black, N. Babar, J. Lo, E. Moreira, L. G. Martins, O. Sanseviero, L. Gonzalez, Z. Gleicher, T. Warkentin, V. Mirrokni, E. Senter, E. Collins, J. Barral, Z. Ghahramani, R. Hadsell, Y. Matias, D. Sculley, S. Petrov, N. Fiedel, N. Shazeer, O. Vinyals, J. Dean, D. Hassabis, K. Kavukcuoglu, C. Farabet, E. Buchatskaya, J.-B. Alayrac, R. Anil, Dmitry, Lepikhin, S. Borgeaud, O. Bachem, A. Joulin, A. Andreev, C. Hardin, R. Dadashi, and L. Hussenot, "Gemma 3 technical report," 2025. [Online]. Available: https://arxiv.org/abs/2503.19786

[71] N. Team, C. Han, G. Li, J. Wu, Q. Sun, Y. Cai, Y. Peng, Z. Ge, D. Zhou, H. Tang et al., "Nextstep-1: Toward autoregressive image generation with continuous tokens at scale," arXiv preprint arXiv:2508.10711, 2025.

[72] P. Tillet, H.-T. Kung, and D. Cox, "Triton: an intermediate language and compiler for tiled neural network computations," in Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, 2019, pp. 10–19.

[73] A. Van Den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural discrete representation learning," Advances in neural information processing systems, vol. 30, 2017.

[74] C. Wang, X. Li, L. Qi, X. Lin, J. Bai, Q. Zhou, and Y. Tong, "Conditional panoramic image generation via masked autoregressive modeling," Advances in Neural Information Processing Systems, vol. 38, pp. 27 654–27 679, 2026.

[75] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in 2021 IEEE international symposium on high-performance computer architecture (HPCA). IEEE, 2021, pp. 97–110.

[76] L. Wang, X. Shi, and Y. Liu, "Foveated rendering: A state-of-the-art survey," Computational visual media, vol. 9, no. 2, pp. 195–228, 2023.

[77] X. Wang, X. Zhang, Z. Luo, Q. Sun, Y. Cui, J. Wang, F. Zhang, Y. Wang, Z. Li, Q. Yu et al., "Emu3: Next-token prediction is all you need," arXiv preprint arXiv:2409.18869, 2024.

[78] Z. J. Wang, E. Montoya, D. Munechika, H. Yang, B. Hoover, and D. H. Chau, "Diffusiondb: A large-scale prompt gallery dataset for text-to-image generative models," in Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers), 2023, pp. 893–911.

[79] C. Wu, J. Li, J. Zhou, J. Lin, K. Gao, K. Yan, S.-m. Yin, S. Bai, X. Xu, Y. Chen et al., "Qwen-image technical report," arXiv preprint arXiv:2508.02324, 2025.

[80] C. Wu, X. Chen, Z. Wu, Y. Ma, X. Liu, Z. Pan, W. Liu, Z. Xie, X. Yu, C. Ruan et al., "Janus: Decoupling visual encoding for unified multimodal understanding and generation," in Proceedings of the Computer Vision and Pattern Recognition Conference, 2025, pp. 12 966–12 977.

[81] xAI. (2024) Grok image generation release. Accessed: 2026-05-20. [Online]. Available: https://x.ai/news/grok-image-generation-release

[82] X. Xiang and F. Tu, "Var-turbo: Unlocking the potential of visual autoregressive models through dual redundancy," in 2026 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2026, pp. 1–16.

[83] X. Xiang and Q. Fan, "Make it efficient: Dynamic sparse attention for autoregressive image generation," arXiv preprint arXiv:2506.18226, 2025.

[84] G. Xiao, J. Tang, J. Zuo, J. Guo, S. Yang, H. Tang, Y. Fu, and S. Han, "Duoattention: Efficient long-context llm inference with retrieval and streaming heads," in International Conference on Learning Representations, vol. 2025, 2025, pp. 37 228–37 253.

[85] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," in International Conference on Learning Representations, vol. 2024, 2024, pp. 21 875–21 895.

[86] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv et al., "Qwen3 technical report," arXiv preprint arXiv:2505.09388, 2025.

[87] X. Yang, S. Shih, Y. Fu, X. Zhao, and S. Ji, "Your vit is secretly a hybrid discriminative-generative diffusion model. arxiv 2022," arXiv preprint arXiv:2208.07791.

[88] H. Yao, R. Zhang, J. Huang, J. Zhang, Y. Wang, B. Fang, R. Zhu, Y. Jing, S. Liu, G. Li et al., "A survey on agentic multimodal large language models," arXiv preprint arXiv:2510.10991, 2025.

[89] P. Yin, J. Zhu, H. Gao, C. Zheng, Y. Huang, T. Zhou, R. Yang, W. Liu, W. Chen, C. Guo et al., "vllm-omni: Fully disaggregated serving for any-to-any multimodal models," arXiv preprint arXiv:2602.02204, 2026.

[90] H. You, Z. Sun, H. Shi, Z. Yu, Y. Zhao, Y. Zhang, C. Li, B. Li, and Y. Lin, "Vitcod: Vision transformer acceleration via dedicated algorithm and accelerator co-design," in 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2023, pp. 273–286.

[91] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang et al., "Big bird: Transformers for longer sequences," Advances in neural information processing systems, vol. 33, pp. 17 283–17 297, 2020.

[92] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett et al., "H2o: Heavy-hitter oracle for efficient generative inference of large language models," Advances in Neural Information Processing Systems, vol. 36, pp. 34 661–34 710, 2023.

[93] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, "Atom: Low-bit quantization for efficient and accurate llm serving," Proceedings of Machine Learning and Systems, vol. 6, pp. 196–209, 2024.

[94] Y. Zhao, D. Wu, and J. Wang, "Alisa: Accelerating large language model inference via sparsity-aware kv caching," in 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA). IEEE, 2024, pp. 1005–1017.

Appendix

1. Abstract

Our artifact provides scripts to reproduce the image quality, latency, and throughput results of diagonal sparse attention and the baseline methods. It contains the source code and a README that describes the exact commands to build the environment, run the experiments, and regenerate Figures 15–17 and Table 2 from the paper.

2. Artifact check-list (meta-information)

- **Algorithm:** Diagonal-aware sparse attention algorithm.
- **Compilation:** CUDA toolkit, PyTorch, Triton
- **Model:** Janus-Pro-1B, Janus-Pro-7B, Lumina-mGPT-7B-512
- **Data set:** GenEval, DPG-Bench
- **Run-time environment:** Docker with NVIDIA Container Toolkit; base image: nvcr.io/nvidia/pytorch:23.12-py3
- **Hardware:** NVIDIA RTX A6000(48GB), Intel Xeon Gold. Evaluators need Ampere-or-newer GPU with at least 48GB memory.
- **Execution:** End-to-end bash scripts; environment initialization, experiments, figure generation.
- **Metrics:** Image quality(GenEval score, DPG-bench score), Latency, Throughput(img/s)
- **Output:** Generated images, figures and table.
- **Experiments:** Figure 15, 16, 17 and Table 2 reported in the paper.
- **How much disk space required (approximately)?:** 80GB free storage.
- **How much time is needed to prepare workflow (approximately)?:** 1 hr
- **How much time is needed to complete experiments (approximately)?:** 1 week for quality experiments(Table 2), 1.5 hr for speedup experiment(Fig 15-17).
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** Creative Commons Attribution 4.0 International, MIT License
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.21712898

3. Description

3.1. How to access. Clone our public Github repository:

```
$ git clone \
https://github.com/casys-kaist/DiagonalAttn
```

3.2. Software dependencies. We distribute the artifact as a Docker image, so evaluators need Docker and the NVIDIA Container Toolkit. We assume the host has CUDA 12.3, matching the version used by the provided base image. Python dependencies are installed with uv, using a provided initialization script.

3.3. Datasets. The prompt datasets for GenEval and DPG-bench are already included in the repository.

3.4. Models. We provide a helper script that automatically downloads all required model weights.

4. Installation

❶ **Set up the Docker Container.** Before launching the Docker container, set the correct `HOST_CACHE_DIR` path in `launch.sh`.

```
$ cd DiagonalAttn/docker
$ sh launch.sh
```

❷ **Initialize environment.** After attaching to the running container, set the environment variables and install the required packages.

```
$ source /workspace/setup/env.sh
$ source /workspace/setup/uv_env.sh
```

❸ **Download models.** Download the model weights from Hugging Face. Evaluators should log in to Hugging Face before downloading.

```
$ hf auth login
$ cd /workspace
$ bash setup/download_models.sh
```

5. Experiment workflow

Quality Experiment. This experiment reproduces Table 2, comparing image quality across the baseline methods and diagonal attention. The scripts first generate images for the baseline methods and diagonal attention. Then, it aggregates the score results written in `/workspace/results` and render them into table. Evaluating all prompts(`--ratio 1.0`) for quality takes multiple GPU-weeks; it may take ∼40 days on 4 RTX A6000 GPUs. We provide `--ratio` to control the fraction of prompts evaluated. The results reported in the paper used `--ratio 1.0`.

```
$ bash evaluation/quality/run_table2.sh \
       --all-cells --gpus 0,1,2,3 --ratio 0.1
$ python evaluation/quality/render_table2.py
```

Speedup Experiment. Each script generates a PDF for its corresponding figure among Figures 15, 16, and 17. For Figure 15, `fig15_quality.sh` must be run first to produce the quality-axis results, which `fig15.sh` then joins with the latency-axis measurements. The script then measures the speedup and automatically renders the figure as a PDF.

```
$ bash speedup/figure/fig15_quality.sh \
       --gpus 0,1,2,3 --ratio 0.1
$ LOCK_GPU_CLOCKS=1 bash speedup/figure/fig15.sh
$ LOCK_GPU_CLOCKS=1 bash speedup/figure/fig16.sh
$ LOCK_GPU_CLOCKS=1 bash speedup/figure/fig17.sh
```

6. Evaluation and expected results

After running the scripts, the artifact generates the table and figures under `./results`, matching the expected results below:

- Table 2
  - results/table2.md
- Figure 15
  - results/figures/fig15.pdf

- Figure 16
  - results/figures/fig16a.pdf
  - results/figures/fig16b.pdf
- Figure 17
  - results/figures/fig17.pdf

The PDF figures and Markdown table correspond to the plots and table reported in the paper. In addition to these final outputs, each experiment also stores raw logs under `results/logs` and `results/speedup`, and image outputs under `results/images`.

7. Experiment customization

We provide several input parameters for customizing the experiments.

`evaluation/quality/run_table2.sh`

- `--gpus`: CUDA GPU IDs to round-robin parallel execution of each cells.
- `--models`: Restrict to specific model (`janus pro1b`, `janus7b`, `lumina`).
- `--benchmarks`: Restrict to specific benchmarks (`geneval`, `dpg`).
- `--methods`: Restrict to specific attention methods (`dense`, `alisa`, `h2o`, `tova`, `window`, `streamingllm`, `diagonal`, `diagonal_sink4`).
- `--sparsity`: Restrict to specific sparsity levels (e.g. `0.950`).
- `--ratio`: Override the benchmark prompt ratio for reduced runtime.

`evaluation/quality/render_table2.py`

- `--ratio`: Select which recorded benchmark-ratio tier's rows to render.
- `--precision N`: Number of decimal places shown in the rendered table.
- `--out-md, --out-tex`: Output table Markdown/LaTex table paths.

`speedup/figure/fig15_quality.sh`

- `--gpus`: CUDA GPU IDs to parallelize the GenEval quality cells across.
- `--ratio`: Subsample GenEval prompts to for reduced runtime.

8. Notes

More information can be found in the `README` file of each directory.