Benchmarking Sparse Attention for Efficient Image and Video Synthesis

Benjamin Chauhan
Duke University, ECE 590: ML Systems

Abstract

The quadratic computational complexity $O(N^2)$ of self-attention is the primary bottleneck preventing real-time, high-fidelity video generation—a capability essential for world models in embodied AI and robotics. This report presents a two-phase empirical study benchmarking sparse attention mechanisms for image and video synthesis in Diffusion Transformers (DiTs). In Phase 1, we implement custom Triton kernels for Block Sparsity, Adaptive Dynamic Sparse Attention (ADSA), and Re-ttention on the DiT-XL-2-256 model and an NVIDIA A6000 GPU, finding that naive kernel implementations fail to deliver wall-clock speedups and produce significant quality degradation. Recognizing the critical importance of production-quality kernels and Hopper-native hardware features, we pivot in Phase 2 to benchmark state-of-the-art production kernels—PISA, SparseVideoGen2 (SVG2), and Video Sparse Attention (VSA)—on an NVIDIA H100 GPU with the Flux.1 [dev] image model and the WAN 2.1 video model. We also include NVIDIA SANA as a non-sparse linear-attention baseline. Our results show that VSA achieves the lowest latency at large sequence lengths (63 ms at 2^{17} tokens versus 225 ms for FlashAttention-3), while SVG2 incurs unexpectedly high VRAM overhead due to k-means clustering buffers. In video generation, sparse methods collectively reduce denoising step time by up to 65% over dense attention, with SVG2 and VSA achieving 19 s and 22 s per step respectively versus 36 s for dense, at the cost of visible quality degradation under default hyperparameters. Due to compute budget constraints, video quality is assessed qualitatively and via attention-only microbenchmarks; full quantitative video metrics (FVD, VBench) remain as future work. We discuss how these findings confirm, partially contradict, and extend the theoretical expectations from each paper's claims, and provide a roadmap for further systematic tuning.

1 Introduction

Real-time, high-fidelity video generation represents a fundamental challenge in modern AI systems. World models—generative models capable of simulating future environment states—are emerging as a cornerstone of embodied intelligence, enabling robots and autonomous agents to mentally simulate the consequences of their actions before executing them [13]. However, the dominant architecture for high-quality video synthesis, the Diffusion Transformer (DiT) [1], suffers from an intrinsic $O(N^2)$ bottleneck in its self-attention layers. At 720p resolution with even modest video durations, token counts quickly reach 10^5 or more, making the attention matrix computation not merely expensive but prohibitive for interactive or real-time deployment.

As a concrete illustration of the problem, Hunyuan-Video [4] requires **16 minutes** to generate a 5-second 720p video on a high-end H100 GPU even with FlashAttention-3 [3]. Analysis of its FLOP distribution reveals that at 115k tokens, attention computation consumes approximately 86% of all floating-point operations, compared to roughly 5% for QKV projection and 9% for the feed-forward network (FFN) [4]. This stark imbalance underscores that optimizing the attention mechanism is both necessary and sufficient for the bulk of latency reduction.

Sparse attention methods address this by computing only a structured or dynamic subset of the full $N \times N$ attention matrix. However, a persistent gap exists between theoretical FLOP reductions and realized wall-clock speedups: generating a sparse mask and dispatching computation to non-contiguous memory regions can introduce overhead that erases the savings unless the kernel is carefully engineered to exploit hardware-level features such as Tensor Memory Accelerator (TMA) and WGMMA instructions found on NVIDIA Hopper GPUs.

This report chronicles a complete research arc from first principles to production-scale evaluation. We begin with custom Triton kernel implementations (Phase 1) on an A6000 GPU, establishing what *does not* work at scale and why. We then pivot to evaluating production-quality sparse kernels (Phase 2) on H100 hardware with modern video and image generation models, and analyze our results against each method's theoretical claims.

2 Background and Related Work

2.1 The Attention Bottleneck in DiTs

The standard self-attention operation is:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \qquad (1)$$

where $Q,K,V \in R^{N\times d}$ are query, key, and value matrices of sequence length N and head dimension d. Computing

$QK^\top$ alone requires $O(N^2 d)$ multiply-add operations and $O(N^2)$ memory for the attention matrix. FlashAttention-2 [2] and FlashAttention-3 [3] minimize memory IO via tiled SRAM computation, but do not reduce the fundamental FLOP count. Sparse attention methods fundamentally reduce this count by replacing the dense attention matrix with a sparse mask $\mathbf{M} \in \{0,1\}^{N\times N}$:

$$\text{SparseAttn}(Q,K,V) = \text{softmax}\left(\frac{QK^\top \odot \mathbf{M}}{\sqrt{d_k}}\right)V \quad (2)$$

The key challenge is constructing $\mathbf{M}$ efficiently, contiguously, and with minimal loss of attention quality.

2.2 Phase 1 Methods: Custom Triton Kernels (A6000)

Block Sparsity. Block sparsity (also called Fixed Pattern Sparsity) partitions the $N \times N$ attention matrix into a grid of $B \times B$ blocks and pre-selects a subset of these blocks for computation, typically restricted to a sliding local window. Because memory access patterns are entirely data-independent and contiguous within each block, this approach is maximally GPU-friendly in principle. The implementation is straightforward: project all tokens into Q, K, V as usual, then mask out non-selected blocks before the softmax. A Triton kernel dispatches one block per GPU thread-block, loading tile-sized chunks of Q and K from HBM into SRAM and computing $QK^\top$ locally, then writing back the masked, softmax-weighted sum into the output.

The fundamental limitation is its *static, data-agnostic* nature. Tokens are bounded to local spatial patches; without explicit global or strided tokens, the model cannot propagate information across distant positions. In image generation via DiT, this manifests as a failure to reconcile locally generated texture patches into globally coherent objects. Our experiments with window size 32 and block size 64 confirmed this: with FID rising from 14.33 to 35.87, the model generated coherent local textures but failed to preserve global object identity, famously generating a fish in place of a human head because the attention window was too small to "see" the held fish when generating the face region.

Adaptive Dynamic Sparse Attention (ADSA). ADSA [11] was originally proposed for *autoregressive* image generation models to handle the differing entropy characteristics of image vs. text tokens. Its design assumes a causal, left-to-right generation process with an explicit KV cache: it classifies cached tokens into three semantic zones—a *Prefix* region (global style and layout tokens, kept entirely), a *Local* region (recent L tokens providing texture continuity), and a *Previous* region (older context). Rather than using a fixed mask, ADSA applies TopK filtering based on value feature similarity: for each query, it ranks all previous-region KV pairs by their V feature cosine similarity and retains only the top k fraction (controlled by `topk_ratio`).

Architectural mismatch with DiT. A critical caveat is that DiT-XL-2-256—and diffusion transformers generally—perform *bidirectional* full attention over all tokens simultaneously, not autoregressive generation with a KV cache. ADSA's Prefix/Local/Previous zone taxonomy is a construct for causal decoding order and has no natural analog in the non-causal diffusion setting. We adapted ADSA to DiT by treating the token sequence as ordered (rasterized patch order) and imposing the zone structure artificially; a departure from the intended use case. The results should therefore be interpreted as an evaluation of the *sparse filtering heuristic* applied to a diffusion model rather than as a faithful ADSA deployment. We include it here to characterize the cost of this domain transfer and to provide a concrete example of the train-test mismatch problem when applying autoregressive sparse methods to bidirectional architectures.

The expected behavior under its native autoregressive setting is a graceful quality-efficiency tradeoff: at `topk_ratio` $= 0.35$, approximately 65% of Previous-region KV pairs are discarded while the most semantically relevant ones are retained. However, in our adaptation with `prefix_len=16` and `local_len=64`, we observed FID of 136.46—a severe degradation driven by two compounding factors: (1) the artificial zone structure does not reflect any true semantic organization of DiT's bidirectional attention patterns, and (2) DiT-XL-2-256 was trained with full attention, so the altered information flow cannot be compensated without fine-tuning on the sparse distribution.

Re-ttention. Re-ttention [12] addresses the *attention collapse* problem that arises when sparsity exceeds $\sim$90%. The issue is a distributional shift in the softmax normalization: removing 90% of KV pairs deflates the denominator of the softmax, artificially inflating remaining attention weights and destroying spatial layout. Re-ttention's solution is the *Attention Statistical Reshape* (ASR) mechanism. During the early (dense) denoising steps, the method caches the per-head softmax denominator (the log-sum-exp value ℓ) and the full-attention output distribution. In later sparse steps, it rescales the sparse attention output by the ratio of cached dense denominators to sparse denominators, effectively correcting the probabilistic normalization shift without retraining.

Our implementation produced catastrophically degraded outputs (FID 331.19 at $s = 0.9$) due to two compounding bugs that were revealed after Phase 1. First, the step-tracking state machine was driven by raw `__call__` invocations rather than global denoising step index; since DiT-XL-2-256 contains 28 transformer blocks, the counter advanced 28$\times$ per denoising step, causing the model to believe it had already exited the dense warm-up phase after the very first step. As a result, no valid denominator cache was ever populated and every step ran with an uncorrected sparse mask. Second, classifier-free guidance doubles the batch dimension, and the denominator caching logic was

not separated for conditional vs. unconditional paths, further corrupting the rescaling factors. Fixing these issues is the prerequisite for any meaningful Re-ttention evaluation and is left as future work as the benchmark shifted towards production kernels.

2.3 Phase 2 Methods: Production Kernels (H100)

The failure of naive Triton kernels to deliver speedups—Block Sparsity ran at 4727 ms vs. 341 ms for dense, a 14× *slowdown*—motivated a complete pivot. Simple masking implementations do not reduce memory allocation (VRAM was identical at 2.09 GB across all methods) and introduce significant kernel launch overhead without exploiting Hopper-native features. We transitioned to an H100 (80 GB HBM3) and evaluated production-grade kernels specifically optimized for SM90 (Hopper) through Tensor Memory Accelerator (TMA) prefetching and WGMMA (Warpgroup Matrix Multiply-Accumulate) instructions.

PISA: Piecewise Sparse Attention. PISA [5] represents a fundamental departure from the standard "keep-or-drop" paradigm. The key insight from the PISA paper is that non-critical attention blocks do not carry *zero* information—they exhibit *distributional stability* across similar token neighborhoods. Rather than discarding these blocks, PISA approximates them via block-wise first-order Taylor expansion of the softmax function.

Formally, given sorted attention blocks, PISA partitions them into "exact" (critical) blocks $\mathscr{S}_{\text{exact}}$ and "approximate" (non-critical) blocks $\mathscr{S}_{\text{approx}}$. For the exact set, standard FlashAttention computation is applied. For the approximate set, PISA derives a lightweight rank-1 approximation using block centroids and covariance-aware routing. The routing strategy is derived from an error analysis that minimizes the Frobenius norm of the approximation error, preferring to assign blocks with high inter-block similarity to the approximate path.

In terms of the attention map visualization shown in Figure 1, PISA achieves "Active Blocks: 100%" with only 20.4% of FLOPs—compared to naive sparse attention which uses 20% of FLOPs but leaves 80% of blocks entirely empty (L1 error 10.34%) and causes severe artifacts. PISA achieves L1 error of only 1.36% at the same computational budget. Its Hopper kernel uses TMA for double-buffered async memory prefetching, keeping tensor cores occupied during the approximate path's centroid lookups.

Expected tradeoffs: PISA should excel in moderate-sparsity regimes ($s \in [0.25, 0.5]$) where approximation error is low and speedup is meaningful. At high sparsity ($s \geq 0.75$), the Taylor expansion will likely become less accurate as block attention distributions diverge from their centroids. It should show minimal quality loss at low sparsity and graceful degradation at high sparsity—a distinctly better quality-efficiency curve than hard-drop sparse attention.

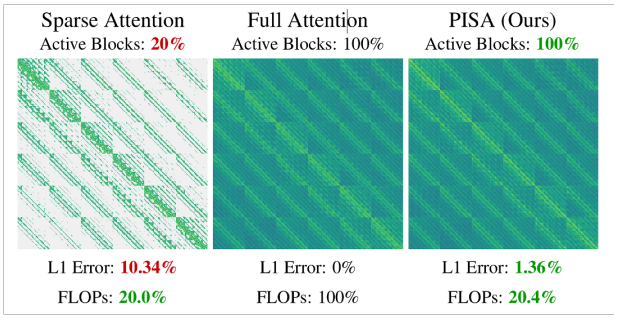


Figure 1: PISA attention block visualization. Left: Standard sparse attention at 20% density leaves 80% blocks empty (L1 error 10.34%). Center: Full attention. Right: PISA at 20% FLOPs covers 100% of blocks via Taylor approximation (L1 error 1.36%).

SparseVideoGen2 (SVG2). SVG2 [6] targets the two core failures of earlier video sparse attention methods: (1) inaccurate critical token identification via position-based clustering, and (2) computational waste from scattered sparse patterns. It introduces *Semantic-Aware Permutation* (SAP) as its central mechanism. SAP applies k-means clustering to all tokens in the attention head based on their query feature vectors, grouping semantically similar tokens together in memory. Once permuted, top-p dynamic budget control selects the most important clusters to compute full attention within. Because semantically similar tokens are now contiguous in memory, the GPU operates on dense, padded blocks rather than scattered indices—eliminating the padding waste that plagues index-based sparse attention.

The process has three stages per attention call: (1) run k-means on the query centroids (reported overhead: $< 1\%$ of step time in the paper), (2) compute centroid-to-centroid attention scores to determine the top-p inter-cluster attention mask, (3) dispatch the permuted attention computation using a custom CUDA kernel. SVG2 achieves up to 2.30× speedup on HunyuanVideo and 1.89× on WAN 2.1 in the paper's evaluation.

Expected tradeoffs: The k-means step, while lightweight on large sequences, is non-trivial at short sequences where its overhead is not amortized. More critically, the clustering requires storing all query centroid features and a permutation buffer, contributing to elevated VRAM overhead—a key discrepancy we observe between paper claims and our measurements. Quality should be high because semantic clustering ensures that tokens attending to each other are genuinely similar, but the fixed-k clustering may struggle with highly dynamic or heterogeneous attention patterns in complex videos.

Video Sparse Attention (VSA). VSA [7] is the only *trainable* method in our Phase 2 evaluation, designed to replace full attention at both training and inference. Its core architecture is a *two-stage hierarchical attention*:

In the **coarse stage**, tokens are pooled into 3D spatio-

temporal cubes of fixed size $(4,4,4)$ in the temporal, height, and width dimensions. A lightweight coarse attention is computed over these cube representations to identify "critical" cubes—those carrying the most attentional mass. This coarse stage produces a *gated output* O_c that provides global context through broadcasting.

In the **fine stage**, token-level attention is computed only within the top-K critical cubes identified by the coarse stage. This fine output O_f provides high-frequency local detail. The final output is a learned gate combination $G \cdot O_c + (1-G) \cdot O_f$, allowing the model to balance global context and local precision.

Because the cube-to-tile mapping is deterministic and the top-K selection is differentiable (via a relaxed sort during training), VSA compiles to a single fused CUDA kernel that avoids any conditional branching at inference time. The kernel achieves 85% of FlashAttention-3 MFU (Memory-bandwidth Utilization). In the paper's evaluation, retrofitting Wan-2.1 1.3B with VSA via fine-tuning reduces end-to-end inference from 31s to 18s ($1.7\times$) on H100.

Expected tradeoffs: VSA's trainable nature means it requires fine-tuning the target model, creating a dependency on training infrastructure. Crucially, when applied directly to a pretrained full-attention model without fine-tuning (as in our inference-only experiments), VSA will produce artifacts because the model's weights encode expectations about full-attention distributions. Tile size is a critical hyperparameter: larger tiles blur the attention pattern while smaller tiles increase kernel launch overhead; the paper recommends $(4,4,4)$ as an optimal configuration.

NVIDIA SANA: Linear Diffusion Transformer. SANA [8] is included as a *non-sparse alternative baseline* representing the "redesign" approach rather than the "retrofit sparse" approach. SANA replaces standard softmax attention entirely with a linear attention mechanism (cost $O(N)$) and combines it with a $32\times$ deep compression autoencoder, Mix-FFN (using depthwise 3×3 convolutions instead of pure linear layers), and a complex human instruction module for prompt enhancement. Unlike the sparse methods benchmarked here, SANA requires full retraining from scratch and cannot be applied as a drop-in to pretrained models. Its linear attention is implemented using a ReLU-based kernel (not softmax), which fundamentally alters the attention distribution and may lose the ability to represent highly peaked attention patterns required for fine-grained texture detail.

Important scope caveat. The SANA 0.6B model is compared against the 12B Flux.1 [dev] model in our latency measurements—a $20\times$ parameter-count difference. The $32.7\times$ step-time reduction observed for SANA on the video pipeline and the $8\times$ reduction on the image pipeline should therefore *not* be interpreted as speedups achievable by applying linear attention to Flux.1 or WAN 2.1. SANA is presented solely to illustrate the upper bound of the "redesign from scratch" strategy relative to sparse-retrofit approaches on equivalent hardware; direct efficiency comparisons between SANA and the sparse methods in this study are not meaningful.

3 Test Harness and Methodology

3.1 Phase 1: Custom Triton Kernels on A6000

Hardware. All Phase 1 experiments ran on a single NVIDIA RTX A6000 GPU (Ampere SM86, 48 GB GDDR6 ECC) on the Duke University CS compute cluster. The Python environment consisted of Python 3.11, PyTorch 2.6.0, and OpenAI Triton 3.x.

Model and Dataset. We used the `facebook/DiT-XL-2-256` pretrained checkpoint from HuggingFace, evaluated on 10,000 class-conditional samples from ImageNet-1k (256×256 resolution). FID was computed against the full ImageNet-1k validation set using the standard Inception-v3 pipeline.

AttentionProcessor Swapping. The `diffusers` library exposes a standardized `AttentionProcessor` interface that decouples the attention computation from the model architecture. Each DiT block contains a `BasicTransformerBlock` with a configurable `attn_processor` field. We implemented custom `BlockSparseAttnProcessor`, `ADSAAttnProcessor`, and `RettentionAttnProcessor` classes, each inheriting from `AttnProcessor2_0` and overriding the `__call__` method with the sparse computation. At inference time, we called `model.set_attn_processor(SparseProcessorDict)` to hot-swap all attention layers without modifying model weights. This design allowed us to use identical model checkpoints for all methods and change only the attention computation graph.

Triton Kernel Design. For each method, we wrote Triton kernels operating at the block tile level. Kernel dispatch followed the standard FlashAttention tiling loop structure: outer loop over query tiles (each assigned to a CUDA thread-block), inner loop over key/value tiles (iterated in SRAM). The sparse mask was computed in Python on CPU and transferred to GPU before the kernel call, adding a non-trivial preprocessing overhead that we did not optimize at this stage. Profiling was captured via W&B and Hydra configuration management, recording mean/p95/p99 latency, peak VRAM allocation (via `torch.cuda.max_memory_allocated`), and FID.

Attention-Only Microbenchmarks. To isolate kernel performance from model overhead, we also ran a standalone attention benchmark: for a single transformer block, we measured forward pass latency across randomly generated Q, K, V tensors at varying sequence lengths, with no model weights or diffusion pipeline involved. These microbenchmarks directly tested the raw kernel speed.

3.2 Phase 2: Production Kernels on H100

Hardware. Phase 2 used a single NVIDIA H100 (80 GB HBM3, SM90 Hopper) accessed via the `brev.dev` cloud platform, enabling Hopper-native TMA and WGMMA instructions unavailable on the A6000.

Image Model: Flux.1 [dev]. Flux.1 [9] is a 12B-parameter flow-matching-based DiT from Black Forest Labs operating in a compressed latent space. Unlike DiT-XL, Flux.1 uses a dual-stream transformer with separate text and image streams, making it a significantly more challenging testbed. The text-to-image pipeline generates 1024×1024 images over 20–28 denoising steps. Sequence lengths at this resolution reach approximately 4096 image tokens plus ˜256 text tokens.

Video Model: WAN 2.1. WAN 2.1 [10] is an open-source video DiT available in 1.3B and 14B variants. It uses a 3D full-attention over flattened spatio-temporal patches, with sequence lengths reaching $>$50k tokens for 720p video generation. We benchmarked the 1.3B variant given H100 single-GPU memory constraints.

Macro and Micro Benchmarks. For macro evaluation, we ran the full generation pipeline and recorded the mean denoising step time (averaged across all steps). For micro benchmarks, we measured isolated attention forward-pass latency across sequence lengths $\{2^{12}, 2^{13}, 2^{14}, 2^{15}, 2^{16}, 2^{17}\}$ with a fixed head configuration, measuring avg latency (ms), peak VRAM (MB), and throughput (tokens/sec).

Compute budget limitations. Video generation on a single H100 with WAN 2.1 1.3B is resource-intensive; end-to-end generation of even short clips exhausts the available cloud budget when run at the scale required for statistically meaningful quantitative metrics. Consequently, the video results in Section 5.5 are based on mean denoising step time (macro latency) and qualitative single-frame inspection. Full quantitative video quality metrics—FVD, VBench, and per-frame PSNR/SSIM—are left as future work and represent the most important gap in the current evaluation.

4 Phase 1 Results: Triton Kernels on A6000

4.1 Quantitative Image Generation Results

Table 1 shows the quantitative results for our Phase 1 evaluation on DiT-XL-2-256 with 10,000 ImageNet-1k samples.

VRAM Discussion. Identically 2.09 GB for all methods reveals a fundamental limitation: our sparse implementations *mask* the attention matrix in PyTorch (setting entries to $-\infty$ before softmax) rather than physically skipping the allocation. The sparse mask sits alongside the full $N \times N$ tensor, meaning no memory savings are realized. True memory reduction requires a kernel that never materializes masked blocks, which our prototype kernels did not achieve.

Table 1: Phase 1 image generation results. DiT-XL-2-256, 10k ImageNet-1k samples, single NVIDIA RTX A6000. FID ↓ is better.

Method	Peak VRAM	Latency (ms)	FID ↓
Full Attention (FA2)	2.09 GB	340.92	14.33
Block Sparsity	2.09 GB	4727.79	35.87
ADSA (adapted)	2.09 GB	454.71	136.46
Re-ttention (s=0.9)	2.19 GB	299.33	331.19

Latency Discussion. Block Sparsity's 4727 ms is particularly striking. Each block selection required CPU-side mask construction (a Python loop over grid positions) followed by a GPU memory transfer before the Triton kernel launched. The Triton kernel itself had no pipelining or TMA prefetching, making it memory-bound. The result is 13.9× *slower* than dense FA2—demonstrating that hardware-unaware sparsity is not merely "not helpful" but actively harmful. ADSA at 454.71 ms is slower than dense but far less catastrophic, because the TopK filtering is implemented as a PyTorch gather operation (GPU-native) rather than a Python loop. Re-ttention's 299.33 ms appears faster than ADSA—but this is because the step-counter bug caused the model to frequently fall back toward full attention, meaning the sparse mask was only partially applied throughout the run.

FID Discussion. Block Sparsity's FID of 35.87 is a 2.5× degradation from baseline. The qualitative evidence is illuminating: the model generates coherent local textures within each attention window but fails to compose globally. The notorious "fish head" artifact (Figure 2) arises because when generating the upper-body region, the local window does not reach the already-generated lower-body tokens showing the held fish, so the model hallucinates a fish in the face position. The ADSA FID of 136.46 is the compounded result of both the train-test mismatch (DiT was trained with full attention) and the architectural domain mismatch described in Section 2: the Prefix/Local/Previous zone structure was designed for causal decoding order and does not correspond to any semantic structure in DiT's bidirectional attention. Re-ttention's FID of 331.19 is essentially a total failure attributable to the implementation bugs described in Section 2: the step counter incremented at the transformer-block level rather than the denoising-step level, so the dense warm-up period was never completed and no valid denominator cache was ever written, leaving every denoising step to run with a fully uncorrected sparse mask.

4.2 The Kernel Overhead Problem

These results collectively confirm the central lesson of Phase 1: **software-level sparsity in Python/Triton prototype form does not deliver wall-clock speedups**. The bottleneck is not the mathematical sparsity but the overhead

(a) Full Attn **(b)** ADSA **(c)** Block Sparse **(d)** Re-ttention

Figure 2: Phase 1 qualitative comparison on DiT-XL-2-256 (ImageNet class 1: goldfish). Block sparse produces a fish in place of the subject's head due to the limited attention window. Re-ttention produces a near-blank image due to the step-counter bug eliminating the dense warm-up. ADSA (adapted from autoregressive to bidirectional DiT) produces a recognizable but semantically incorrect subject.

of mask construction, memory allocation, and unoptimized kernel dispatch. Production sparse attention requires co-design of the sparsity pattern, the memory layout, and the CUDA kernel at a microarchitecture level. This motivated the transition to production kernels in Phase 2.

5 Phase 2 Results: Production Kernels on H100

5.1 Microbenchmark: Latency vs. Sequence Length

Figure 3 shows the forward latency (ms) as a function of sequence length from 2^{12} to 2^{17} tokens, comparing FlashAttention-3 (FA3, dense baseline), SVG2, VSA, and random sparsity (a structurally simple sparse baseline).

At short sequences (2^{12}–2^{14} tokens), all methods are roughly equivalent ($<$10 ms), which is expected: the sparse overhead (mask construction, k-means clustering for SVG2, coarse-stage pooling for VSA) is not yet offset by the FLOP reduction at short sequence lengths. This matches the PISA paper's observation that sparse attention is inefficient at fewer than $\sim$4k tokens.

The divergence begins at 2^{15} (32k tokens) and becomes dramatic at 2^{16}–2^{17}. At 2^{17} tokens (the rightmost data point):

- **FA3 (dense)**: 225 ms
- **SVG2**: 160 ms (1.41$\times$ speedup)
- **Random**: 117 ms (1.92$\times$ speedup)
- **VSA**: 63 ms (3.57$\times$ speedup)

VSA's dominance is consistent with the paper's claims of retaining 85% of FA3 MFU: its cube-partitioned kernel maps directly to contiguous block-sparse computation with no runtime dispatching overhead. SVG2's latency is higher than random sparsity despite its semantic-aware design, likely because the k-means clustering step at 2^{17} tokens is itself expensive, partially eroding the speedup. The paper reports SVG2's overhead as $< 1\%$ of step time, but this was benchmarked on the full video generation pipeline where attention dominates; in our attention-only microbenchmark, the clustering overhead is more visible.

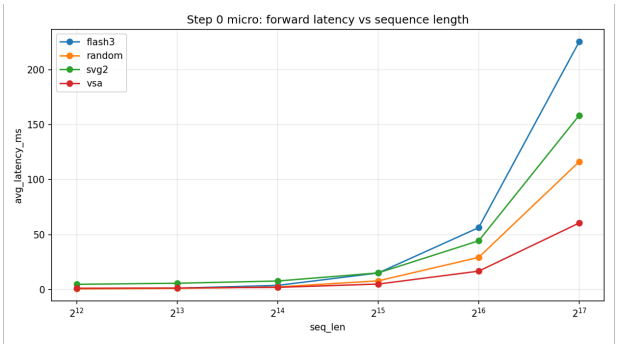


Figure 3: Microbenchmark: avg latency (ms) vs. sequence length for FA3, SVG2, VSA, and random sparsity. VSA achieves 3.57$\times$ speedup at 2^{17} tokens. Divergence begins at 2^{15} (32k tokens).

5.2 Microbenchmark: Peak VRAM vs. Sequence Length

Figure 4 reveals a surprising finding: **SVG2 uses dramatically more VRAM than any other method**, reaching $\sim$6.8 GB at 2^{17} tokens versus 1.05 GB for FA3. This directly contradicts the expectation that sparse methods save memory.

The root cause is SVG2's clustering buffers. The k-means algorithm must store all token feature vectors ($N \times d$) plus cluster assignment arrays (N) plus centroid matrices ($k \times d$) throughout the attention computation. At 2^{17} tokens with head dimension 128, this totals $\sim$5.5 GB of auxiliary storage—dwarfing the attention computation itself. VSA is more memory-efficient (2.75 GB at 2^{17}), as its cube pooling is a reduction operation that does not require storing the full token set in a separate buffer. FA3, random, and PISA all scale roughly linearly from $\sim$50 MB at 2^{12} to $\sim$1.05 GB at 2^{17}.

For video generation, where baseline VRAM usage already approaches 60–70 GB on an H100, SVG2's $\sim$5$\times$ memory overhead is a significant practical concern.

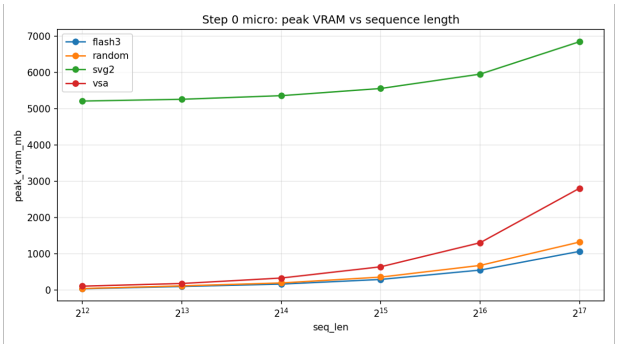


Figure 4: Microbenchmark: peak VRAM (MB) vs. sequence length. SVG2 reaches $\sim$6.8 GB at 2^{17} tokens due to k-means clustering buffers, far exceeding all other methods.

5.3 Microbenchmark: Throughput vs. Sequence Length

Figure 5 shows tokens/sec as a function of sequence length. Throughput peaks at intermediate sequence lengths (2^{13}–2^{14}) for all methods and declines at longer sequences as the kernel becomes memory-bandwidth bound. VSA peaks at ∼8.3M tokens/sec at 2^{14}, matching random sparsity. FA3 peaks at ∼7M tokens/sec at 2^{13} before degrading sharply. SVG2 peaks last (at 2^{14}) with ∼2.1M tokens/sec, constrained by clustering overhead. At 2^{17}, VSA and random maintain ∼2.2M tokens/sec versus FA3's 0.75M tokens/sec—confirming the latency speedup from a throughput perspective.

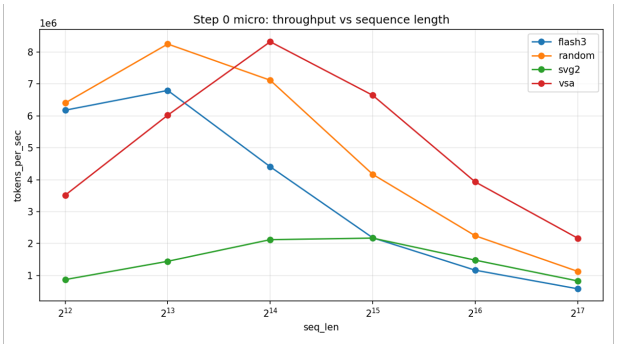


Figure 5: Microbenchmark: throughput (tokens/sec) vs. sequence length. VSA peaks at ∼8.3M tokens/sec at 2^{14}; SVG2 is constrained by clustering overhead across all lengths.

5.4 Image Generation: Flux.1 [dev]

Step Time vs. Sparsity. Figure 6 (left) shows mean step time as a function of sparsity for PISA and random sparsity on Flux.1. Both methods exhibit the expected monotonically decreasing latency trend: higher sparsity (more blocks dropped) reduces computation time. PISA ranges from 232 ms (s=0.15) to 206 ms (s=0.9), while random ranges from 241 ms to 205 ms. The two curves converge at high sparsity because at s=0.9, both methods compute nearly identically few blocks—the approximation scheme in PISA provides no further advantage when so few blocks are non-critical.

The absolute speedup over dense is modest (∼1.15×). This is below PISA's reported 1.2× on Flux.1, and far below SVG2's 1.89× on WAN 2.1. Two factors explain this: (1) Flux.1 operates at shorter sequence lengths (4096 tokens) than video models (50k+ tokens), where sparse attention overhead is not yet amortized; (2) Flux.1 uses a dual-stream architecture where text tokens always attend to all image tokens, limiting the effective sparsity budget in the joint attention layers.

Step Time by Method. Figure 6 (right) shows step time grouped by sparsity for each method including SANA linear attention. SANA (linear) achieves dramatically lower step time of ˜26 ms, but as noted in Section 2, this figure represents a separate 0.6B model rather than a speedup applied to Flux.1 (12B), so the gap is not a meaningful efficiency comparison. Both PISA and random sparsity show very similar step times across the $s \in [0.15, 0.9]$ range, suggesting that Flux.1's attention patterns are not sufficiently sparse (i.e., do not have a clear block structure) for PISA's approximation to gain significant advantage over random dropping.

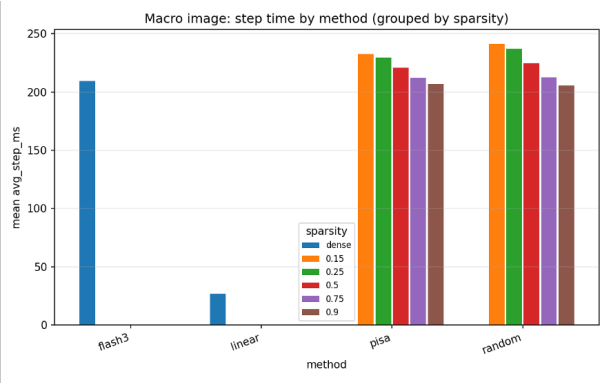


Figure 6: Macro image benchmark: mean denoising step time (ms) on Flux.1 [dev]. Left: PISA vs. random sparsity step time vs. sparsity ratio. Both decrease monotonically. Right: step time by method grouped by sparsity. Note that SANA (26 ms) reflects a separate 0.6B model and is not directly comparable to the 12B Flux.1 sparse results.

Qualitative Analysis. Figure 7 shows the PISA sparsity sweep. Full side-by-side comparisons against random sparsity and the all-methods fireplace grid appear in Appendix Figures 10 and 12. PISA maintains faithful, sharp dew drops across s=0.15–0.9, showing the Taylor approximation effectively preserves high-frequency local structure throughout the sparsity range. In contrast, random sparsity degrades to tile artifacts at s=0.75–0.9, consistent with its inability to selectively preserve high-attention blocks. For the fireplace and cat prompt, PISA at s=0.5 is nearly indistinguishable from the dense baseline, while random sparsity at s=0.9 collapses into a plaid pattern covering the entire image.

5.5 Video Generation: WAN 2.1

Scope and limitations. Due to compute budget constraints, video generation results are limited to mean denoising step time and qualitative single-frame inspection on a single test prompt. Full quantitative video metrics (FVD, VBench, PSNR, SSIM) were not feasible within the available H100 allocation. The speedup numbers below should therefore be read as efficiency measurements only; quality conclusions are necessarily qualitative and may not generalize across prompts, resolutions, or video lengths.

Figure 8 shows denoising step time (ms) for each video method on WAN 2.1 1.3B. Dense attention requires

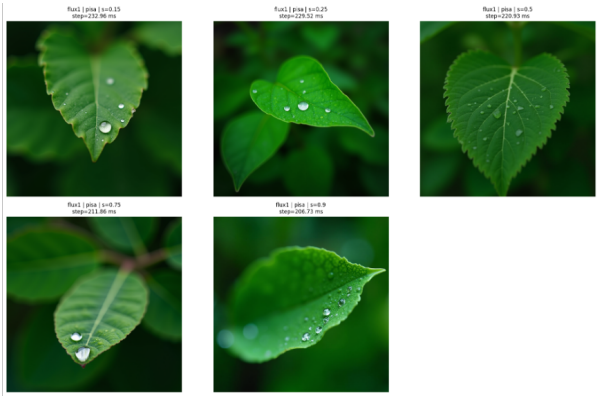


Figure 7: PISA sparsity sweep on Flux.1 [dev]: "A close-up of a single leaf with dew drops," $s \in \{0.15, 0.25, 0.5, 0.75, 0.9\}$. Image quality and dew-drop sharpness are preserved across all sparsity levels; only subtle softening is visible at $s = 0.9$.

˜36,000 ms per step, SVG2 reduces this to ˜19,000 ms (1.89×), VSA to ˜22,000 ms (1.64×), and SANA linear achieves ˜1,100 ms (32.7×). As with the image benchmark, the SANA figure reflects a separate, much smaller model architecture (0.6B vs. 1.3B for WAN 2.1) and is included for reference only; it does not represent a speedup achievable by applying linear attention to WAN 2.1.

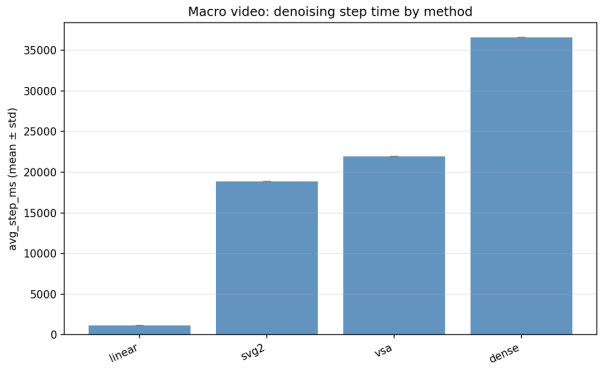


Figure 8: Macro video benchmark: mean denoising step time (ms) on WAN 2.1 1.3B by method. SVG2 and VSA reduce dense attention from ∼36s to ∼19s and ∼22s per step respectively. SANA (∼1s) is a separate 0.6B model and is shown for reference only.

The step-time speedups for SVG2 and VSA broadly align with paper claims (1.89× and 1.7× respectively). However, qualitative evaluation reveals a stark quality-efficiency tension shown in Figure 9.

Dense attention produces a photorealistic red ball resting on a warm hardwood floor with a correct perspective shadow and coherent background geometry. Linear (SANA) also generates a high-fidelity scene—a sharper, more brightly lit ball on a clean wooden surface—though stylistically distinct from the dense output, consistent with its separate training regime.

SVG2 produces the most severe degradation of any method tested. The ball is almost entirely absent: only a faint reddish smear at roughly the correct spatial position hints at the intended subject. The remainder of the frame is an undifferentiated orange-brown blur with no recognizable floor geometry. This is far worse than the paper's reported results and substantially below expectations given its claimed speedup with preserved quality. The most likely explanation is that SVG2's semantic k-means clustering at default hyperparameters groups the large homogeneous background region into the same cluster as the ball—both are warm-colored, low-frequency regions when viewed globally. With the ball tokens outvoted in every cluster by background tokens, they are effectively excluded from all top-p critical clusters, and the subject is never reconstructed. Tuning the cluster count k and top-p budget for WAN 2.1 specifically—rather than using defaults calibrated for HunyuanVideo—is a necessary prerequisite for usable SVG2 video quality. Without quantitative metrics across multiple prompts, however, we cannot rule out that this failure is prompt-specific.

VSA fares better: the ball is clearly present and correctly shaped. However, the background exhibits severe color bleeding—vivid red-orange and yellow streaks radiate from the ball into the surrounding space, and the floor texture is entirely lost. This pattern is characteristic of a miscalibrated coarse gating output O_c. Without the fine-tuning step that VSA requires, the learned gate weights that blend coarse and fine attention outputs are not adapted to WAN 2.1's weight distribution. The coarse stage over-attends globally to the ball region and broadcasts that signal everywhere via the gate, flooding the background with ball-colored information. Even a short fine-tuning run of a few hundred steps on WAN 2.1 would likely eliminate this artifact by teaching the gate to properly suppress coarse broadcasting in background regions.

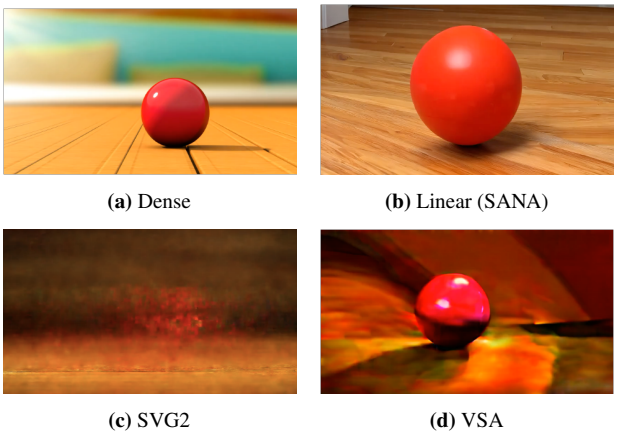
6 Discussion

6.1 Production Kernels vs. Theory

Our Phase 2 results both confirm and complicate the theoretical claims of each paper:

VSA achieves the largest latency reduction in our microbenchmarks (3.57× at 2^{17} tokens), consistent with the paper's 85% FA3 MFU claim. However, its quality in inference-only deployment is poor, confirming that VSA's design fundamentally depends on fine-tuning to adapt the model's weight distributions to the sparse attention pattern. The color-bleeding artifact pattern points specifically to uncalibrated coarse gate weights.

SVG2 achieves the expected latency reduction but at dramatically higher VRAM cost and catastrophically low video quality under default hyperparameters. The $< 1\%$ clustering overhead claim applies to video-scale full-pipeline runs; our attention-only microbenchmarks expose



(a) Dense **(b)** Linear (SANA)

(c) SVG2 **(d)** VSA

Figure 9: WAN 2.1 video generation qualitative comparison (single frame, prompt: "A red ball rolling on a hardwood floor"). Dense and Linear (SANA) produce coherent, high-fidelity frames. SVG2 catastrophically fails under default hyperparameters. VSA preserves the ball geometry but exhibits severe color bleeding attributable to miscalibrated coarse gate weights without fine-tuning. Quality conclusions are based on a single prompt; quantitative video metrics were not collected due to compute budget constraints.

the true cost. The near-total loss of the video subject on WAN 2.1 demonstrates that default hyperparameters do not transfer across models, and per-model calibration of cluster count and top-p budget is essential. This conclusion is limited by the single-prompt evaluation; systematic VBench sweeps are needed to confirm generality.

PISA shows the most theoretically-aligned behavior: graceful quality degradation with increasing sparsity, low L1 error (1.36%), and modest but real latency savings. It is the only method whose quality-efficiency tradeoff is acceptable for inference-only deployment without any fine-tuning or hyperparameter search.

SANA demonstrates that the "redesign" approach (linear attention + deep compression) can achieve dramatically lower per-step latency than sparse attention retrofits, though this comes at the cost of retraining from scratch with an entirely different attention mechanism. Because SANA is a substantially smaller model (0.6B) than the sparse-method targets used here (12B Flux.1, 1.3B WAN 2.1), the latency figures are not directly comparable and should not be interpreted as an upper bound on what sparse methods could achieve at matched scale.

6.2 The Sparsity Hyperparameter Problem

A persistent theme across all methods is the sensitivity to hyperparameters. PISA's sparsity s, SVG2's k and top-p, and VSA's tile size K all require careful tuning on a per-model, per-resolution basis. No single configuration works well across all settings. This creates a significant practical barrier: deploying sparse attention in production requires an offline profiling stage to calibrate hyperparameters, which may require storing representative activation statistics across the full generative distribution.

6.3 The Train-Test Gap

Our most fundamental finding is that inference-time sparse attention on full-attention-trained models carries an irreducible quality cost. PISA's approximation scheme mitigates this through its Taylor expansion, which preserves the expected attention magnitude even for dropped blocks. VSA and SVG2's hard-drop strategies cannot avoid the distributional shift. Fine-tuning is the correct solution but requires training infrastructure that may not be available for all deployment scenarios.

7 Conclusion and Future Work

We have presented a two-phase empirical benchmarking study of sparse attention for image and video synthesis DiTs. Phase 1 established that custom Triton kernels without hardware-aware optimization not only fail to accelerate inference but actively harm it. Phase 2 demonstrated that production-quality kernels on Hopper hardware achieve meaningful speedups (1.4–3.6× depending on sequence length and method), with VSA providing the best raw latency and PISA providing the best quality-efficiency tradeoff for inference-only deployment. SVG2 produced catastrophically low video quality at default hyperparameters, underscoring that per-model calibration is not optional. Video quality conclusions remain qualitative; quantitative metric collection (FVD, VBench) is the highest-priority next step.

Our key actionable findings are:

1. **Systematic Hyperparameter Sweeps:** SVG2 and VSA require per-model tuning before any production use. A structured sweep over cluster count k, top-p, tile size, and sparsity ratio using VBench metrics is the most urgent next step.
2. **Fine-Tuning Investment:** VSA has the highest quality ceiling once fine-tuned. Given its throughput advantage, 2–5k fine-tuning steps on WAN 2.1 is a high-priority next experiment.
3. **Holistic Stack:** Sparse attention alone provides 1.4–2× savings; combining it with FP8 quantization and step distillation could yield a practical 6–8× end-to-end reduction.
4. **Test Other Methods:** Flex Attention, NATTEN, STA, and Token Merging represent promising directions not yet evaluated.
5. **Quantitative Video Metrics:** Systematic FVD/VBench evaluation on production kernels would provide the final Pareto analysis needed to rank methods conclusively and validate or refute the qualitative video quality conclusions drawn here.

References

[1] W. Peebles and S. Xie, "Scalable Diffusion Models with Transformers," *ICCV*, 2023. [arXiv:2212.09748]

[2] T. Dao, "FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning," *ICLR*, 2024. [arXiv:2307.08691]

[3] J. Shah, G. Bikshandi, Y. Zhang, V. Thambidurai, A. Ramani, and T. Dao, "FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision," *NeurIPS*, 2024. [arXiv:2407.08608]

[4] W. Kong, Q. Tian, Z. Zhang et al., "HunyuanVideo: A Systematic Framework For Large Video Generative Models," *arXiv:2412.03603*, 2024. [arXiv:2412.03603]

[5] H. Li, S. Shao, W. Zhong, Z. Zhou, L. Bai, H. Xiong, and Z. Xie, "PISA: Piecewise Sparse Attention Is Wiser for Efficient Diffusion Transformers," *arXiv:2602.01077*, 2026. [arXiv:2602.01077]

[6] S. Yang, H. Xi, Y. Zhao, M. Li, J. Zhang, H. Cai, Y. Lin, X. Li, C. Xu, K. Peng, J. Chen, S. Han, K. Keutzer, and I. Stoica, "Sparse VideoGen2: Accelerate Video Generation with Sparse Attention via Semantic-Aware Permutation," *NeurIPS* (Spotlight), 2025. [arXiv:2505.18875]

[7] P. Zhang, H. Huang, Y. Chen, W. Lin, Z. Liu, I. Stoica, E. Xing, and H. Zhang, "Faster Video Diffusion with Trainable Sparse Attention," *NeurIPS*, 2025. [arXiv:2505.13389]

[8] E. Xie, J. Chen, J. Chen, H. Cai, H. Tang, Y. Lin, Z. Zhang, M. Li, L. Zhu, Y. Lu, and S. Han, "SANA: Efficient High-Resolution Image Synthesis with Linear Diffusion Transformers," *ICLR* (Oral), 2025. [arXiv:2410.10629]

[9] Black Forest Labs, "FLUX.1," 2024. [GitHub]

[10] WanTeam: A. Wang, B. Ai, B. Wen et al., "Wan: Open and Advanced Large-Scale Video Generative Models," *arXiv:2503.20314*, 2025. [arXiv:2503.20314]

[11] X. Xiang and Z. Liu, "Make It Efficient: Dynamic Sparse Attention for Autoregressive Image Generation," *arXiv:2506.18226*, 2025. [arXiv:2506.18226]

[12] R. Chen, K. G. Mills, L. Jiang, C. Gao, and D. Niu, "Re-ttention: Ultra Sparse Visual Generation via Attention Statistical Reshape," *NeurIPS*, 2025. [arXiv:2505.22918]

[13] Y. LeCun, "A Path Towards Autonomous Machine Intelligence," *OpenReview*, 2022. [OpenReview]

A Extended Image Grids

This appendix provides full-width qualitative comparison grids referenced in the main text. All images were generated with Flux.1 [dev] on an NVIDIA H100.

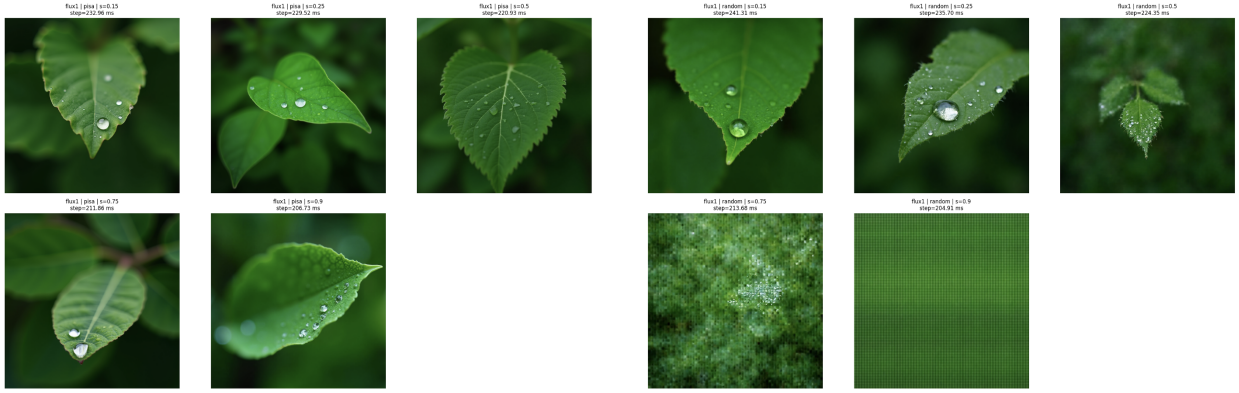


Figure 10: PISA vs. Random Sparsity – "A close-up of a single leaf with dew drops." Each group shows sparsity levels $s \in \{0.15, 0.25, 0.5, 0.75, 0.9\}$ (step times annotated above each image). **PISA (left group):** All five images retain sharp leaf veins and crisp, spherical dew drops throughout the sweep. Only at $s = 0.9$ is any softening of fine detail visible, and the image remains semantically faithful. This demonstrates that the Taylor approximation for non-critical blocks prevents the distributional collapse that hard-drop methods suffer. **Random Sparsity (right group):** Images at $s \leq 0.5$ are comparable to PISA, but at $s = 0.5$ slight artifacts emerge, and above $s = 0.75$ coarse artifacts begin to emerge as the uniform random mask fails to preferentially preserve high-attention dew-drop tokens. At $s = 0.9$ the output fully collapses into a tiled green texture with no recognizable subject. Step times across both groups are comparable at each sparsity level, making PISA's quality advantage essentially free.

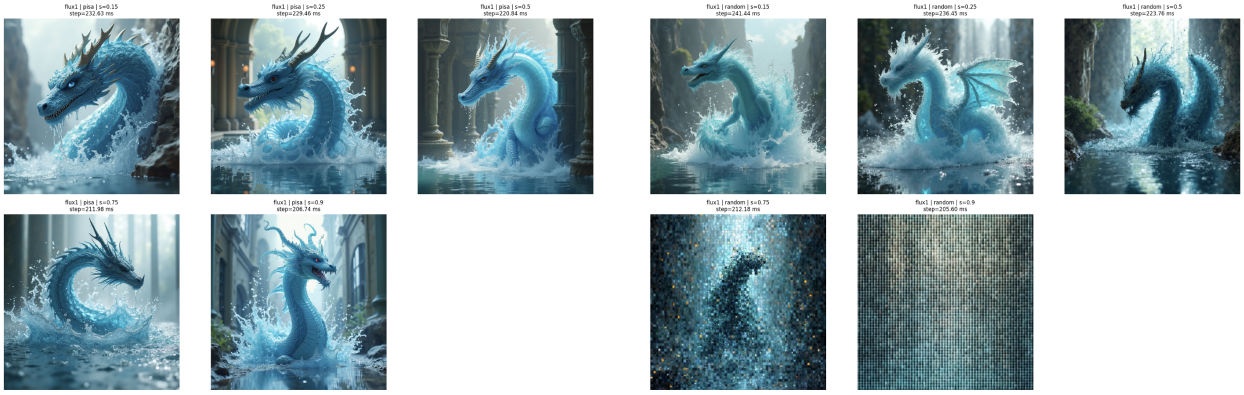


Figure 11: PISA vs. Random Sparsity – "A dragon made of liquid water crashing through a glass palace" Each group shows sparsity levels $s \in \{0.15, 0.25, 0.5, 0.75, 0.9\}$ (step times annotated above each image). **PISA (left group):** All five images retain sharp water droplet and dragon details, with quality dropping almost imperceptibly as sparsity increases, demonstrating the robustness of the PISA method. **Random Sparsity (right group):** Images at $s < 0.5$ are comparable to PISA, but at $s = 0.5$ the image is noticeably coarser due to the random masking. At $s = 0.75$ the output is significantly pixelated, and at $s = 0.9$ the output fully collapses into a tiled texture with no recognizable subject. Step times across both groups are comparable at each sparsity level, making PISA's quality advantage clear throughout the high-sparsity regime.

Figure 12: All-methods comparison – "A cozy living room with a fireplace and a cat on a rug." Top row (left to right): Dense / FlashAttention-3 (∼209 ms/step); PISA $s = 0.5$ (∼221 ms/step); PISA $s = 0.9$ (∼207 ms/step). **Bottom row:** Random $s = 0.5$ (∼225 ms/step); Random $s = 0.9$ (∼205 ms/step); SANA linear (∼27 ms/step). Dense produces a photo-realistic scene with accurate fireplace geometry, detailed rug texture, and a correctly-posed cat. PISA $s = 0.5$ is nearly indistinguishable from dense: fireplace, mantle decorations, and cat posture are all preserved at only 5% latency overhead. PISA $s = 0.9$ retains scene layout and subject identity with moderate softening of fine detail (rug texture, cat fur), demonstrating graceful degradation under the Taylor approximation. Random $s = 0.5$ also preserves the scene convincingly—at moderate sparsity the randomly retained blocks still cover most high-attention interactions. Random $s = 0.9$, however, collapses catastrophically into a plaid-like interference pattern: the uniform random mask fails to preserve any coherent block of high-attention interactions, and the softmax renormalizes over a fully incoherent subset. SANA (linear) generates a stylistically distinct but high-quality image in 27 ms; note that SANA is a separate 0.6B model and this step time does not reflect a speedup achievable on 12B Flux.1.