Performance Analysis and Optimization of 3D Generative Diffusion Models across GPU Architectures

Jeeho Ryoo
Fairleigh Dickinson University
Vancouver, BC, Canada
j.ryoo@fdu.edu

Yongchan Jung
Fairleigh Dickinson University
Vancouver, BC, Canada
y.jung@student.fdu.edu

Muhammad Ali Khaliq
The University of Colorado at Colorado Springs
Colorado Springs, CO, USA
mkhaliq@uccs.edu

Weidong Zhang
Northeastern University
Vancouver, BC, Canada
zhang.weid@northeastern.edu

Jiatong Han
Fairleigh Dickinson University
Vancouver, BC, Canada
j.han2@student.fdu.edu

Byeong Kil Lee
The University of Colorado at Colorado Springs
Colorado Springs, CO, USA
blee@uccs.edu

Abstract

Diffusion models have become essential for high-fidelity 3D MRI synthesis, yet their deployment remains constrained by substantial GPU resource demands arising from hundreds of U-Net evaluations per sample and a highly heterogeneous kernel behavior. This paper performs a comprehensive performance analysis of the state-of-the-art medical diffusion model, Med-DDPM, across three generations of NVIDIA architectures to study kernel-level runtime breakdowns, instruction-mix characteristics, memory system utilization, warp-level activities, and profiler priority-score estimates. We show that training is overwhelmingly dominated by cuDNN convolution and implicit-GEMM kernels, with inefficiencies arising from memory-access patterns, tensor-layout conversions, and limited Tensor Core utilization. Guided by these insights, we evaluate two architecture-aware optimizations TF32 Tensor Core activation and a 3D channels-last layout and demonstrate that they reduce SM cycles by up to 100×, cut dynamic instructions by 100×, raise Tensor Core utilization from 1.45 to 9.98×, and increase IPC by 7% on A100, all without degrading synthesis quality.

CCS Concepts

• **Computer systems organization → Parallel architectures**; • **Computing methodologies → Machine learning**.

Keywords

Performance Analysis; GPU Optimization; Tensor Cores; TF32; channels-last; Diffusion Models

ACM Reference Format:
Jeeho Ryoo, Yongchan Jung, Muhammad Ali Khaliq, Weidong Zhang, Jiatong Han, and Byeong Kil Lee. 2026. Performance Analysis and Optimization of 3D Generative Diffusion Models across GPU Architectures. In *Proceedings of the 17th ACM/SPEC International Conference on Performance Engineering (ICPE '26), May 04–08, 2026, Florence, Italy*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3777884.3797012

ICPE '26, Florence, Italy

ACM ISBN 979-8-4007-2325-4/2026/05
https://doi.org/10.1145/3777884.3797012

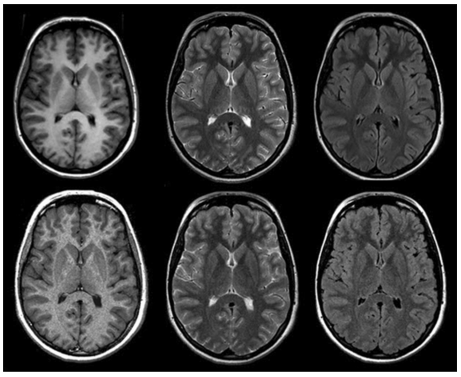

Figure 1: Conventional (top) and Mask-conditioned Synthetic (bottom) T1, T2, and FLAIR MRIs

1 Introduction

Magnetic Resonance Imaging (MRI) is central to clinical neurology and neuroscience research, providing high-resolution 3D views of brain anatomy across complementary modalities such as T1-weighted, T2-weighted, and FLAIR scans [12]. These contrasts reveal tissue boundaries, fluid characteristics, and lesion patterns essential for diagnosis and longitudinal monitoring. However, large-scale MRI datasets remain difficult to assemble due to privacy constraints, scanner variability, long acquisition times, and the need for expert annotation [3, 44]. To address these limitations, generative models have become an important tool for augmenting datasets with privacy-preserving images that reflect modality-specific appearance patterns. Figure 1 illustrates these modality-specific contrasts by showing conventional T1, T2, and FLAIR slices alongside the corresponding mask-conditioned synthetic outputs. [2, 40].

High-fidelity synthetic MRIs match anatomical boundaries, relative intensity profiles, and lesion-like hyperintensities, although they may differ subtly in texture or global scaling. These nuances motivate the evaluation through both quantitative metrics and qualitative expert review, as well as the use of explicit mask conditioning to control lesion placement and morphology for augmentation, harmonization, and privacy-preserving dataset expansion [59]. Recent advances in denoising diffusion probabilistic models (DDPMs) have further improved the realism of 2D and 3D MRI synthesis, consistently demonstrating higher structural fidelity than GAN-based

approaches [25]. Within this landscape, Med-DDPM stands out as a state-of-the-art yet still relatively new framework for 3D, mask-guided MRI synthesis, offering strong reconstruction quality and anatomically coherent outputs.

However, these accuracy benefits come with substantial computational cost. A single diffusion sample requires hundreds of sequential U-Net evaluations, each executing dozens of 3D convolutions, normalization layers, and elementwise operations [53]. This results in heterogeneous GPU execution patterns with frequent kernel launches, alternating compute-bound and memory-bound phases, high arithmetic intensity for convolutions, and bandwidth-limited behavior for normalization and reduction kernels. Consequently, training 3D diffusion models demands hundreds of GPU-hours even for modest datasets. Model improvements alone cannot resolve these bottlenecks as their practical deployment depends critically on architectural features such as tensor-core design, memory bandwidth, layout handling, and precision modes across GPU generations [18].

This systems perspective is largely absent from existing medical imaging research, where diffusion models are typically evaluated only on image quality. In contrast, this work performs a full-stack GPU performance study of Med-DDPM across three NVIDIA architectures Volta (V100), Ampere (A100), and Hopper (H100) using NVIDIA's Nsight Compute profiler to characterize the model from kernel level down to microarchitectural scheduling behavior. Beyond kernel-level execution characteristics, we analyze detailed kernel mixes, dynamic instruction-mix profiles, priority-score distributions, SM-level utilization, memory-hierarchy behavior, warp occupancy and warp-scheduler efficiency, and stall-type breakdowns involving long-latency dependencies, math-pipeline throttling, and divergent-branch resolution. These multi-level measurements reveal that Med-DDPM is overwhelmingly dominated by cuDNN 3D convolution and implicit-GEMM kernels, and that inefficiencies arise not only from memory-access irregularities, tensor-layout conversions, and precision-mode constraints, but also from fragmented warp scheduling, cache residency effects, and underutilized Tensor Core pipelines on Ampere/Hopper architectures. Building on these profiling results, we apply two targeted, architecture-aware interventions that directly address the dominant bottlenecks identified in our analysis. We show how enabling Tensor Core execution on Ampere/Hopper and restructuring Med-DDPM's memory layout fundamentally changes the balance of compute-, memory-, and scheduler-driven behavior across the model. These findings lead to the following contributions:

- We present a detailed system-level and microarchitectural GPU characterization of Med-DDPM across three different architectures (V100, A100, and H100).
- We identify architecture-specific bottlenecks in convolution, normalization, and layout-conversion kernels using kernel-mix breakdowns, IPC stacks, and priority-score analysis.
- We apply and evaluate two system level optimizations TF32 Tensor Core activation and a 3D channels-last layout derived from microarchitectural implications.

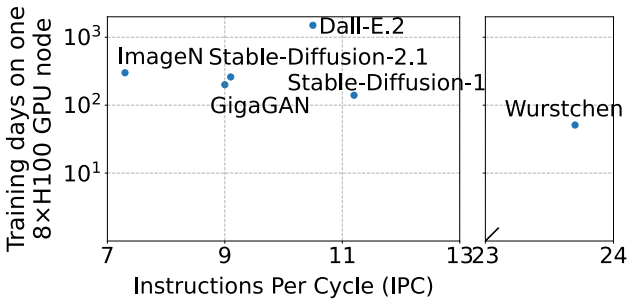

Figure 2: IPC versus Training Duration for Representative Generative Models

- We show that TF32 Tensor Core activation reduces SM cycles by up to ∼ 5× on A100, while a 3D channels-last layout exposes new memory-bound microkernels that require further fusion to reach peak throughput.
- We analyze architectural behavior shifts with the proposed optimizations in resource utilization, kernel mix, memory, active warps, and stalls. Based on the analysis, practical guidelines are presented for optimizing general 3D diffusion models on modern GPUs.

2 Background

2.1 Med-DDPM Architecture

Diffusion models have emerged as a leading paradigm for medical image synthesis due to their stability and ability to preserve anatomical structure, outperforming GAN-based approaches that often exhibit mode collapse and training instability in clinical settings [19, 23, 39]. These limitations motivate the shift toward diffusion-based approaches, which provide more stable optimization behavior and stronger structural fidelity. Med-DDPM adapts this paradigm for 3D brain MRI generation under semantic guidance, using segmentation masks to address challenges such as limited annotated datasets and privacy concerns while enabling explicit control over structural placement [12, 45]. Beyond sampling stability, diffusion models can integrate structured guidance signals that further improve anatomical control. This mask-conditioned generation strategy allows the model to synthesize anatomically faithful and pathology-aware MRIs suitable for augmentation, controlled simulation, and dataset expansion in clinical workflows [65].

Diffusion models are highly compute-intensive: each synthesized sample requires hundreds to thousands of iterative denoising steps, so every training batch executes repeated full U-Net passes rather than the single forward–backward cycle typical of GANs. For 3D MRI, this cost grows further as large volumetric tensors stress memory bandwidth, kernel-launch rates, and SM utilization. Figure 2 situates this burden by comparing training duration and IPC across diffusion and non-diffusion generative models [47], revealing that diffusion architectures cluster in a medium-IPC, long-duration regime.

Med-DDPM inherits this algorithmic structure, motivating the detailed GPU-level analysis in this paper. Even well-optimized diffusion models require days of uninterrupted GPU time and exhibit wide hardware-efficiency variation. In contrast, models like

Wurstchen achieve higher IPC and much shorter training times because they avoid iterative denoising. This fundamental disparity iterative refinement versus single-pass generation explains why diffusion models are slower, less compute-efficient, and far more resource-intensive despite their superior fidelity, posing a major scalability challenge for large 3D medical generators such as Med-DDPM [37].

Architecturally, Med-DDPM follows a standard DDPM forward process with a cosine noise schedule, gradually corrupting a clean MRI volume with Gaussian noise before reversing the corruption through a conditioned 3D U-Net [12]. The reverse network predicts the noise added at each timestep while concatenating a one-hot encoded segmentation mask with the noisy input, ensuring anatomical awareness throughout the denoising process [59]. This design enables Med-DDPM to generate structurally coherent MRIs even from limited training data, producing higher-quality images and stronger downstream segmentation performance than GAN-based and latent-diffusion baselines [12].

Figure 3 summarizes the full workflow, pairing the forward diffusion trajectory with the conditioned reverse denoising pathway. In the reverse phase, the U-Net processes mask-aware feature maps through the Initial Block and Encoder Blocks ❶, composed of Residual Blocks ❹, Down-sample layers, and Attention Blocks, before integrating global features in the Bottleneck Block ❷. Decoder Blocks ❸ then reconstruct the anatomical volume using skip connections, with the segmentation mask guiding feature refinement at each timestep [16, 51]. This conditioning mechanism enforces anatomical boundaries and pathological regions during sampling, enabling Med-DDPM to generate clinically realistic, mask-guided 3D MRI variations despite the substantial computational burden associated with diffusion training.

Unlike CNNs or transformers, diffusion models execute long chains of heterogeneous kernels, large convolutions, normalization layers, elementwise ops, and small reductions repeated hundreds of times per sample. This creates alternating compute-bound and memory-bound phases that expose architectural bottlenecks not visible in conventional training workloads. A detailed GPU-profiling study is therefore essential to understand how diffusion workloads interact with Tensor Cores, memory hierarchy, warp scheduling, and cache behavior.

2.2 GPU Architectures

Diffusion models stress GPUs differently from conventional CNN or transformer workloads. Because Med-DDPM consists of hundreds of sequential U-Net evaluations, its computational footprint interacts strongly with the underlying GPU architecture. Each sampling trajectory executes hundreds of U-Net blocks composed of convolutions, GroupNorm, residual connections, and pointwise operations, producing a rapid stream of small and medium-sized kernels with alternating compute and memory-bound behavior. As a result, the achievable performance depends closely on core GPU architectural features such as SM organization, tensor-core capabilities, memory hierarchy, cache design, and supported precision modes. Table 1 summarizes the architectural characteristics of the three NVIDIA GPU generations evaluated in this study [8].

Modern NVIDIA GPUs organize computation around Streaming Multiprocessors (SMs), which integrate scalar and vector ALUs,

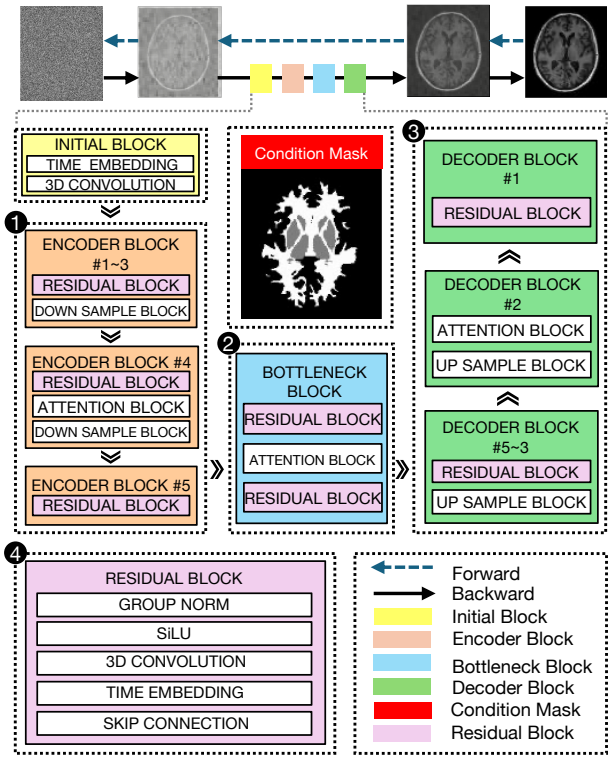

Figure 3: Overview of the Med-DDPM architecture

Table 1: Comparison of GPU Architectures and Features

Feature	H100	A100	V100
Architecture	Hopper	Ampere	Volta
CUDA Cores	14592	6912	5120
Memory Size (GB)	80	40 / 80	16 / 32
Memory Type	HBM3	HBM2e	HBM2
Memory BW (GB/s)	3,350	2,000	900
Peak FP32 (TFLOPS)	60	19.5	15.7
Tensor Cores	Gen4 + FP8	Gen3	Gen1
NVLink BW (GB/s)	900	600	300
PCIe Version	Gen5	Gen4	Gen3 / Gen4
Multi-Instance GPU	Yes	Yes	No
Transformer Engine	Yes	No	No
L1 Cache (per SM)	256 KB	192 KB	128 KB
L2 Cache	50 MB	40 MB	6 MB
Supported Precision	FP16/32/64, TF32 INT8, INT4, FP8	FP16/32/64, TF32 INT8, INT4	FP16/32/64 INT8

warp schedulers, register files, shared memory, and L1 cache. The L1 cache size has increased across generations from 128KB per SM in Volta to 192KB in Ampere and 256KB in Hopper, improving intra-block data reuse and reducing latency for small kernel operations. These architectural factors directly influence how efficiently diffusion workloads can be scheduled and executed. Convolutions in U-Net blocks map to implicit GEMM operations, making tensor-core throughput a primary determinant of overall diffusion speed. Volta's V100 provides Gen1 tensor cores optimized for FP32 and INT8, Ampere's A100 introduces Gen3 tensor cores with TF32 and INT8 support, and Hopper's H100 adds Gen4 tensor cores with FP8 and INT4 acceleration. These architectural differences directly influence how effectively diffusion models can exploit mixed-precision execution without degrading denoising stability [60].

The memory hierarchy is equally important because many kernels in diffusion models (elementwise ops, normalization layers, and layout conversions) are memory-bound. HBM bandwidth increases substantially across generations (V100's 900 GB/s, A100's 2,000 GB/s, H100's 3,350 GB/s), and L2 cache capacity grows from 6 MB in Volta to 40 MB in Ampere and 50 MB in Hopper. These improvements reduce stalls for bandwidth-limited stages of the U-Net and improve feature-map reuse across layers [33].

Precision support also evolves across generations. V100 primarily targets FP32 and INT8 execution, A100 adds TF32 and INT4 paths for both training and inference, and H100 enables FP8 acceleration for tolerant layers. Diffusion models accumulate numerical error across timesteps, so the availability of multiple precision modes is critical for balancing stability and throughput in medical imaging tasks [22].

Finally, diffusion workloads generate a high frequency of small kernel launches. Higher SM counts and improved warp scheduling mechanisms in A100 and H100 help sustain throughput under such fragmented execution patterns. For 3D medical diffusion models like Med-DDPM where tensor volumes are large and precision demands strict, these architectural differences have direct impact on achievable sampling time and fidelity. Overall, the progression from Volta to Ampere to Hopper provides increasingly favorable conditions for high-resolution diffusion workloads [10].

3 Experimental Methodology

This section describes the experimental methodology used to profile Med-DDPM training across H100, A100, and V100. The methodology covers hardware and software setup, measurement strategy, and profiling configuration.

3.1 Hardware and Software Setup

The profiling and performance evaluations were performed on three platforms summarized in Table 1. The profiling on H100 and A100 was performed inside Docker containers on Indiana Jetstream2 and TACC Stampede3, while the V100 runs used a Singularity/Apptainer image on SDSC Expanse to comply with site policies [4–6]. Since Docker relies on a root-privileged daemon, many sites restrict its use due to security concerns. Instead, they support user-level containers such as Singularity/Apptainer, which align better with system security policies. We used those containerized environments to ensure that the Med-DDPM workload, software dependencies, and CUDA/cuDNN versions remained bit-identical across all GPU architectures, eliminating run-to-run variability caused by divergent system libraries or site-specific configurations. The software stack for all runs used PyTorch 2.0.1 and Python 3.11.4. Nsight Compute CLI (`ncu`) version 2025.2.1 (CUDA Toolkit 12.9, cuDNN version 8.9.2) was used for profiling in the kernel-level, architecture-level, and instruction-level [9]. For every kernel-level profile, Nsight Compute provides instruction mix, warp execution efficiency, SM utilization, Tensor core usage, etc. Microarchitecture-level profiling information includes pipeline stalls and L1/L2 cache behavior, while SASS (machine instructions) and per-instruction stall reasons are provided at instruction-level [7].

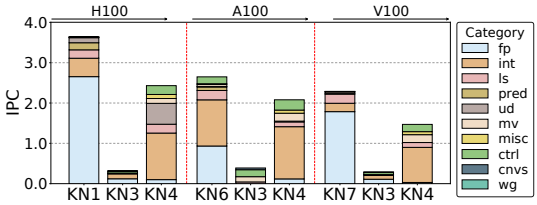

Figure 4: IPC Stack Bars

3.2 Performance Measurement

Med-DDPM training was executed with the same hyperparameters across platforms: 25 epochs, `batchsize=1`, `input_size=128`, `depth_size=128`, `num_channels=64`, and the same `resume_weight`. The profiling script invoked Nsight Compute with the following key options: `–set full`, `–call-stack`, `–nvtx`, `–csv`, `–export`, `–launch-skip 10`, `–launch-count 100`, and `–target-processes all`. The option `–launch-skip 10` avoids warm-up noise, while `–launch-count 100` provides statistical stability for repeated kernel launches [57]. We could not collect data using such performance monitoring counters `smsp__warps_issue_stalled_long_scoreboard.sum`, `smsp__warps_issue_stalled_math_pipe_throttle.sum`, and `smsp__warps_issue_stalled_branch_resolving.sum` as they do not exist in a relatively old V100 architecture. Thus, we do not report these metrics in §6.5.

The Med-DDPM training workload generates many short, recurrent kernels primarily convolutions, elementwise operations, and reductions making kernel-level profiling essential for understanding end-to-end behavior. For each GPU platform, we used Nsight Compute (`ncu`) under the same training configuration to collect per-kernel execution statistics and microarchitectural metrics. The primary metrics for cross-platform comparison were kernel duration, instruction count, and cycle count, which we used to compute per-kernel IPC (instructions per cycle) and an overall IPC via the geometric mean of per-kernel IPC. In addition, we recorded memory bandwidth by kernel duration, and the profiler's priority score, which ranks kernels by estimated optimization potential (See §4.1) [58]. Together, these measurements provide a consistent basis for characterizing kernel behavior and comparing training performance across architectures.

4 Performance Analysis of Med-DDPM

Diffusion-driven MRI synthesis places sustained pressure on GPUs because each reverse-diffusion step invokes a full 3D U-Net pass composed of convolutional blocks, normalization stages, and auxiliary tensor operations. Although modern samplers reduce the total number of denoising steps, the execution remains dominated by a large collection of kernels whose behavior varies significantly across architectures. To understand how these workloads interact with contemporary GPUs and where inefficiencies arise, we now shift from algorithmic context to a detailed systems-level study. This section presents a comprehensive performance analysis of Med-DDPM, examining kernel runtimes, instruction behavior, and priority-score profiles to reveal the architectural bottlenecks that govern end-to-end training throughput.

Table 2: Kernel Name Abbreviations

Abbr	Kernel Name	Abbr	Kernel Name
KN1	sm80_xmma_fprop_implicit	KN5	nhwcToNchwKernel
KN2	sm90_xmma_fprop_implicit	KN6	implicit_convolveNd
KN3	RowwiseMomentsCUDAKernel	KN7	_5x_cudnn_volta_scudnn
KN4	elementwise_kernel	KN8	nhwcAddPaddingKernel

4.1 System Performance Analysis

We extracted instruction-per-cycle (IPC) stack profiles for representative kernels in Med-DDPM's reverse-diffusion loop. Table 2 shows representative kernel name abbreviations used in the profiling. As illustrated in Figure 4, diffusion workloads exhibit persistent inefficiencies across architectures. On the V100, the dominant cuDNN 3D convolution incurs $\sim 3.9 \times 10^9$ cycles with an aggregate IPC of 2.38, of which nearly 78% are FP instructions indicating that the model saturates neither memory pipelines nor Tensor-Core pathways. On the A100, the corresponding tensor-core GEMM completes in $\sim 1.18 \times 10^9$ cycles (a 3.3× reduction), yet its IPC stack (INT: 42%, FP: 25%) shows that memory movement and reduction operations still occupy a significant fraction of execution. The H100 further reduces execution to $\sim 8.2 \times 10^8$ cycles, but its IPC decomposition shows more than 50% of instructions in UD (tensor-core micro-ops) and only 2% in classical FP pipelines, with the remainder dominated by LS, MV, and CTRL instructions. Across all GPUs, diffusion kernels achieve suboptimal utilization because memory access, layout conversions, and repeated small reductions impose structural bottlenecks that compute improvements alone cannot overcome.

To understand not only how kernels behave, but also which kernels matter most, we analyze Nsight Compute's *priority score*. This metric estimates potential optimization benefit by combining kernel duration and the percentage of achievable speedup as shown in Equation 1.

$$\text{Priority Score} = \text{Duration} \times \text{Estimated Speedup} \tag{1}$$

Nsight Compute computes the *Estimated Speedup* by comparing each kernel's measured throughput against the maximum throughput that the GPU could theoretically sustain for its dominant bottleneck. Formally, Equation 2 explains how the profiler quantifies this remaining performance headroom as clamped to the range [0,1] to represent the percentage of potential improvement. A kernel operating near its hardware limit yields an estimated speedup close to zero, whereas kernels far below their achievable ceilings exhibit larger values and therefore higher optimization potential.

$$\text{Estimated Speedup} \approx \frac{\text{Achievable Performance}}{\text{Observed Performance}} - 1 \tag{2}$$

Summing these scores across kernel categories allows us to quantify total optimization opportunity. The result for Med-DDPM is shown in Figure 5 where normalization kernels account for the largest cumulative priority, followed by tensor-core GEMMs and then elementwise operations. This reveals that diffusion workloads are not dominated by a single computational primitive, but rather by a heterogeneous mixture of memory-bound, reduction-bound, and shape-sensitive kernels.

Taken together, the IPC and priority-score patterns highlight that faster samplers or improved denoising schedules cannot, by themselves, resolve Med-DDPM's inefficiencies. As long as the underlying kernels remain memory-bound, reduction-heavy, or dominated by non-compute micro-operations, algorithmic refinements will yield only incremental gains. Likewise, moving to newer

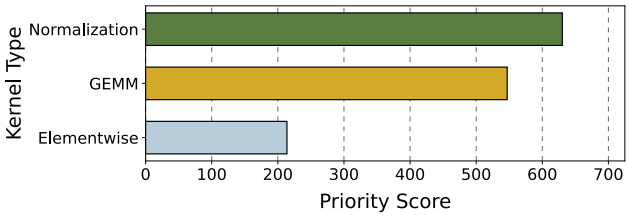

Figure 5: Priority Scores of Med-DDPM

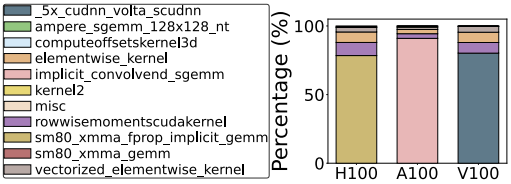

Figure 6: Med-DDPM Kernel Mix

GPUs such as A100 or H100 does not automatically eliminate bottlenecks when substantial fractions of instructions arise from layout handling, address generation, or control flow rather than arithmetic work. These observations frame the broader motivation for our study that synthetic MRI generation's diffusion-based pipelines remain prohibitively expensive to train at scale.

Takeaway 1: Med-DDPM underutilizes GPU pipelines across all architectures due to persistent memory, layout, and reduction overheads.

4.2 Kernel-Level Analysis

Figure 6 presents the kernel-level composition of Med-DDPM's reverse-diffusion loop across V100, A100, and H100. Each bar aggregates the proportion of total kernel time spent in major operator classes including cuDNN convolutions, implicit-GEMM kernels, normalization kernels, and a variety of elementwise and layout-conversion kernels. Architecture-specific kernel mix directly shapes hardware utilization patterns, which can lead to imbalanced utilization across compute, memory, and scheduling resources. Consequently, kernel-level characterization is essential for identifying the dominant bottlenecks and guiding targeted optimization strategies. Also, this breakdown highlights how the 3D U-Net structure translates into GPU workloads and makes clear which operations dominate execution on each architecture.

Across all three GPUs, a single cuDNN or implicit-GEMM kernel accounts for the vast majority of runtime. On V100, the `_5x_cudnn_volta_scudnn` convolution kernel consumes nearly 80% of total execution, with normalization (`RowwiseMomentsCUDAKernel`) and elementwise kernels contributing only marginally. The skew is even stronger on A100, where `implicit_convolveNd_sgemm` approaches 90% of runtime. H100 follows the same trend: its primary kernel, `sm80_xmma_fprop_implicit_gemm`, accounts for roughly 78% of observed activity. The remaining kernels' pointwise operations, padding and layout transforms, and small reductions appear only as thin slices in the kernel mix.

This distribution reflects the computational structure of the Med-DDPM U-Net, in which 3D convolutions dominate both forward and backward passes and are mapped by PyTorch to cuDNN's implicit-GEMM backends. Architecturally, the V100's dominant convolution

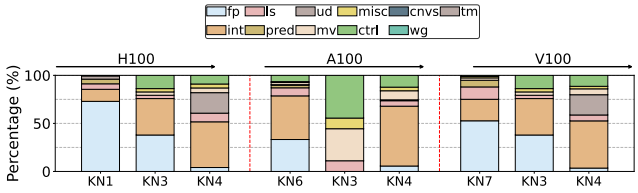

Figure 7: Med-DDPM Instruction Mix

kernel is mostly constrained by FP32/FP16 FMA throughput rather than by memory bandwidth, leading to a compute-bound regime with limited Tensor Core engagement. A100 and H100, by contrast, route these same convolutional workloads through more advanced MMA pipelines (TF32, BF16, FP16, and Hopper's enhanced Tensor Cores). Yet despite their higher theoretical throughput, both architectures remain bottlenecked by how efficiently PyTorch and cuDNN manage tile loading, shared-memory reuse, and tensor-layout handling. The persistent dominance of the conv/GEMM pathway across all generations underscores a central result: architectural advances improve throughput but do not alter the fundamental fact that convolution remains the structural bottleneck in Med-DDPM's training pipeline.

Takeaway 2: A single cuDNN convolution/GEMM kernel dominates runtime, making convolution throughput the primary optimization target.

4.3 Instruction-Level Analysis

The instruction mix in Figure 7 provides a complementary view of the underlying bottlenecks from a hardware-focused perspective, showing how each architecture distributes its dynamic instructions within the dominant kernels. Across three architectures, the primary convolution and GEMM kernels contain a notably high proportion of floating-point instructions and comparatively modest load/store activity. On V100, more than half of all dynamic instructions in the main convolution kernel correspond to floating-point operations, with the remainder consisting of integer arithmetic, load/store instructions, and predicate or control flow. This profile reflects a kernel whose behavior is shaped largely by Tensor Core throughput and extensive register reuse rather than by memory-system demands.

A100 exhibits a different distribution, characterized by a larger fraction of integer and control instructions relative to floating-point operations. This pattern aligns with Ampere's implicit-GEMM design, where MMA pipelines rely on extensive index computation and loop control to coordinate Tensor Core execution. The comparatively small share of load/store instructions indicates that the kernel's behavior is influenced more by scheduler activity and intra-SM execution than by external memory bandwidth. On H100, the dominant GEMM kernel is even more heavily weighted toward floating-point operations, with over 70% of its dynamic instructions belonging to FP pipelines. This profile suggests that Hopper's Tensor Core pathways drive the majority of the kernel's execution, with memory and control-flow operations contributing minor shares.

The instruction-level characteristics of elementwise kernels differ markedly from those of convolution and GEMM kernels. Elementwise operations contain relatively few floating-point instructions and instead rely heavily on integer arithmetic, uniform datapath operations, register movement, and control flow. This distribution

Algorithm 1: TF32 Tensor-Core Acceleration

Require: 3D UNet training loop with convolution- and GEMM-heavy operations
Ensure: Use TF32 Tensor Cores on Ampere+ GPUs

```
MatmulBackend.allow_tf32 = True;
CuDNNBackend.allow_tf32 = True;
M ← diffusion_model
foreach step ∈ training_steps do
    (x, y) = select_batch();
    M.forward_tf32_conv3D(x, y);
    loss = M.calc_loss(x, y);
    M.backpropagate_tf32_conv3D_matmul(loss);
    M.update_parameters();
end
```

reflects the nature of PyTorch's fused pointwise kernels, which typically perform simple arithmetic but require substantial address computation, conditional evaluation, and data movement. Such kernels commonly exhibit sensitivity to memory bandwidth, register usage, and warp-level control behavior. Because their arithmetic intensity is low, the overall instruction mix is dominated by operations that support memory access and per-element control rather than direct computation.

Normalization and reduction kernels, such as `RowwiseMomentsCUDAKernel`, display yet another instruction profile. These kernels rely heavily on load/store instructions, integer arithmetic, and control logic as they repeatedly access feature-map tiles, compute per-channel statistics, and coordinate partial reductions across threads. Their behavior is closely tied to shared-memory bandwidth and warp-synchronous data exchange rather than to floating-point throughput.

Taken together, the instruction-mix data across all architectures reveal distinct execution regimes for convolution, elementwise, and reduction kernels. Convolution and GEMM kernels are dominated by floating-point pipelines and are shaped primarily by Tensor Core execution. Elementwise kernels rely heavily on integer and control operations, reflecting their memory-bound and address-intensive nature. Reduction kernels emphasize load/store and control behavior, consistent with their communication and synchronization patterns. These distinctions illustrate the heterogeneous instruction-level structure of the Med-DDPM model and highlight the diverse hardware resources engaged across different kernel types.

Takeaway 3: Convolution, elementwise, and reduction kernels stress different instruction paths, requiring multi-faceted rather than single-point optimization.

5 Optimization Strategies

In this section, we relate the optimization strategies to the bottlenecks identified in §4. The kernel and instruction-mix analysis showed that Med-DDPM's training loop is dominated by convolution and GEMM kernels, with execution shaped largely by Tensor Core throughput and surrounding memory-access patterns. The strategies below therefore target these two factors by enabling high-throughput Tensor Core execution and adopting memory layouts that minimize layout conversions and improve data locality.

Algorithm 2: 3D Channels-Last Memory Layout

Require: Input size (N, C, H, W, D), UNet hyperparameters
Ensure : UNet executes with channels-last 3D activations

$UNet$ = create_UNet($input\_size$, $channels$, $blocks$);
$UNet$ = to_channels_last_3D($UNet$);
M = GaussianDiffusion($UNet$, size_of(H, W, D), $timestaps$);
foreach *step* ∈ *training_steps* **do**
  (x, y) = select_batch();
  (x, y) = to_channels_last_3D(x, y);
  x_t = M.generate_noisy(x);
  $\tilde{x}_t$ = M.predict_noise(x_t);
  M.calc_loss($x_t, \tilde{x}_t$);
  M.update_parameters();
end

5.1 TF32 Activation for Tensor Core

Starting from the Ampere architecture, third generation Tensor Cores can be utilized with broader precision support, and we leverage these tensor cores to accelerate high-dimensional tensor operations such as 3D convolution and matrix multiplication, thereby optimizing performance. Algorithm 1 summarizes this TF32 acceleration process. TF32 is an efficient numeric format that combines the wide dynamic range of FP32 with the computation speed of FP16, and it is natively supported by tensor cores. This allows us to achieve significant speedups while maintaining convergence accuracy nearly equivalent to FP32. The key idea of the algorithm is to explicitly enable TF32 computation in the backend of the PyTorch framework before the training loop begins (the red highlighted). The `MatmulBackend.allow_tf32` setting accelerates matrix multiplication (GEMM) operations used in components such as Linear layers. `CuDNNBackend.allow_tf32` setting instructs the cuDNN library to process 3D convolution operations which are most critical for the performance of the 3D U-Net using tensor cores. These settings directly affect the two most computationally intensive stages of training. First is the forward pass (blue highlighted), where all 3D convolution operations inside the U-Net are accelerated by TF32. Second is the backward pass (the green highlighted), where convolution and matrix multiplication operations required to compute gradients from the loss are efficiently handled by tensor cores.

5.2 Channels-Last Memory Layout

Along with TF32 computation acceleration, we applied a memory layout optimization strategy to maximize the execution efficiency of the 3D U-Net. Modern hardware accelerators such as NVIDIA GPU tensor cores exhibit higher compute throughput and memory bandwidth efficiency when operating on a channels-last memory layout rather than the channels-first layout. Algorithm 2 provides a detailed overview of how this channels-last memory layout is applied to the training of the 3D diffusion model.

Immediately after creating the base structure of the 3D U-Net model using the `create_UNet` function, we convert the model's memory format to channels-last through the `to_channels_last_3D function` (the red highlighted). This transformation modifies the internal structure so that the model's weights and layers, such as Conv3D, accept and output channels-last tensors instead of channels-first. Once the main training loop begins (for all steps), the sampled batch's x (image) and y (mask) remain in the standard format (channels-first). Therefore, immediately before feeding these tensors into the model, we explicitly convert them to a five-dimensional format, where the dimensions follow the order of batch size (N), height (H), width (W), depth (D), and channel count (C), using `to_channels_last_3D` (the blue highlighted). When the core training operation `M.predict_noise` (the green highlighted) is executed, both the model and the input data are aligned in the channels-last format. As a result, the entire forward and backward pass inside the U-Net executes in a manner that is highly optimized for tensor cores, producing a synergistic effect with TF32 acceleration (Algorithm 1).

6 Evaluation

We evaluate the impact of the proposed optimization strategies on Med-DDPM by comparing four configurations: **Baseline**, the unmodified Med-DDPM implementation; **OPT1**, which applies the TF32 tensor-core optimization in isolation; **OPT2**, which applies the 3D channels-last memory layout optimization in isolation; and **OPT12**, which integrates both **OPT1** and **OPT2** simultaneously.

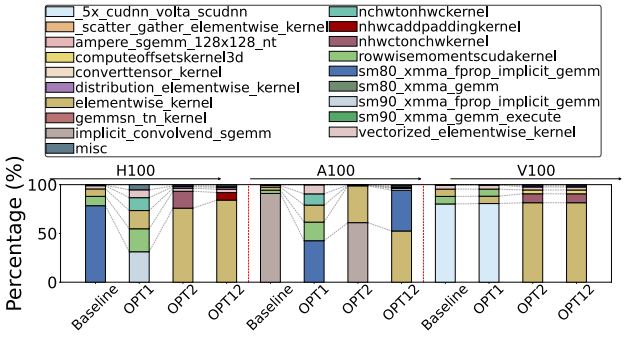

Figure 8: Kernel Mix Bar Chart for Optimizations

6.1 Overall Performance Analysis

Figure 8 shows that baseline Med-DDPM is dominated by large cuDNN convolution/GEMM kernels across all GPUs `_5x_cudnn_volta_scudnn` on V100 (~80%), `implicit_convolveNd_sgemm` on A100 (~91%), and `sm80_xmma_fprop_implicit_gemm` on H100 (~78%). Enabling TF32 (OPT1) leaves V100 unchanged but substantially reshapes the Ampere/Hopper architectures: on A100, the conv/GEMM share drops to ~43% with row-wise moments and elementwise kernels rising to ~19% and ~17%, while H100 sees Tensor Core work fall to ~31% as normalization, pointwise ops, and layout transforms collectively exceed 55%. These shifts confirm that TF32 shortens the dominant conv/GEMM path and exposes bottlenecks previously hidden behind long FP32 kernels. Correspondingly, normalized SM cycles and instructions reduce sharply on A100/H100 (cycles at 0.18× and 0.39×; instructions at 0.09× and 0.18×) while IPC remains near baseline (1.02× and 0.95×), consistent with high-throughput MMA instructions compressing the arithmetic footprint without underutilizing the schedulers.

OPT2 (channels-last), by contrast, collapses the conv/GEMM contribution and rebalances execution almost entirely toward

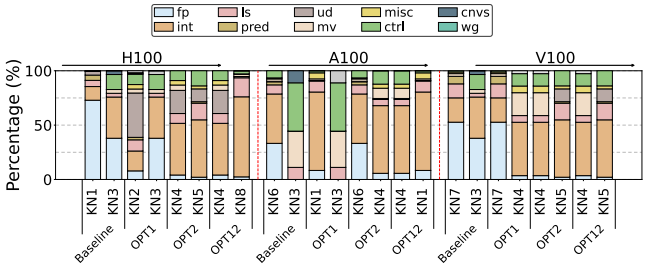

Figure 9: Instruction Mix of Baseline and Optimizations

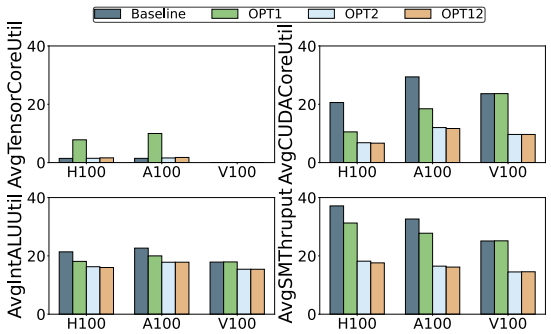

Figure 10: Average Tensor/CUDA/Int ALU Core Utilization and SM Throughput

`elementwise_kernel` (~73–80%) plus NHWC padding and transform kernels, fragmenting the 3D U-Net into many small memory-bound kernels. This is reflected in extreme reductions in dynamic work on A100/H100 cycles falling to 0.01–0.02× and instructions to 0.01× with IPC slightly increasing (~1.07×) because these lighter kernels retire efficiently at the SM level despite offering little Tensor Core utilization. OPT12 partially restores Tensor Core usage on A100 (GEMM at ~42%), but V100 and H100 remain dominated by elementwise and layout transforms (>80%), confirming that channels-last fundamentally shifts the bottleneck away from compute throughput. Microarchitecturally, OPT1 is a true arithmetic compression mechanism that preserves SM efficiency via dense MMA execution, whereas OPT2/OPT12 alter the kernel landscape itself, trading away Tensor Core acceleration for a layout-bound pipeline whose performance will only improve through reducing NHWC↔NCHW transforms, fusing pointwise ops, and improving data locality.

Figure 9 complements this kernel-level view by showing how the dynamic instruction mix evolves under each optimization. On A100, the baseline `implicit_convolveNd_sgemm` kernel is split between integer and floating-point work (Integer ≈ 45%, FP ≈ 33%), with the remainder spread across load/store (≈ 8%), control (≈ 7%), and small fractions of movement, predicate, and uniform-datapath instructions. Under OPT1, the corresponding Tensor Core path becomes strongly integer-heavy (Integer ≈ 72%, FP ≈ 10%), while on H100 the FP share drops from ≈ 73% to ≈ 41% and uniform-datapath plus movement/miscellaneous instructions roughly double. This shift reflects how TF32 MMAs are implemented as compact matrix micro-ops that consume integer and uniform-datapath resources rather than a long stream of scalar FP instructions, increasing predicate/control traffic for tile scheduling but shrinking the FP portion of the dynamic mix. In contrast, the channels-last pipeline (OPT2/OPT12) moves the aggregate mix toward the profile of elementwise and layout kernels, which are dominated by integer address arithmetic, control flow, and load/store/movement operations. On Ampere/Hopper, this is visible in the elevated shares of control, predicate, and movement instructions in Figure 9, indicating that performance is increasingly limited by index generation and tensor-coordinate conversions rather than by raw floating-point throughput. In summary, OPT1 primarily re-encodes arithmetic into MMA-dominated instruction streams, whereas OPT2/OPT12 restructure Med-DDPM into a memory- and layout-bound regime with substantially higher non-arithmetic instruction overhead.

6.2 Microarchitectural Analysis

The hardware-usage counters reinforce the earlier kernel-mix and cycle/IPC analysis by making explicit how OPT1 redistributes work between Tensor Cores and the traditional CUDA FP32 pipelines on Ampere and Hopper. On H100 and A100, baseline Med-DDPM barely engages Tensor Cores (Avg Tensor Core Util ≈ 1.4%), consistent with the dominance of FP32 cuDNN-style convolutions. Under OPT1, Tensor Core utilization jumps to 7.81% on H100 and 9.98% on A100, a 5–7× increase, while average CUDA core utilization drops from 20.60% → 10.53% (H100) and 29.39% → 18.46% (A100) as shown in Figure 10. This matches the earlier observation that TF32 activation compresses scalar FMA work into dense MMA instructions: the SM warp schedulers feed a smaller number of Tensor Core–resident conv/GEMM instructions that retire far more arithmetic per issue, allowing the FP32 pipelines to idle more without hurting throughput. The modest reduction in average SM throughput (37.14% → 31.27% on H100 and 32.64% → 27.77% on A100) should therefore be read alongside the large drops in cycles and instructions: the SMs are less "busy" in a normalized sense because there is simply less dynamic work to perform per sample, not because they are starved. The integer ALU utilization also decreases slightly (e.g., H100 21.38% → 18.11%, A100 22.66% → 20.00%), indicating that address-generation and loop-control overheads shrink with the shorter, more Tensor Core–dominated conv loops. On V100, Tensor Core utilization remains at zero for all configurations, and both CUDA core and SM throughput metrics stay essentially flat between baseline and OPT1 (Avg CUDA Core Util ≈ 23.6%, Avg SM Thruput ≈ 25.1%), which is consistent with the cycle, instruction, and kernel-mix data. Volta operates exclusively on the FP32 convolution path, as it lacks TF32-capable Tensor Cores.

The channels-last optimization (OPT2) and its combination with TF32 (OPT12) expose a different architectural regime where the SMs are underutilized even as total cycles and instructions collapse. On H100, OPT2 and OPT12 leave Tensor Core utilization near baseline levels (1.47% and 1.62% versus 1.44%), while CUDA core utilization falls sharply to 6.81% and 6.66% and SM throughput to 18.18% and 17.60%, approximately half of the baseline. A100 shows the same pattern: Tensor Core utilization only nudges up to 1.59%–1.76%, whereas CUDA core utilization drops from 29.39% to 12.01%–11.69% and SM throughput from 32.64% to 16.48%–16.16%. Combined with the kernel-mix data which shows that channels-last shifts execution toward elementwise kernels and NHWC-related padding and

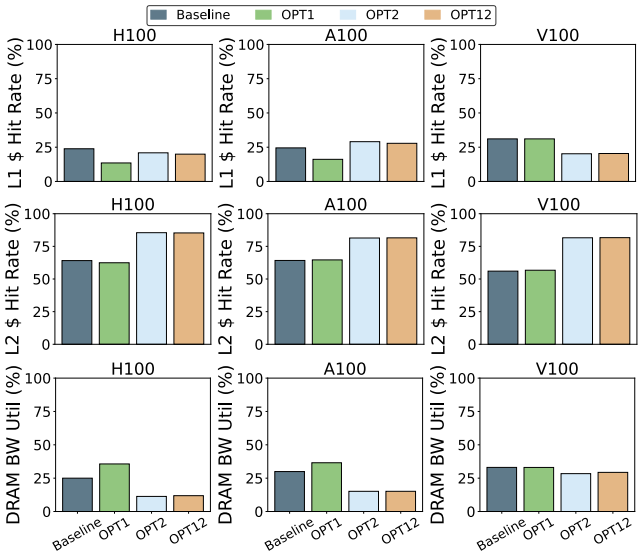

Figure 11: L1/L2 Hit Rate, DRAM BW Utilization

transform operations the hardware counters indicate that the workload has entered a highly memory-bound, low-arithmetic-intensity regime. At the same time, neither CUDA cores nor Tensor Cores are being driven near their compute limits as the SMs spend more time stalled on data movement and kernel launch/tear-down overhead than doing floating-point work. On V100, OPT2/OPT12 drop CUDA core utilization from 23.63% to 9.65% and SM throughput from 25.11% to ∼ 14.5%, with Tensor Core utilization remaining at zero; this aligns with the earlier observation of much fewer total instructions and cycles but a higher relative IPC, implying that when warps do execute, they use the FP32 pipelines more efficiently, yet the overall SM residency is lower due to fragmentation into many small elementwise/layout kernels. Taken together, these hardware-usage trends show that OPT1 achieves genuine compute-path compression by shifting work from FP32 cores to high-throughput Tensor Cores on Ampere/Hopper, whereas the current channels-last implementation (OPT2/OPT12) reduces overall work but fails to keep either compute engine saturated, making kernel fusion and more aggressive Tensor Core–aware tiling essential for turning the observed cycle reductions into sustained, architecture-scaled speedups.

6.3 Memory System Analysis

The cache and DRAM statistics show that OPT1 fundamentally changes how Med-DDPM uses the memory hierarchy on Ampere and Hopper, in a way that is consistent with the Tensor Core–centric execution pattern identified earlier. On H100 and A100, OPT1 reduces the L1 cache hit rate from 23.86% → 13.51% and 24.50% → 16.15%, respectively, while leaving the L2 cache hit rate essentially unchanged relative to baseline (64.10% → 62.44% on H100 and 64.21% → 64.63% on A100), as shown in Figure 11. At the same time, DRAM bandwidth utilization increases sharply, from 25.00% → 35.66% on H100 and 29.95% → 36.55% on A100, with V100 showing flat L1/L2 behavior and essentially unchanged DRAM utilization. Microarchitecturally, this pattern suggests that TF32-enabled Tensor Core convolutions are issuing larger, more contiguous global-memory tiles that stream through the SMs with

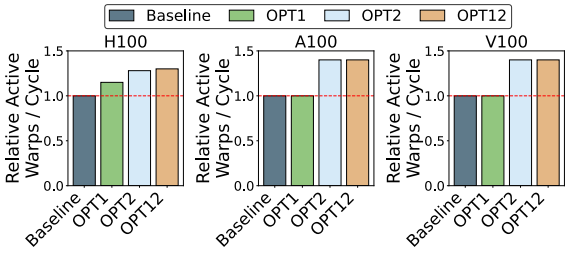

Figure 12: Relative Active Warps/Cycle

less fine-grained temporal locality at L1. The per-tile reuse window is narrower, so fewer accesses are captured in the small L1, but the accesses are still regular enough to be serviced efficiently by the L2. The higher DRAM bandwidth utilization reflects the fact that, per unit of SM active time, the Tensor Core path pulls data more aggressively from memory to keep the MMA pipelines fed, even though the total number of cycles and instructions per sample decreases. In other words, OPT1 trades some L1 locality for higher sustained memory throughput into Tensor Cores, aligning with the earlier observation that conv/GEMM kernels shrink in time while Tensor Core utilization rises and CUDA core/INT ALU activity drops.

In contrast, the channels-last optimization (OPT2) and its combination with TF32 (OPT12) push the workload into a regime where the L2 cache absorbs most of the traffic and DRAM is significantly underutilized, reinforcing the view that these configurations are dominated by low-intensity elementwise and layout kernels rather than dense convolutions. Across all three architectures, the L2 hit rate jumps dramatically under OPT2/OPT12, from 64.10% → 85.47% on H100, 64.21% → 81.40% on A100, and 56.03% → 81.57% on V100 (with nearly identical values for OPT12), while DRAM bandwidth utilization collapses to 11.32%–11.87% on H100 and 15.14%–15.15% on A100 and only slightly decreases on V100 (33.07% → 28.39%). L1 behavior becomes more architecture- and layout-sensitive: on A100, the L1 hit rate *increases* to 29.03% (OPT2) and 27.84% (OPT12), consistent with channels-last improving spatial locality for many pointwise and padding kernels, whereas on H100 and V100, the L1 hit rate either recovers only partially from OPT1 or drops below baseline. Taken together with the earlier kernel-mix and SM utilization results, these memory statistics indicate that OPT2/OPT12 dramatically reduce the amount of data that must be fetched from DRAM and concentrate traffic in L2, but they do so by replacing large, high-flop Tensor Core convolutions with numerous small, memory-bound kernels whose working sets fit comfortably in cache. The SMs thus spend more time stalled on control and launch overhead with neither Tensor Cores nor CUDA cores saturated, and the memory system is over-provisioned relative to the reduced arithmetic intensity. This explains why OPT2/OPT12 achieve large reductions in cycles and instructions without translating into proportionate, architecture-scaled speedups. The memory hierarchy is no longer the bottleneck, but the compute engines are also not being driven at their design limits.

6.4 Warp Efficiency Analysis

The warp-level behavior further clarifies the execution regimes created by the two optimizations and aligns closely with the earlier kernel, memory, and SM-utilization findings. Figure 12 presents relative active warps per cycle. OPT1 shows only modest changes

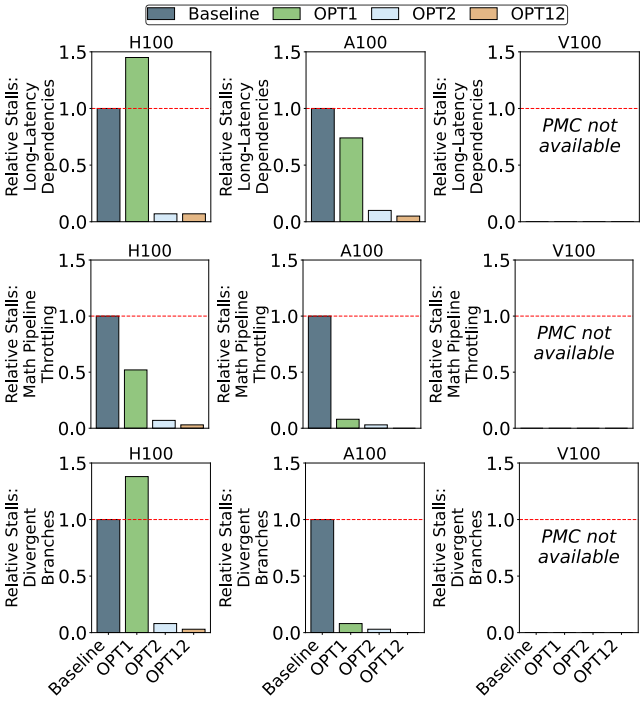

Figure 13: Relative Stall Breakdown

in active warps per cycle H100 increases slightly to 1.15× baseline while A100 and V100 remain at 1.00× because TF32-accelerated convolutions still launch large, well-tiled kernels whose occupancy and scheduling structure broadly match the baseline FP32 path. The main effect of OPT1 is reducing dynamic instructions and cycles, not altering warp residency. In contrast, OPT2 and OPT12 substantially increase relative active warps on three GPUs, reaching 1.30× on H100 and 1.40× on A100/V100. This reflects the fragmentation of 3D U-Net execution into many smaller elementwise, padding, and layout kernels whose working sets fit comfortably in L2 and often in L1, enabling high resident-warp counts even as arithmetic intensity collapses. These kernels are lightweight enough that the scheduler can keep more warps in flight, but as shown by the drastic drops in SM throughput, Tensor Core utilization, and DRAM bandwidth, they do not generate sufficient compute or memory pressure to saturate the hardware. Thus, high active-warp density under OPT2/OPT12 does not translate into performance as Warp occupancy is high, but execution efficiency remains low. This mirrors the earlier observation that the channels-last path shifts the workload from dense, compute-bound convs into a high-occupancy yet low-efficiency regime dominated by memory-bound micro-kernels.

6.5 Scheduling Efficiency Analysis

The scheduler-level stall breakdown clarifies why OPT1 shifts MedDDPM into a Tensor-Core–dominated execution regime on Ampere and Hopper and aligns with the earlier kernel, memory, and warp analyses. Figure 13 shows relative stalls due to dependencies, throttling, and divergent branches. On H100, OPT1 increases long-latency-dependency stalls to 1.45× baseline and divergent-branch stalls to 1.38×, while sharply reducing math-pipeline throttling to 0.52×. This pattern reflects Tensor Core–enabled convolutions that issue large, deep MMA tiles whose operand availability often depends on multi-stage global-memory pipelines (as previously evidenced by lower L1 hit rates, stable L2 behavior, and significantly higher DRAM bandwidth). The drop in math-throttling stalls indicates that the FP32 pipelines are no longer the bottleneck Instead, the SMs increasingly wait on long dependency chains for Tensor Core operands, spanning global-memory loads, shared-memory staging, and tile setup. A100, however, shows a smoother schedule under OPT1. All stall categories decrease (long-latency 0.74×, math throttling 0.08×, divergent 0.08×), consistent with the earlier findings that TF32 greatly reduces cycles and instructions while keeping IPC near or above baseline.

In contrast, OPT2 and OPT12 push the model into a near "stall-free" regime on both H100 and A100, with all stall categories collapsing to 0.03–0.10× baseline. When viewed alongside the extremely high L2 hit rates, minimal DRAM bandwidth usage, and elevated active-warp density, this reveals that the channels-last path decomposes the 3D U-Net into many short, cache-resident elementwise/layout kernels with minimal dependency depth and little opportunity for either math-pipeline pressure or long memory-latency exposure. However, as earlier SM-throughput and Tensor/Core CUDA utilization results showed, this state is not performance optimal: although warps rarely stall, they also perform very little useful computation per issue, fail to engage Tensor Cores meaningfully, and operate at low arithmetic intensity. Thus, OPT2/OPT12 eliminate classic stall sources not because the pipeline is maximally utilized, but because the workload has been transformed into a high-occupancy, low-productivity sequence of micro-kernels explaining why dramatic reductions in cycles and instructions do not translate into architecture-scaled speedups. On V100, all three stall classes remain effectively zero across configurations, confirming that neither TF32 nor channels-last materially alters its FP32-centric scheduling behavior.

7 Related Work

Medical Imaging using Machine Learning: Machine learning has reshaped medical imaging by enabling automated diagnosis and prognosis prediction. Recent advances in deep learning architectures have significantly improved performance in tumor detection, organ segmentation, and disease classification [15, 27–30]. The models trained on multimodal datasets now support precision oncology by integrating radiology, pathology, and genomics data. Also, regulatory bodies such as the FDA have begun approving AI-based imaging tools, underscoring their clinical relevance [46, 52, 56, 61, 63].
Denoising Diffusion Probabilistic Models: Denoising Diffusion Probabilistic Models (DDPMs) have emerged as powerful generative frameworks for medical image synthesis and reconstruction [12, 19, 64]. Unlike GANs, DDPMs offer stable training and controllable sampling, making them suitable for high-fidelity 3D medical imaging. Recent work such as Medical Diffusion demonstrates DDPMs applied to volumetric MRI generation, achieving realistic anatomical structures [17, 31, 55]. However, the computationally heavy workload of these models demands the need for optimized GPU kernels and efficient architectural approaches to make training and inference practical at scale. [24, 31, 34, 35, 41].
Synthetic Image Generation: Synthetic data generation addresses key limitations in medical imaging, including data scarcity, privacy constraints, and annotation cost [20, 21, 49]. GAN-based methods

have been widely used to generate synthetic radiographs, CT scans, and MRIs [14, 36, 43]. More recently, diffusion models have been employed to produce high-resolution synthetic 3D images with anatomical labels [48, 50]. Generating such datasets requires significant load on GPU clusters, requiring efficient design of data pipelines, mixed-precision execution, and distributed data generation workflows to maintain throughput. [38, 42].

Performance Analysis of Diffusion Models: Diffusion models are computationally intensive, requiring GPU-level optimization. Recent work explores acceleration techniques such as FP8 quantization and TensorRT integration to reduce latency and cost on Hopper GPUs [13, 62]. However, there is no in-depth profiling of diffusion models. [1, 11, 32]. Profiling tools like Nsight Compute provide performance analysis with PMCs (Performance Monitoring Counters), guiding kernel fusion and layout transformations. Studies also emphasize the role of memory bandwidth and launch overhead in shaping performance bottlenecks [26, 54].

8 Conclusion

This work provides a comprehensive, system-level analysis of MedDDPM across three NVIDIA GPU architectures and demonstrates that 3D diffusion workloads are dominated by cuDNN convolution and implicit-GEMM kernels whose inefficiencies stem from memory-access behavior, tensor-layout conversions, and limited Tensor Core engagement. Using detailed Nsight Compute profiling, we show that enabling TF32 Tensor Core paths and adopting a 3D channels-last layout materially reshapes the microarchitectural execution profile. TF32 consistently compresses arithmetic work on A100 and H100, reducing SM cycles by up to 5× while preserving IPC and memory balance, whereas the channels-last layout exposes a new regime dominated by elementwise and layout kernels that achieve high cache locality but underutilize compute pipelines. Together, these findings highlight that effective acceleration of large 3D diffusion models requires aligning precision modes, memory formats, and kernel behavior with underlying GPU microarchitecture, and they establish TF32 as a robust optimization baseline.

References

[1] C. Chen, C. Giannoula, and A. Moshovos. 2024. Low-Bitwidth Floating Point Quantization for Efficient High-Quality Diffusion Models. In *Proceedings of the 2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, Vancouver, BC, Canada, 181–193. doi:10.1109/IISWC63097.2024.00025

[2] Chen Chen, Chen Qin, Huaqi Qiu, Cheng Ouyang, Shuo Wang, and Daniel Rueckert. 2020. Realistic Adversarial Data Augmentation for MR Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI) (Lecture Notes in Computer Science, Vol. 12261)*. Springer, 667–677. doi:10.1007/978-3-030-59710-8_65

[3] Hyungjin Chung, Eun Sun Lee, and Jong Chul Ye. 2023. MR Image Denoising and Super-Resolution Using Regularized Reverse Diffusion. *IEEE Transactions on Medical Imaging* 42, 4 (2023), 922–934. doi:10.1109/TMI.2022.3220681

[4] NVIDIA Corporation. 2017. *NVIDIA Tesla V100 GPU Architecture*. Technical Report. NVIDIA. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[5] NVIDIA Corporation. 2020. *NVIDIA A100 Tensor Core GPU Architecture*. Technical Report. NVIDIA. https://www.nvidia.com/content/dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

[6] NVIDIA Corporation. 2022. *NVIDIA H100 Tensor Core GPU Architecture*. Technical Report. NVIDIA. https://resources.nvidia.com/en-us-hopper-architecture/nvidia-h100-tensor-c

[7] NVIDIA Corporation. 2023. *Nsight Compute Kernel Profiling Guide*. Technical Report. NVIDIA Corporation. https://docs.nvidia.com/nsight-compute/2023.2/pdf/ProfilingGuide.pdf v2023.2.2.

[8] NVIDIA Corporation. 2023. NVIDIA Hopper H100 GPU: Scaling Performance. *IEEE Micro* 43, 4 (2023), 56–65. doi:10.1109/MM.2023.10070122

[9] NVIDIA Corporation. 2025. *Nsight Compute Profiling Guide*. https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html Version 2025.3.1.

[10] Bill Dally. 2023. The Secret to NVIDIA's AI Success. *IEEE Spectrum* (2023). https://spectrum.ieee.org/nvidia-gpu

[11] M. Dombrowski, H. Reynaud, J. P. Müller, M. Baugh, and B. Kainz. 2024. Trade-Offs in Fine-Tuned Diffusion Models between Accuracy and Interpretability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. AAAI Press, 21037–21045. doi:10.1609/aaai.v38i19.30095

[12] Z. Dorjsembe, H.-K. Pao, S. Odonchimed, and F. Xiao. 2024. Conditional Diffusion Models for Semantic 3D Brain MRI Synthesis. *IEEE Journal of Biomedical and Health Informatics* 28, 7 (July 2024), 4084–4093. doi:10.1109/JBHI.2024.3385504

[13] J. Ekelund, S. Markidis, and I. Peng. 2025. Boosting Performance of Iterative Applications on GPUs: Kernel Batching with CUDA Graphs. In *Proceedings of the 2025 33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*. IEEE, Turin, Italy, 70–77. doi:10.1109/PDP66500.2025.00019

[14] N. Gaggion, L. Mansilla, C. Mosquera, D. H. Milone, and E. Ferrante. 2023. Improving Anatomical Plausibility in Medical Image Segmentation via Hybrid Graph Neural Networks: Applications to Chest X-Ray Analysis. *IEEE Transactions on Medical Imaging* 42, 2 (February 2023), 546–556. doi:10.1109/TMI.2022.3224660

[15] Irena Galić, Marija Habijan, Hrvoje Leventić, and Krešimir Romić. 2023. Machine Learning Empowering Personalized Medicine: A Comprehensive Review of Medical Image Analysis Methods. *Electronics* 12, 21, Article 4411 (2023). doi:10.3390/electronics12214411

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*.

[17] P. Guo, Y. Mei, J. Zhou, S. Jiang, and V. M. Patel. 2024. ReconFormer: Accelerated MRI Reconstruction Using Recurrent Transformer. *IEEE Transactions on Medical Imaging* 43, 1 (January 2024), 582–593. doi:10.1109/TMI.2023.3314747

[18] Bagus Hanindhito and Lizy K. John. 2024. Accelerating ML Workloads using GPU Tensor Cores: The Good, the Bad, and the Ugly. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24)*. ACM, 12. doi:10.1145/3629526.3653835

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[20] Shu-Hui Hsu, Zhaohui Han, Jonathan E. Leeman, Yue-Houng Hu, Raymond H. Mak, and Atchar Sudhyadhom. 2022. Synthetic CT generation for MRI-guided adaptive radiotherapy in prostate cancer. *Frontiers in Oncology* 12 (2022). doi:10.3389/fonc.2022.969463

[21] I. Irmakci, Z. E. Unel, N. Ikizler-Cinbis, and U. Bagci. 2022. Multi-Contrast MRI Segmentation Trained on Synthetic Images. In *Proceedings of the 2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, Glasgow, Scotland, United Kingdom, 5030–5034. doi:10.1109/EMBC48229.2022.9871119

[22] Zhe Jia, Michael Garland, and Yuandong Tian. 2016. Dissecting GPU Memory Hierarchy Through Microbenchmarking. *IEEE Transactions on Parallel and Distributed Systems* 27, 7 (2016), 1944–1957. doi:10.1109/TPDS.2016.2531642

[23] Chutian Jiang. 2021. Efficient Quantization Techniques for Deep Neural Networks. In *Proceedings of the 2021 International Conference on Signal Processing and Machine Learning (CONF-SPML)*. IEEE, 271–277. doi:10.1109/CONF-SPML54095.2021.00059

[24] H. Jiang, Z. Wang, D. Liu, L. Guo, et al. 2025. Fast-DDPM: Fast Denoising Diffusion Probabilistic Models for Medical Image-to-Image Generation. *IEEE Journal of Biomedical and Health Informatics* 29, 10 (October 2025), 7326–7335. doi:10.1109/JBHI.2025.3565183

[25] Mingfeng Jiang, Peihang Jia, Xin Huang, Zihan Yuan, Dongsheng Ruan, Feng Liu, and Ling Xia. 2025. Frequency-Aware Diffusion Model for Multi-Modal MRI Image Synthesis. *Journal of Imaging* 11, 5 (2025), 152. doi:10.3390/jimaging11050152

[26] W. Kong et al. 2024. Cambricon-D: Full-Network Differential Acceleration for Diffusion Models. In *Proceedings of the 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, Buenos Aires, Argentina, 903–914. doi:10.1109/ISCA59077.2024.00070

[27] R. R. Kumar, S. V. Shankar, R. Jaiswal, et al. 2025. Advances in Deep Learning for Medical Image Analysis: A Comprehensive Investigation. *Journal of Statistical Theory and Practice* 19, 1 (2025), 9. doi:10.1007/s42519-024-00422-2

[28] Rachel Lawrence, Emma Dodsworth, Efthalia Massou, Chris Sherlaw-Johnson, Angus I. G. Ramsay, Holly Walton, Tracy O'Regan, Fergus Gleeson, Nadia Crellin, Kevin Herbert, Pei Li Ng, Holly Elphinstone, Raj Mehta, Joanne Lloyd, Amanda Halliday, Stephen Morris, and Naomi J. Fulop. 2025. Artificial intelligence for diagnostics in radiology practice: a rapid systematic scoping review. *eClinicalMedicine* 83 (2025). doi:10.1016/j.eclinm.2025.103228

[29] H. Laçi, K. Sevrani, and S. Iqbal. 2025. Deep learning approaches for classification tasks in medical X-ray, MRI, and ultrasound images: a scoping review. *BMC Medical Imaging* 25, 1 (2025), 156. doi:10.1186/s12880-025-01701-5

[30] Mengfang Li, Yuanyuan Jiang, Yanzhou Zhang, and Haisheng Zhu. 2023. Medical image analysis using deep learning algorithms. *Frontiers in Public Health* 11 (2023). doi:10.3389/fpubh.2023.1273253

[31] D. Liu, Z. Wang, and L. Guo. 2025. A Plug-and-Play Diffusion-Styled Conversion Model for Domain Discrepancies in Medical Image Segmentation. In *Proceedings of the 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Hyderabad, India, 1–5. doi:10.1109/ICASSP49660.2025.10889167

[32] Y. Liu, Y. Feng, J. Cheng, H. Zhan, and Z. Zhu. 2025. MambaDiff: Mamba-Enhanced Diffusion Model for 3D Medical Image Segmentation. *IEEE Transactions on Image Processing* 34 (2025), 5761–5775. doi:10.1109/TIP.2025.3607615

[33] Yifan Liu and Xipeng Shen. 2021. Analyzing and Leveraging Decoupled L1 Caches in GPUs. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 1–11. doi:10.1109/ISPASS48437.2021.9407080

[34] Z. Liu, A. Song, N. Sabar, and W. Li. 2024. Evolving a Better Scheduler for Diffusion Models. In *PRICAI 2023: Trends in Artificial Intelligence (Lecture Notes in Computer Science, Vol. 14326)*, F. Liu, A. A. Sadanandan, D. N. Pham, P. Mursanto, and D. Lukose (Eds.). Springer, Singapore. doi:10.1007/978-981-99-7022-3_37

[35] Y. Luo, Q. Yang, Y. Fan, H. Qi, and M. Xia. 2024. Measurement Guidance in Diffusion Models: Insight from Medical Image Synthesis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 12 (December 2024), 7983–7997. doi:10.1109/TPAMI.2024.3399098

[36] Alessio Luschi, Linda Tognetti, Alessandra Cartocci, Elisa Cinotti, Giovanni Rubegni, Laura Calabrese, Martina D'onghia, Martina Dragotto, Elvira Moscarella, Gabriella Brancaccio, Giulia Briatico, Camila Scharf, Dario Buononato, Vittorio Tancredi, Carmen Cantisani, Camilla Chello, Luca Ambrosio, Pietro Scribani Rossi, Marco Virone, Giovanni Pellacani, Gabriele Cevenini, Pietro Rubegni, and Ernesto Iadanza. 2025. Design and development of a systematic validation protocol for synthetic melanoma images for responsible use in medical artificial intelligence. *Biocybernetics and Biomedical Engineering* 45, 4 (2025), 608–616. doi:10.1016/j.bbe.2025.09.001

[37] Gustav Müller-Franzes, David Zimmerer, Fabian Isensee, and Klaus H. Maier-Hein. 2023. A Multimodal Comparison of Latent Denoising Diffusion Probabilistic Models and Generative Adversarial Networks for Medical Image Synthesis. *Scientific Reports* 13, 1 (2023), 12456. doi:10.1038/s41598-023-39278-0

[38] Maham Nazir, Muhammad Aqeel, and Francesco Setti. 2025. Diffusion-Based Data Augmentation for Medical Image Segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*. IEEE, 1330–1339.

[39] Alexander Q. Nichol and Prafulla Dhariwal. 2021. Improved Denoising Diffusion Probabilistic Models. In *Proceedings of the 38th International Conference on Machine Learning*.

[40] IEEE Transactions on Medical Imaging. 2024. Special Issue on Score-Based Generative Models for Medical Imaging. *IEEE Transactions on Medical Imaging* (2024).

[41] Geon Yeong Park, Sang Wan Lee, and Jong Chul Ye. 2025. Inference-Time Diffusion Model Distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 4049–4058.

[42] J. Peng et al. 2022. Knowledge-Driven Generative Adversarial Network for Text-to-Image Synthesis. *IEEE Transactions on Multimedia* 24 (2022), 4356–4366. doi:10.1109/TMM.2021.3116416

[43] Matteo Pozzi, Shahryar Noei, Erich Robbi, Luca Cima, Monica Moroni, Enrico Munari, Evelin Torresani, and Giuseppe Jurman. 2024. Generating and evaluating synthetic data in digital pathology through diffusion models. *Scientific Reports* 14, 1 (November 2024), 28435. doi:10.1038/s41598-024-79602-w

[44] Chen Qian, Haoyu Zhang, Dan Ruan, Yirong Zhou, and Xiaobo Qu. 2023. Physics-Informed Deep Diffusion MRI Reconstruction: Break the Bottleneck of Training Data in Artificial Intelligence. In *Proceedings of the IEEE International Symposium on Biomedical Imaging (ISBI)*. IEEE, 1–5. doi:10.1109/ISBI53787.2023.10230567

[45] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, 234–241. doi:10.1007/978-3-319-24574-4_28

[46] Ravi K. Samala, Karen Drukker, Amita Shukla-Dave, Heang-Ping Chan, Berkman Sahiner, Nicholas Petrick, Hayit Greenspan, Usman Mahmood, Ronald M. Summers, Georgia Tourassi, Thomas M. Deserno, Daniele Regge, Janne J. Näppi, Hiroyuki Yoshida, Zhimin Huo, Quan Chen, Daniel Vergara, Kenny H. Cha, Richard Mazurchuk, Kevin T. Grizzard, Henkjan Huisman, Lia Morra, Kenji Suzuki, III Armato, Samuel G., and Lubomir Hadjiiski. 2024. AI and machine learning in medical imaging: key points from development to translation. *BJR Artificial Intelligence* 1, 1 (April 2024), ubae006. doi:10.1093/bjrai/ubae006

[47] Vikash Sehwag, Xianghao Kong, Jingtao Li, Michael Spranger, and Lingjuan Lyu. 2025. Stretching Each Dollar: Diffusion Training from Scratch on a Micro-Budget. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 28596–28608.

[48] Isabella Barbosa Silva, Elsa Oliveira, Ricardo Melo, Luís Rosado, César Gálvez-Barrón, Irene Bernadet Heijink, Sem Hoogteijling, and Iñigo Gabilondo. 2025. Designing for Qualitative Evaluation of Synthetic Medical Data. In *Extended Abstracts of the 2025 CHI Conference on Human Factors in Computing Systems (CHI EA '25)*. Association for Computing Machinery, New York, NY, USA, Article 185, 11 pages. doi:10.1145/3706599.3720274

[49] Elena Sizikova, Andreu Badal, Jana G. Delfino, Miguel Lago, Brandon Nelson, Niloufar Saharkhiz, Berkman Sahiner, Ghada Zamzmi, and Aldo Badano. 2024. Synthetic data in radiological imaging: current state and future outlook. *BJR Artificial Intelligence* 1, 1 (May 2024), ubae007. doi:10.1093/bjrai/ubae007

[50] Jinzhuo Wang, Kai Wang, Yunfang Yu, Yuxing Lu, Wenchao Xiao, Zhuo Sun, Fei Liu, Zixing Zou, Yuanxu Gao, Lei Yang, Hong-Yu Zhou, Hanpei Miao, Wenting Zhao, Lisha Huang, Lingchao Zeng, Rui Guo, Ieng Chong, Boyu Deng, Linling Cheng, Xiaoniao Chen, Jing Luo, Meng-Hua Zhu, Daniel Baptista-Hon, Olivia Monteiro, Ming Li, Yu Ke, Jiahui Li, Simiao Zeng, Taihua Guan, Jin Zeng, Kanmin Xue, Eric Oermann, Huiyan Luo, Yun Yin, Kang Zhang, and Jia Qu. 2025. Self-improving generative foundation model for synthetic medical image generation and clinical applications. *Nature Medicine* 31, 2 (February 2025), 609–617. doi:10.1038/s41591-024-03359-y

[51] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik. 2003. Multi-Scale Structural Similarity for Image Quality Assessment. In *Proceedings of the 37th Asilomar Conference on Signals, Systems and Computers*.

[52] Asim Waqas, Aakash Tripathi, Ravi P. Ramachandran, Paul A. Stewart, and Ghulam Rasool. 2024. Multimodal data integration for oncology in the era of deep neural networks: a review. *Frontiers in Artificial Intelligence* 7 (2024). doi:10.3389/frai.2024.1408843

[53] George Webber and Andrew J. Reader. 2024. Diffusion Models for Medical Image Reconstruction. *BJR|Artificial Intelligence* 1, 1 (2024), ubae013. doi:10.1093/bjrai/ubae013

[54] Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, Christian Rupprecht, Daniel Cremers, Peter Vajda, and Jialiang Wang. 2024. Cache Me if You Can: Accelerating Diffusion Models through Block Caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 6211–6220.

[55] K. Xu, S. Lu, B. Huang, W. Wu, and Q. Liu. 2024. Stage-by-Stage Wavelet Optimization Refinement Diffusion Model for Sparse-View CT Reconstruction. *IEEE Transactions on Medical Imaging* 43, 10 (October 2024), 3412–3424. doi:10.1109/TMI.2024.3355455

[56] Tony Xu, Sepehr Hosseini, Chris Anderson, Anthony Rinaldi, Rahul G. Krishnan, Anne L. Martel, and Maged Goubran. 2025. A generalizable 3D framework and model for self-supervised learning in medical imaging. *npj Digital Medicine* 8, 1 (2025), 639. doi:10.1038/s41746-025-02035-w

[57] Charlene Yang, Thorsten Kurth, and Samuel Williams. 2020. Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System. *Concurrency and Computation: Practice and Experience* 32, 24 (2020), e5547. doi:10.1002/cpe.5547

[58] Charlene Yang, Yunsong Wang, Thorsten Kurth, Samuel Williams, and Steven Farrell. 2020. Hierarchical Roofline Performance Analysis for Deep Learning Applications. In *Proceedings of SC '20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE/ACM. doi:10.1109/SC41405.2020.00045

[59] Xin Yi, Ekta Walia, and Paul Babyn. 2019. Generative Adversarial Network in Medical Imaging: A Review. *Medical Image Analysis* (2019).

[60] Haoyu Zhang, Chen Qian, and Xiaobo Qu. 2023. A Reconfigurable Processing Element for Multiple-Precision Floating/Fixed-Point HPC. *IEEE Transactions on Circuits and Systems II: Express Briefs* 70, 10 (2023), 3456–3460. doi:10.1109/TCSII.2023.10272667

[61] T. Zhang, X. Chen, C. Qu, A. Yuille, and Z. Zhou. 2024. Leveraging AI Predicted and Expert Revised Annotations in Interactive Segmentation: Continual Tuning or Full Training?. In *Proceedings of the 2024 IEEE International Symposium on Biomedical Imaging (ISBI)*. IEEE, Athens, Greece, 1–5. doi:10.1109/ISBI56570.2024.10635518

[62] J. Zhao and S. Li. 2025. Radiomics-Driven Diffusion Model and Monte Carlo Compression Sampling for Reliable Medical Image Synthesis. *IEEE Journal of Biomedical and Health Informatics* (2025). doi:10.1109/JBHI.2025.3602674

[63] Z. Zhao, F. Zhou, K. Xu, Z. Zeng, C. Guan, and S. K. Zhou. 2023. LE-UDA: Label-Efficient Unsupervised Domain Adaptation for Medical Image Segmentation. *IEEE Transactions on Medical Imaging* 42, 3 (March 2023), 633–646. doi:10.1109/TMI.2022.3214766

[64] Zhenyu Zhou, Defang Chen, Can Wang, Chun Chen, and Siwei Lyu. 2024. Simple and Fast Distillation of Diffusion Models. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 40831–40860. doi:10.52202/079017-1291

[65] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 2016. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI*.