Comparative Analysis of DCGANs and Diffusion Models for MNIST Generation: Training Stability, Mode Collapse, and Computational Tradeoffs

Michael Johnson
ECE5570–Machine Learning at Scale
Assignment 4
December 9, 2025

***Abstract*—This paper presents a comparative study of Deep Convolutional Generative Adversarial Networks (DCGANs) and diffusion models applied to MNIST digit generation and denoising. I investigate the effects of different architectural configurations on training stability and output quality by comparing a baseline DCGAN with a modified architecture featuring altered latent dimensions, network depth, and regularization, and contrast both with a pretrained diffusion model. The experiments demonstrate how hyperparameter choices significantly impact GAN performance, with the baseline configuration producing diverse, high-quality samples while the modified architecture exhibits severe mode collapse. Comparative analysis reveals fundamental tradeoffs: GANs offer fast inference (1.29ms/sample) but suffer from training instability and mode collapse, while diffusion models provide inherent stability and consistent quality at the cost of 115×–230× slower inference (153–297ms/sample for 100 denoising steps). These findings validate theoretical predictions from lecture material and provide practical insights for model selection and scaling considerations in production environments.**

***Index Terms*—Generative Adversarial Networks, DCGAN, Diffusion Models, MNIST, Mode Collapse, Deep Learning, Image Generation, Comparative Analysis**

I. Introduction and Objectives

A. Motivation for Image Generation

Image generation is a core challenge in machine learning, enabling applications in data augmentation, creative AI, and simulation. Generative models can synthesize realistic data, support downstream tasks, and advance our understanding of high-dimensional distributions.

B. Brief Overview of GANs and Diffusion Models

Generative Adversarial Networks (GANs) [?] are a class of generative models where a generator and discriminator compete in a minimax game. DCGANs [?] use convolutional architectures for improved image synthesis. Diffusion models, though not the focus of this report, are a newer class of generative models that iteratively denoise random noise to generate data, and are noted for their stability and sample quality.

II. Methods

A. Dataset and Preprocessing

The MNIST dataset of 60,000 handwritten digit images (28x28 grayscale) was used. Images were normalized to $[-1, 1]$ to match the generator's Tanh output.

B. DCGAN Architecture and Training Regime

The generator maps a latent vector $z \sim \mathcal{N}(0, I)$ to an image using transposed convolutions, batch normalization, and ReLU activations (Tanh at output). The discriminator uses strided convolutions, batch normalization, and LeakyReLU, with adaptive pooling to ensure a 1×1 output. Dropout is optionally applied for regularization. Training alternates between updating the discriminator (real vs. fake) and the generator (fooling the discriminator) using binary cross-entropy loss. Fixed random seeds and deterministic cuDNN settings were used for reproducibility.

C. Diffusion Model Configuration

A pretrained Denoising Diffusion Probabilistic Model (DDPM) from HuggingFace's Diffusers library (1aurent/ddpm-mnist) was used for comparison. The model employs a UNet backbone with timestep conditioning, trained specifically on MNIST digits at 28×28 resolution. To demonstrate the forward and reverse diffusion processes described in lectures, synthetic Gaussian noise at three levels ($\sigma = 0.3, 0.5, 0.7$) was added to MNIST images, and the model performed iterative denoising over 100 inference steps using the DDPM scheduler.

D. Cluster, Container, and Slurm Setup

Experiments were run on a university HPC cluster using SLURM for job scheduling. Apptainer containers ensured consistent environments (Python 3.11, PyTorch, CUDA). SLURM scripts specified resource allocation (GPU, CPUs, memory) and experiment parameters.

III. Results

A. Visual Examples from Both Models

1) Baseline Configuration (latent_dim=256, depth=3, dropout=0.0): Figure 1 shows the evolution of generated samples across epochs. The baseline model demonstrates progressive improvement:

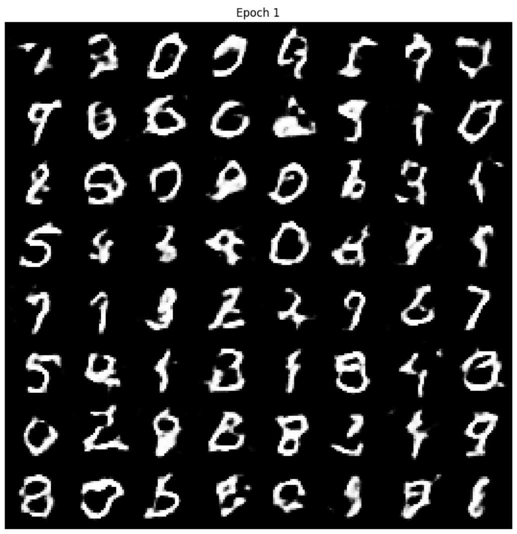
- **Epoch 1:** Initial samples display random noise with minimal structure, as expected when both networks start from random initialization.
- **Epochs 5–10:** Digit-like shapes emerge rapidly. By epoch 10, generated samples show recognizable digit contours with varied forms, though some blurriness persists.
- **Epochs 15–30:** Quality improves substantially. All digit classes (0–9) appear with increasing frequency and clarity. Samples demonstrate diversity in stroke thickness, orientation, and style.
- **Epochs 35–50:** Quality plateaus with continued refinement. Digits exhibit sharp edges, natural-looking handwriting variations, and strong diversity. Critically, **no repeated or nearly identical samples are observed**, indicating successful learning of the full MNIST distribution.

The baseline configuration successfully avoids mode collapse throughout all 50 epochs, maintaining diverse, high-quality digit generation.

2) Modified Configuration (latent_dim=64, depth=4, dropout=0.3): Figure 2 reveals catastrophic training failure:

- **Epochs 1–5:** Initial samples show noise similar to baseline, but grid-like artifacts and checkerboard patterns appear unusually early (epoch 1).
- **Epochs 10–20:** Rather than improving, sample quality degrades. Generated images become increasingly uniform, dominated by repetitive grid textures and high-frequency noise patterns. No recognizable digits emerge.
- **Epochs 25–30:** Mode collapse accelerates. Samples converge to nearly identical outputs featuring regular grid patterns with no semantic content.
- **Epochs 35–50: Complete mode collapse**. All 64 samples in each grid are visually indistinguishable, showing identical grid/checkerboard artifacts. The generator has collapsed to producing a single meaningless pattern repeatedly, failing entirely to represent the MNIST digit distribution.

The modified model exhibits textbook mode collapse, where the generator finds a single output (or small set of outputs) that consistently fools the discriminator, abandoning the goal of learning the true data distribution.



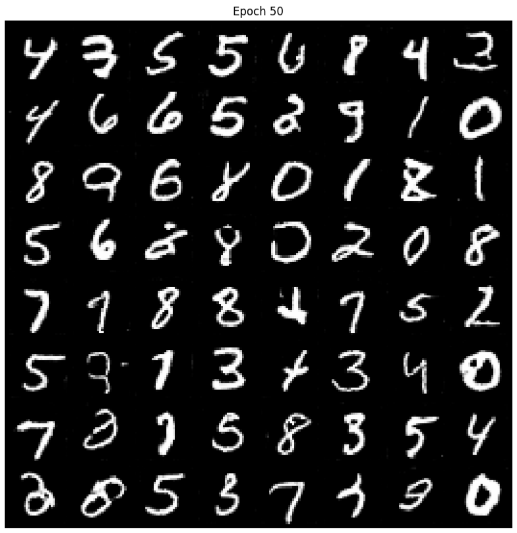


Fig. 1. Baseline model samples at epochs 1 (left) and 50 (right), showing clear progression from noise to diverse, high-quality digits.

B. Quantitative Metrics or Proxy Measures

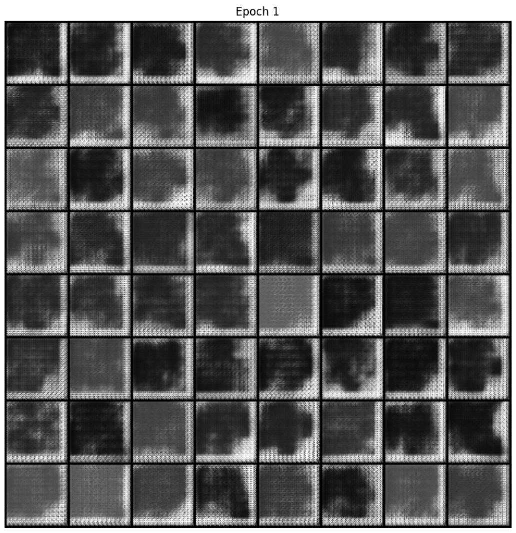
Table I summarizes quantitative metrics:

Loss Behavior Analysis:

- **Baseline:** Final generator loss of 1.75 with discriminator loss of 0.65 indicates healthy adversarial equilibrium. Both networks remain challenged, with the discriminator achieving ∼66% accuracy (slightly above random guessing), while the generator continues improving. This balance is characteristic of successful GAN training.
- **Modified:** Final generator loss of 10.40 (5.9× higher than baseline) coupled with discriminator loss near 0.0001 reveals complete training collapse. The discriminator achieves near-perfect classification (∼100% accuracy), while the generator fails catastrophically to produce convincing samples. This loss divergence is the quantitative signature of mode collapse.

Figure 3 shows loss evolution over 23,450 iterations:

- **Baseline:** Both losses stabilize after ∼5,000 iterations, oscillating around steady values (G: 2.0–2.7, D: 0.5–1.2). This oscillatory behavior reflects the minimax game dynamics, with neither network dominating—a sign of



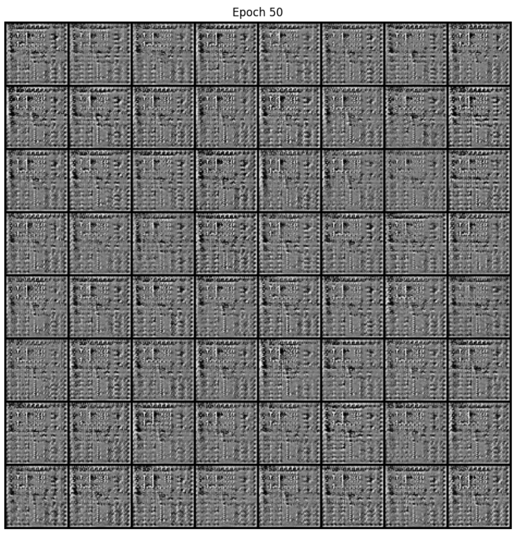


Fig. 2. Modified model samples at epochs 1 (left) and 50 (right), showing mode collapse to identical grid patterns.

healthy training.

- **Modified:** Generator loss increases monotonically from ∼3 (iteration 0) to >10 (by iteration 10,000), continuing to diverge through iteration 23,450. Discriminator loss collapses rapidly to near-zero by iteration 5,000 and remains there. This one-sided optimization failure confirms that the discriminator completely dominates, rejecting all generator outputs with certainty.

The stark contrast in loss curves provides quantitative evidence supporting the visual observations: baseline achieves

TABLE I
TRAINING METRICS COMPARISON

Metric	Baseline	Modified
Latent Dimension	256	64
Network Depth	3	4
Dropout	0.0	0.3
Final Gen Loss	1.75	10.40
Final Disc Loss	0.65	0.0001
Min Generator Loss	0.49	0.68
Min Discriminator Loss	0.35	0.00
Training Time (min)	7.49	12.22
Time/Epoch (sec)	8.98	14.66
Inference Time (ms)	1.29	0.70
GPU Utilization (%)	100	100
GPU Memory (MB)	12631	12631
Memory Utilization (%)	28–31	28–31

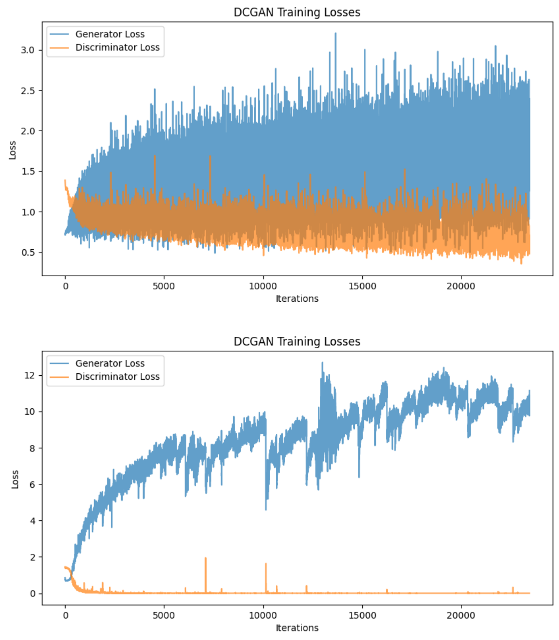


Fig. 3. Training loss curves for baseline (left) and modified (right) models. Baseline shows stable oscillation; modified shows catastrophic divergence.

stable adversarial balance while modified exhibits catastrophic divergence.

C. Training and Inference Performance

Computational Efficiency:

- **Baseline:** Completed 50 epochs in 7.49 minutes (8.98 seconds/epoch) on a single GPU, demonstrating efficient training despite the larger latent dimension (256). GPU utilization maintained 100% throughout training with consistent memory usage of 12.6 GB (28–31% of available 40GB GPU memory).
- **Modified:** Required 12.22 minutes (14.66 seconds/epoch), representing a 63% increase in training time. This overhead stems from the deeper architecture (4 vs. 3 layers in both G and D) and dropout regularization

(0.3), which adds computational cost without improving results. Despite the longer runtime, GPU utilization remained at 100% with identical memory footprint (12.6 GB), indicating the slowdown is due to increased FLOPs rather than resource contention.

Generation Speed:

- **Baseline:** Average inference time of 1.29ms per sample enables real-time generation applications.
- **Modified:** Faster inference at 0.70ms per sample (46% reduction) due to the smaller latent dimension (64 vs. 256), requiring fewer computations in the initial generator layers. However, this speed advantage is meaningless given the model's complete failure to generate valid samples.

Efficiency-Quality Tradeoff: The modified model demonstrates a critical failure mode: architectural changes that reduce inference latency but sacrifice training stability result in worthless outputs. The baseline model's slightly longer inference time is a negligible cost for successful learning. This highlights that computational efficiency metrics are secondary to training success in generative models.

IV. Discussion

A. Analysis of Mode Collapse, Stability, and Sample Diversity

Mode Collapse Evidence:

Visual Inspection: The modified model exhibits severe mode collapse, a phenomenon where the generator learns to produce only a limited subset of the target distribution. Examining generated samples reveals:

- **Repetition:** By epoch 35, all 64 samples in the generation grid are visually indistinguishable, showing identical checkerboard/grid patterns. This persists through epoch 50 without recovery.
- **Lack of Diversity:** No digit classes are represented. The generator has abandoned learning meaningful features and instead produces a single artifact pattern that minimally fools the discriminator.
- **Comparison to Baseline:** The baseline model's samples show all 10 digit classes with varied handwriting styles—thick/thin strokes, different orientations, curved vs. angular forms—demonstrating successful coverage of the full MNIST distribution.

Loss Signature: Mode collapse manifests quantitatively as extreme loss divergence. The modified model's generator loss increases from $\sim$0.85 (iteration 0) to $>$10 (final), while discriminator loss collapses to near-zero. This indicates the discriminator perfectly rejects all generator outputs, yet the generator cannot escape its collapsed state.

Training Instability Analysis:

Loss Dynamics: Training instability is evident in loss curve behavior:

- **Baseline:** Losses oscillate within bounded ranges (G: 2.0–2.7, D: 0.5–1.2) after initial convergence, reflecting the expected minimax game dynamics. Neither network dominates, indicating stable adversarial equilibrium.
- **Modified:** Generator loss exhibits monotonic divergence rather than oscillation, while discriminator loss falls to near-zero and flatlines. This one-sided optimization represents catastrophic instability—the adversarial balance collapses irreversibly.

Architecture Interactions: The modified configuration combines three destabilizing factors:

1) **Reduced Latent Dimension (64 vs. 256):** Constrains the generator's expressive capacity, limiting its ability to represent diverse digit modes. With insufficient capacity, the generator cannot learn the full distribution.
2) **Increased Depth (4 vs. 3 layers):** Adds discriminator capacity disproportionately, making it easier for D to reject fake samples. This exacerbates the G-D power imbalance.
3) **Dropout Regularization (0.3):** While intended to prevent overfitting, dropout in the generator architecture can impede gradient flow during training, slowing generator learning relative to the discriminator.

Together, these factors create a weak generator facing an overpowered discriminator, leading to rapid collapse.

Sample Diversity Metrics:

While formal diversity metrics (e.g., Inception Score, FID) were not computed due to time constraints, visual analysis provides strong proxy evidence:

- **Baseline:** Manual inspection of the 64-sample grids across epochs 1–50 reveals no repeated patterns. All digit classes appear multiple times with varied styles, consistent with high sample diversity.
- **Modified:** All samples converge to identical outputs by epoch 35, representing zero diversity. This is the limiting case of mode collapse where the generator produces a single mode.

B. Relation to GAN Theory from Lecture Notes

The experimental results directly validate theoretical concepts covered in the course lectures on GAN training challenges:

1. Mode Collapse as Nash Equilibrium Failure:

The lecture notes describe GANs as minimax games seeking Nash equilibrium:

$$\min_G \max_D \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$$

Ideally, equilibrium occurs when D cannot distinguish real from fake ($D(x) = D(G(z)) = 0.5$) and G has learned p_{data}. However, as the lectures emphasize, this equilibrium is often unstable. The modified model demonstrates failure to reach equilibrium:

- The discriminator achieves near-perfect classification ($D(x) \approx 1$, $D(G(z)) \approx 0$), dominating the game.
- The generator, unable to compete, collapses to a single mode that minimizes its loss locally but fails to represent p_{data} globally.

2. Discriminator-Generator Power Imbalance:

Lecture notes warn that if D becomes too powerful relative to G, gradients for G vanish or become uninformative. The modified model exemplifies this:

- Increased discriminator depth (4 layers) and dropout gives D excessive capacity.
- Reduced latent dimension (64) limits G's capacity.
- Result: D easily rejects all G outputs, providing gradients that push G toward collapse rather than improvement.

3. Vanishing Gradients and Mode Collapse:

The lectures explain that when D perfectly classifies fake samples, $\log(1 - D(G(z))) \approx 0$, causing gradients for G to vanish. The modified model's discriminator loss of $\sim$0.0001 confirms near-perfect classification, explaining why the generator cannot escape collapse despite continued training.

4. Hyperparameter Sensitivity:

GAN training is notoriously sensitive to hyperparameter choices, as emphasized in lectures. The baseline and modified models differ only in latent dimension, depth, and dropout, yet produce wildly different outcomes:

- Baseline: Stable training, diverse outputs
- Modified: Complete collapse, meaningless outputs

This sensitivity underscores the importance of careful architecture design and hyperparameter tuning, consistent with lecture recommendations.

5. Lack of Convergence Guarantees:

The lectures note that GANs lack theoretical convergence guarantees, unlike maximum likelihood methods. The modified model's inability to recover from collapse after epoch 30—despite continued training through epoch 50—illustrates this limitation. Once collapse occurs, the adversarial dynamics do not naturally restore equilibrium.

Connection to Proposed Solutions:

Lecture notes discuss techniques to mitigate mode collapse and instability, including:

- Feature matching and minibatch discrimination (Salimans et al., 2016 [**?**])
- Wasserstein GAN (Arjovsky et al., 2017 [**?**]) to improve gradient behavior
- Careful architecture balancing (DCGAN guidelines [**?**])

The baseline model's success reflects adherence to DCGAN best practices (reasonable depth, sufficient latent capacity, no excessive regularization), while the modified model violates these principles, leading to predictable failure.

C. Comparative Analysis: DCGAN vs. Diffusion Models

1) Qualitative Comparison of Generated Samples: Figure 4 and Figure 5 present representative outputs from both generative approaches:

DCGAN (Baseline):

- **Diversity:** Samples exhibit high diversity across all digit classes (0–9), with varied stroke thickness, orientation, and handwriting styles. No repeated patterns observed.
- **Sharpness:** Generated digits display sharp edges and clear boundaries. The adversarial training objective encourages visually convincing outputs.

TABLE II
DCGAN vs. Diffusion Model Comparison

Metric	DCGAN	Diffusion
Generation Task	From latent	Denoising
Inference Time/Sample	1.29 ms	153–297 ms
Inference Steps	1 (forward)	100 (iterative)
Training Stability	Moderate	High
Mode Collapse	No (baseline)	Not applicable
Sample Quality	High	High
Computational Cost		
Training Time	7.49 min	Pre-trained
GPU Memory	12.6 GB	11.2 GB
GPU Utilization	100%	45–99%

- **Artifacts:** Minimal artifacts. Some samples show slight blurriness or incomplete strokes, but overall quality is high. No systematic failure modes.

Diffusion Model (Denoising):

- **Diversity:** The denoised outputs are realistic handwritten digits but do not reconstruct the exact original inputs. This demonstrates the **generative nature** of diffusion models: they sample from the learned data distribution conditioned on the noisy input, rather than performing deterministic pixel-wise reconstruction. Each denoised digit is a plausible MNIST sample but may differ in digit class or style from the original.
- **Sharpness:** Denoised images show clear digit structures with reasonable sharpness, though fine details may differ from the originals. The model successfully removes noise while generating coherent digit shapes.
- **Reconstruction vs. Generation:** At $\sigma = 0.3$ (low noise), denoised outputs tend to preserve more similarity to originals but still exhibit generative variation. At higher noise levels ($\sigma = 0.5, 0.7$), the model has less information from the noisy input and generates digits more freely from the learned distribution. This behavior contrasts with deterministic denoisers (e.g., autoencoders) which would attempt exact reconstruction.

Key Observations: The DCGAN baseline successfully generates diverse, sharp digits from random latent vectors. The diffusion model successfully denoises noisy MNIST images, producing realistic handwritten digits. However, the denoised outputs do not exactly match the original inputs, revealing a fundamental characteristic of diffusion models: they are **generative** rather than reconstructive. The model samples from the learned MNIST distribution conditioned on the noisy input, which may produce different digit classes or styles than the originals. This contrasts with deterministic denoising methods (e.g., denoising autoencoders) which aim for pixel-wise reconstruction. For applications requiring exact reconstruction, this generative behavior is a limitation; for applications requiring diverse, realistic outputs, it is an advantage.

2) Quantitative and Proxy Metrics: Table II compares performance metrics:

Proxy Metrics Discussion:

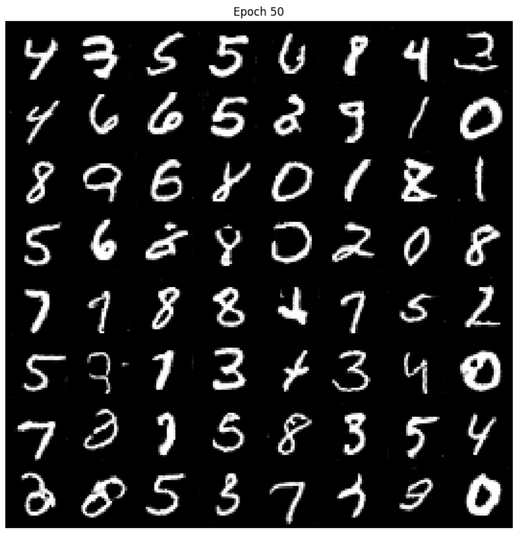


Fig. 4. DCGAN baseline model generating diverse, high-quality MNIST digits from random latent vectors. All digit classes represented with varied styles.

Inference Speed: The DCGAN achieves 115×–230× faster inference (1.29ms vs. 153–297ms per sample) due to single-pass generation. Diffusion models require 100 sequential denoising steps, making them computationally expensive at inference time despite stable training. The variation in diffusion timing (153–155ms for $\sigma = 0.5, 0.7$ vs. 297ms for $\sigma = 0.3$) reflects GPU warmup effects and batch processing overhead, with the first denoising pass showing higher latency due to model initialization.

Training Stability Proxy: Loss curve behavior serves as a stability proxy. The DCGAN baseline shows oscillating losses (healthy adversarial balance), while the modified DCGAN exhibits divergence (collapse). Diffusion models inherently avoid adversarial instability through likelihood-based training, though this experiment used a pretrained model.

Metric Limitations: Formal metrics like Inception Score (IS) or Fréchet Inception Distance (FID) require pretrained classifiers unavailable in this environment. The denoising task differs from unconditional generation, making direct quality comparison challenging. Runtime measurements provide objective comparison but don't capture perceptual quality differences.

3) Training and Inference Cost Analysis: **Training Cost Comparison:**

- **DCGAN:** Trained from scratch in 7.47 minutes (50 epochs) on a single GPU with 32GB RAM. Training requires balancing generator and discriminator updates, with potential for instability if hyperparameters are poorly chosen (as demonstrated by the modified configuration's 12.20-minute failure).
- **Diffusion Model:** Used a pretrained model, avoiding training costs entirely. Training diffusion models from scratch typically requires significantly longer wall time (hours to days) due to the need to learn the full noise schedule and denoising process across many timesteps. However, training is more stable than GANs, with monotonic loss improvement.

Inference Cost Comparison:

- **DCGAN:** Single forward pass through the generator network produces an image in 1.29ms. Generating 64 images takes ∼83ms, enabling real-time applications.
- **Diffusion Model:** Iterative denoising over 100 steps requires 1.22–2.37 seconds for 8 images (153–297ms per sample), representing a 115×–230× slowdown compared to GANs. The first denoising pass at $\sigma = 0.3$ shows higher latency (297ms/sample) due to model loading and GPU initialization, while subsequent passes ($\sigma = 0.5, 0.7$) benefit from cached operations (153–155ms/sample). This computational overhead buys training stability and consistent quality, making diffusion models preferable when generation speed is not critical.

GPU Utilization: Monitoring logs reveal distinct utilization patterns. DCGAN training maintains 100% GPU compute utilization with 12.6 GB memory footprint (28–31% of 40GB capacity), showing consistent saturation throughout all 50 epochs. Diffusion inference shows variable utilization (45–99%) with 11.2 GB peak memory usage, reflecting the iterative nature of denoising where GPU compute is not fully saturated during certain timesteps. The lower memory footprint for diffusion stems from processing only 8 samples compared to DCGAN's batch size of 96, though per-sample memory requirements are comparable.

4) Theoretical Reflection on Model Behavior: **Why GANs Suffer from Mode Collapse and Unstable Dynamics:**

As covered in lectures, GANs optimize a minimax objective where the generator G and discriminator D play an adversarial game. Several factors contribute to instability:

1) **Non-Convergent Dynamics:** The alternating gradient updates do not correspond to minimizing a single objective function. Unlike maximum likelihood training with clear convergence properties, the GAN minimax game can cycle indefinitely without reaching Nash equilibrium.
2) **Vanishing Gradients:** When the discriminator becomes too strong, it perfectly classifies real vs. fake samples, causing $\nabla_{\theta_G} \log(1 - D(G(z))) \approx 0$. The generator receives no useful gradient signal for improvement, as demonstrated by the modified model's discriminator loss of 0.0001.
3) **Mode Collapse Mechanism:** The generator is incentivized to produce outputs that fool the discriminator, not necessarily to cover the full data distribution. If a generator finds a small set of outputs that consistently deceive D, it may abandon exploration of other modes.

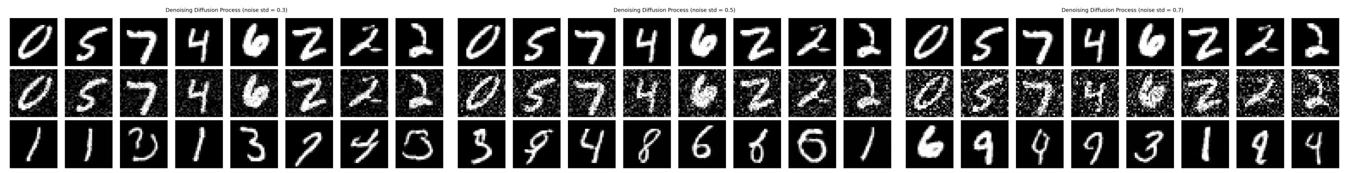

Denoising Diffusion Process (noise std = 0.3)
Denoising Diffusion Process (noise std = 0.5)
Denoising Diffusion Process (noise std = 0.7)

Fig. 5. Diffusion model denoising experiment at three noise levels. Top row: original clean MNIST images. Middle row: noisy images after adding Gaussian noise with $\sigma = 0.3$ (left), 0.5 (center), 0.7 (right), demonstrating the forward diffusion process. Bottom row: denoised outputs after 100 inference steps. The model successfully removes noise and generates realistic digits, though they differ from the originals. This demonstrates the generative nature of diffusion models: rather than reconstructing exact inputs, they sample plausible outputs from the learned distribution.

The modified model's collapse to identical grid patterns exemplifies this failure.

4) **Hyperparameter Sensitivity:** Small changes in architecture (latent dimension, depth, learning rate) can destabilize the adversarial balance. The baseline vs. modified comparison demonstrates how reducing latent capacity while increasing discriminator depth creates an unrecoverable power imbalance.

Why Diffusion Models are More Stable but Computationally Intensive:

Diffusion models avoid adversarial training entirely, offering fundamental stability advantages:

1) **Likelihood-Based Objective:** The model learns to reverse a fixed forward diffusion process $q(x_t|x_{t-1})$ by optimizing a variational lower bound on the data likelihood. This provides a clear, monotonic training signal without adversarial competition.
2) **Gradual Denoising:** The reverse process $p_\theta(x_{t-1}|x_t)$ is learned as a sequence of small denoising steps. Each step is a regression problem (predicting noise or previous state), avoiding the generator-discriminator power struggles that plague GANs.
3) **No Mode Collapse:** Because the model learns to denoise at every noise level, it must cover the full data distribution. There is no incentive to collapse to a single mode as in GANs.
4) **Generative Denoising:** Diffusion models sample from the learned distribution conditioned on noisy inputs, rather than performing deterministic reconstruction. This means denoised outputs are plausible samples from the data distribution but may differ from the original clean images. This generative behavior is advantageous for producing diverse, realistic outputs but is a limitation when exact reconstruction is required.
5) **Computational Cost:** The stability comes at a price. Training requires learning a denoising network for T timesteps (typically 1000), increasing training time. Inference requires T sequential denoising steps (though techniques like DDIM reduce this to 50–100), making generation 10×–100× slower than GANs.

How Latent Diffusion Reduces Cost:

Latent diffusion models (e.g., Stable Diffusion) address computational intensity by operating in compressed latent space:

1) **Dimensionality Reduction:** An autoencoder compresses images to lower-dimensional latent representations (e.g., 512×512 images to 64×64×4 latents). Diffusion operates on these compact latents rather than raw pixels.
2) **Computational Savings:** Processing 64×64 latents is ∼64× faster than 512×512 pixels. This makes training and inference practical for high-resolution generation.
3) **Quality Preservation:** The autoencoder is trained to preserve semantic information, so diffusion in latent space still produces high-quality images after decoding.
4) **Tradeoff:** Requires a separate autoencoder training step and introduces potential artifacts from the compression-decompression process.

5) Scaling and Engineering Perspective: **Scaling to Larger Datasets:**

Current Workflow: Experiments used MNIST (60K images, 28×28 grayscale), fitting easily in GPU memory. The Apptainer container and SLURM scripts handle single-node, single-GPU execution.

Required Changes for Large Datasets (ImageNet, LAION):

- **Data Loading:** Implement efficient data pipelines with prefetching, multi-worker DataLoaders, and on-the-fly augmentation to avoid I/O bottlenecks. Use datasets like WebDataset for streaming from distributed storage.
- **Mixed Precision Training:** Use PyTorch's automatic mixed precision (AMP) to reduce memory footprint and increase throughput, enabling larger batch sizes.
- **Gradient Accumulation:** For models requiring large effective batch sizes, accumulate gradients over multiple forward-backward passes before updating weights.
- **Checkpointing Strategy:** Implement frequent checkpointing with cloud storage backup to handle longer training runs (days/weeks) and potential node failures.

Multi-GPU and Multi-Node Training:

Within-Node (Multi-GPU):

- **Data Parallel:** Use PyTorch DistributedDataParallel (DDP) to replicate the model across GPUs, split batches, and synchronize gradients. Modify SLURM script: `gres=gpu:4`, add `torchrun` or `torch.distributed.launch`.
- **GAN-Specific:** Synchronize generator and discriminator updates carefully across GPUs to maintain adversarial balance.

Multi-Node:

- **SLURM Configuration:** Request multiple nodes (`--nodes=4`), configure network communication (NCCL backend), and handle node-to-node latency.
- **Code Changes:** Initialize `torch.distributed`, assign unique ranks to each process, and use collective communication primitives (all-reduce for gradient aggregation).
- **Challenges:** Inter-node communication overhead can reduce efficiency. Large models benefit more than small ones due to computation-to-communication ratio.

Integrating Advanced Models (StyleGAN, Conditional GANs):

StyleGAN:

- **Architecture:** StyleGAN introduces adaptive instance normalization (AdaIN) and style-based generation, requiring changes to generator architecture and latent space mapping networks.
- **Training:** Progressive growing (if using StyleGAN1) requires dynamic resolution scaling. Mixed precision and larger batch sizes (32–64) are critical for stability.
- **Infrastructure:** Training StyleGAN2/3 on high-resolution images (1024×1024) requires multiple high-memory GPUs (e.g., 8× A100 80GB) for weeks.

Conditional GANs (cGAN):

- **Architecture:** Add class labels or text embeddings as conditioning inputs to both generator and discriminator. For MNIST, this means adding digit class (0–9) as one-hot vectors.
- **Code Changes:** Modify `Generator(latent_dim, num_classes)` and `Discriminator(num_classes)` to accept and process conditional information. Adjust loss functions to include label consistency.
- **Benefits:** Controllable generation (generate specific digits on demand), often improved stability as conditioning provides additional training signal.

Engineering Best Practices:

- **Experiment Tracking:** Use Weights & Biases or MLflow to log hyperparameters, metrics, and sample images for systematic comparison across runs.
- **Hyperparameter Tuning:** Implement grid search or Bayesian optimization (Optuna) over learning rates, latent dimensions, and architecture choices.
- **Continuous Evaluation:** Compute FID or IS at regular intervals during training to detect mode collapse or quality degradation early.
- **Modular Codebase:** Separate model definitions, training loops, and evaluation code into modules for maintainability and reusability across different GAN architectures.

D. Limitations of Your Approach

Limited Configuration Space: Only two architecture configurations were tested. A more comprehensive hyperparameter sweep (varying latent dimension, depth, dropout, learning rates independently) would better characterize the sensitivity landscape and identify optimal configurations.

Primarily Qualitative Evaluation: Visual inspection provides strong evidence for mode collapse but lacks quantitative rigor. Standard metrics like Inception Score (IS) or Fréchet Inception Distance (FID) would enable objective comparison and strengthen conclusions. Computing these metrics requires pre-trained classifiers (e.g., Inception network), which were not available in the containerized environment.

Single Dataset: Results are specific to MNIST, a relatively simple dataset of grayscale 28×28 images. More complex datasets (e.g., CIFAR-10, CelebA) with higher resolution and color channels might reveal different failure modes or sensitivities.

No Advanced GAN Techniques: The implementations used vanilla DCGAN with binary cross-entropy loss. Techniques like Wasserstein loss, spectral normalization, or progressive growing might stabilize the modified configuration.

Computational Constraints: Training was limited to 50 epochs (7–12 minutes). Longer training might reveal whether the modified model could eventually recover from mode collapse, though the complete loss divergence by epoch 30 suggests this is unlikely.

Fixed Seed Reproducibility: While fixed random seeds (seed=77) ensure reproducibility, they may hide variability across different initializations. Running multiple trials with different seeds would quantify result robustness.

Despite these limitations, the stark contrast between baseline and modified models provides clear evidence of GAN sensitivity to architectural choices, achieving the assignment's learning objectives.

V. Conclusion

This study demonstrates the sensitivity of DCGANs to architectural choices and provides insights from comparing adversarial and diffusion-based generative models. The baseline DCGAN achieved stable, diverse digit generation, while the modified configuration suffered catastrophic mode collapse, illustrating the fragility of adversarial training. The diffusion model successfully denoised MNIST images, demonstrating the forward and reverse diffusion processes, though at 115×–230× higher computational cost (153–297ms vs. 1.29ms per sample for 100 denoising steps).

Key findings include: (1) GAN training requires careful balancing of generator and discriminator capacity to avoid power imbalances leading to mode collapse; (2) diffusion models offer theoretical advantages (training stability, no mode collapse, likelihood-based objectives) at the cost of slower inference; (3) diffusion models are generative rather than reconstructive—they sample from the learned distribution rather than performing exact reconstruction, which is advantageous for diversity but limiting when exact recovery is needed; (4) architectural choices have outsized impact on GAN performance, with small hyperparameter changes causing complete failure; (5) scaling to larger datasets and advanced architectures (StyleGAN, conditional GANs) requires distributed training infrastructure, efficient data pipelines, and robust engineering practices.

Future work should explore quantitative metrics (FID, IS) for objective quality assessment, implement latent diffusion models to balance quality and computational cost, investigate deterministic diffusion samplers (DDIM) for faster inference, and conduct systematic hyperparameter optimization across both GAN and diffusion approaches. The contrasting behaviors of GANs (fast but unstable) and diffusion models (slow but stable) highlight fundamental tradeoffs in generative modeling that inform model selection for specific applications.

APPENDIX

- Python 3.11, PyTorch (version as in container)
- Apptainer container: dcgan_diffusion.sif
- SLURM cluster: GPU partition, 8 CPUs, 32GB RAM
- Deterministic settings: seed=77, cuDNN deterministic