By: Yurii Voievidka

(10 pt.) Implement and train (unconditional) Diffusion model using DDPM. Implement inference using DDIM. (pixel-based)

(based on previous HW) you could use MNIST
(based on previous HW) try to use training pipeline from previous HW, but you could use pytorch lightning (or any other) tool for training

---

DDPM (Denoising Diffusion Probabilistic Models) generate images by gradually adding noise and then learning to reverse this process step by step. DDIM (Denoising Diffusion Implicit Models) is a faster variant that skips steps in a structured way, reducing the number of inference steps while maintaining quality. The code I have written implements a **DDPM-based model** using a **U-Net** conditioned on **sinusoidal time embeddings**. It consists of an encoder-decoder architecture with residual connections, downsampling, and upsampling layers. The **DiffusionModel** class defines the forward diffusion process, where Gaussian noise is added progressively. During training, the model learns to predict this noise using an **MSE loss**. At inference, the model can sample new images using a **DDIM-like deterministic procedure** by skipping timesteps.

During training, the model **learns to predict noise** added at each timestep using a simple **MSE loss**.The **forward_diffusion** function applies the noise corruption formula

$$x_t = \hat{\alpha} t_{x0} + 1 - \hat{\alpha}_t \epsilon x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}$$

\epsilon where ϵ\epsilon is Gaussian noise.
The **sampling function** implements **DDIM inference**, using a reduced number of steps compared to standard DDPM sampling.. Instead of iterating through all **1000 timesteps**, the model **selects 50 steps** for inference, significantly speeding up generation.

Key points in architecture of Unet:

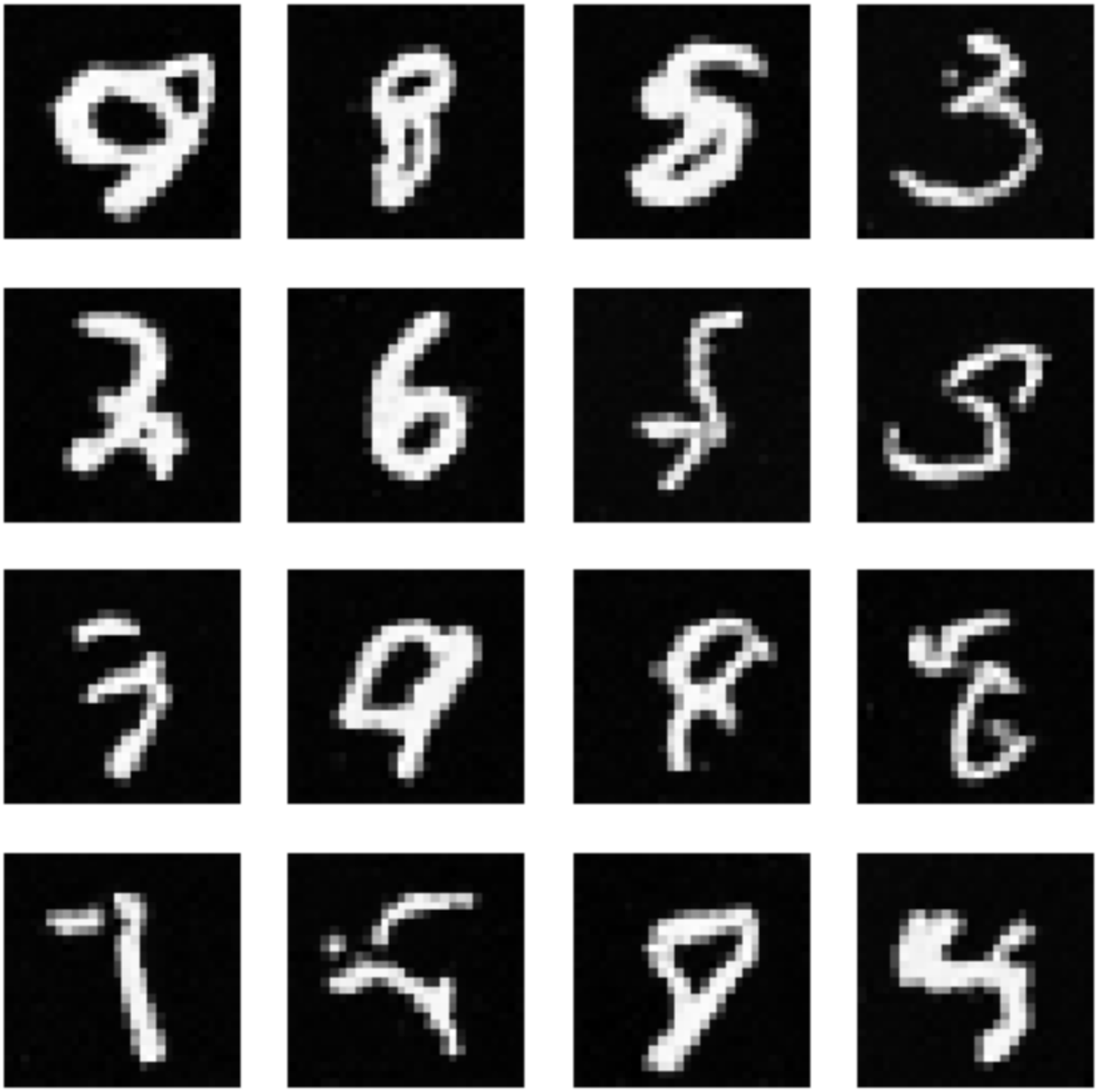
- Sinusoidal Positional Embedding

The **Sinusoidal Positional Embedding (SinusoidalPosEmb)** is used to encode **time information** in the diffusion process, allowing the U-Net to process images differently at each noise level. It provides a **continuous and structured representation** of timesteps, helping the model generalize across different diffusion steps. The sinusoidal function ensures that similar timesteps have similar embeddings, improving **temporal awareness** and stability during training. Unlike learned embeddings, sinusoidal embeddings offer **better generalization** to unseen timesteps and avoid overfitting. By injecting these embeddings into the **UncondBlock** layers, the model effectively learns how to denoise images based on their noise level.

While UNet is quite complex, the DDPM architecture is pretty simple:

```
self.model = nn.Sequential(
        nn.Linear(latent_dim, 128),
        nn.ReLU(),
        nn.Linear(128, latent_dim)
)
```

Results:

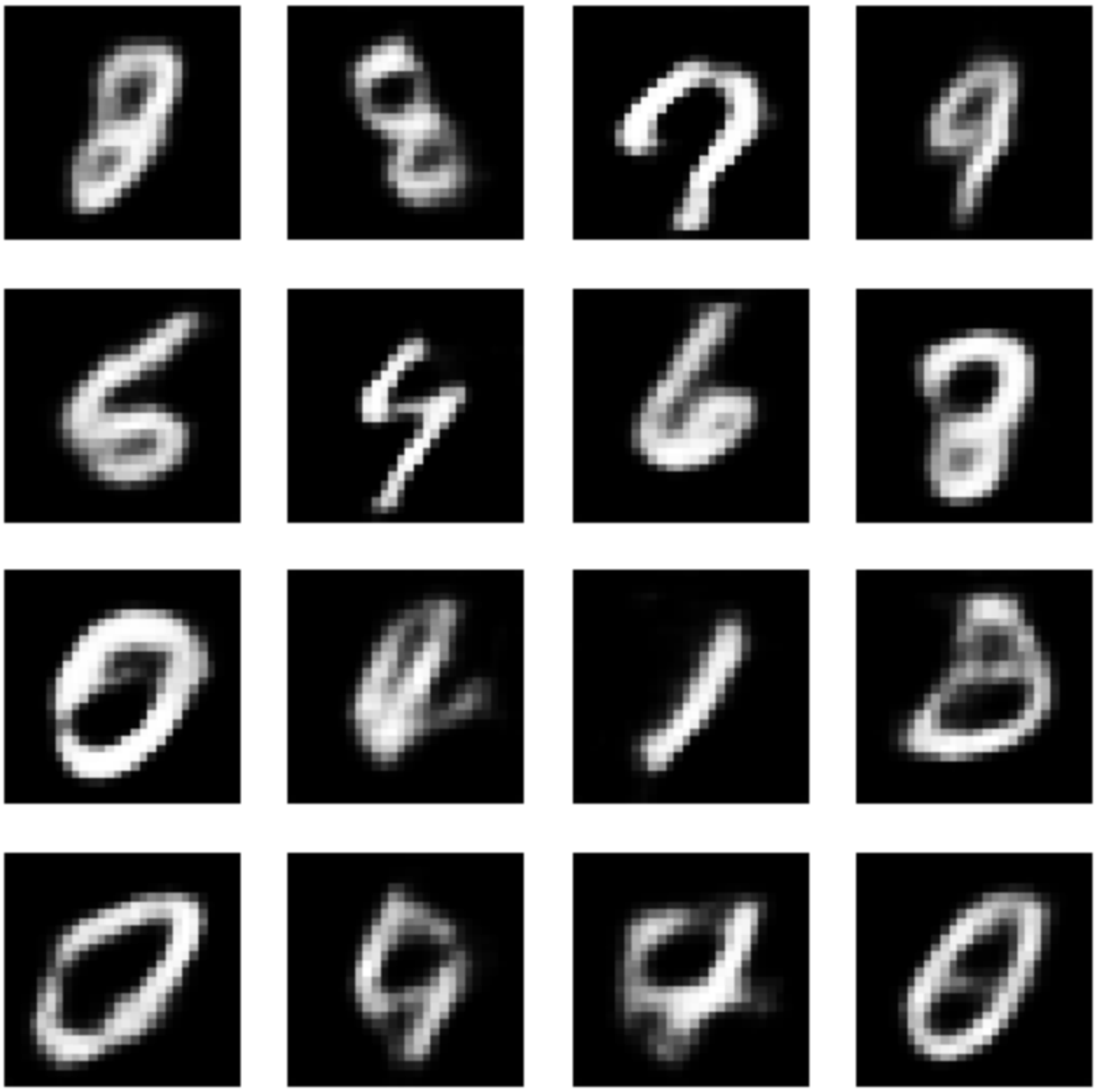


(2 pt.) Using VAE from previous HW (re-train on MNIST if need) and diffusion implementation from above, implement and train latent diffusion model, compare inference from DDPM and DDIM

The **VAE** consists of an encoder that compresses input images into a lower-dimensional latent representation and a decoder that reconstructs images from this latent space. It also introduces a probabilistic structure by learning a mean and variance for each latent vector, using the **reparameterization trick** to sample from a Gaussian distribution. The **DDPM** is trained in this latent space instead of the image space, learning to denoise latent

vectors over multiple timesteps. The DDPM model is a simple multi-layer perceptron (MLP) that predicts the added noise at each timestep, allowing the diffusion process to be reversed during inference. Training first occurs for the VAE on the MNIST dataset, using **reconstruction loss** and a **KL-divergence term** to enforce a structured latent space. Once the VAE is trained, the DDPM is then trained using the encoded latent representations from the VAE. The **DDIM sampling function** gradually refines randomly sampled latent vectors by iteratively subtracting predicted noise over a predefined number of steps. Finally, the denoised latent vectors are passed through the **VAE decoder**, reconstructing images from the learned latent space.

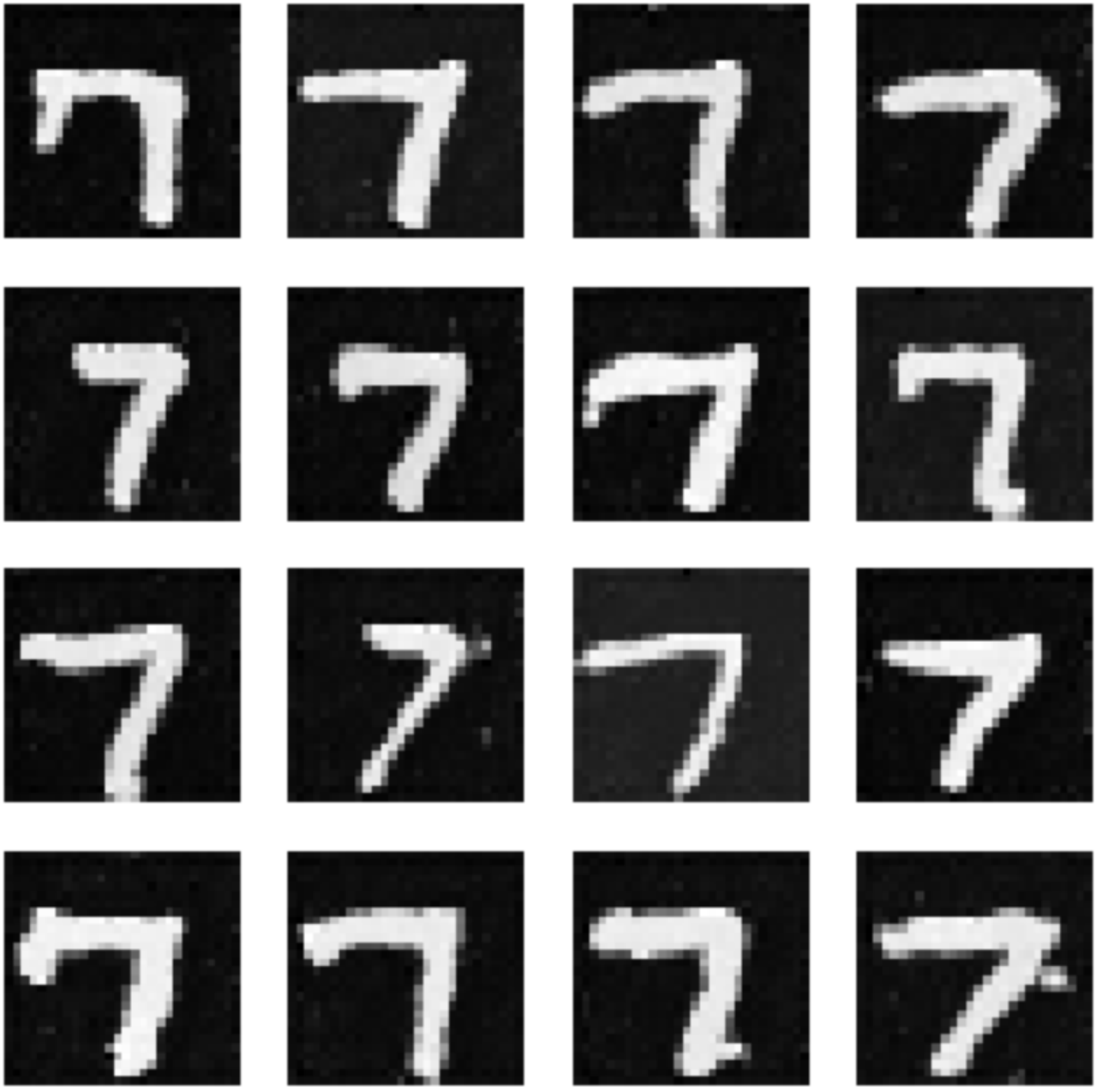
Results:



(3 pt.) Implement and train classifier-free guidance using class label. Condition U-net thought input channel and using cross-attention

classifier-free guidance uses exactly same principles as DDPM, it just takes a label of mnist image as condition and guided by that condition while sampling

Results:



(10pt.) Implement and train diffusion model (pixel and latent based) using Rectified Flow

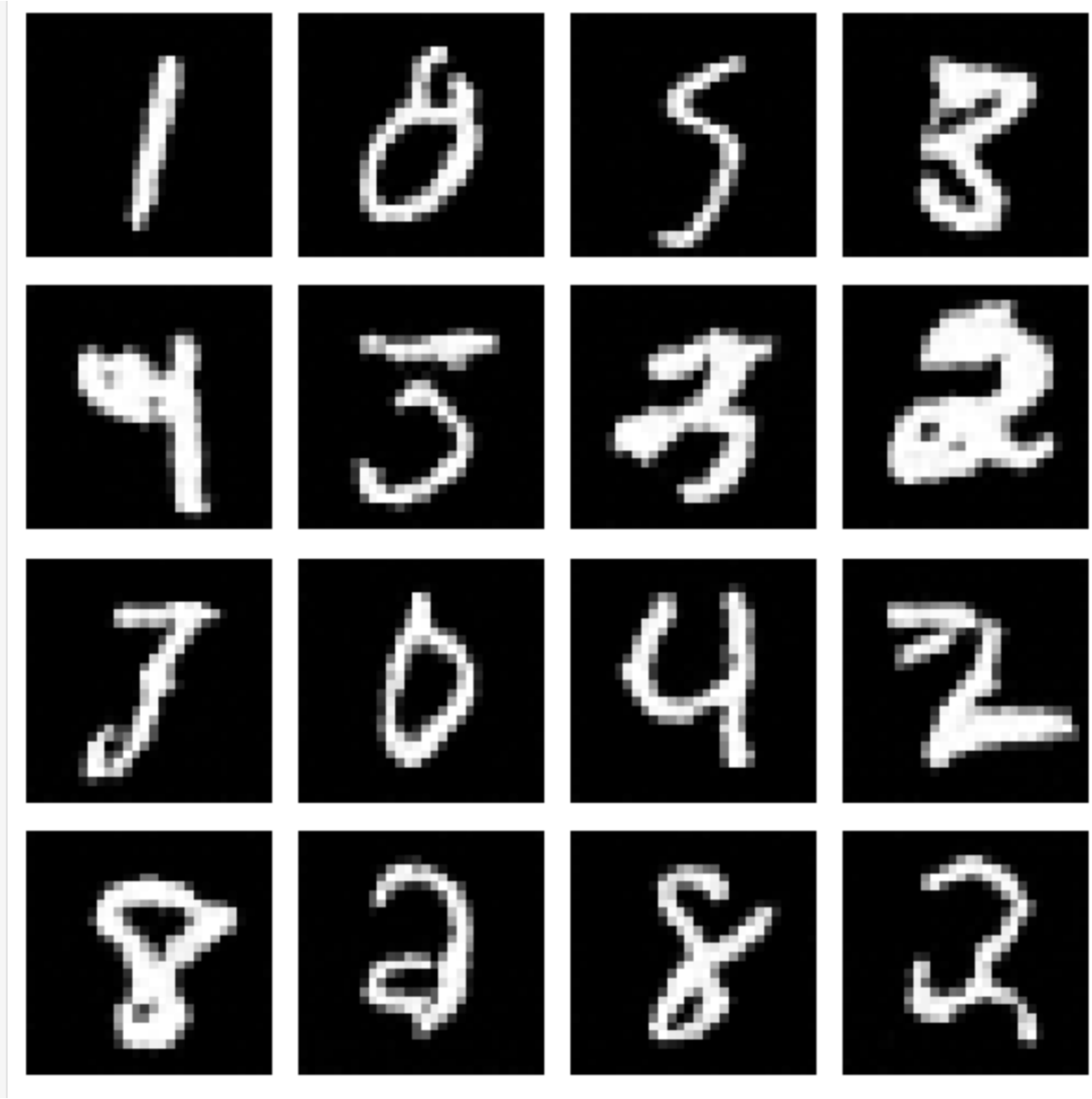
Rectified Flow is a novel approach to generative modeling that aims to improve the efficiency and quality of sampling in diffusion-based models. Unlike traditional diffusion models, which rely on a noisy forward process and a learned denoising reverse process, Rectified Flow directly learns an **optimal transport path** between noise and real data. This means that instead of simulating a complex stochastic process, the model smoothly interpolates data along a rectified, well-behaved trajectory. The key idea is to define a **vector field** that maps points in latent space toward real data in a direct and controlled manner. This eliminates the need for simulating a full stochastic differential equation (SDE) or an ordinary differential equation (ODE) as in standard diffusion models. As a result, Rectified Flow achieves faster convergence, requires fewer inference steps, and generates higher-quality samples. It also reduces issues like **loss of details** or **sampling artifacts**, which can occur when approximating long diffusion trajectories. Training a Rectified Flow model involves optimizing a trajectory that minimizes the discrepancy between data and noise while maintaining a smooth interpolation. Empirically, Rectified Flow has been shown to improve both image generation quality and computational efficiency compared to standard diffusion models. Overall, it represents a promising alternative for training and sampling from generative models more effectively.

The key difference between **Latent Rectified Diffusion** and **Pixel-Based Diffusion** lies in where the diffusion process takes place:

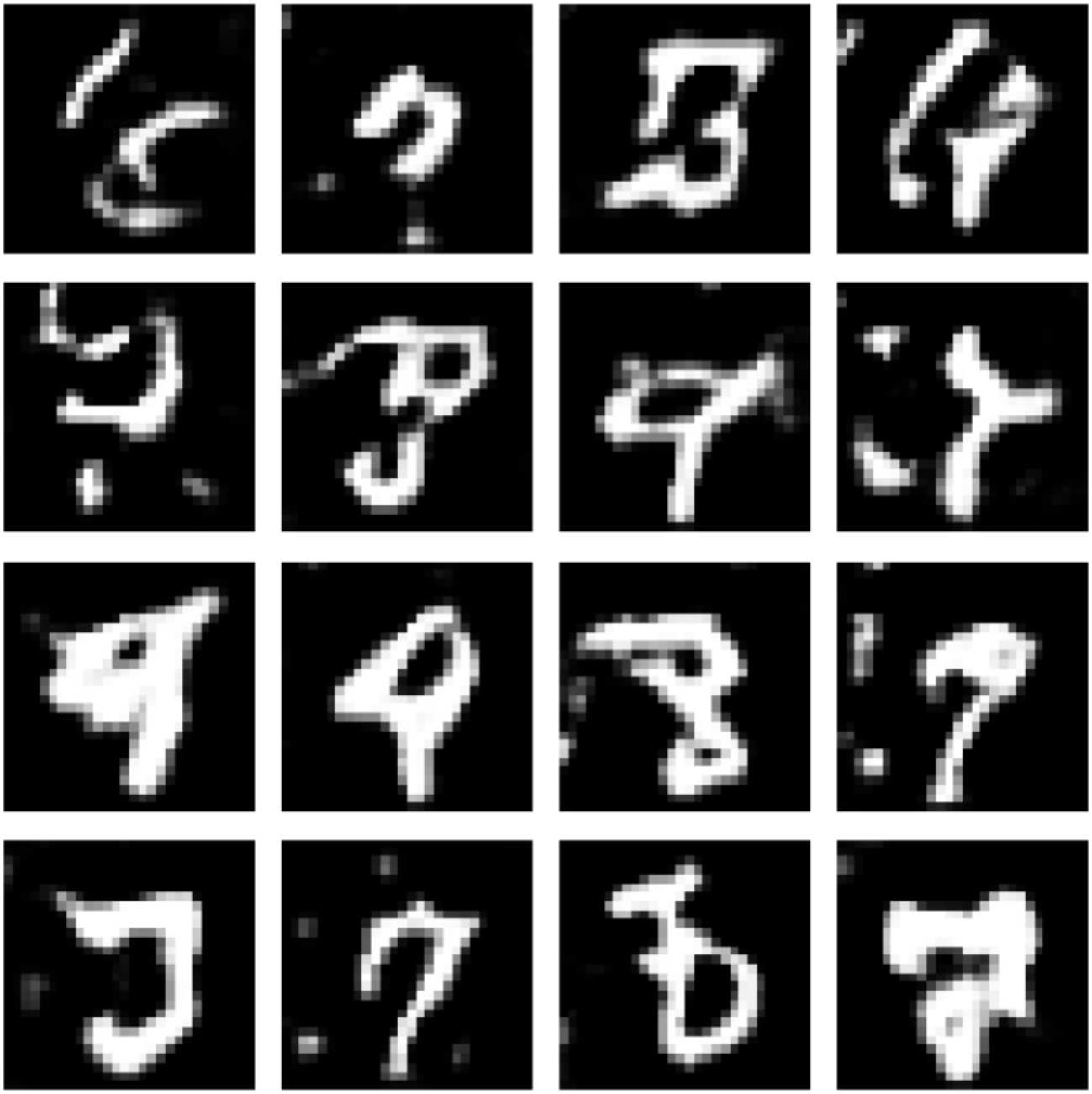
1. **Latent Rectified Diffusion** operates in a **lower-dimensional latent space**, meaning that instead of applying diffusion directly to raw image pixels, it first encodes images into a compressed representation using a pre-trained encoder, such as a **Variational Autoencoder (VAE)** or an autoencoder-based model. The rectified flow is then applied within this latent space, where data is **more structured and compact**, allowing for **faster training and inference** with fewer steps. The denoised latent representations are then **decoded back into pixel space** using a decoder network. This method is particularly useful for **high-resolution images**, as it reduces computational complexity and avoids the inefficiencies of working directly with pixel-level noise.
2. **Pixel-Based Diffusion** applies the diffusion process **directly to pixel values**, meaning that the noise is added and removed at the level of the raw image. This method captures **fine-grained details** and operates at the full image resolution, but it is often computationally expensive, requiring **many more diffusion steps** to achieve high-quality samples. While pixel-based diffusion models (such as DDPMs and DDIMs) can produce highly detailed images, they are typically **slower** and require significantly more resources for training and inference.

Results:

- Pixel Based Diffusion model generated



- Latent Based Diffusion model generation



(3pt.) Implement DeepCache for U-Net and research balance between skip steps and final result.

The `RectifiedFlowDiffusionWithDeepCache` class implements the training, validation, and sampling process for the diffusion model. Instead of traditional noise-based forward diffusion, it uses **rectified flow**, meaning it directly interpolates between noise and real data in a smooth trajectory. The model is trained by computing the true velocity field (difference between real data and noise) and optimizing an MSE loss to predict it. The **Euler integration method** is used during sampling, where the model iteratively refines noisy samples by subtracting the predicted velocity field at each time step. The

DeepCache system can be turned on or off during sampling and supports different cache skip steps, allowing trade-offs between speed and accuracy.

Results:

The results highlight how DeepCache accelerates inference while maintaining image quality, making it particularly useful for large-scale diffusion models. The **rectified flow approach eliminates the need for traditional diffusion noise schedules**, making sampling more efficient while maintaining high fidelity in generated images. The model uses a **U-Net encoder-decoder structure** to refine image generation progressively, similar to standard diffusion models but optimized for faster inference. The DeepCache mechanism is especially beneficial when generating multiple samples, reducing redundant computations by reusing stored feature maps.

This approach represents an **efficient variant of diffusion models**, leveraging **rectified flow for smoother interpolation** and **caching for faster sampling**. By storing and reusing intermediate computations, DeepCache enables large batch generation without excessive computational overhead. The **benchmarking and visualization tools** included in the code provide insights into DeepCache's effectiveness, ensuring that improvements in speed do not degrade sample quality. The combination of **U-Net-based rectified flow diffusion and DeepCache** offers a promising direction for scalable generative models in image synthesis.

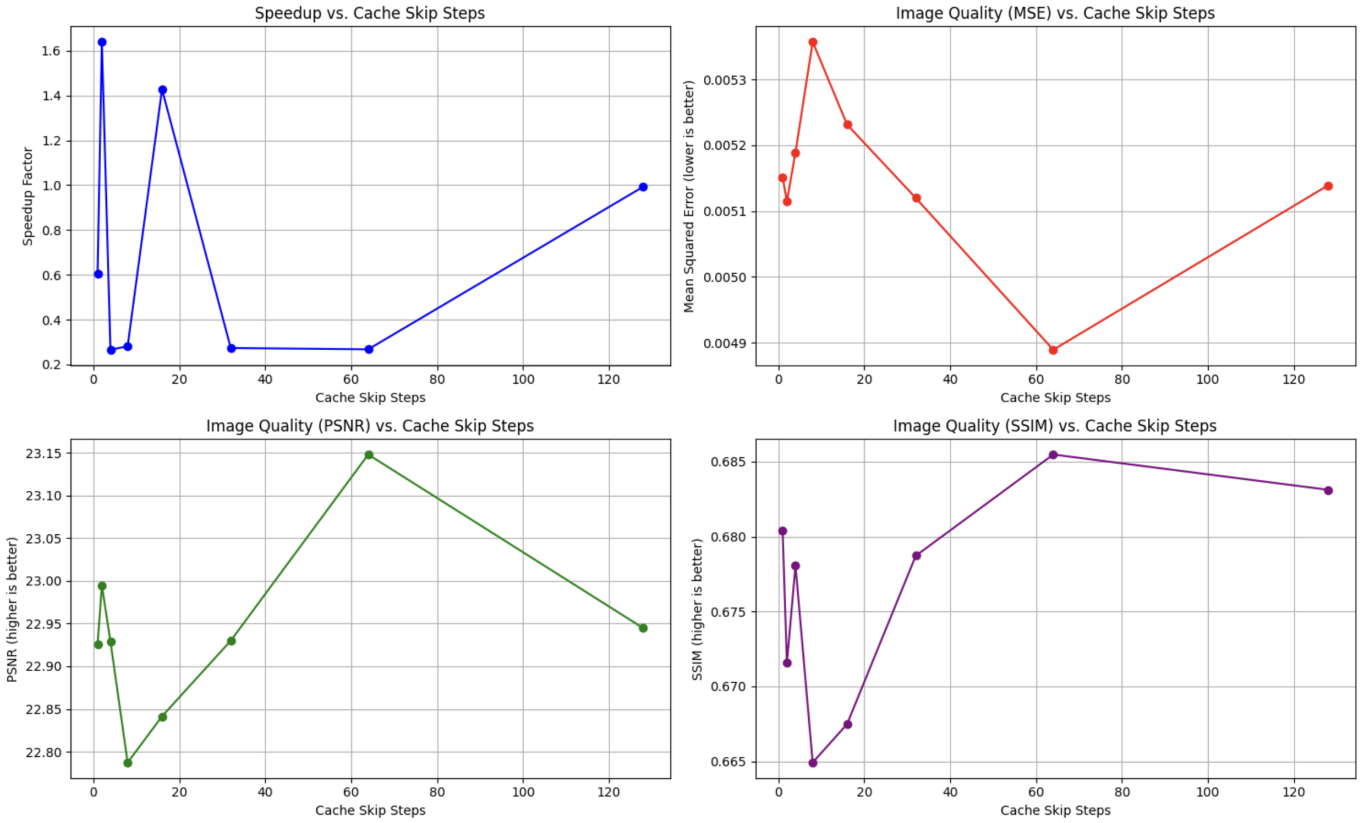
1 sampling step
Results and interpretation:

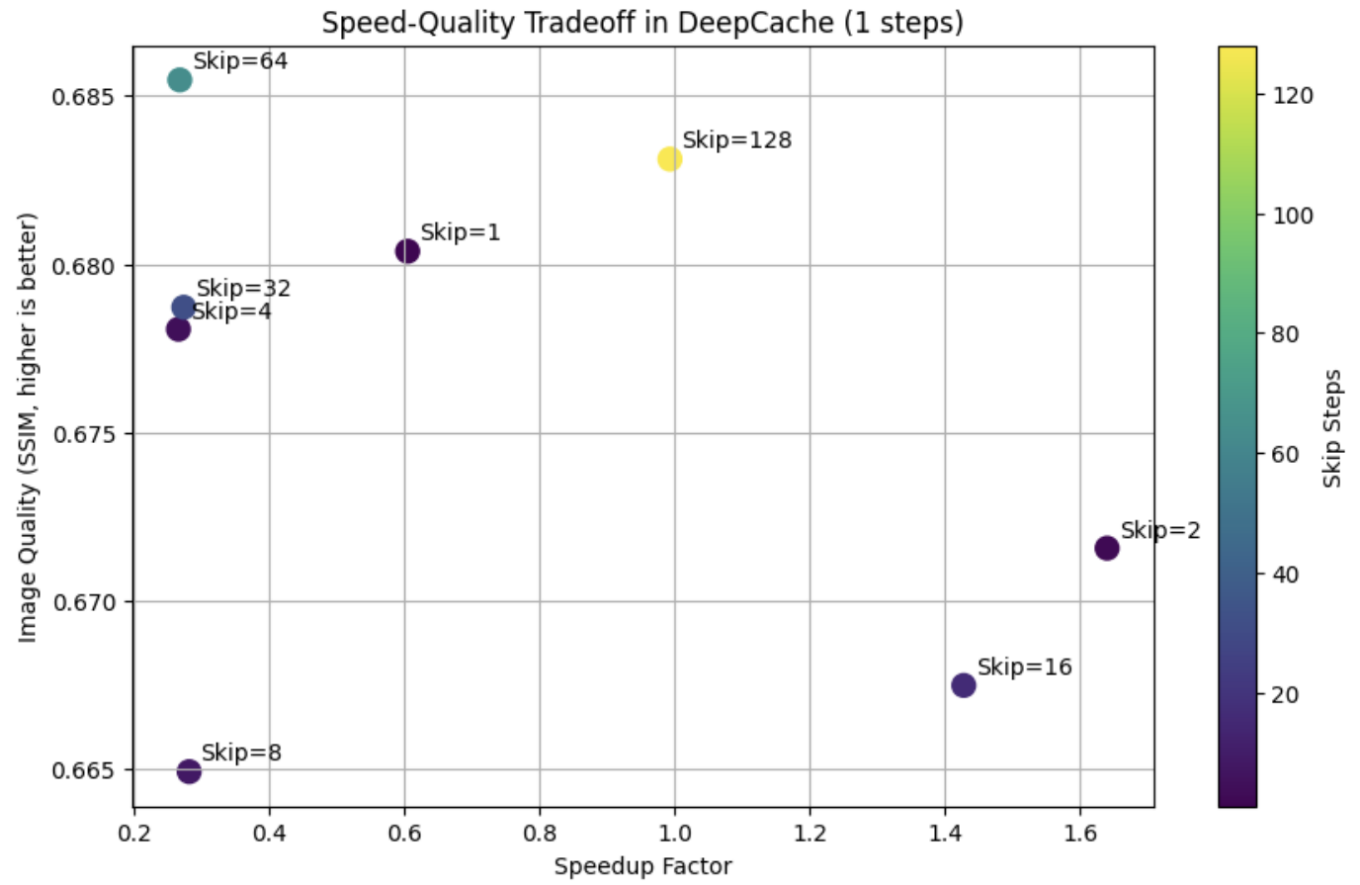
- The speedup factor fluctuates significantly for low cache skip values, then stabilizes.
- Higher skip values seem to regain some speedup benefits.
- The Mean Squared Error (MSE) initially increases with skip steps but then decreases, suggesting an optimal range.
- Too high a skip step causes MSE to increase again, meaning the image reconstruction degrades.
- The Peak Signal-to-Noise Ratio (PSNR) improves initially but then degrades at higher skip values.
- Indicates that a moderate skip step might yield better quality.
- The Structural Similarity Index (SSIM) follows a similar trend to PSNR.
- The highest SSIM is at intermediate cache skip values.

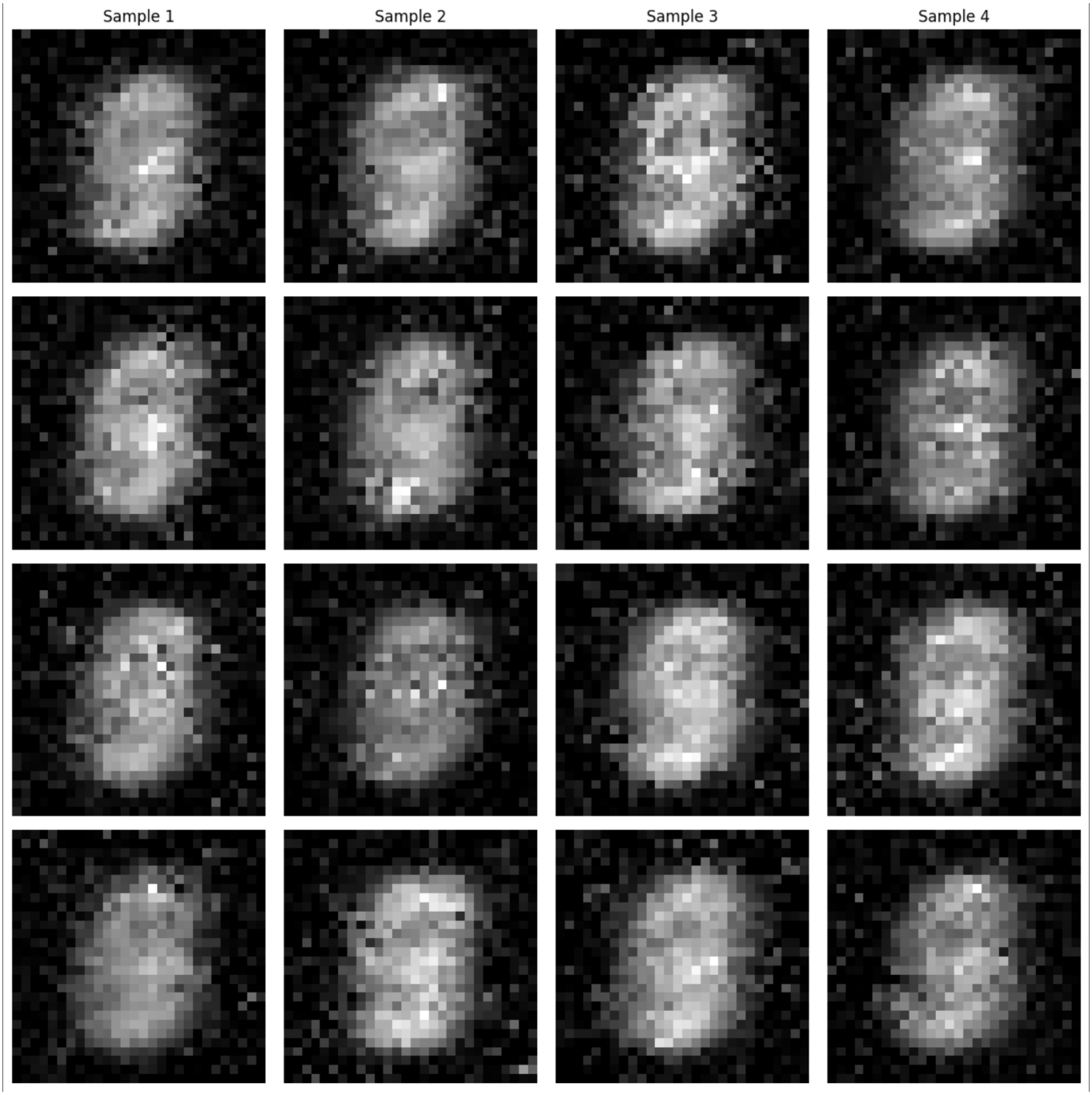
The scatter plot illustrates the trade-off between speedup and image quality (SSIM) in DeepCache. Low skip values (e.g., Skip=1) preserve high SSIM but offer minimal speedup, while high skip values (e.g., Skip=16 and Skip=128) achieve significant speedup at the cost of lower SSIM. Moderate skips (Skip=32 and Skip=64) provide a balanced trade-off, maintaining both reasonable speed and quality. This suggests that an optimal caching strategy lies in the mid-range of skip values.

- Higher skips introduce **blurriness and artifacts**.
- Moderate skips (middle rows) may be acceptable.

DeepCache Analysis with 1 Sampling Steps





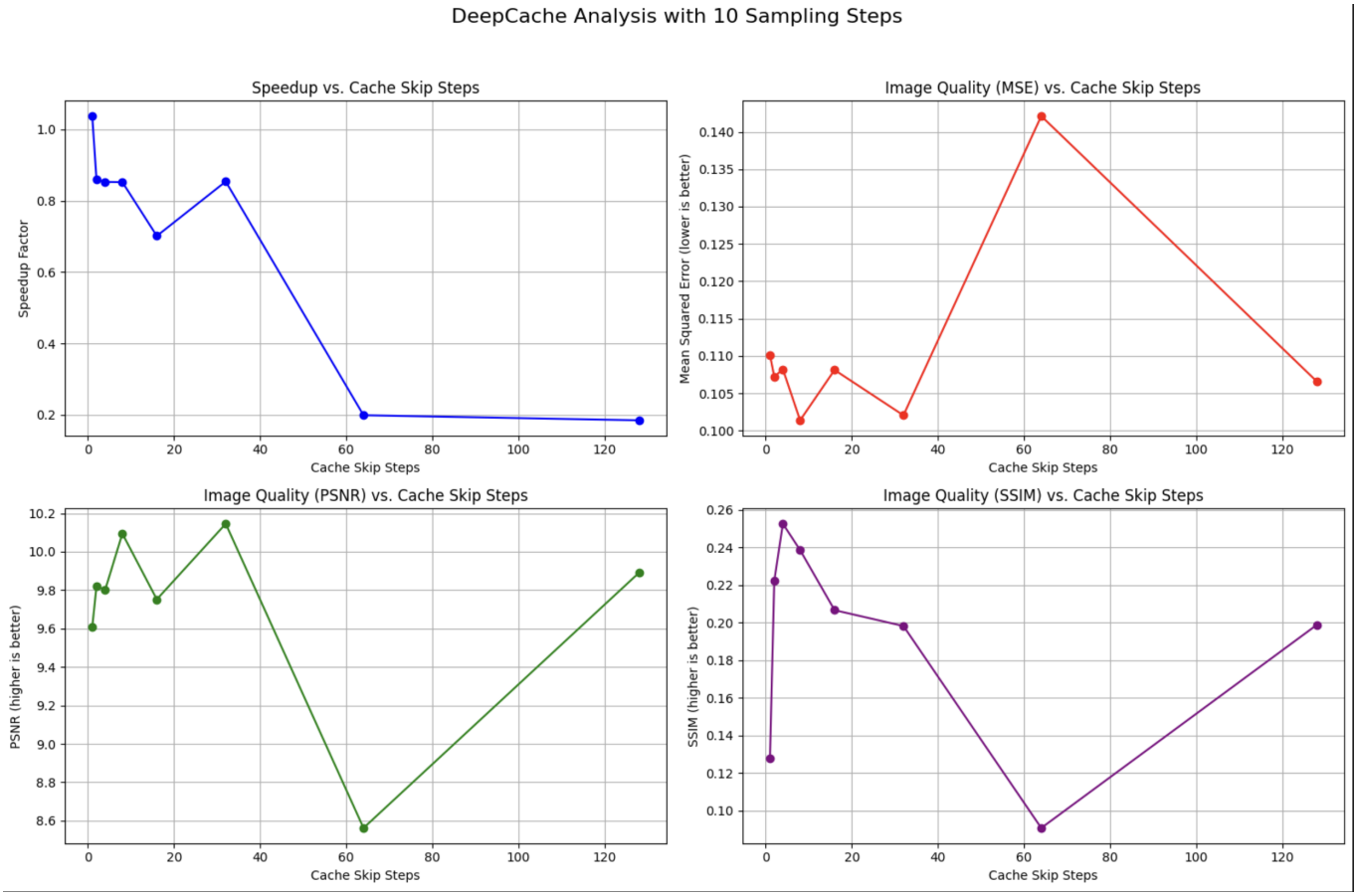


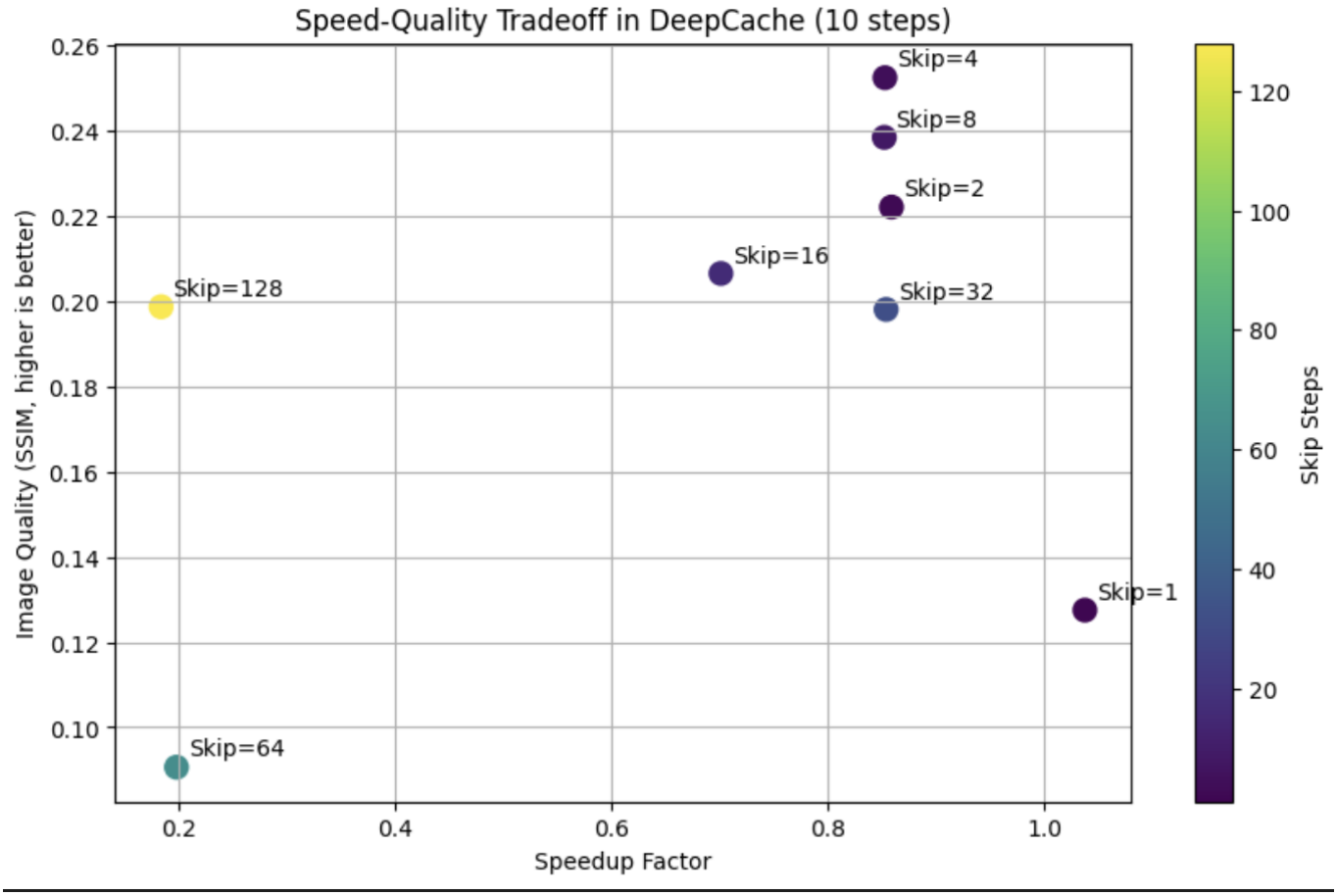
10 sampling steps
Results and interpretations:

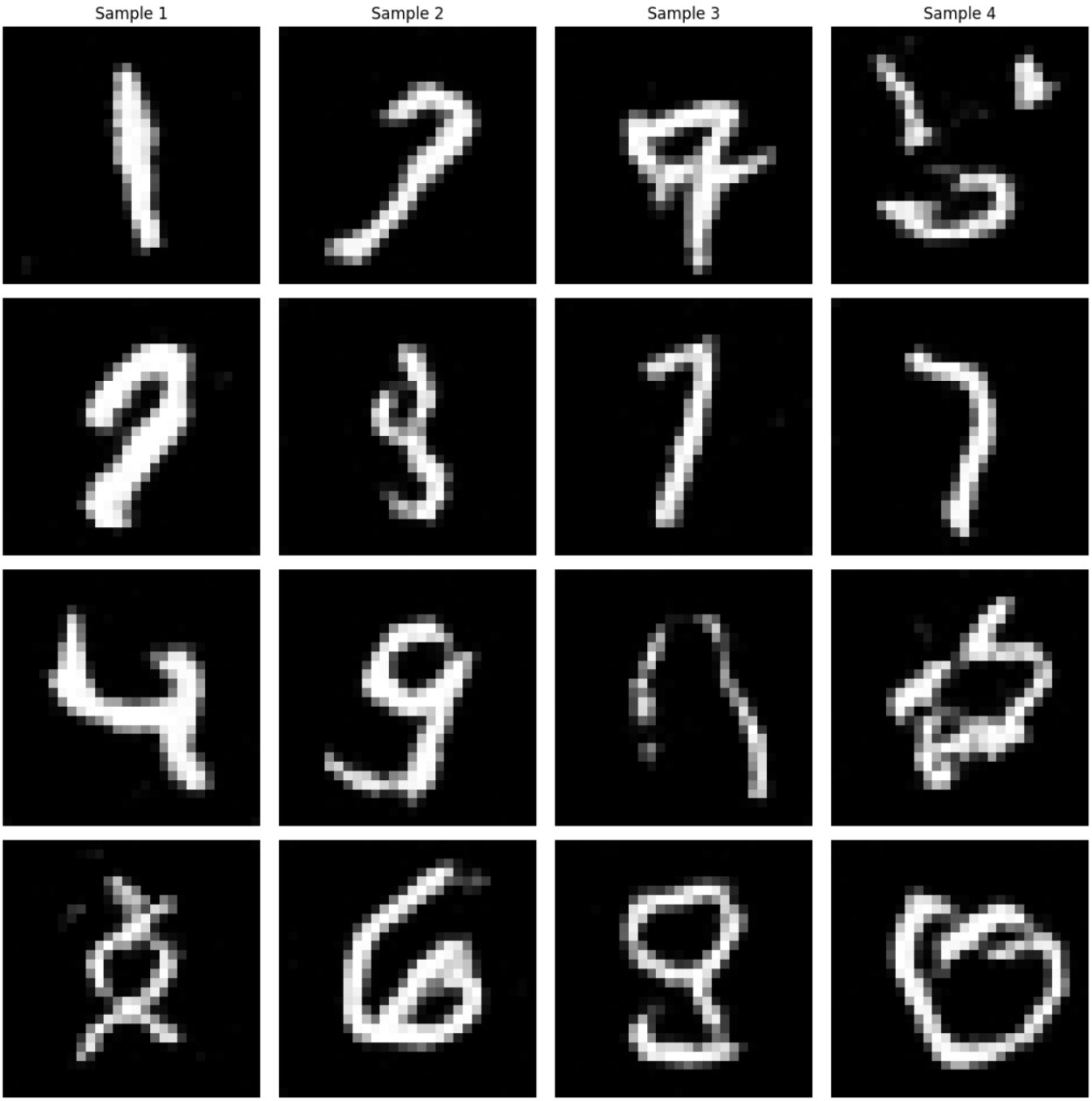
DeepCache with 10 sampling steps, showing that speedup declines as cache skip steps increase, with a sharp drop beyond **skip=60**. MSE fluctuates at lower skip values but spikes significantly at **skip=60**, indicating poor reconstruction, before slightly improving. PSNR remains relatively stable with small skips, peaks around **skip=40**, and then collapses, mirroring the quality degradation seen in MSE. SSIM exhibits a similar pattern, with **high SSIM at small skips**, a significant drop at **skip=60**, and partial recovery at **skip=120**.

The second plot presents the **speed-quality trade-off**, showing **skip=1** maintaining high SSIM but achieving minimal speedup. Skips **2, 4, 8, and 16** provide incremental speed improvements while preserving relatively high SSIM. **Skip=32** achieves better speedup but starts to sacrifice SSIM, whereas **skip=64 and skip=128** experience major quality degradation. The qualitative visualization confirms these findings, with **high skips introducing severe distortions and artifacts** in the reconstructed images.

Overall, **moderate cache skips (around 8-16) seem optimal**, balancing speed and image quality. **Extreme skipping (64+) severely degrades quality** despite improved speed. This suggests an optimal **skip range of 8-32**, depending on the acceptable trade-off between speed and fidelity.




Speed-Quality Tradeoff in DeepCache (10 steps)
Image Quality (SSIM, higher is better)
Speedup Factor
Skip Steps
Skip=4
Skip=8
Skip=2
Skip=16
Skip=32
Skip=128
Skip=1
Skip=64
0.26
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0.10
0.2
0.4
0.6
0.8
1.0
120
100
80
60
40
20



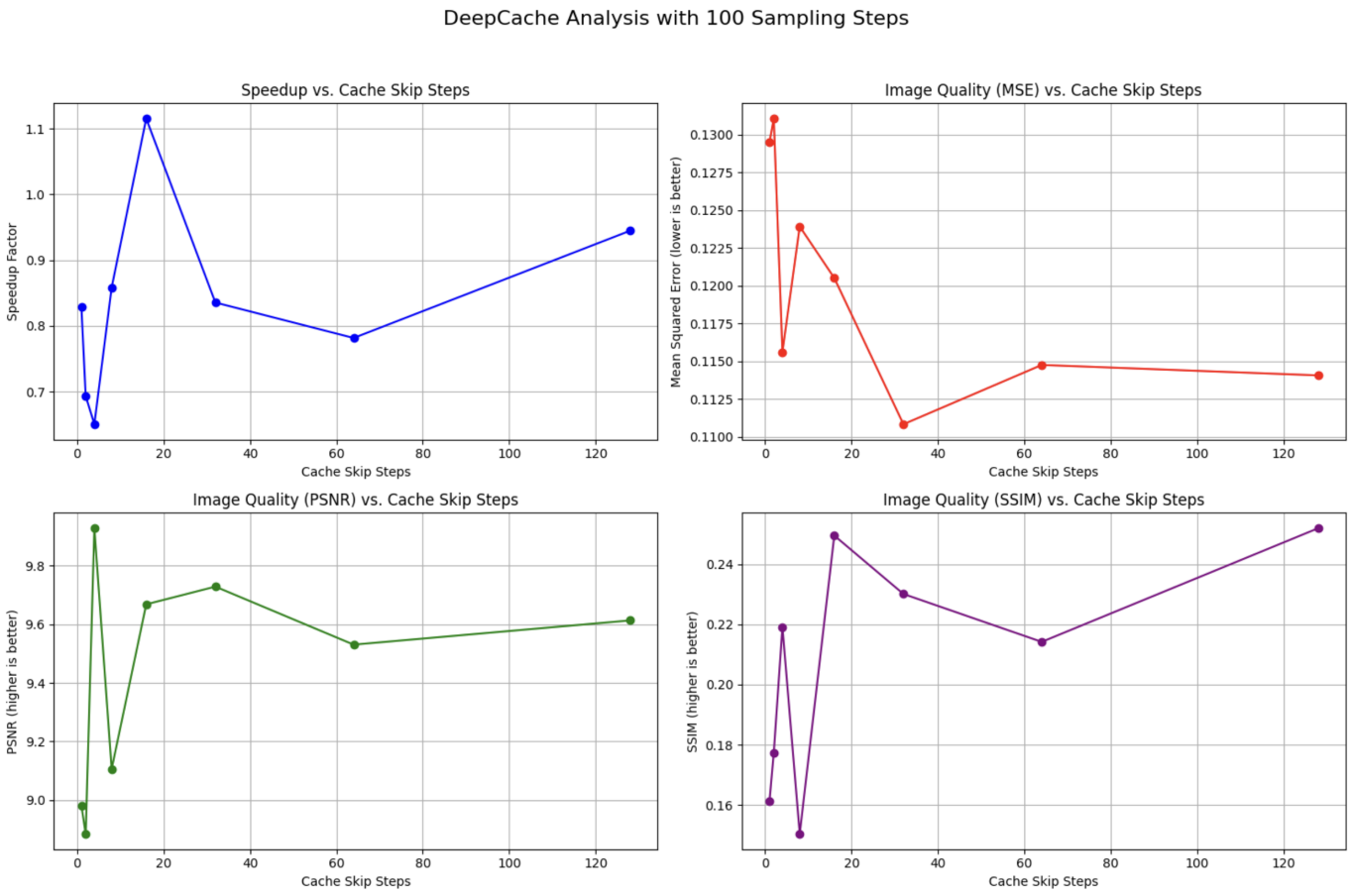
100 sampling steps:

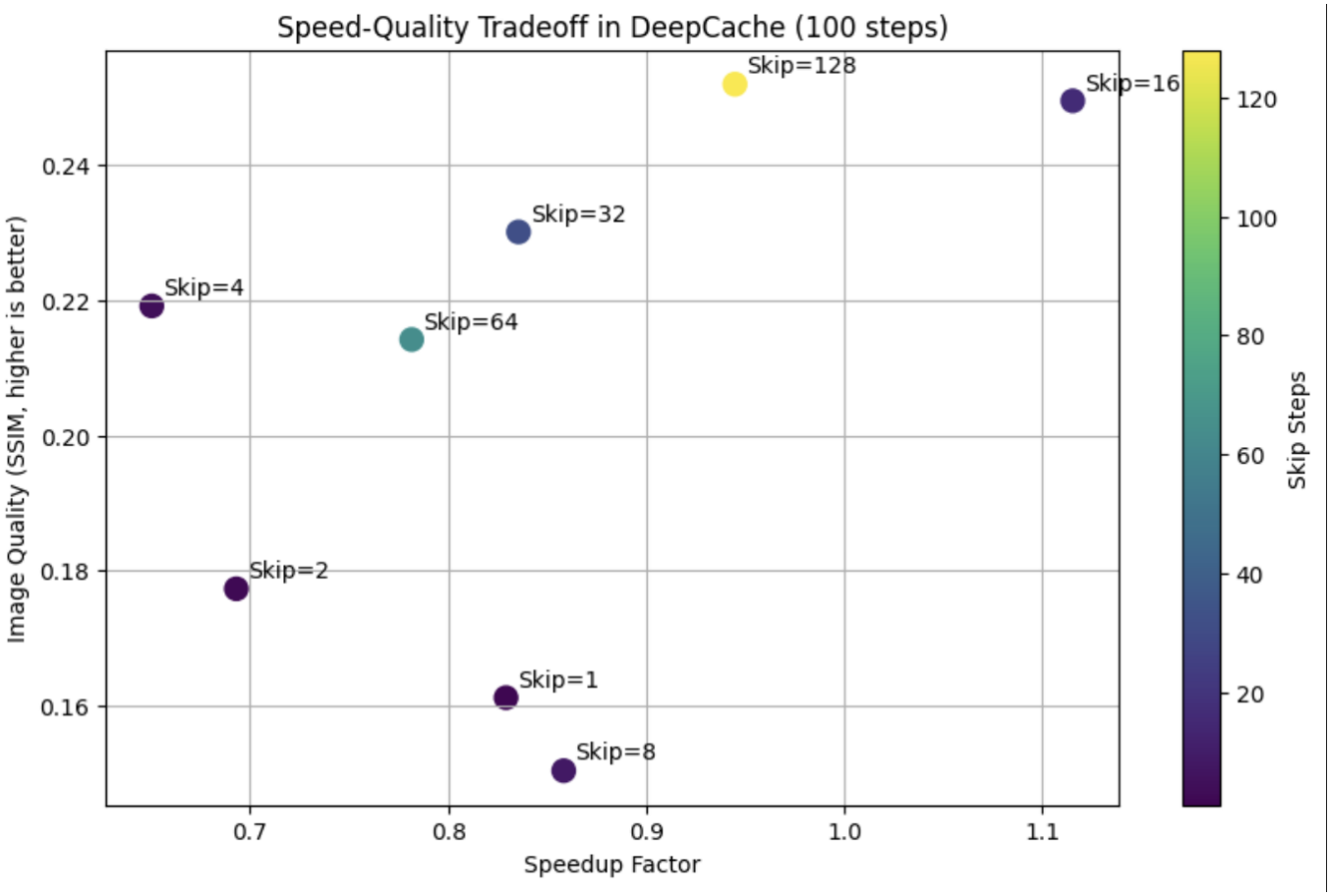
Results and interpretations:

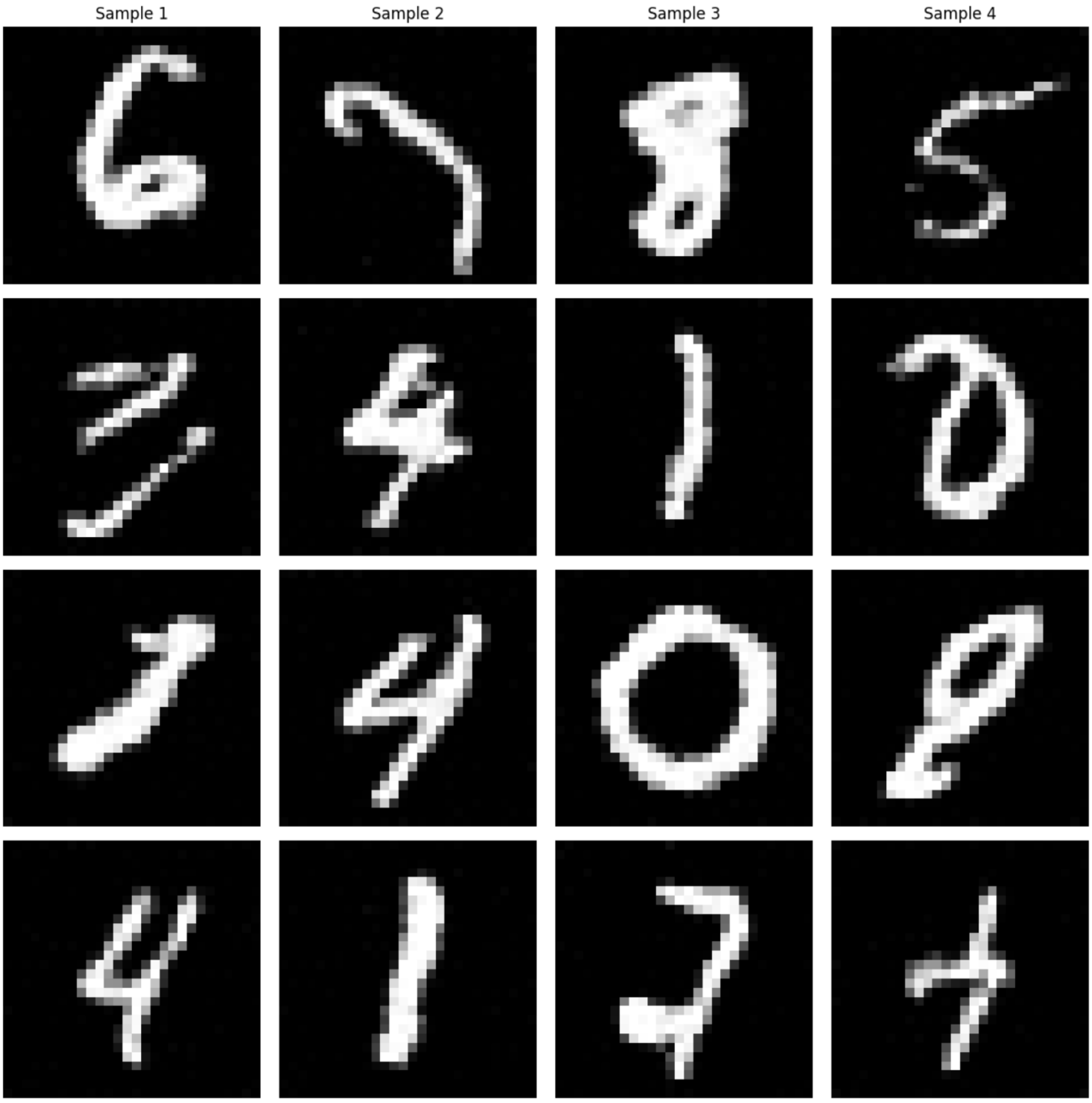
The first plot set examines **DeepCache with 100 sampling steps**, where speedup fluctuates at lower cache skip values but improves steadily beyond **skip=40**. MSE is **highest at very low skips**, stabilizes, and slightly increases beyond **skip=60**, indicating that errors are better distributed. PSNR follows a similar trend, peaking early, dropping slightly, but remaining stable beyond **skip=60**. SSIM shows high variance at small skips, a drop at **skip=40-60**, and recovery at larger skips.

The second plot presents the **speed-quality trade-off**, where **skip=1 has poor speedup but high SSIM**, while **skip=16 and skip=128 achieve the best speed but at the cost of some SSIM degradation**. **Skip=4, 32, and 64 maintain a good balance** between speed and SSIM, making them potential optimal choices. **Skip=2 and skip=8 suffer from both lower speedup and lower SSIM**, making them less favorable choices.

The third image visually confirms the trend, showing **gradual degradation in image quality with increasing skips**, but **skip=32 and skip=64 still maintain recognizable structures**. **Higher skips (e.g., 128) lead to noticeable distortions**, but the digits are still somewhat readable compared to lower sampling runs. This analysis suggests that **skip values around 32-64 are the best choices**, offering a reasonable trade-off between speed and quality.




Speed-Quality Tradeoff in DeepCache (100 steps)
Skip=128
Skip=16
Skip=32
Skip=4
Skip=64
Skip=2
Skip=1
Skip=8
Image Quality (SSIM, higher is better)
0.24
0.22
0.20
0.18
0.16
0.7
0.8
0.9
1.0
1.1
Speedup Factor
120
100
80
60
40
20
Skip Steps



(3pt.) Implement and train conditioning with ControlNet using canny edges

This work explores the integration of **ControlNet** with diffusion-based generative models for structured image synthesis. Specifically, we train a **ControlNet-guided U-Net** to reconstruct MNIST digits from their **Canny edge representations**, demonstrating how edge-based conditioning can guide generative models. The dataset **CannyEdgeMNIST** is constructed by applying the **Canny edge detection algorithm** to MNIST images, creating a paired dataset where the model learns to map edge images back to their corresponding digit images.

The architecture consists of a **ControlNet module** that extracts feature representations from the edge image and a **U-Net model** that synthesizes images based on these extracted features. The training process follows a **self-supervised reconstruction approach**, where the model is trained to generate images from only their edge representations using an **L1 loss** to measure reconstruction accuracy. The **UNetForControlNet** component takes the encoded control features and progressively refines the output through a series of downsampling and upsampling layers.

During training, the model receives a blank image and an edge-conditioned input, aiming to predict the corresponding original MNIST digit. The optimization is performed using the **Adam optimizer** with a learning rate of **0.001**, while training runs for **10 epochs** with a batch size of **64**.

By leveraging **edge-based conditioning**, this model aligns with recent advancements in **diffusion models with external guidance**, such as **Stable Diffusion's ControlNet implementation**. The integration of ControlNet into diffusion frameworks enables **more structured and interpretable generation**, reducing randomness while maintaining generative diversity. This experiment demonstrates that even with **basic edge detection, a generative model can reconstruct high-quality images**, showcasing the importance of incorporating external information in generative models.

The model's effectiveness could be further improved by **combining multiple conditioning signals**, such as gradients, depth maps, or semantic segmentation masks. The work provides insights into **conditioning strategies for generative models**, showing that **diffusion-based generation can be refined with explicit structural cues**.

Results:

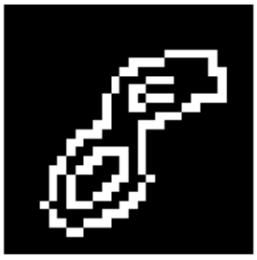
Original (Label: 2)

Canny Edge

Generated

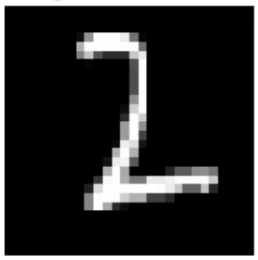
Original (Label: 8)

Canny Edge



Generated

Original (Label: 2)

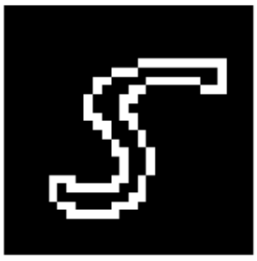


Canny Edge

Generated

Original (Label: 5)

Canny Edge



Generated

Original (Label: 4)

Canny Edge

Generated