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

---

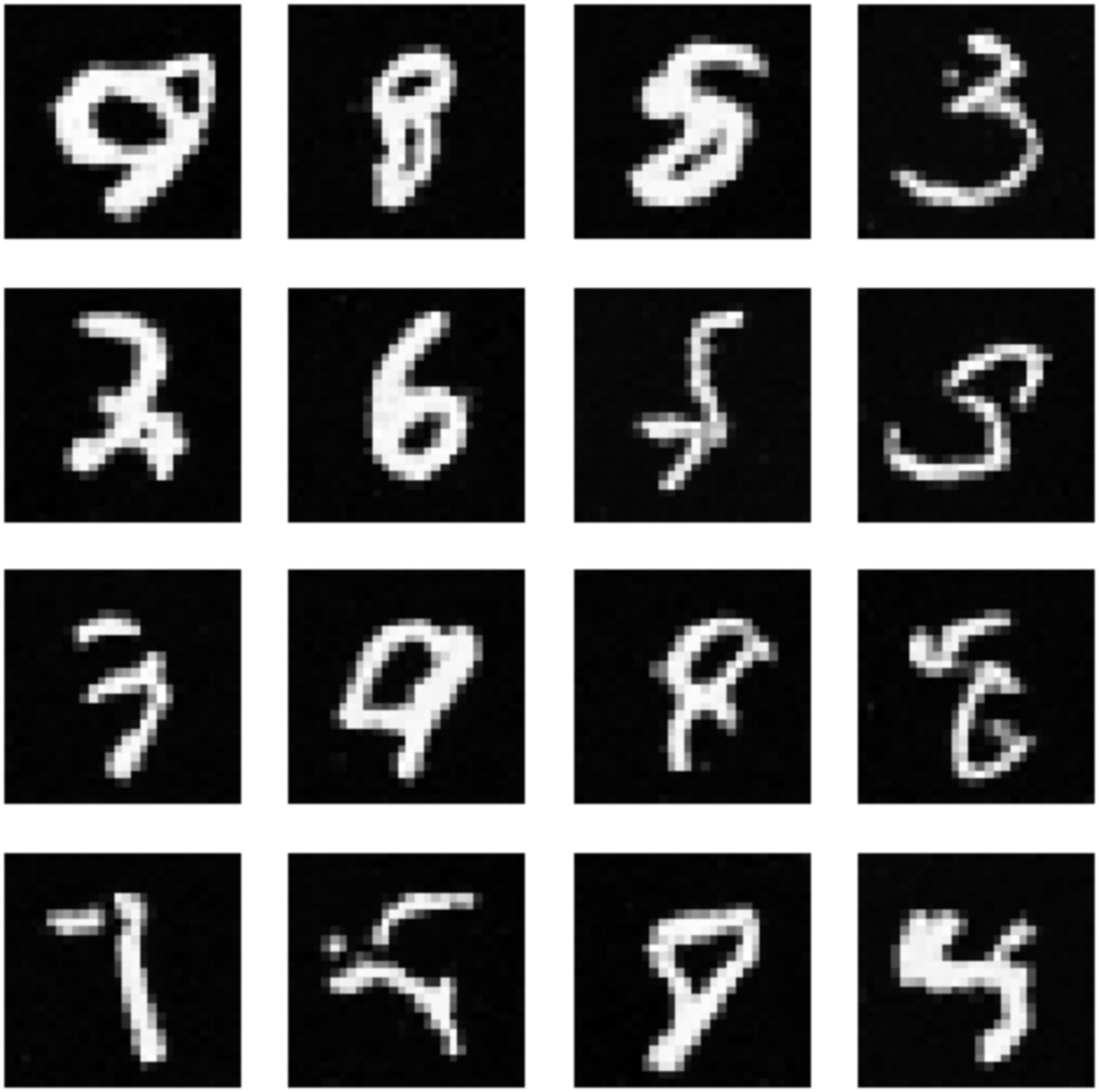
DDPM (Denoising Diffusion Probabilistic Models) generate images by gradually adding noise and then learning to reverse this process step by step. DDIM (Denoising Diffusion Implicit Models) is a faster variant that skips steps in a structured way, reducing the number of inference steps while maintaining 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

$$xt = \alpha^{-}tx0 + 1 - \alpha^{-}t\epsilon x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}$$

\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.

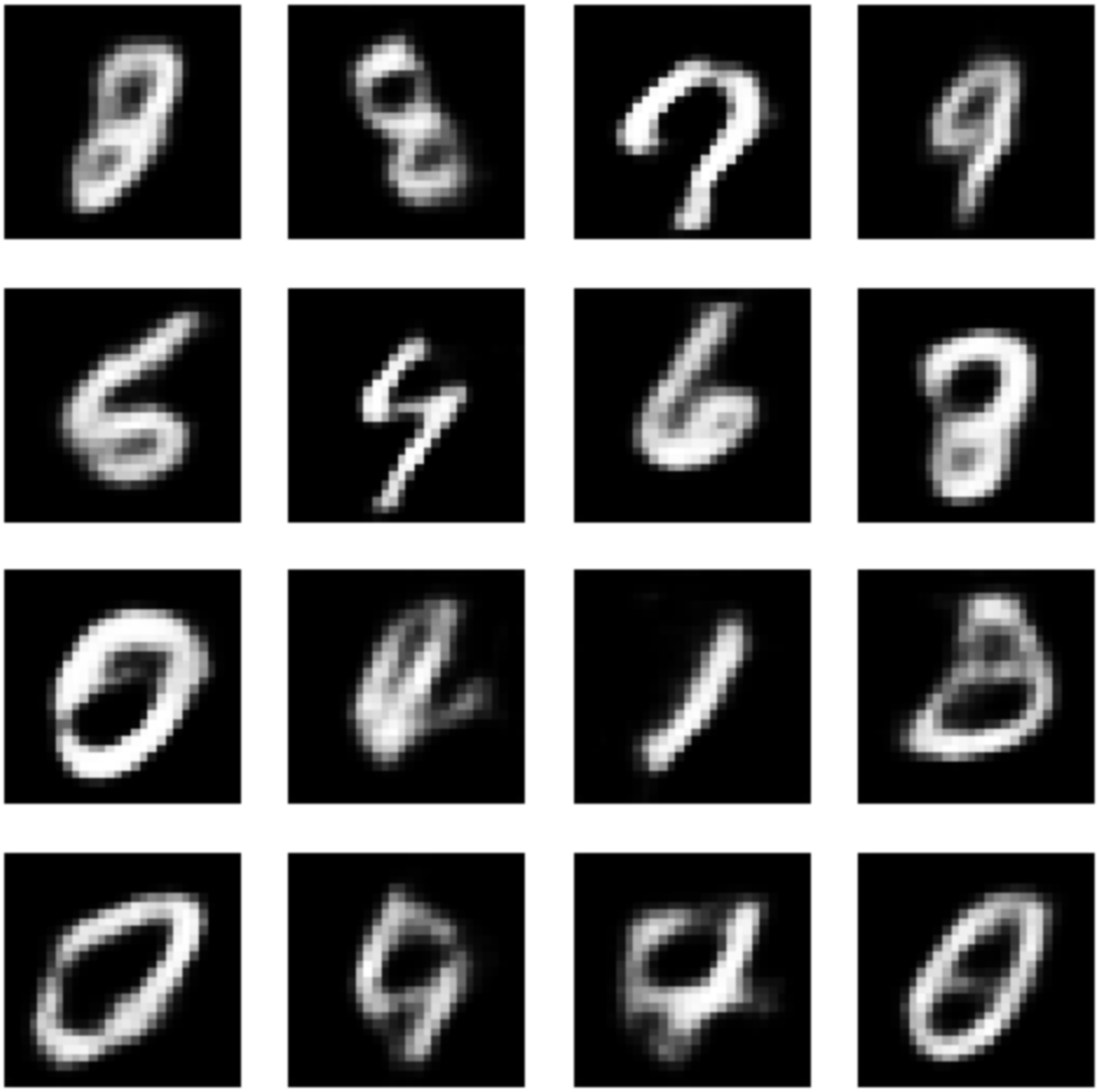
Results:



(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.

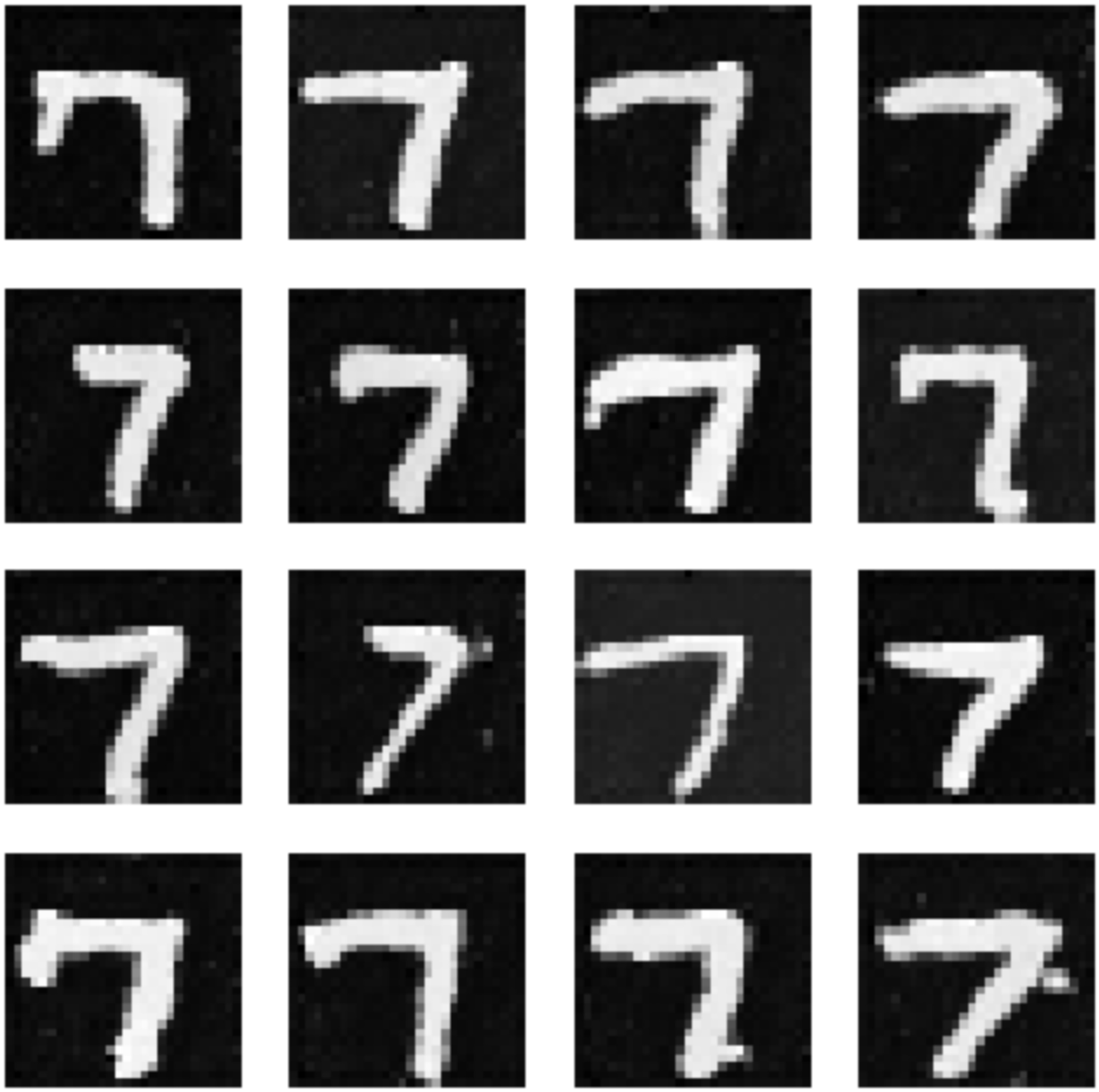
Results:



(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:

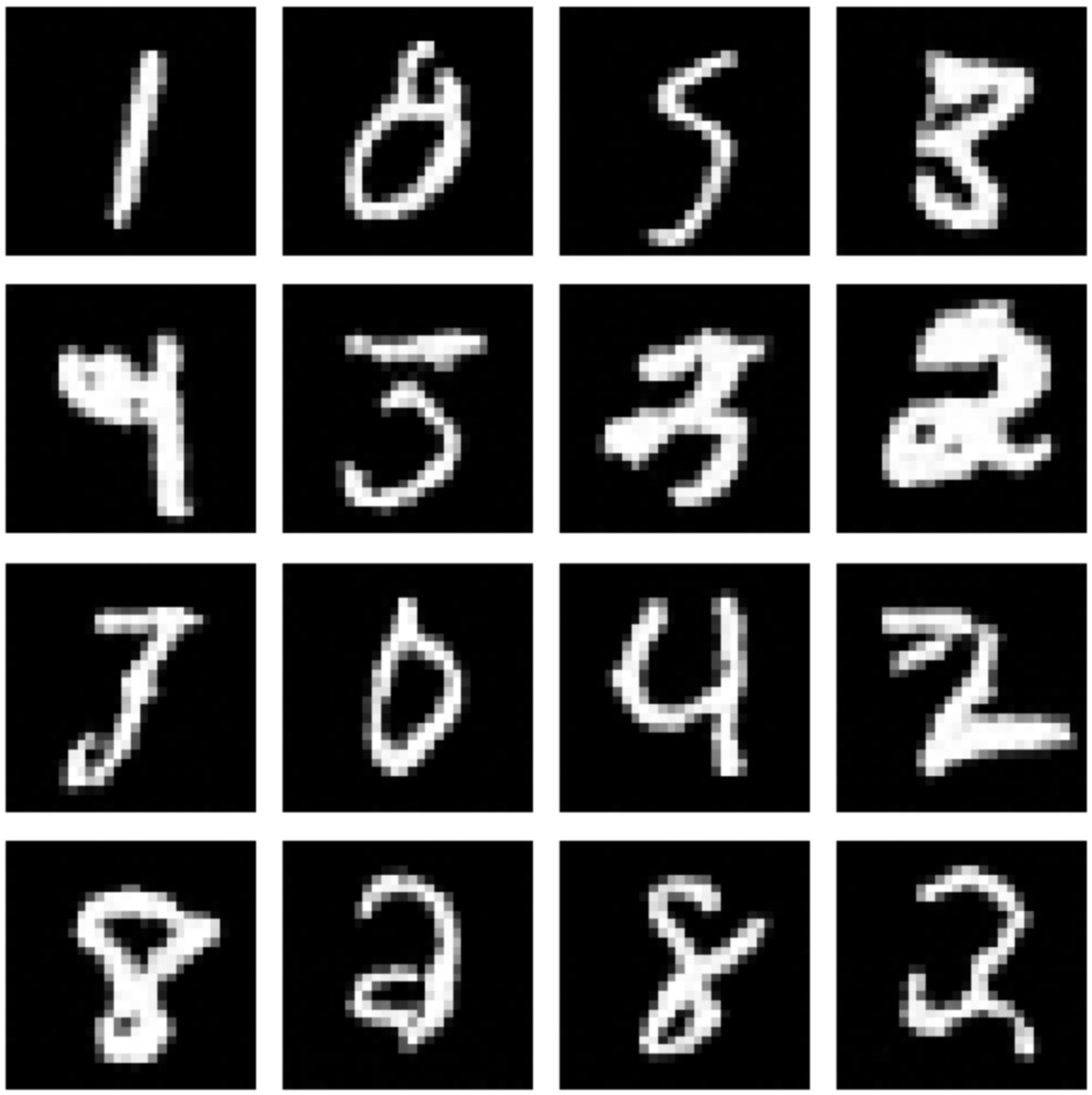


(10pt.) Implement and train diffusion model (pixel and latent based) using Rectified Flow

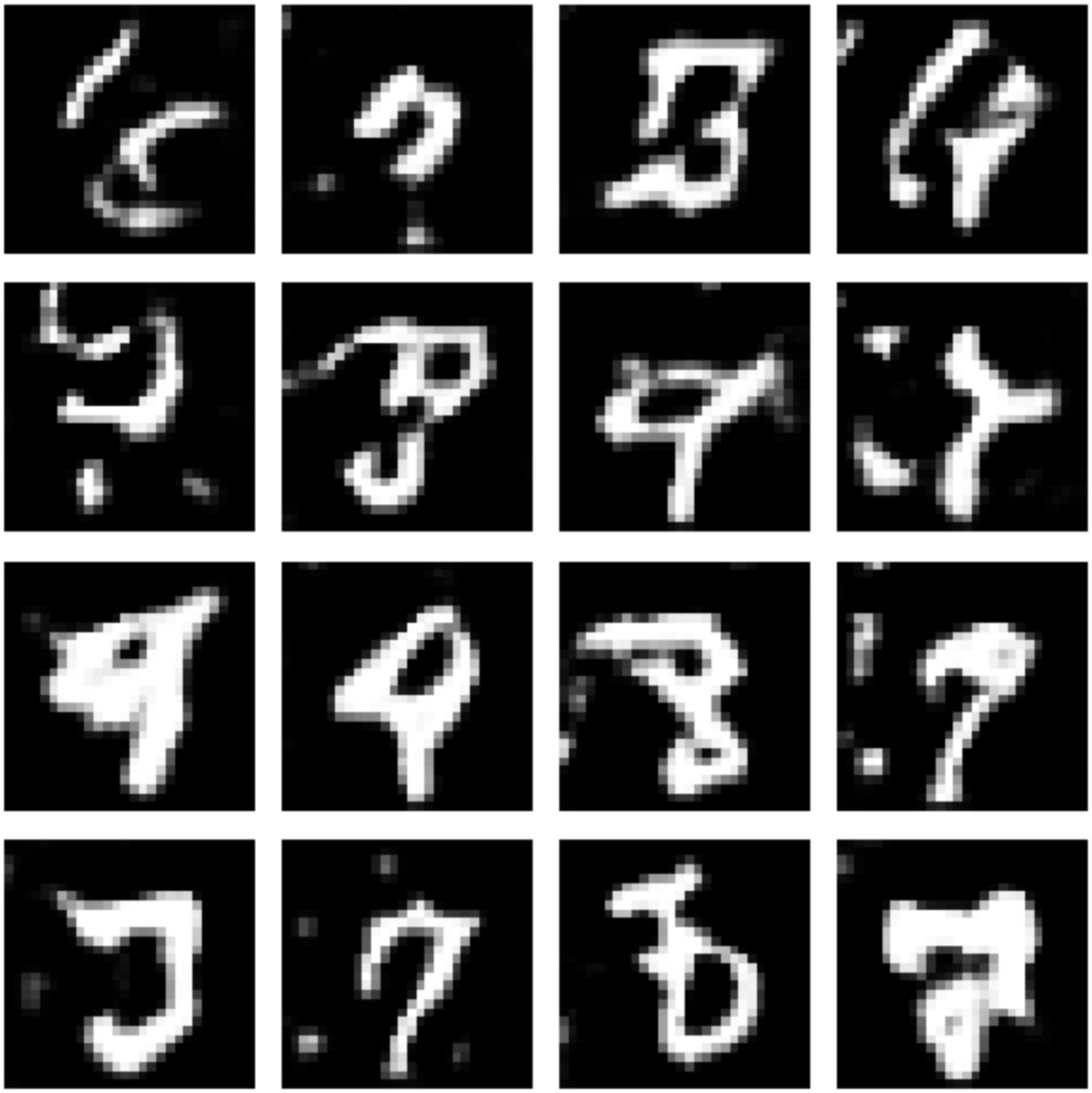
Rectified Flow is a novel approach to generative modeling that aims to improve the efficiency and quality of sampling in diffusion-based models. Unlike traditional diffusion models, which rely on a noisy forward process and a learned denoising reverse process, Rectified Flow directly learns an **optimal 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.

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.

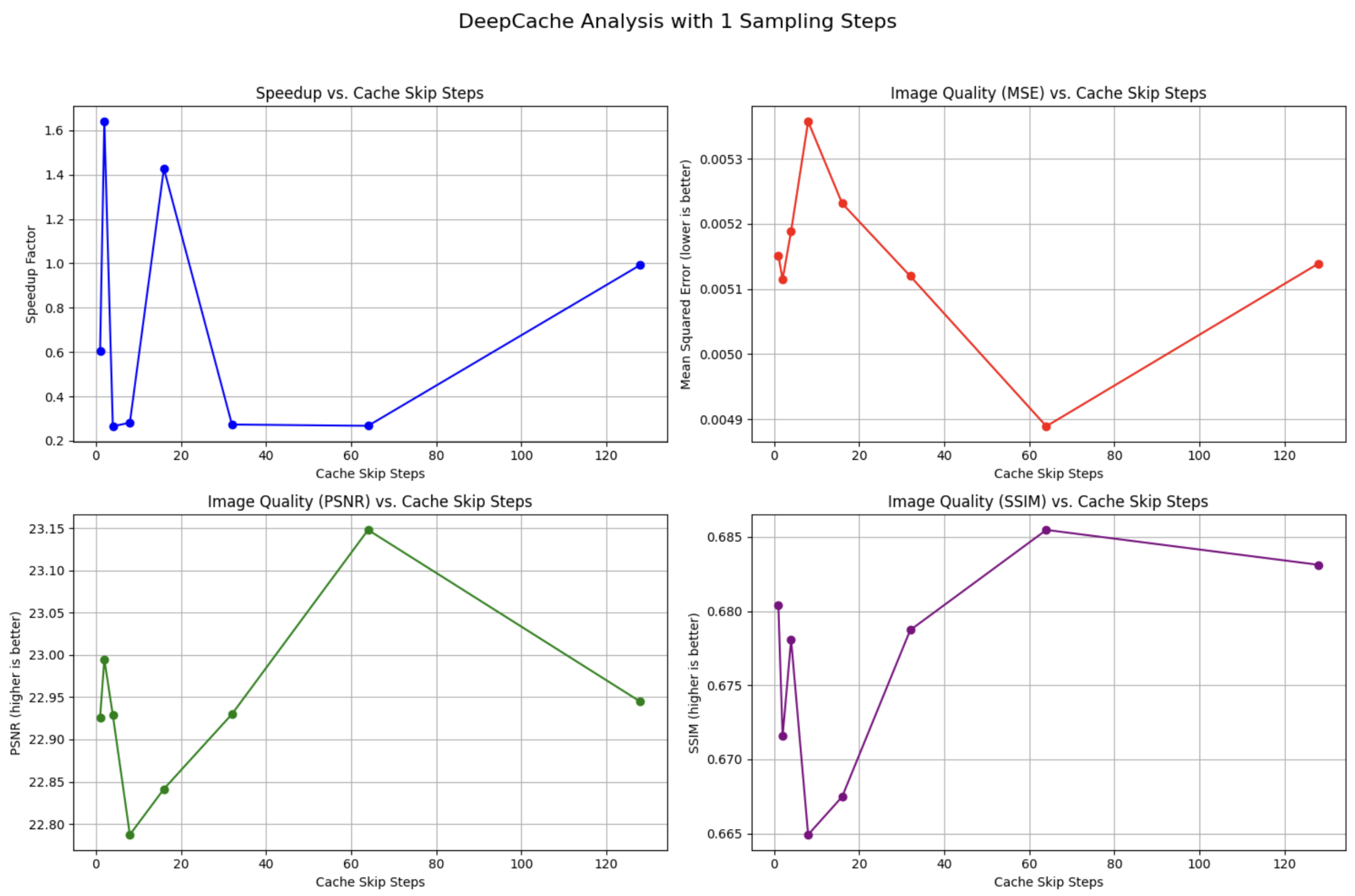
The `RectifiedFlowDiffusionWithDeepCache` class implements the training, validation, and sampling process for the diffusion model. Instead of traditional noise-based forward diffusion, it uses **rectified flow**, meaning it directly interpolates between noise and real data in a smooth trajectory. The 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

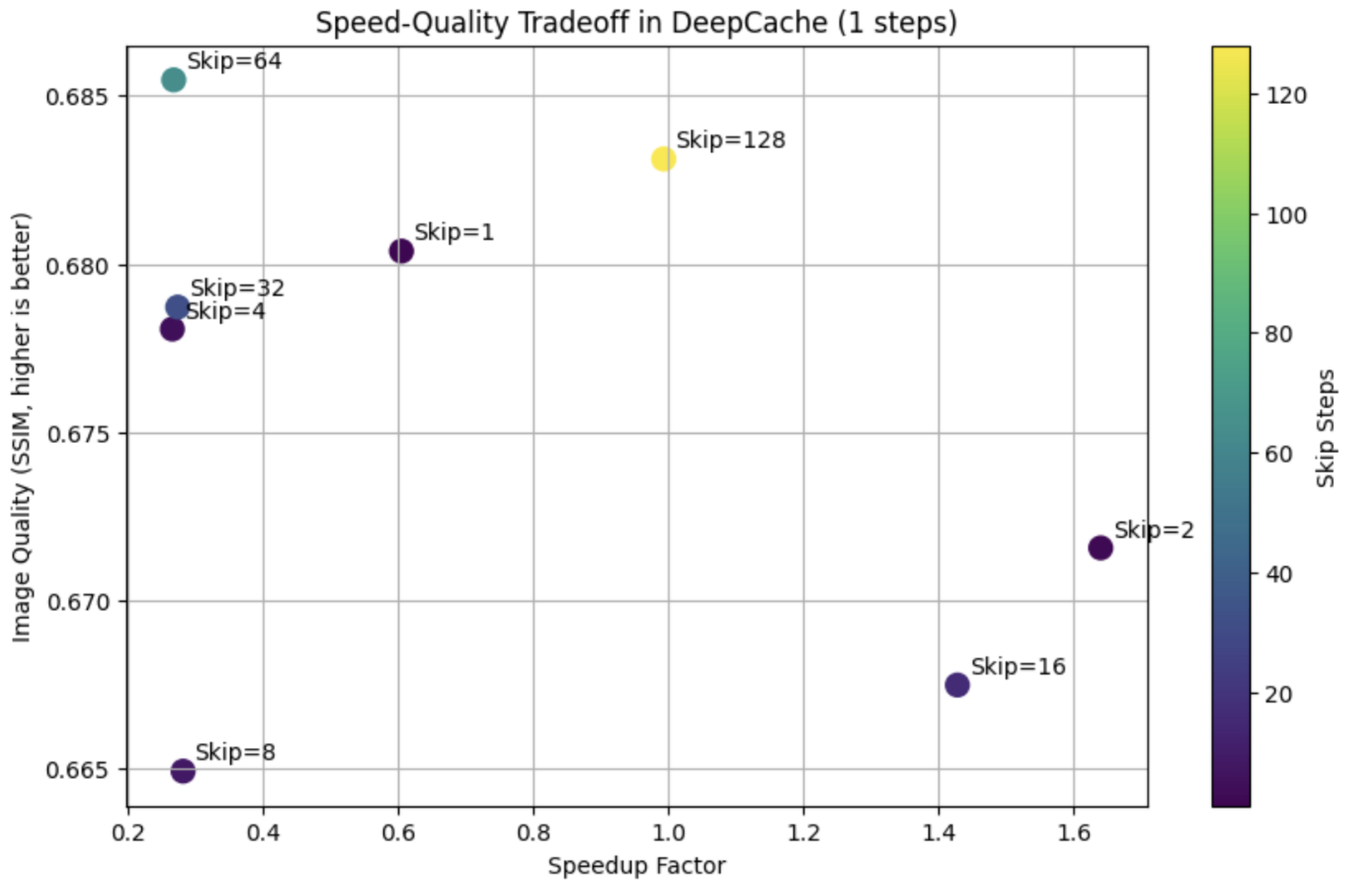
DeepCache system can be turned on or off during sampling and supports different cache skip steps, allowing trade-offs between speed and accuracy.

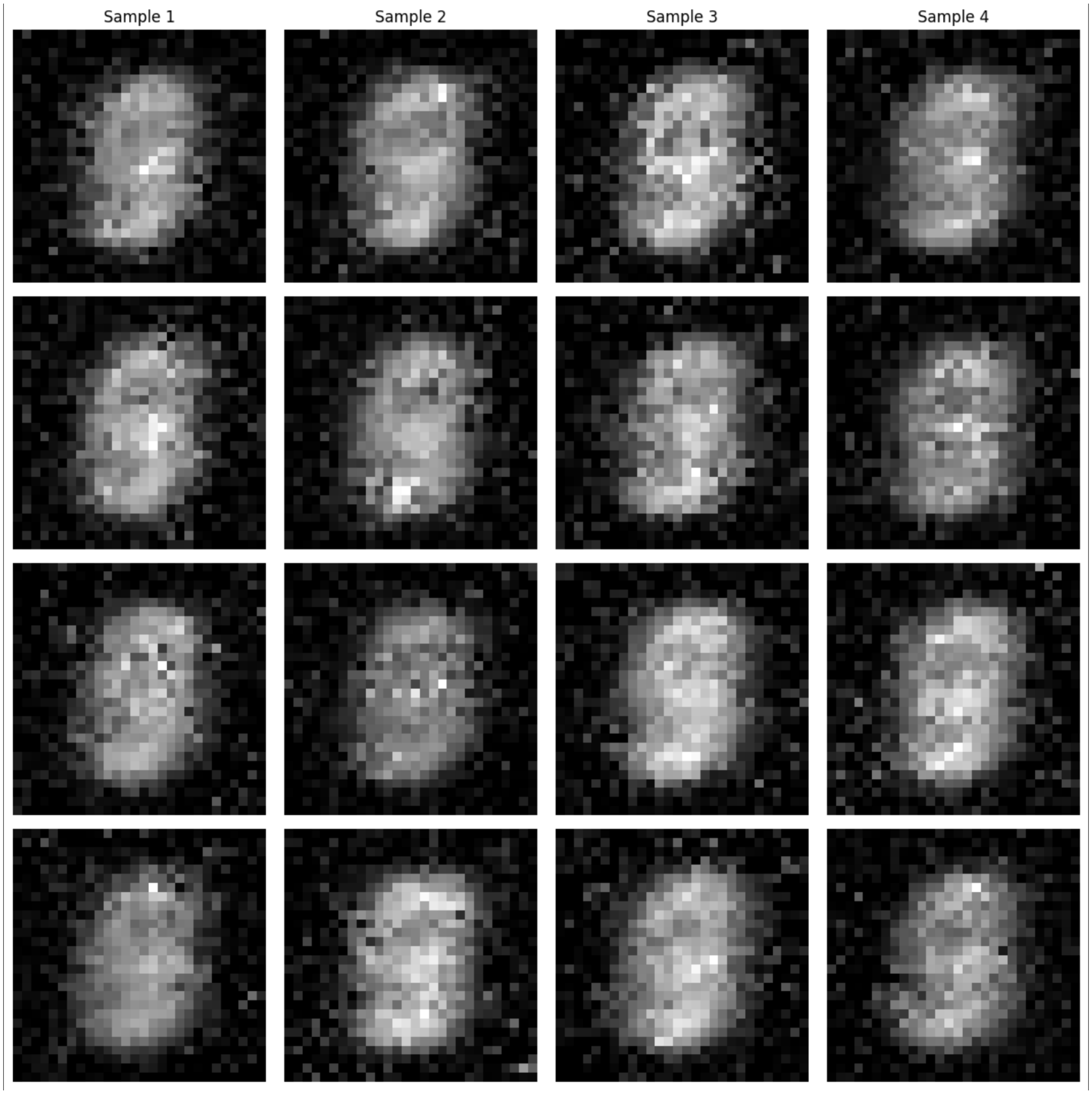
Results:

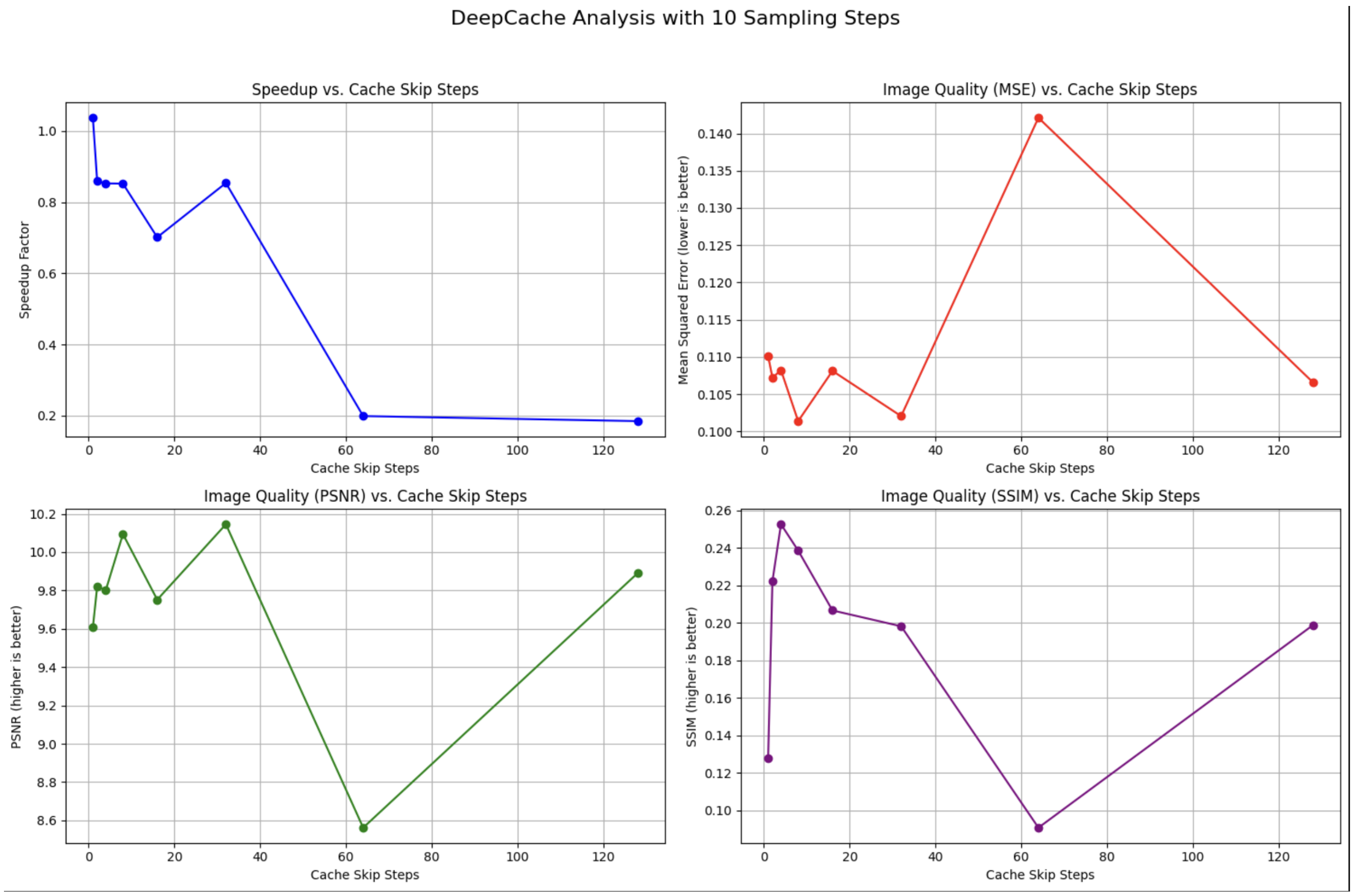
The results highlight how DeepCache accelerates inference while maintaining image quality, making it particularly useful for large-scale diffusion models. The **rectified flow approach eliminates the need for traditional diffusion noise schedules**, making sampling more efficient while maintaining high 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.

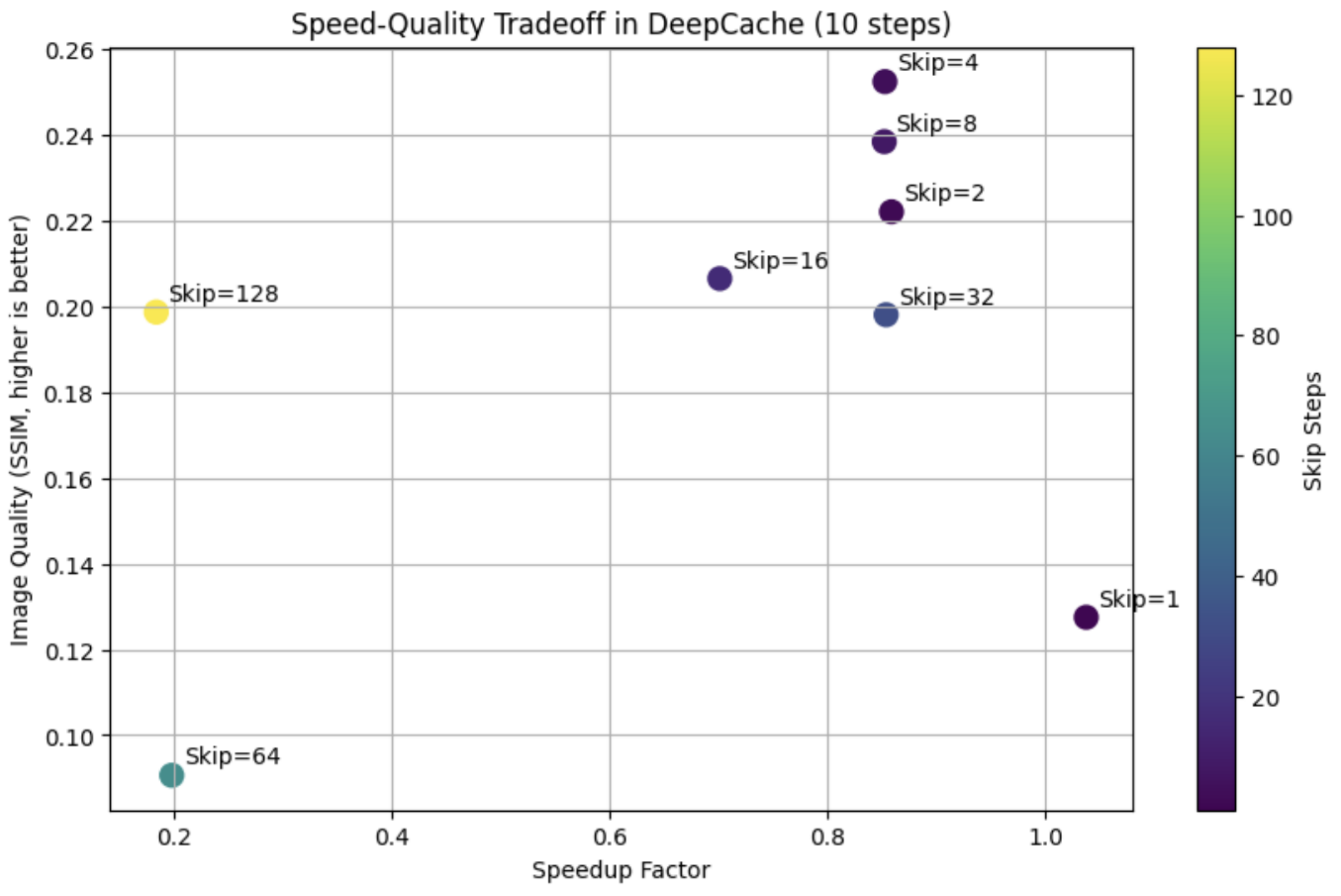
This approach represents an **efficient variant of diffusion models**, leveraging **rectified flow for smoother interpolation** and **caching for faster sampling**. By storing and reusing intermediate computations, DeepCache enables large batch generation without excessive computational overhead. The **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.

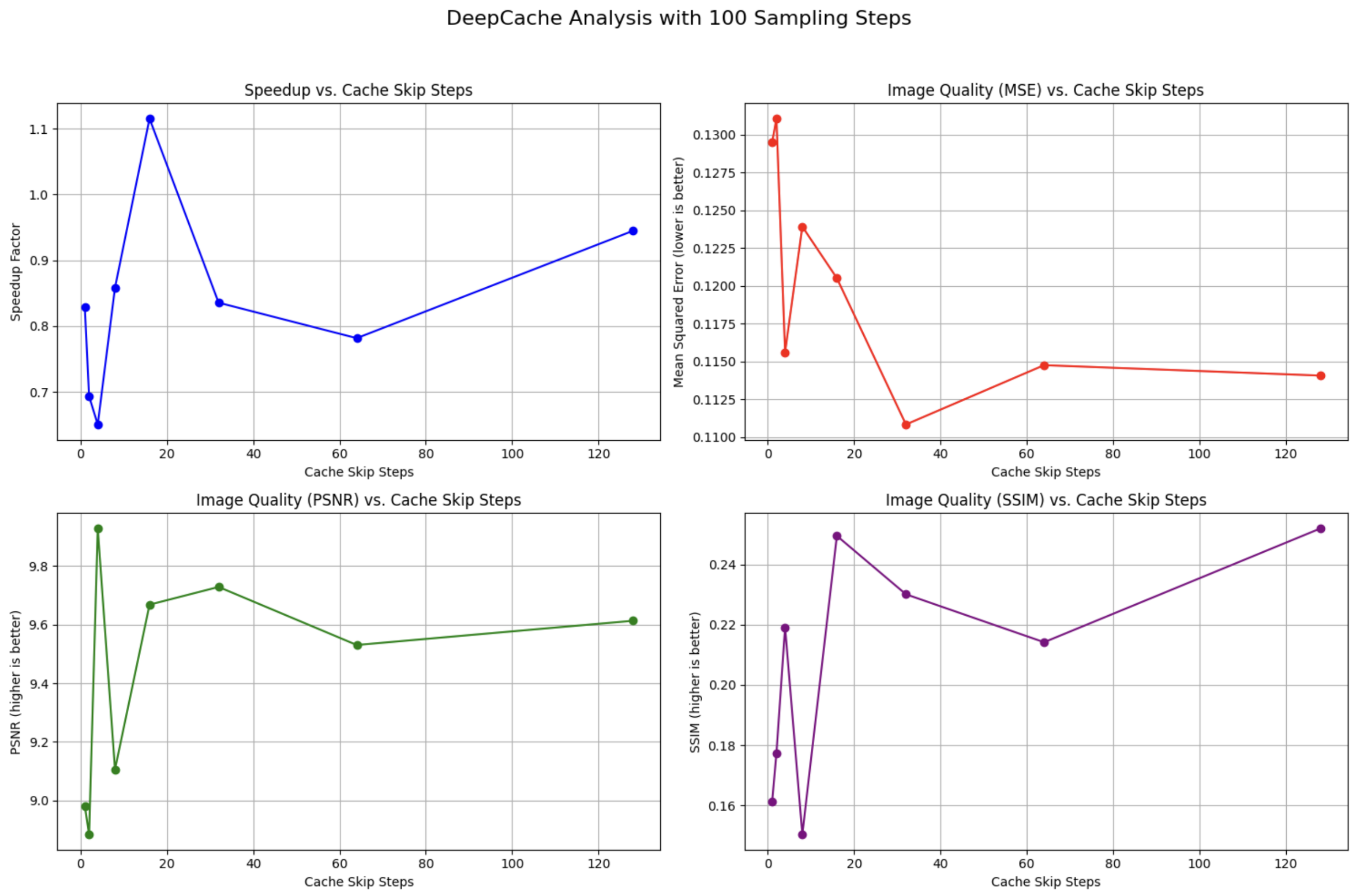

DeepCache Analysis with 1 Sampling Steps
Speedup vs. Cache Skip Steps
Speedup Factor
Cache Skip Steps
Image Quality (MSE) vs. Cache Skip Steps
Mean Squared Error (lower is better)
Cache Skip Steps
Image Quality (PSNR) vs. Cache Skip Steps
PSNR (higher is better)
Cache Skip Steps
Image Quality (SSIM) vs. Cache Skip Steps
SSIM (higher is better)
Cache Skip Steps

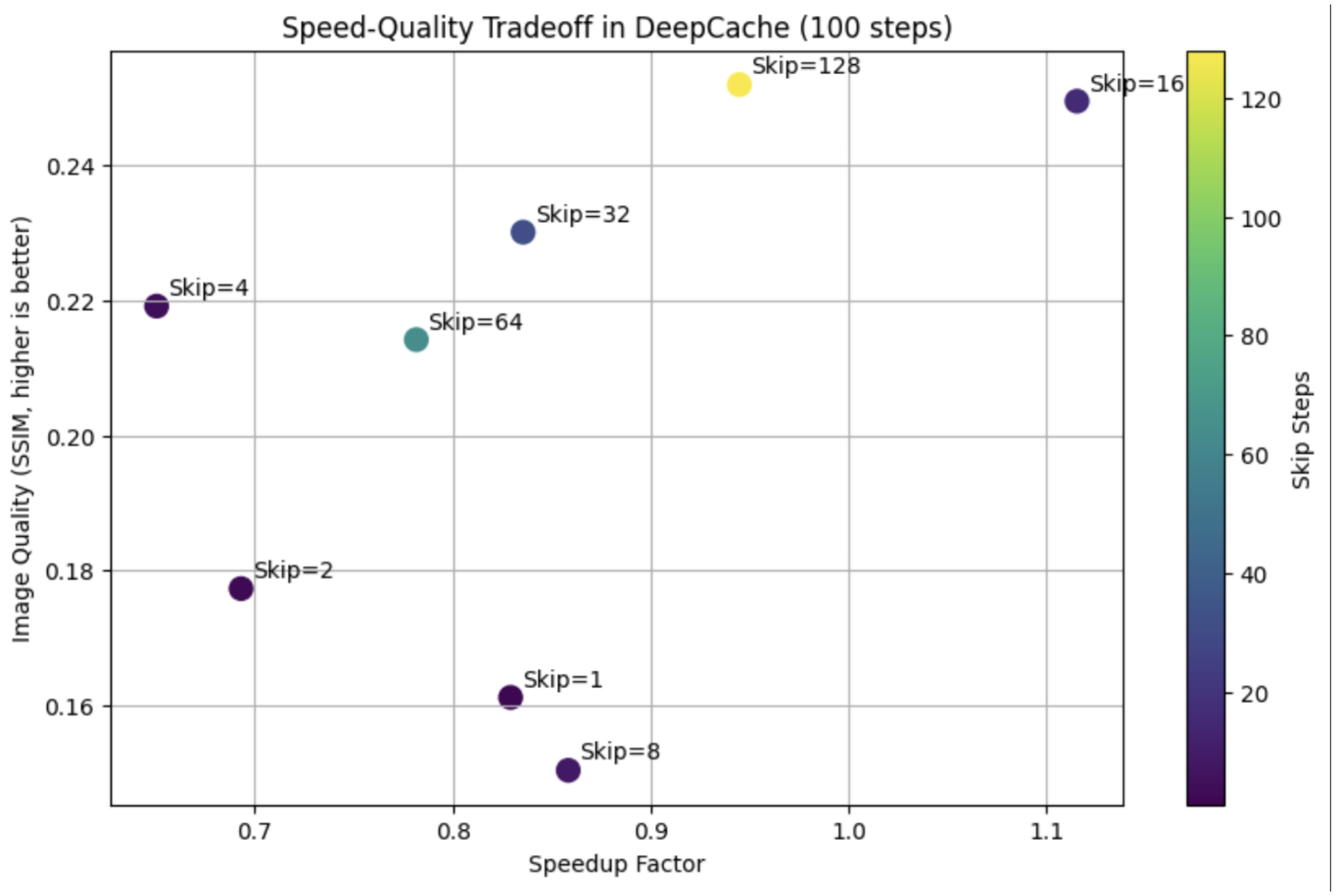

Speed-Quality Tradeoff in DeepCache (1 steps)
Image Quality (SSIM, higher is better)
Speedup Factor
Skip Steps
Skip=64
Skip=128
Skip=1
Skip=32
Skip=4
Skip=2
Skip=16
Skip=8

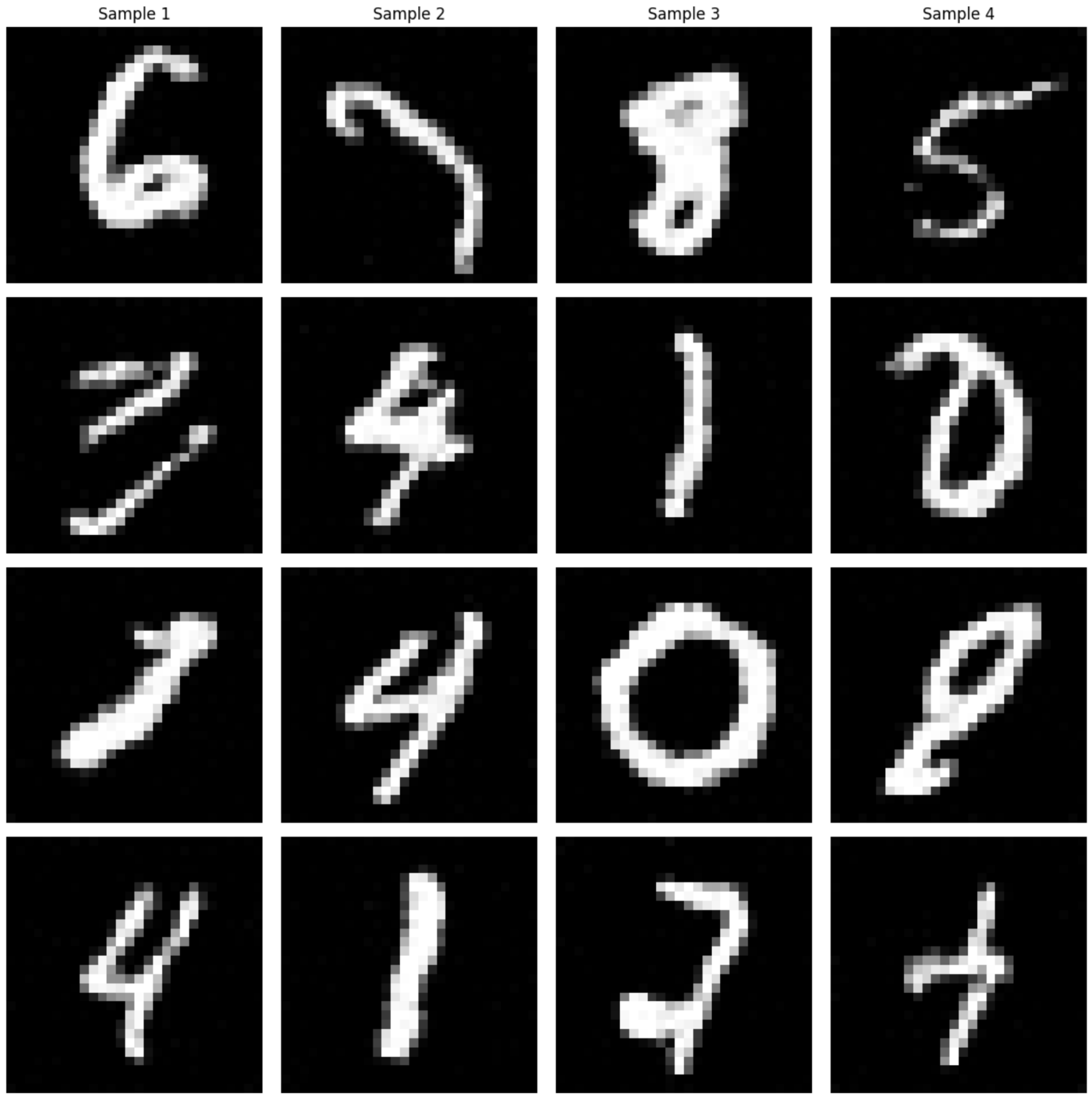

Sample 1
Sample 2
Sample 3
Sample 4


DeepCache Analysis with 10 Sampling Steps
Speedup vs. Cache Skip Steps
Speedup Factor
Cache Skip Steps
Image Quality (MSE) vs. Cache Skip Steps
Mean Squared Error (lower is better)
Cache Skip Steps
Image Quality (PSNR) vs. Cache Skip Steps
PSNR (higher is better)
Cache Skip Steps
Image Quality (SSIM) vs. Cache Skip Steps
SSIM (higher is better)
Cache Skip Steps


Speed-Quality Tradeoff in DeepCache (10 steps)
Skip=4
Skip=8
Skip=2
Skip=16
Skip=128
Skip=32
Skip=1
Skip=64
Image Quality (SSIM, higher is better)
Speedup Factor
Skip Steps


DeepCache Analysis with 100 Sampling Steps
Speedup vs. Cache Skip Steps
Speedup Factor
Cache Skip Steps
Image Quality (MSE) vs. Cache Skip Steps
Mean Squared Error (lower is better)
Cache Skip Steps
Image Quality (PSNR) vs. Cache Skip Steps
PSNR (higher is better)
Cache Skip Steps
Image Quality (SSIM) vs. Cache Skip Steps
SSIM (higher is better)
Cache Skip Steps


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)
Speedup Factor
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.

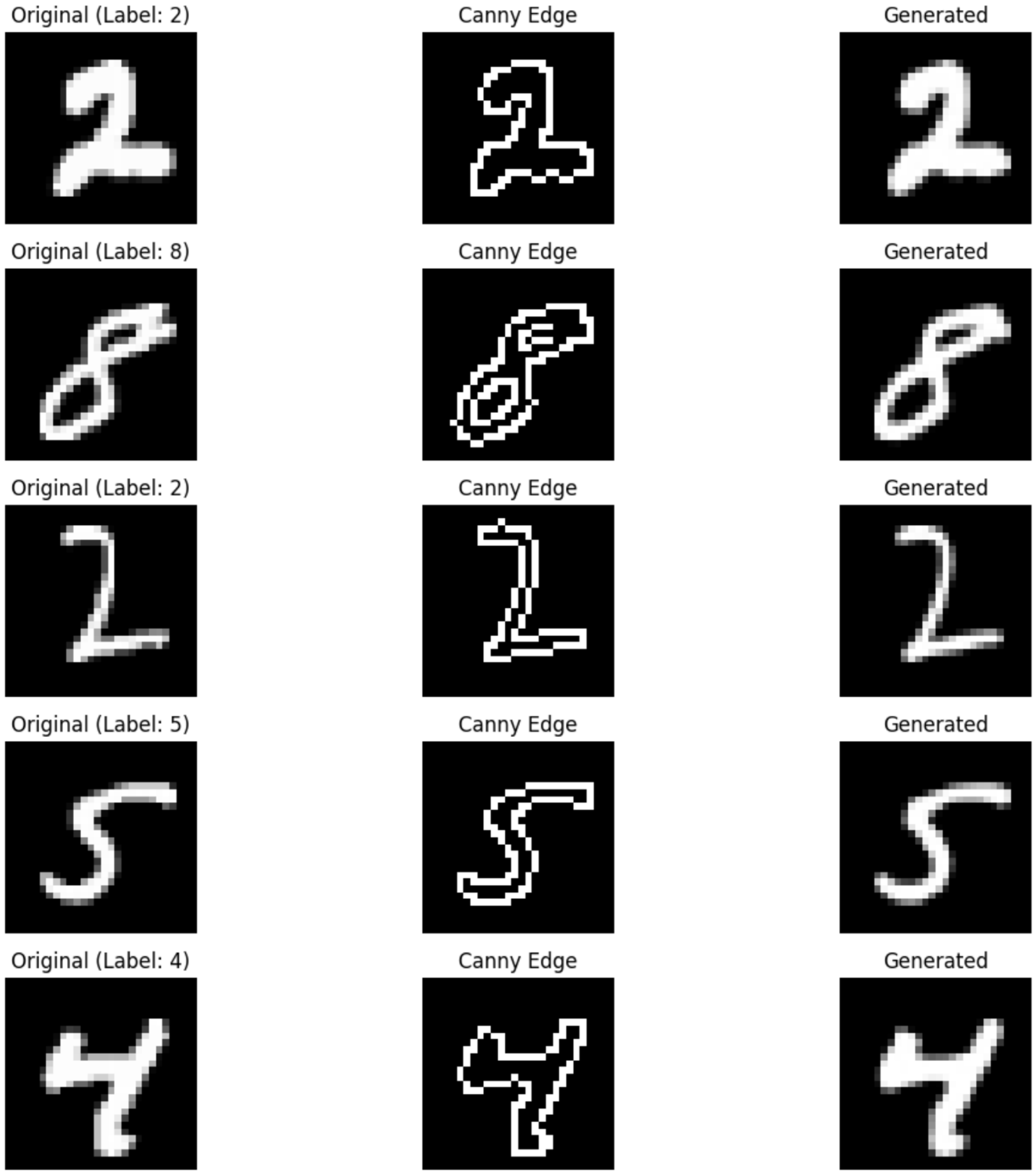
The architecture consists of a **ControlNet module** that extracts feature representations from the edge image and a **U-Net model** that synthesizes images based on these extracted features. The training process follows a **self-supervised reconstruction approach**, where the model is trained to generate 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