# Training Pipeline

All experiments were conducted using PyTorch Lightning for efficient training and reproducibility. Models were trained on the MNIST dataset, using Adam optimization and appropriate loss functions, with GPU acceleration where available.

# Main part

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

In scope of this task, a UNet and a diffusion model (DDPM with DDIM sampling) was developed. The UNet is built around sinusoidal time embeddings ( `SinusoidalPosEmb` ), residual blocks ( `ResBlock` ) that combine convolutional layers and time-driven projections, and a self-attention mechanism ( `AttentionBlock` ). Down/Up blocks reduce or restore spatial dimensions via strided and transpose convolutions while preserving context through skip connections.

#### Components

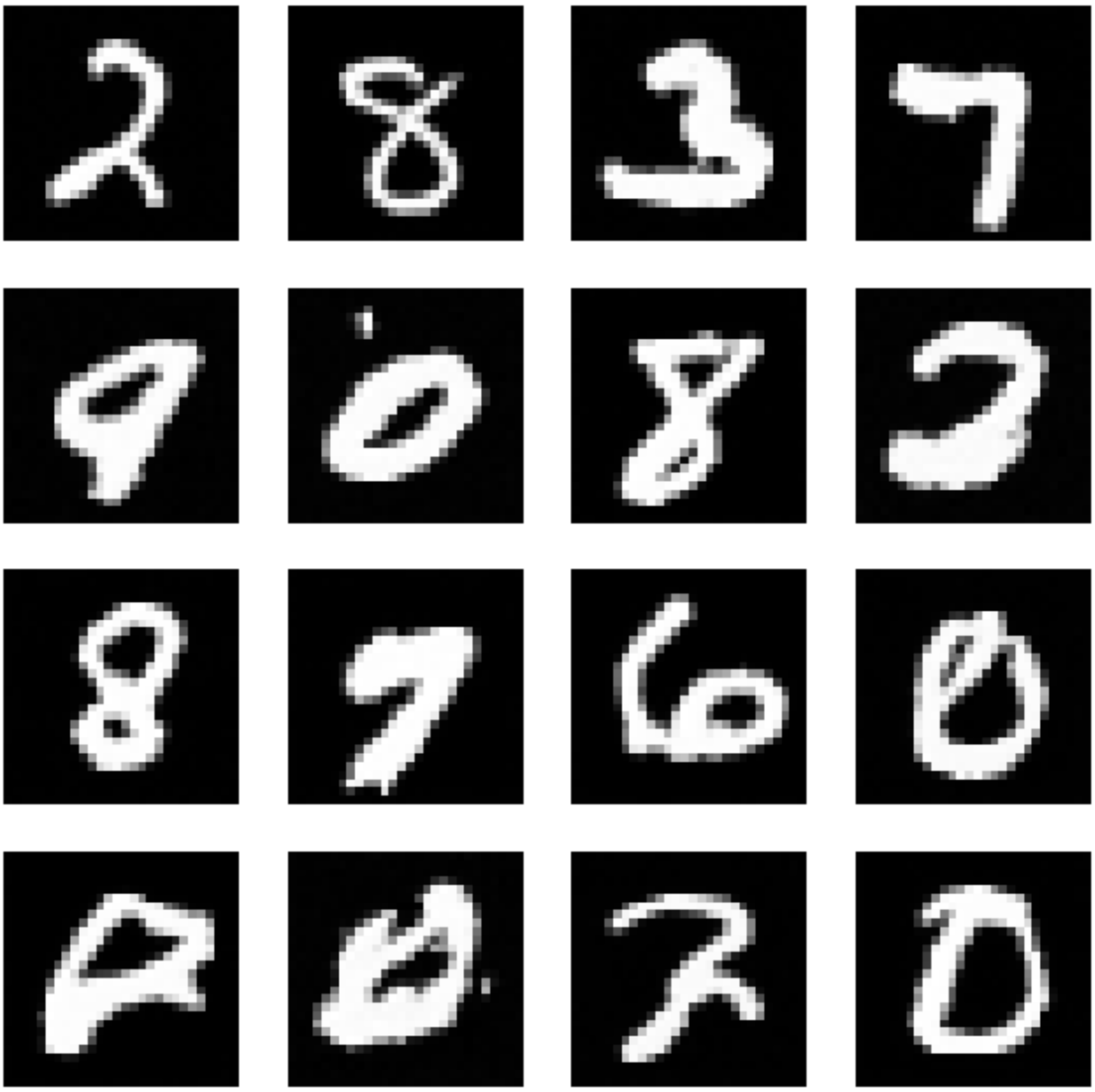
- **SinusoidalPosEmb**: Encodes time steps into sinusoidal embeddings.
- **ResBlock**: Residual block with time-conditioning.
- **AttentionBlock**: Self-attention mechanism for feature refinement.
- **Downsample / Upsample**: Resizes feature maps via convolutional layers.
- **DownBlock / UpBlock**: Stacks residual blocks, adds skip connections.
- **UNet**: Full architecture integrating encoder-decoder structure.

#### Network Flow

```
Input Image (B, C, H, W)
│
├── Initial Convolution ──> [Base Channels]
│
├── Downsampling Path:
│   ├── DownBlock 1 ──> [Base * 1]  (ResBlocks + Optional Attention +
Downsample)
│   ├── DownBlock 2 ──> [Base * 2]
│   ├── DownBlock 3 ──> [Base * 4]
│
├── Bottleneck:
│   ├── ResBlock ──> [Base * 4]
│   ├── AttentionBlock
│   ├── ResBlock ──> [Base * 4]
│
```

```
├── Upsampling Path:
│   ├── UpBlock 3 ──> [Base * 2]  (Upsample + Skip Connection + ResBlocks
+ Optional Attention)
│   ├── UpBlock 2 ──> [Base * 1]
│   ├── UpBlock 1 ──> [Base]
│
├── Final Normalization + Activation
│
└── Output Convolution ──> (B, C_out, H, W)  →  Denoised Image
```

The generated images exhibit recognizable digit structures, but some samples show distortions and inconsistencies, suggesting room for improvement in the model’s generative quality.



## 1.2 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

Here, a latent diffusion model was developed by integrating a VAE with the diffusion process, allowing generative modeling in a lower-dimensional latent space. The `SpatialVAE` compresses input images into a structured latent representation, using convolutional encoders to extract features and produce a mean ( `mu` ) and variance ( `logvar` ). The reparameterization trick samples latent vectors, which are then decoded back into images using transposed convolutions.

The `LatentDiffusionModel` applies a UNet-based denoising network to the VAE's latent space instead of raw pixels, making the process more efficient. A diffusion schedule (linear beta schedule) determines noise levels across timesteps, and the model learns to reverse this process by predicting noise at each step. Inference is performed using both DDIM and DDPM, where DDIM allows for faster sampling by skipping steps while maintaining sample quality.

### Components

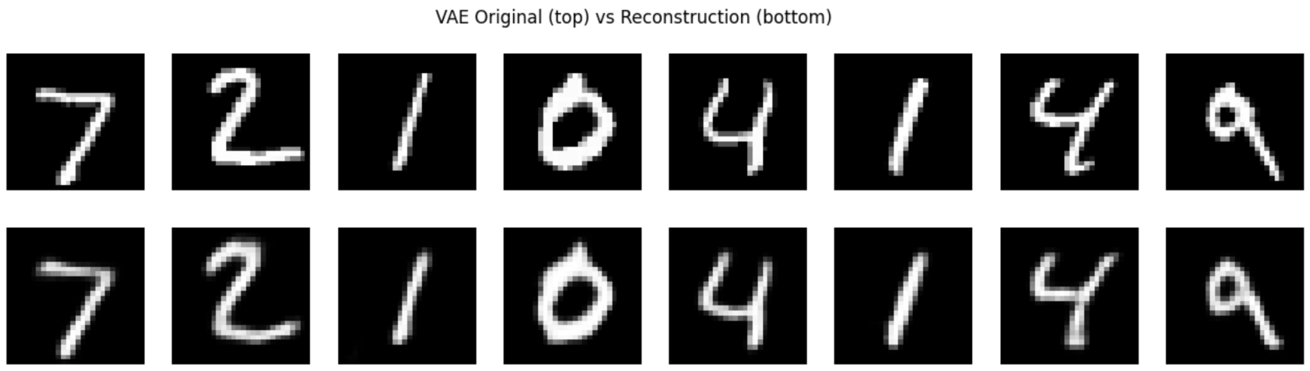
- **Encoder**: Compresses input images into a lower-dimensional latent space.
- **Latent Space**:
    - Outputs **mean** ( `mu` ) and **log-variance** ( `logvar` ).
    - Uses the **reparameterization trick** to sample latent vectors.
- **Decoder**: Reconstructs images from latent vectors using transposed convolutions.

### Network Flow

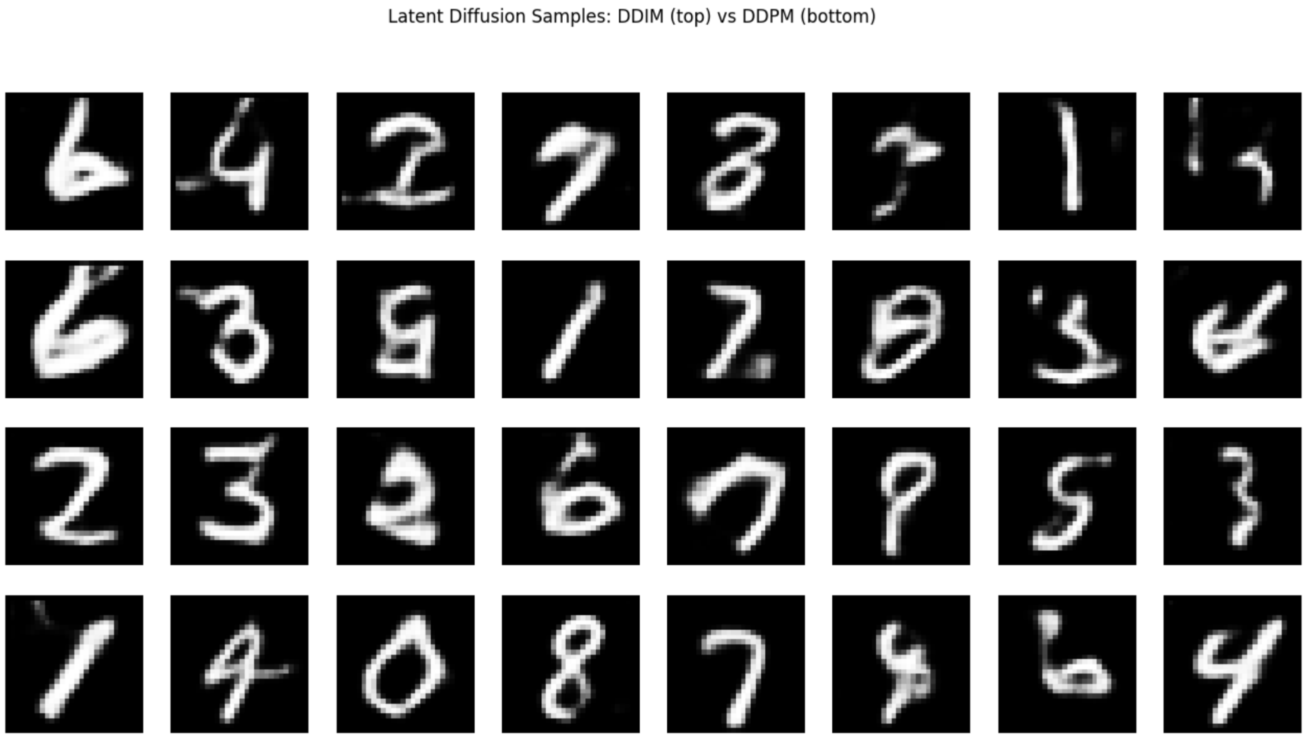
```
Input Image (B, C, 28, 28)
│
├── Encoder:
│   ├── Conv2D (C → 32) → ReLU → Downsample (28 → 14)
│   ├── Conv2D (32 → 64) → ReLU → Downsample (14 → 7)
│   ├── Conv2D (64 → 128) → ReLU → Downsample (7 → 4)
│
├── Latent Space:
│   ├── Conv2D (128 → Latent Channels) → Mean (`mu`)
│   ├── Conv2D (128 → Latent Channels) → Log-Variance (`logvar`)
│   ├── Reparameterization: `z = mu + std * epsilon`
│
├── Decoder:
│   ├── ConvTranspose2D (Latent → 128) → ReLU → Upsample (4 → 7)
│   ├── ConvTranspose2D (128 → 64) → ReLU → Upsample (7 → 14)
│   ├── ConvTranspose2D (64 → 32) → ReLU → Upsample (14 → 28)
│   ├── Conv2D (32 → C) → Sigmoid (Final Output)
│
└── Reconstructed Image (B, C, 28, 28)
```

Results include VAE reconstructions, showing that the VAE successfully compresses and reconstructs MNIST digits, and latent diffusion samples, comparing DDIM and DDPM-based generations. The generated images demonstrate the model's ability to produce realistic digits, though some samples still exhibit blurriness and distortions.

### VAE Reconstructions



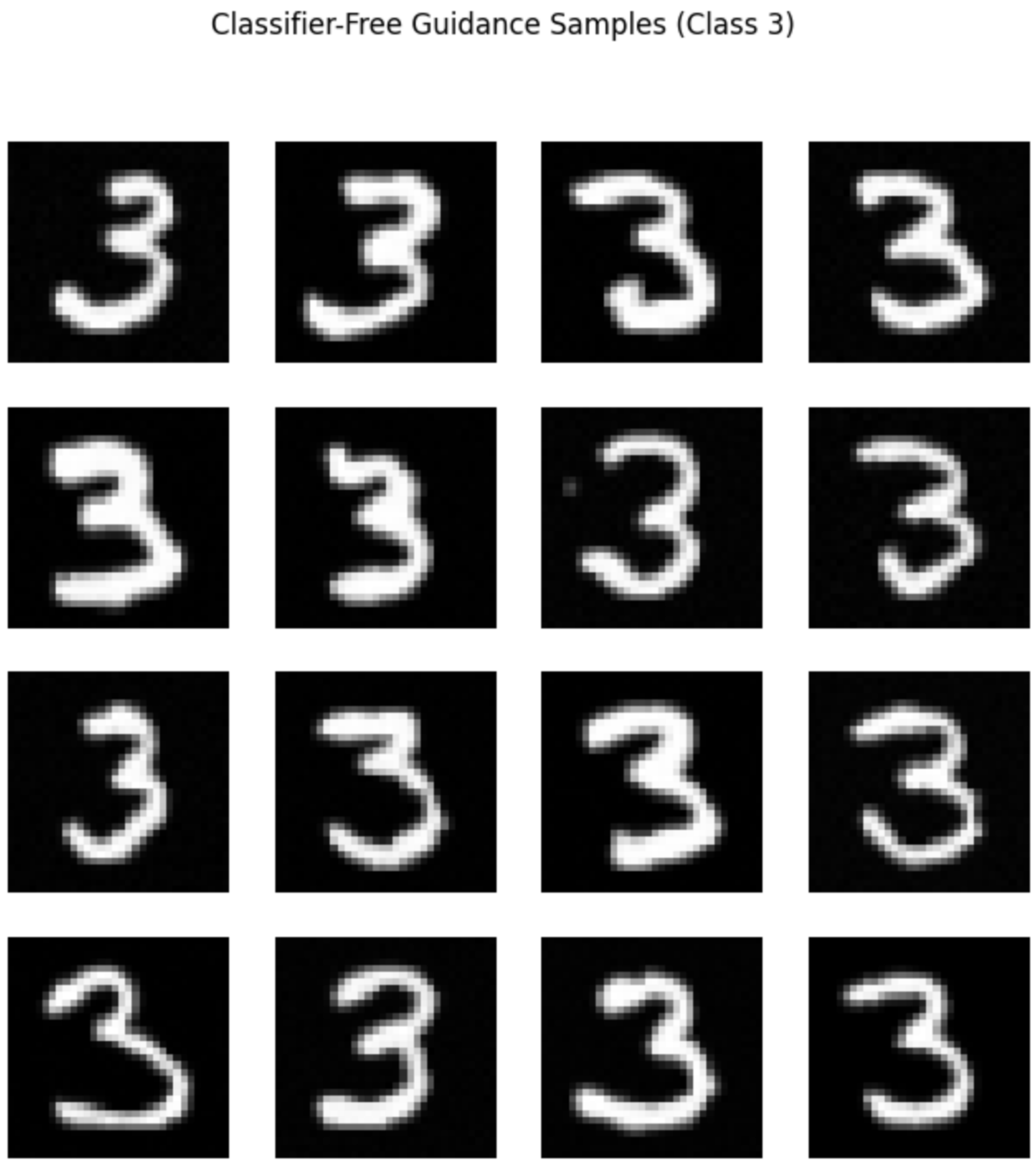
### Generated Images



## 1.3 Implement and train classifier-free guidance using class label. Condition U-net thought input channel and using cross-attention

Classifier-free guidance was implemented using a Conditional UNet and a diffusion model. The Conditional UNet extends the standard UNet by integrating cross-attention to condition the generation on class labels. A learned class embedding maps labels to a latent vector, which influences the network's mid-block through an attention mechanism. During training, the model randomly drops class conditions with a probability ( `cond_drop_prob` ), enabling classifier-free guidance.

The Classifier-Free Diffusion Model follows the standard diffusion training procedure but distinguishes between conditional and unconditional predictions. At inference, it generates samples by blending both predictions, scaled by a guidance weight to enhance class-consistency. Results show high-quality digit "3" generations, demonstrating improved control over the output while preserving diversity.

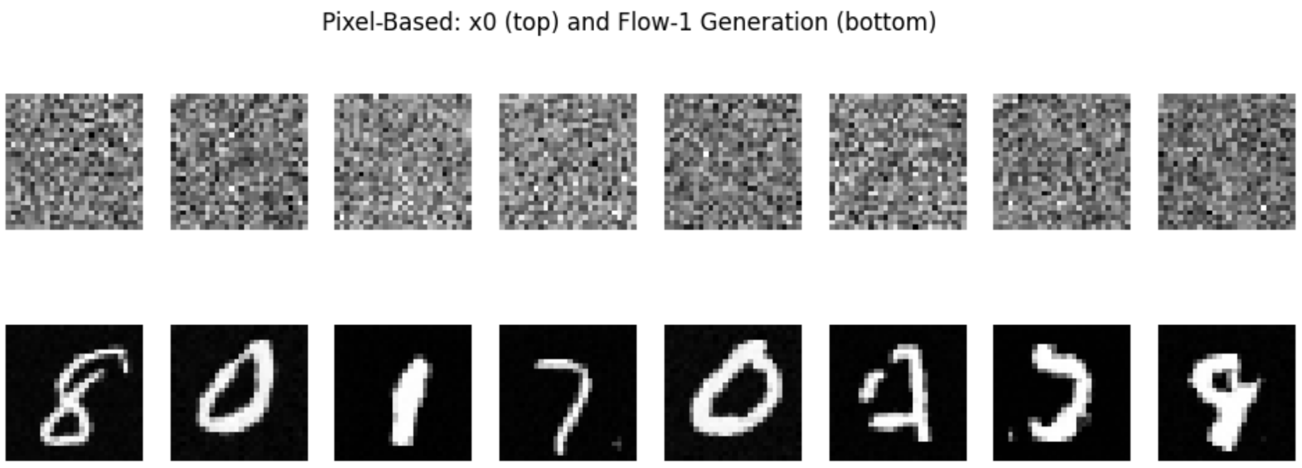


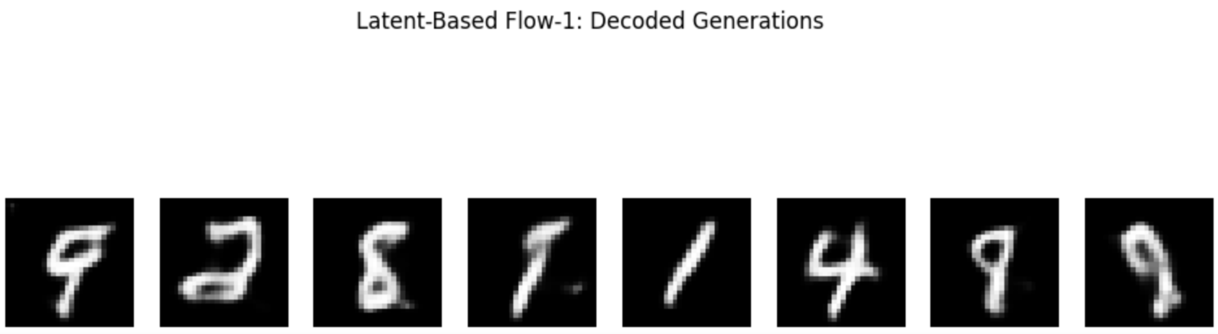
## 1.4 Implement and train diffusion model (pixel and latent based) using Rectified Flow

In this task, Rectified Flow (RF) was used to train diffusion models in both pixel space and latent space. RF replaces the standard stochastic noise-based training with a velocity field approach, learning to map noise directly to images in a more efficient way.

The pixel-based rectified flow model (Flow-1) was trained to progressively transform random noise into MNIST digits using an Euler integration scheme. A second-stage model (Flow-2) further refined this transformation by training on precomputed noisy-clean image pairs. The generated images demonstrate clear MNIST-like structures with smooth transformations from noise.

For latent-based rectified flow, a VAE was used to encode images into a structured latent space. Flow-1 was trained in this latent space, producing structured generative trajectories, followed by Flow-2 for further refinement. The decoded images from the latent model show smooth and well-defined digits, suggesting improved generation efficiency compared to pixel-space models.


Pixel-Based: x0 (top) and Flow-1 Generation (bottom)


Latent-Based Flow-1: Decoded Generations

## Bonus

### 3. Implement and train classifier guidance (based on class). (Note, you need to train classifier on noisy images)

The classifier was used to guide the generation process toward a target class during DDIM-based sampling. The diffusion model was trained using a UNet-based DDPM, predicting noise at different timesteps. A classifier was trained separately on MNIST images with noise, learning to classify digits in degraded conditions.

During inference, classifier gradients were used to adjust the predicted noise in each DDIM sampling step, steering the generation toward a specific class (e.g., digit "7"). The results show that the not all generated samples similar to target digit, which may be a problem of implementation

Classifier-Guided DDIM (Target: 7)