

Project5 : Fun With Diffusion Models!

Part A: The Power of Diffusion Models!

1.1 Implementing the Forward Process

The core of the forward diffusion process is to iteratively add noise to an input image, gradually degrading it from a clear image to random noise. The key formula is:



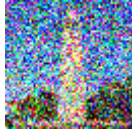
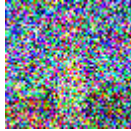
$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon},$$

where:

- $\sqrt{\bar{\alpha}_t}$ controls the proportion of the original signal retained;
- $\sqrt{1 - \bar{\alpha}_t}$ determines the noise strength;
- $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$ is Gaussian noise.

The algorithm iteratively computes this for different timesteps t , progressively adding noise to the image. This demonstrates the gradual replacement of signal by noise, forming the foundation for subsequent denoising and generation tasks.

result:

origin image	t = 250	t = 500	t = 750
			

1.2 Classical Denoising

1. Gaussian Blur Basics

Gaussian blur is a spatial-domain filtering technique that smooths an image by convolving it with a Gaussian kernel. The Gaussian kernel is defined as:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right),$$

where:

- (x, y) are offsets within the filter window;
- *sigma* is the standard deviation, controlling the blur intensity;
- The kernel size $k \times k$ is typically proportional to *sigma*.

Gaussian blur effectively suppresses high-frequency noise while preserving low-frequency structural information.

2. Iterative Denoising Process

For a noisy input image \mathbf{x}_t , the denoising process applies Gaussian blur at each timestep t , producing a denoised image $\hat{\mathbf{x}}_t$. This can be described as:

$$\hat{\mathbf{x}}_t = \mathbf{K}_\sigma * \mathbf{x}_t,$$

where:

- \mathbf{K}_σ is the Gaussian kernel with standard deviation σ ;
- $*$ denotes the convolution operation;
- \mathbf{x}_t is the image at timestep t ;
- $\hat{\mathbf{x}}_t$ is the denoised output.

The denoising algorithm iteratively uses the output of the previous step as the input for the next step, refining the image at each stage.

3. Parameter Effects

- **Kernel Size $k \times k$** : Larger kernels create stronger blur effects, improving noise suppression but potentially losing fine details.
- **Standard Deviation σ** : Controls the level of smoothing, where higher σ reduces more noise but risks oversmoothing the image.



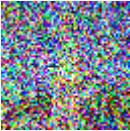



4. Iterative Process

Starting with the initial noisy image \mathbf{x}_0 and a series of timesteps $\{t_1, t_2, \dots, t_n\}$, the algorithm updates the image iteratively as:

$$\mathbf{x}_{t+1} = \mathbf{K}_\sigma * \mathbf{x}_t,$$

where each iteration reduces the noise while preserving the overall structure. The resulting sequence of images reflects varying levels of denoising at different timesteps.

result:

t = 250	t = 500	t = 750
noisy image	noisy image	noisy image
		
denoised image	denoised image	denoised image
		

1.3 One-Step Denoising

1. Forward Process

The first step involves generating a noisy image \mathbf{x}_t at a specific timestep t using the forward diffusion process. The process adds Gaussian noise to the clean image \mathbf{x}_0 , described by:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon},$$

where:

- $\bar{\alpha}_t$ is the cumulative product of diffusion coefficients α_i , controlling the retained signal ratio.
- $\sqrt{1 - \bar{\alpha}_t}$ scales the Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$.

For each specified t , this step generates noisy images \mathbf{x}_t , simulating progressively noisier inputs.

2. Noise Estimation

The algorithm estimates the noise $\boldsymbol{\epsilon}_{\text{est}}$ in \mathbf{x}_t using a neural network (UNet) conditioned on the timestep t and a prompt embedding \mathbf{e} . Mathematically, the network models:

$$\boldsymbol{\epsilon}_{\text{est}} = \text{UNet}(\mathbf{x}_t, t, \mathbf{e}),$$

where:

- \mathbf{x}_t : Noisy input image.
- t : Current timestep, guiding the network's understanding of the noise level.
- \mathbf{e} : Prompt embedding providing context or additional information.

The output $\boldsymbol{\epsilon}_{\text{est}}$ represents the estimated noise present in \mathbf{x}_t .

3. Clean Image Reconstruction

Using the estimated noise $\boldsymbol{\epsilon}_{\text{est}}$, the algorithm reconstructs the clean image $\hat{\mathbf{x}}_0$ by reversing the noise addition process. The reconstruction is given by:

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \cdot \boldsymbol{\epsilon}_{\text{est}}}{\sqrt{\bar{\alpha}_t}},$$


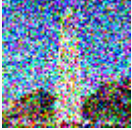
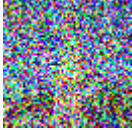



where:

- $\sqrt{1 - \bar{\alpha}_t} \cdot \boldsymbol{\epsilon}_{\text{est}}$ removes the estimated noise component.
- Division by $\sqrt{\bar{\alpha}_t}$ rescales the signal to approximate the original clean image.

This step is iterated for each timestep t to generate reconstructed images at varying noise levels.

result:

result:

t = 250	t = 500	t = 750
noisy image	noisy image	noisy image
		
denoised image	denoised image	denoised image
		

1.4 Iterative Denoising

1. Cumulative Coefficient Calculation

For timestep t and its previous timestep t' , compute:

$$\alpha = \frac{\bar{\alpha}_t}{\bar{\alpha}_{t'}}, \quad \beta = 1 - \alpha,$$

where:

- $\bar{\alpha}_t$ is the cumulative product of diffusion coefficients at t .
- α and β determine the weights for combining the predicted clean image and the current noisy image.

2. Noise Estimation

The neural network UNet estimates the noise ϵ_{est} in the noisy image:

$$\epsilon_{\text{est}} = \text{UNet}(\mathbf{x}_t, t, \mathbf{e}),$$

where \mathbf{x}_t is the noisy image, t is the timestep, and \mathbf{e} is the prompt embedding.

3. Clean Image Prediction

Using the noise estimate ϵ_{est} , the clean image \mathbf{x}_0 is predicted as:



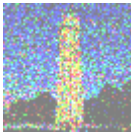
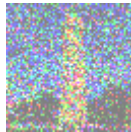
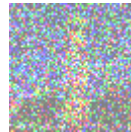
$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon_{\text{est}}}{\sqrt{\bar{\alpha}_t}}.$$



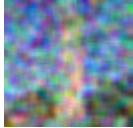
4. Refine the Image

The algorithm predicts the less noisy image at the previous timestep $\mathbf{x}_{t'}$ using:

$$\mathbf{x}_{t'} = \sqrt{\bar{\alpha}_{t'}} \cdot \hat{\mathbf{x}}_0 + \sqrt{1 - \bar{\alpha}_{t'}} \cdot \epsilon_{\text{est}},$$

combining the predicted clean image and residual noise for stability.

t = 90	t = 240	t = 390	t = 540	t = 690
				

original	Iteratively Denoised Campanile	One-Step Denoised Campanile	Gaussian Blurred Campanile
			

1.5 Diffusion Model Sampling

1. Initial Noise Generation

The process begins with generating random Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$, where:

$$\mathbf{x}_T = \text{random noise.}$$

This noise serves as the starting point for sampling and will be refined in subsequent steps.

2. Iterative Denoising

The denoising process refines the noisy input \mathbf{x}_T through a series of timesteps $T \rightarrow 0$. At each timestep t , the following operations are performed:

1. Noise Estimation:

A neural network UNet estimates the noise component ϵ_{est} present in the noisy image:

$$\epsilon_{\text{est}} = \text{UNet}(\mathbf{x}_t, t, \mathbf{e}),$$

where:

- \mathbf{x}_t : The noisy image at timestep t .
- t : The current timestep.
- \mathbf{e} : The prompt embedding, guiding the model toward specific image features (e.g., "a high-quality photo").

2. Clean Image Prediction:

Using the estimated noise, the algorithm predicts the clean image at timestep t :

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon_{\text{est}}}{\sqrt{\bar{\alpha}_t}},$$

where α is the cumulative diffusion coefficient.

3. Refinement:

The image for the previous timestep \mathbf{x}_{t-1} is computed by reintroducing some noise for stochasticity:

$$\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \cdot \hat{\mathbf{x}}_0 + \sqrt{1 - \bar{\alpha}_{t-1}} \cdot \epsilon_{\text{est}}.$$

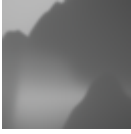

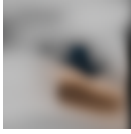
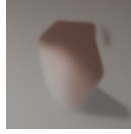
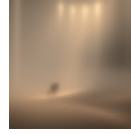
This step progressively reduces noise while preserving structural details guided by the prompt embedding.

3. Image Sampling

The algorithm repeats the denoising process for $T \rightarrow 0$, generating a clean image from the initial random noise. Multiple images can be generated by repeating this process independently, with different initial noise samples.

For N samples, the output set is:

$$\{\hat{\mathbf{x}}_0^{(1)}, \hat{\mathbf{x}}_0^{(2)}, \dots, \hat{\mathbf{x}}_0^{(N)}\}.$$

Sample 1	Sample 2	Sample 3	Sample 4	Sample 5
				

1.6 Classifier-Free Guidance (CFG)

1. Starting from Noise

The process begins with a random Gaussian noise image $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$ at the maximum timestep T . This noise serves as the starting point for progressively generating a meaningful image.

2. Reverse Diffusion with CFG

The algorithm uses the reverse diffusion process to iteratively refine the noisy image. At each timestep t , it calculates a cleaner image \mathbf{x}_{t-1} by incorporating guidance from a conditional prompt and an unconditional prompt.

3. Noise Estimation

For each timestep t , two noise estimates are generated:

1. **Conditional Noise Estimate** ϵ_{cond} :

Estimated noise when conditioning on a specific prompt embedding \mathbf{e}_{cond} :

$$\epsilon_{\text{cond}} = \text{UNet}(\mathbf{x}_t, t, \mathbf{e}_{\text{cond}})$$

2. **Unconditional Noise Estimate** ϵ_{uncond} :

Estimated noise without any specific prompt guidance:

$$\epsilon_{\text{uncond}} = \text{UNet}(\mathbf{x}_t, t, \mathbf{e}_{\text{uncond}}),$$

where $\mathbf{e}_{\text{uncond}}$ is an empty embedding.

4. Classifier-Free Guidance

The algorithm combines the two noise estimates using the **Classifier-Free Guidance** formula to enhance the influence of the conditional guidance:

$$\epsilon_{\text{CFG}} = \epsilon_{\text{uncond}} + s \cdot (\epsilon_{\text{cond}} - \epsilon_{\text{uncond}}),$$

where:

- s is the guidance scale, controlling how strongly the conditional prompt influences the generation.

5. Clean Image Prediction

Using the combined noise estimate ϵ_{CFG} , the algorithm predicts the clean image $\hat{\mathbf{x}}_0$ at the current timestep t :

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon_{\text{CFG}}}{\sqrt{\bar{\alpha}_t}},$$

where α is the cumulative diffusion coefficient.



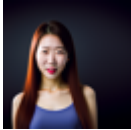
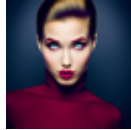

6. Refining the Image

The image at the previous timestep \mathbf{x}_{t-1} is computed by combining $\hat{\mathbf{x}}_0$ with some noise for stochasticity:

$$\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \cdot \hat{\mathbf{x}}_0 + \sqrt{1 - \bar{\alpha}_{t-1}} \cdot \epsilon_{\text{CFG}}.$$

This ensures that the generation process balances noise reduction with flexibility in generating variations.

result:

Sample 1	Sample 2	Sample 3	Sample 4	Sample 5
				

1.7 Image-to-image Translation

1. Adding Noise

The clean image \mathbf{x}_0 is corrupted by noise using the forward diffusion process:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \cdot \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon,$$

where ϵ is random Gaussian noise, and $\bar{\alpha}_t$ controls how much of the original image remains. This creates progressively noisier images for various noise levels.

2. Iterative Denoising

To restore the image, the algorithm iteratively removes noise for each timestep $t \rightarrow t - 1$:

1. Noise Estimation:

Two noise estimates are made:

- ϵ_{cond} : Guided by the text prompt (e.g., "a high-quality photo").
- ϵ_{uncond} : Without any prompt guidance.

2. Combining Noise Estimates:

The two estimates are blended using:

$$\epsilon_{\text{CFG}} = \epsilon_{\text{uncond}} + s \cdot (\epsilon_{\text{cond}} - \epsilon_{\text{uncond}}),$$

where s is the guidance strength.






















3. Updating the Image:

The clean image \hat{x}_0 is predicted, and the noisy image is refined step by step until the final output is achieved.

3. Multiple Noise Levels

The process is repeated for various noise levels, demonstrating the model's ability to handle both mild and extreme noise.

result:

	SDEdit with i_start=1	SDEdit with i_start=3	SDEdit with i_start=5	SDEdit with i_start=7	SDEdit with i_start=10	SDEdit with i_start=20	origin
1							
2							
3							

1.7.1 Editing Hand-Drawn and Web Images

1. Image Preparation:

The input image (from the web or drawn manually) is resized, normalized, and converted to a format suitable for the model.

2. Adding Noise:

Noise is added at different levels ($t=1,3,5,7,10,20$; $t=1, 3, 5, 7, 10, 20$; $t=1,3,5,7,10,20$) to simulate progressively degraded versions of the image.






















3. Iterative Denoising:

The model removes noise step by step, guided by a text prompt (e.g., "a high-quality photo"). This restores the image by combining conditional and unconditional noise estimates.

4. Result Display:

Cleaned images are displayed for each noise level, showing how the model reconstructs high-quality outputs even from heavily degraded inputs.



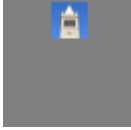





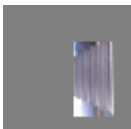
result:

	1	3	5	7	10	20	origin
web							
draw1							
draw2							

1.7.2 Inpainting

Mask Application: At each step, the updated image is combined with the original unmasked region to ensure only the masked area is modified:

$$\mathbf{x}_{t-1} = \mathbf{x}_{t-1} \cdot \text{mask} + \mathbf{x}_t \cdot (1 - \text{mask}).$$

	Image	Mask	ToReplace
1			
2			
3			

1. Image and Mask Preparation:

The input image is resized and normalized to a range of $([-1, 1])$. A binary mask defines the region to be replaced, while unmasked areas remain unchanged.

2. Adding Noise:

Noise is added to the masked region using the forward diffusion process:

$$\mathbf{x}_t = \sqrt{\alpha_t} \cdot \mathbf{x}_0 + \sqrt{1 - \alpha_t} \cdot \epsilon,$$

where \mathbf{x}_t is the noisy image, and \mathbf{x}_0 is the clean image.

3. Iterative Denoising:

- Two noise estimates are computed:

$$\epsilon_{\text{cond}}, \quad \epsilon_{\text{uncond}}$$

for conditional and unconditional prompts.

- Classifier-Free Guidance combines them:

$$\epsilon_{\text{CFG}} = \epsilon_{\text{uncond}} + s \cdot (\epsilon_{\text{cond}} - \epsilon_{\text{uncond}}),$$













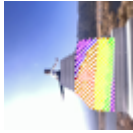

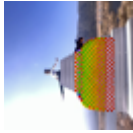
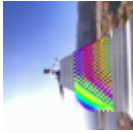
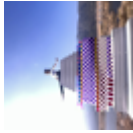
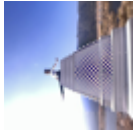
where s controls the prompt influence.

- The image is iteratively updated to refine the masked region while preserving unmasked areas:

$$\mathbf{x}_{t-1} = \mathbf{x}_{t-1} \cdot \text{mask} + \mathbf{x}_t \cdot (1 - \text{mask}).$$

4. Final Output:

After all iterations, the masked region is reconstructed based on the prompt and seamlessly blends with the rest of the image.

	1	3	5	7	10	20
1						
2						
3						

1.7.3 Text-Conditional Image-to-image Translation

1. Prepare the Image:

The input image is resized and normalized for the model.

2. Add Noise:

Noise is added at different levels to simulate degraded images.














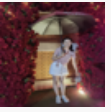



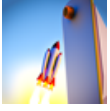



3. Denoise with Guidance:

The noisy image is refined step by step, guided by a text prompt (e.g., "a rocket ship") to align the result with the prompt.

4. Output:

The final edited image is displayed for each noise level, showing how the model reconstructs and edits the image.

Result:

	1	3	5	7	10	20	origin
1							
2							
3							

1.8 Visual Anagrams

1. Starting with Noise

The algorithm begins with a random Gaussian noise image:

$$\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I}),$$

which serves as the starting point for refinement.

2. Noise Predictions

At each timestep t , noise predictions are computed for:

- **Original Orientation:** Using prompt embedding \mathbf{e}_1 , generating $\epsilon_{\text{cond},1}$.
- **Flipped Orientation:** Using prompt embedding \mathbf{e}_2 on the flipped image, generating $\epsilon_{\text{cond},2}$, which is then flipped back.

Additionally, an unconditional noise estimate ϵ_{uncond} is computed without prompt guidance.

3. Blending Noise Predictions:

The conditional noise estimates are blended using weights:

$$\epsilon_{\text{combined}} = w \cdot \epsilon_{\text{cond},1} + (1 - w) \cdot \epsilon_{\text{cond},2},$$

where w controls the influence of each prompt.

4. Classifier-Free Guidance:

The combined noise estimate is enhanced using CFG:

where s is the guidance scale.

5. Denoising the Image:

The clean image estimate $\hat{\mathbf{x}}_0$ is calculated:

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon_{\text{CFG}}}{\sqrt{\bar{\alpha}_t}},$$

where $\bar{\alpha}_t$ is the cumulative diffusion coefficient.

The less noisy image \mathbf{x}_{t-1} for the next step is updated as:

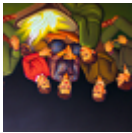





6. Final Outputs:

- The generated image is refined through all timesteps.
- A flipped version of the final image is created to form a visual anagram illusion.





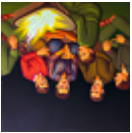















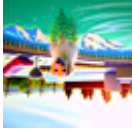
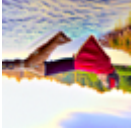

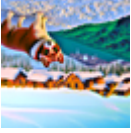

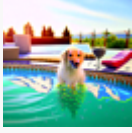

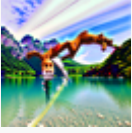
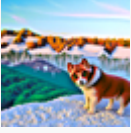

prompt:

- The first prompt is :
 - an oil painting of an old man
 - an oil painting of people around a campfire
- The second prompt is :
 - a lithograph of waterfalls
 - a man wearing a hat
- The third prompt is:
 - an oil painting of a snowy mountain village
 - a photo of a dog

result:

	image	flipped image
1		
2		
3		

process:

	1	2	3	4	5
generate 1					
flipped 1					
generate 2					
flipped 2					
generate 3					
flipped 3					

1.9 Hybrid Images

1. Starting with Noise

The algorithm begins with a random Gaussian noise image

$$\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I}),$$

which is refined iteratively into the final hybrid image.

2. Conditional and Unconditional Predictions

For each timestep t :

- Two noise predictions are generated:
 - $\epsilon_{\text{cond},1}$: Using the first text prompt \mathbf{e}_1 (e.g., "an oil painting of a snowy mountain village").
 - $\epsilon_{\text{cond},2}$: Using the second text prompt \mathbf{e}_2 (e.g., "a photo of a dog").
 - An unconditional prediction ϵ_{uncond} is generated without any text prompt guidance.
-

3. Spectral Filtering

Each noise prediction is decomposed into low-pass and high-pass components:

- Low-pass components lowpass_1 and lowpass_2 are extracted using Gaussian blur.
- High-pass components highpass_1 and highpass_2 are computed by subtracting the blurred image from the original prediction.

The blended noise estimate is constructed as:

$$\epsilon_{\text{combined}} = \text{lowpass}_1 + (\epsilon_{\text{cond},2} - \text{highpass}_1),$$

allowing the first prompt to dominate low-frequency features and the second prompt to contribute high-frequency details.

4. Classifier-Free Guidance

The blended noise estimate is refined using CFG:

$$\epsilon_{\text{CFG}} = \epsilon_{\text{uncond}} + s \cdot (\epsilon_{\text{combined}} - \epsilon_{\text{uncond}}),$$

where s is the guidance scale that amplifies the influence of the prompts.

5. Denoising and Image Update

The algorithm iteratively updates the noisy image \mathbf{x}_t toward the clean image:

1. Estimate the clean image:

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon_{\text{CFG}}}{\sqrt{\bar{\alpha}_t}},$$

where $\bar{\alpha}_t$ is the cumulative diffusion coefficient.

2. Update the image for the next timestep:

$$\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \cdot \hat{\mathbf{x}}_0 + \sqrt{1 - \bar{\alpha}_{t-1}} \cdot \epsilon_{\text{CFG}}.$$

6. Final Hybrid Image




After iterating through all timesteps, the algorithm produces a hybrid image combining:

- Low-frequency details (e.g., overall structure) from the first prompt.
- High-frequency details (e.g., textures) from the second prompt.


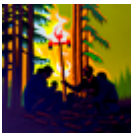

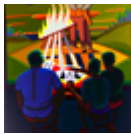









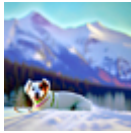

prompt:

- The first prompt is :
 - an oil painting of an old man
 - an oil painting of people around a campfire
- The second prompt is :
 - a lithograph of waterfalls
 - a man wearing a hat
- The third prompt is:
 - an oil painting of a snowy mountain village
 - a photo of a dog

result:

1	2	3
		

process:

	1	2	3	4	final
1					
2					
3					

Part B: Diffusion Models from Scratch!

Part 1: Training a Single-Step Denoising UNet

1.1 Implementing the UNet

Construct a neural network based on this diagram.

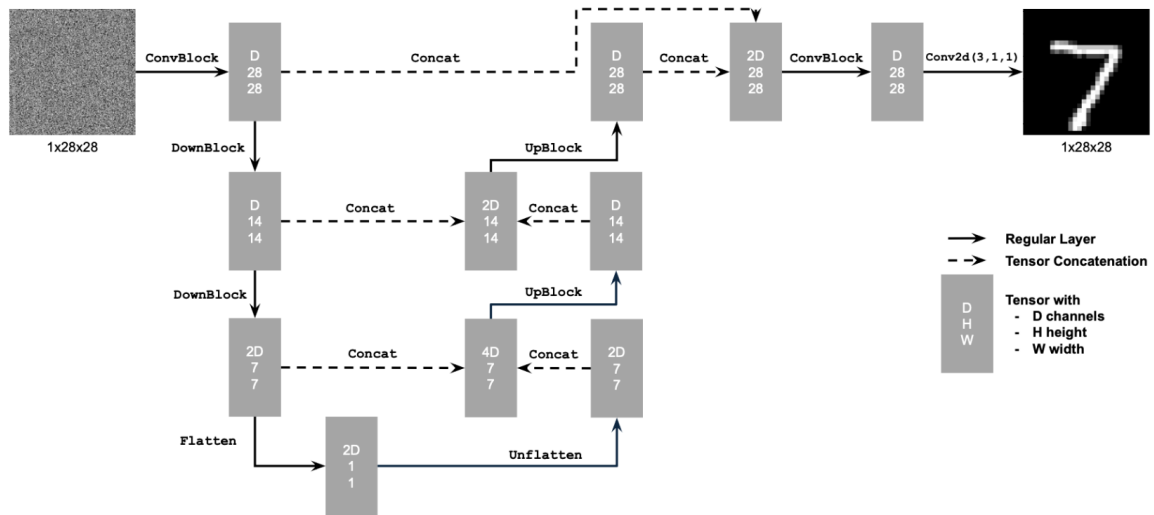
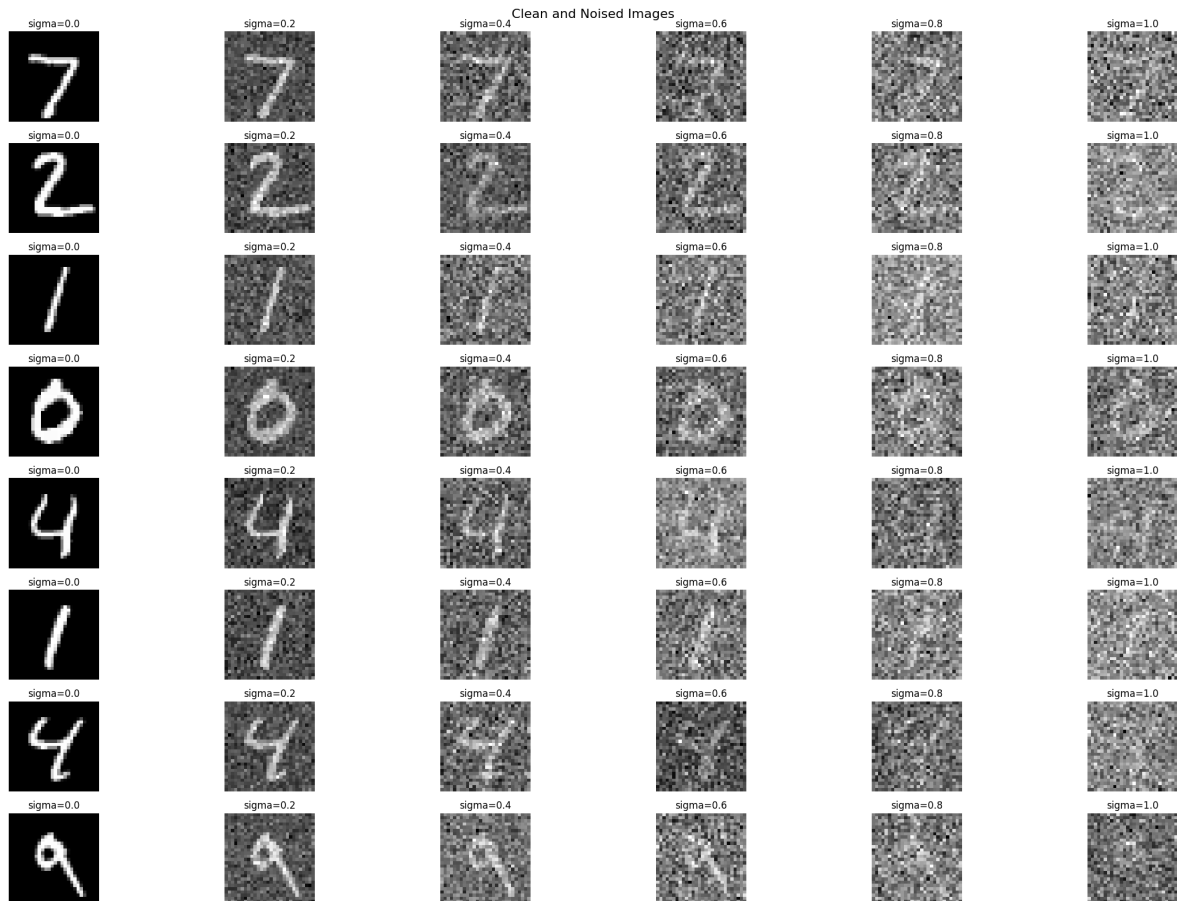
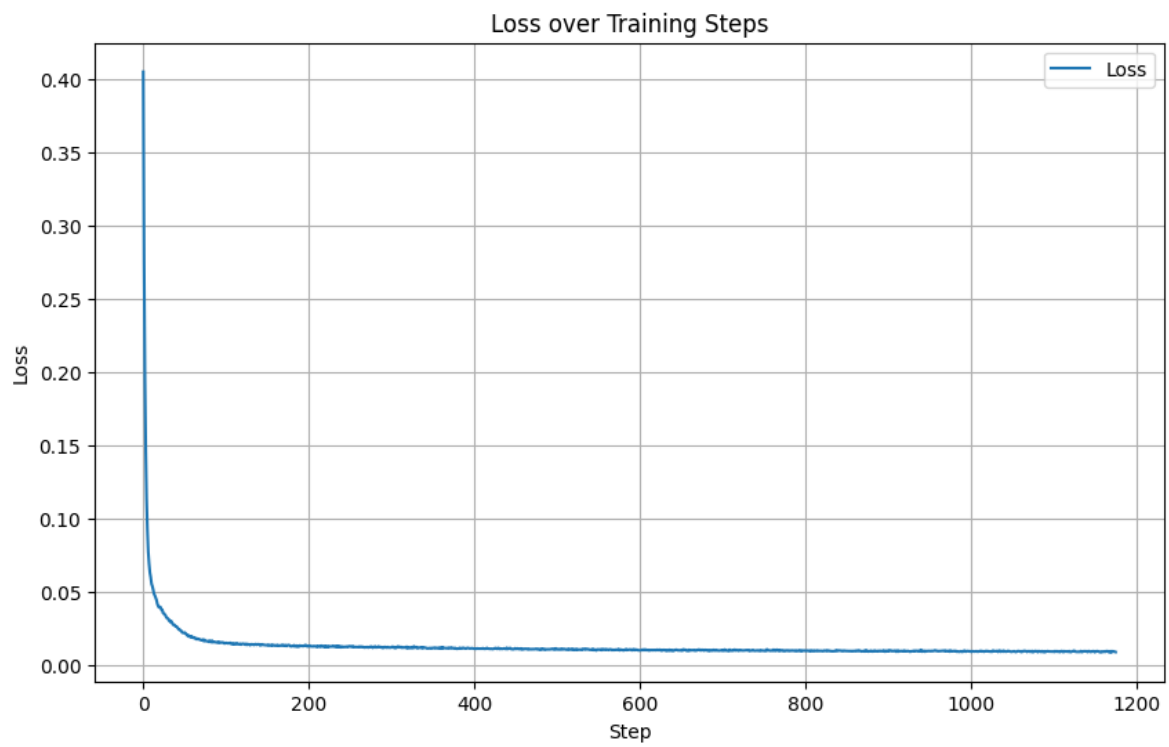


Figure 1: Unconditional UNet

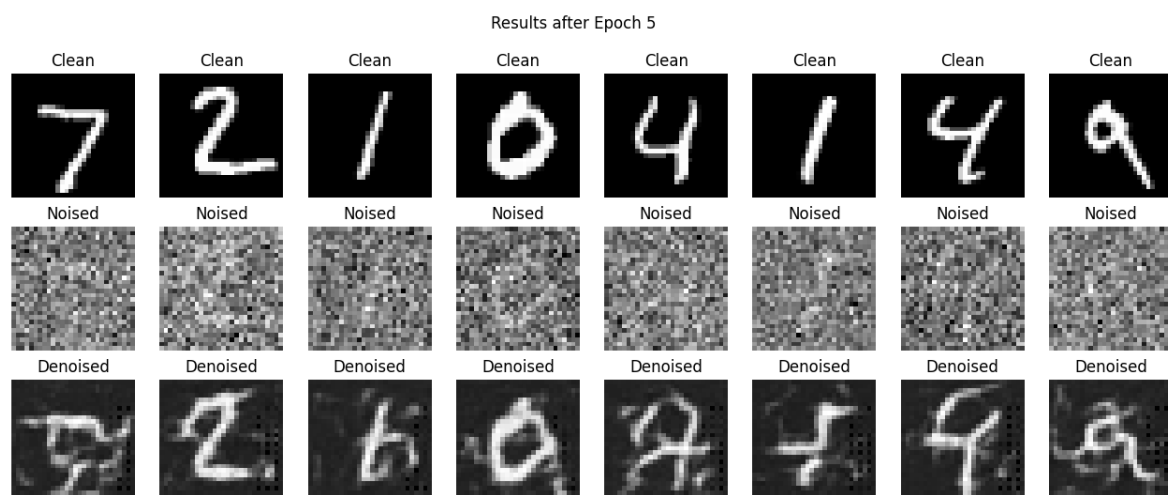
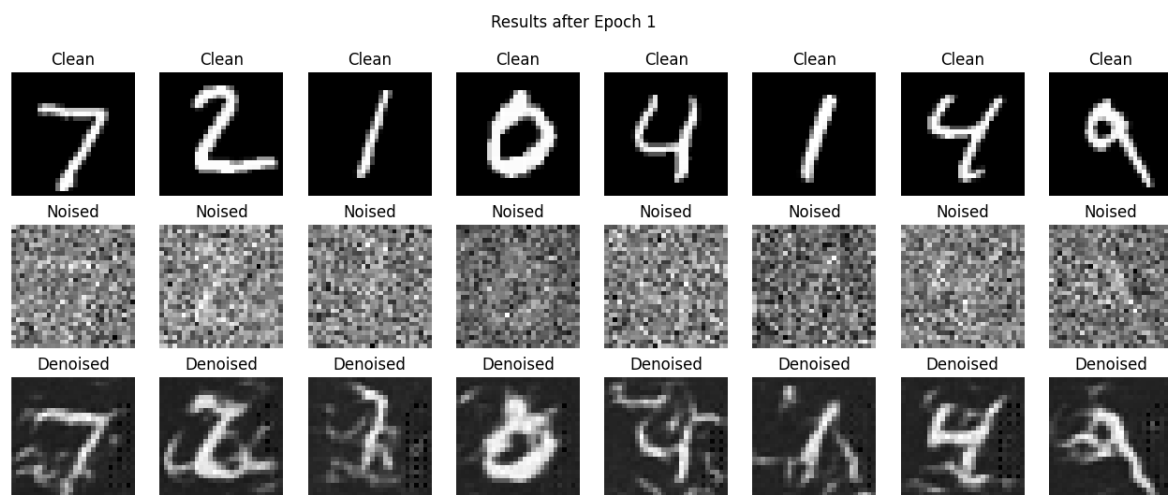
Then add different levels of noise to the image



Use the constructed model and the images from the MNIST dataset with progressively added noise as data to train the model. Then, obtain the loss:



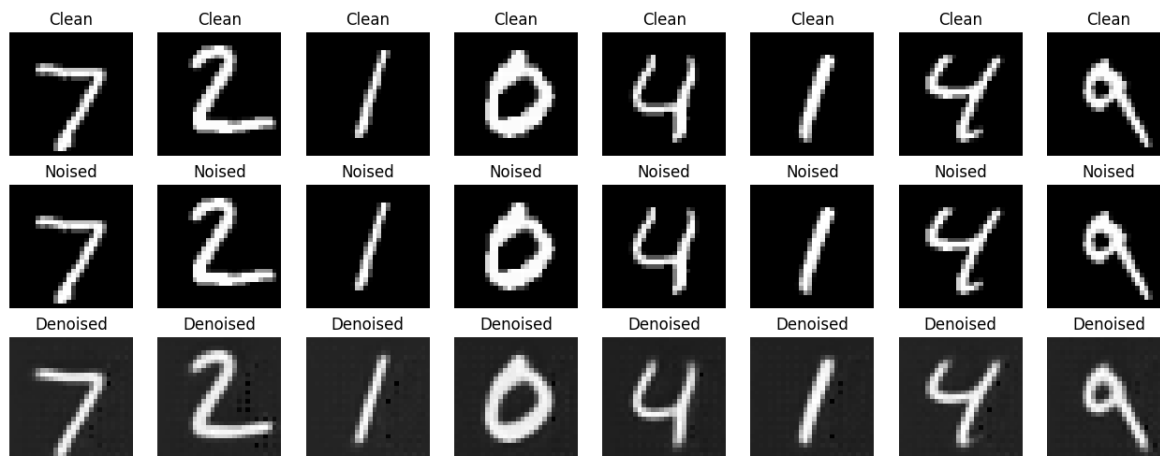
The result after train:



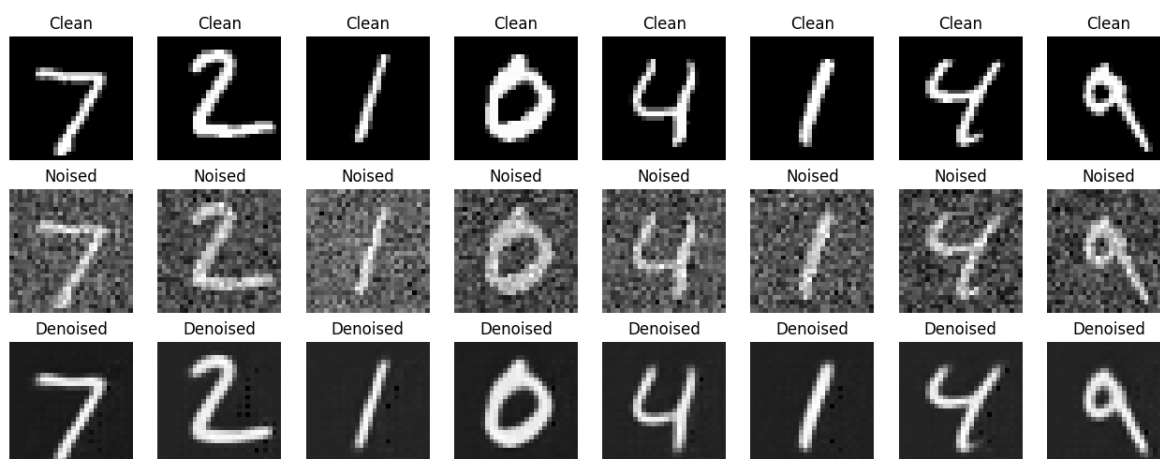
1.2.2 Out-of-Distribution Testing

Transfer the model to other datasets to test its denoising performance:

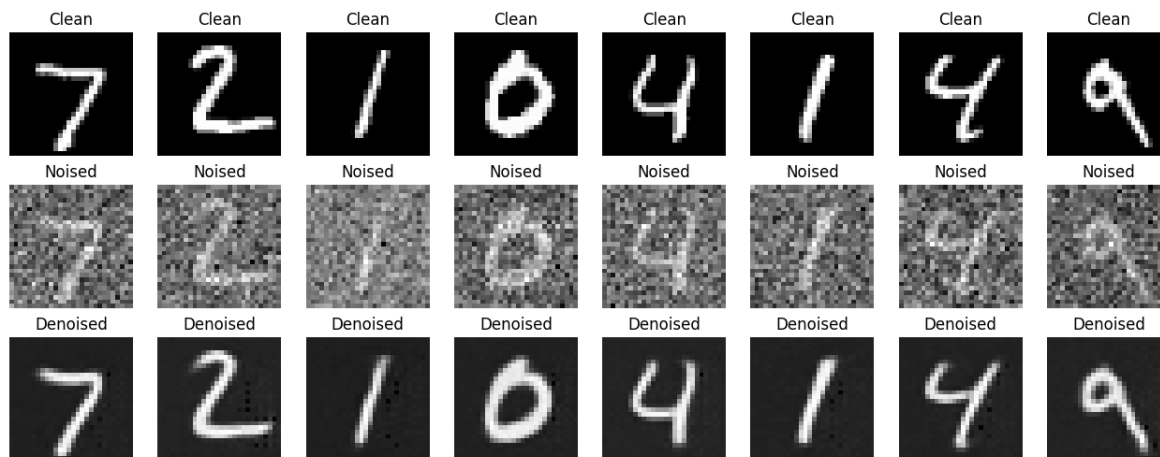
Results after Epoch out-of-distribution sigma=0.0



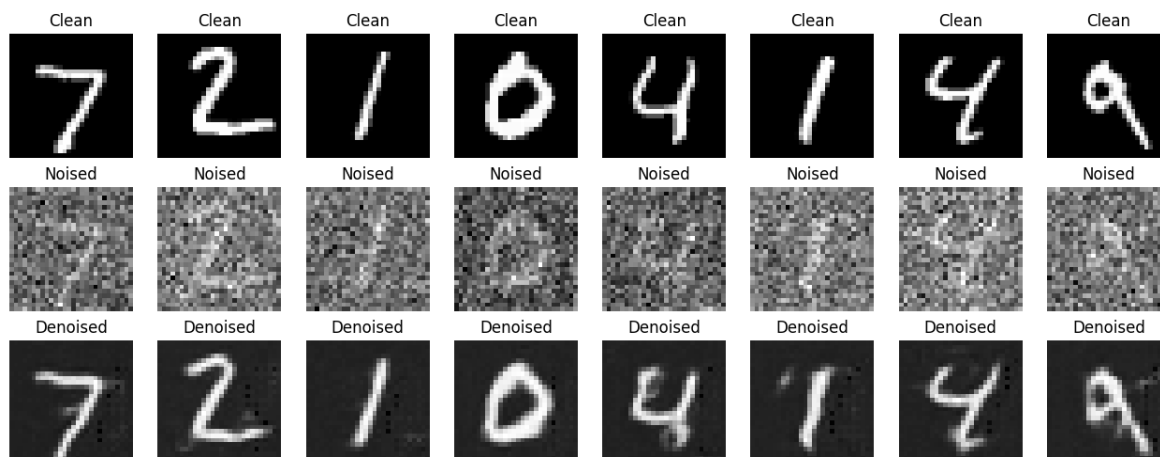
Results after Epoch out-of-distribution sigma=0.2

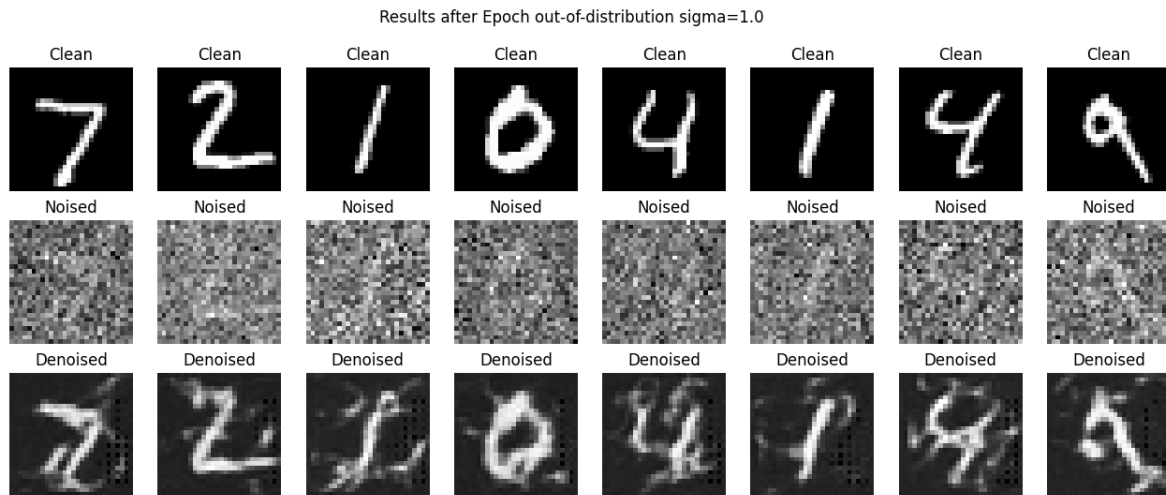
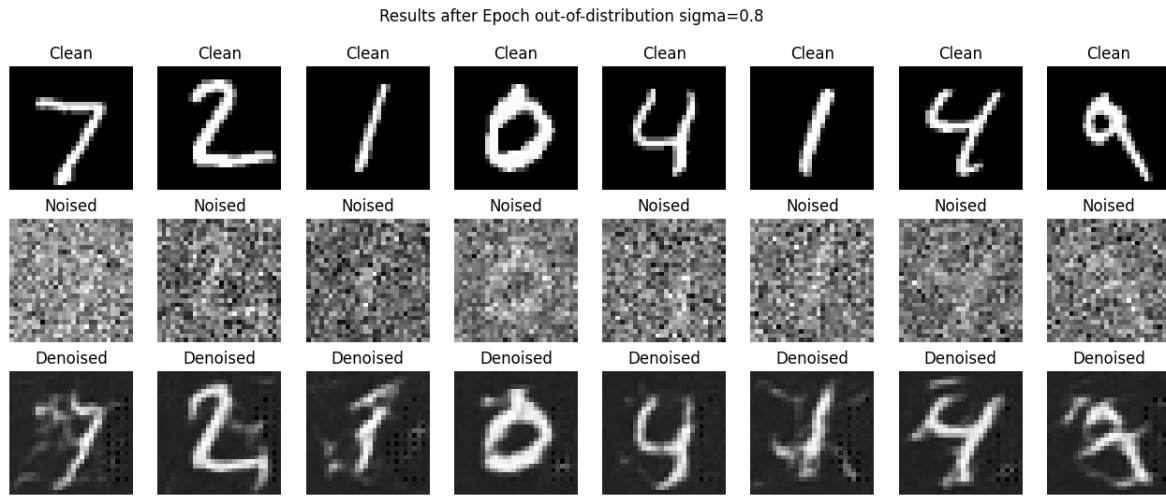


Results after Epoch out-of-distribution sigma=0.4



Results after Epoch out-of-distribution sigma=0.6





Part 2: Training a Diffusion Model

1. Adding Time Conditioning to UNet

Algorithm Description

1. Time Conditional U-Net

The U-Net serves as the backbone for predicting noise at each step of the diffusion process. The structure consists of:

- **Initial Convolution Block and Downsampling:**

$$x_1 = \text{ConvBlock}(x)$$

Downsampling is performed iteratively to reduce spatial resolution and extract hierarchical features.

- **Time-Conditional FCBlock:** The time step t is processed through a fully connected block to condition the network:

$$t_1 = \text{FCBlock}_1(t)$$

This mapping transforms t into a higher-dimensional representation.

- **Upsampling with Skip Connections:** During upsampling, lower-resolution features are concatenated with higher-resolution ones via skip connections:

$$x_{\text{up}} = \text{concat}(x_{\text{low_res}}, x_{\text{high_res}})$$

- **Output Transformation:** The output is adjusted via a final convolution to match the input dimensions.
-

2. Forward Diffusion Process

In the forward process, Gaussian noise is added progressively to the original image x_0 over T timesteps. The noise addition is defined as:

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

where:

- x_0 is the original image.
- $\epsilon \sim \mathcal{N}(0, I)$ is sampled Gaussian noise.
- $\bar{\alpha}_t = \prod_{i=1}^t \alpha_i$, and $\alpha_i = 1 - \beta_i$.

The noise schedule β_t controls the diffusion rate:

β_t =linear interpolation between β_1 and β_2

3. Reverse Diffusion Process

The reverse process aims to reconstruct x_0 from x_T by iteratively denoising. The U-Net predicts the noise $\epsilon_\theta(x_t, t)$, and x_{t-1} is computed as:

$$x_{t-1} = \frac{\sqrt{\bar{\alpha}_{t-1}\beta_t}}{1 - \bar{\alpha}_t} \hat{x}_0 + \frac{\sqrt{\bar{\alpha}_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t + \sqrt{\beta_t} z$$

4. Loss Function

The loss function minimizes the prediction error for noise:

$$\mathcal{L} = \mathbb{E}_{x_0, t, \epsilon} [\|\epsilon - \epsilon_\theta(x_t, t)\|^2]$$

- t is randomly sampled from $[1, T]$.
 - The model learns to predict ϵ , the noise added at timestep t .
-

Training and Sampling

Training Phase

1. At each iteration:
 - Sample $t \sim \mathcal{U}(1, T)$.
 - Add noise ϵ to the input x_0 to generate x_t .
 - Predict the noise $\epsilon_\theta(x_t, t)$ using the U-Net.
 - Minimize the loss \mathcal{L} .

Sampling Phase

1. Start with random noise $x_T \sim \mathcal{N}(0, I)$.
2. Iteratively compute x_{t-1} using the reverse process equation.
3. Continue until $t = 1$, producing the reconstructed x_0 .

Schedule and Hyperparameters

- **Noise Schedule:** A linear schedule is defined between β_1 and β_2 :

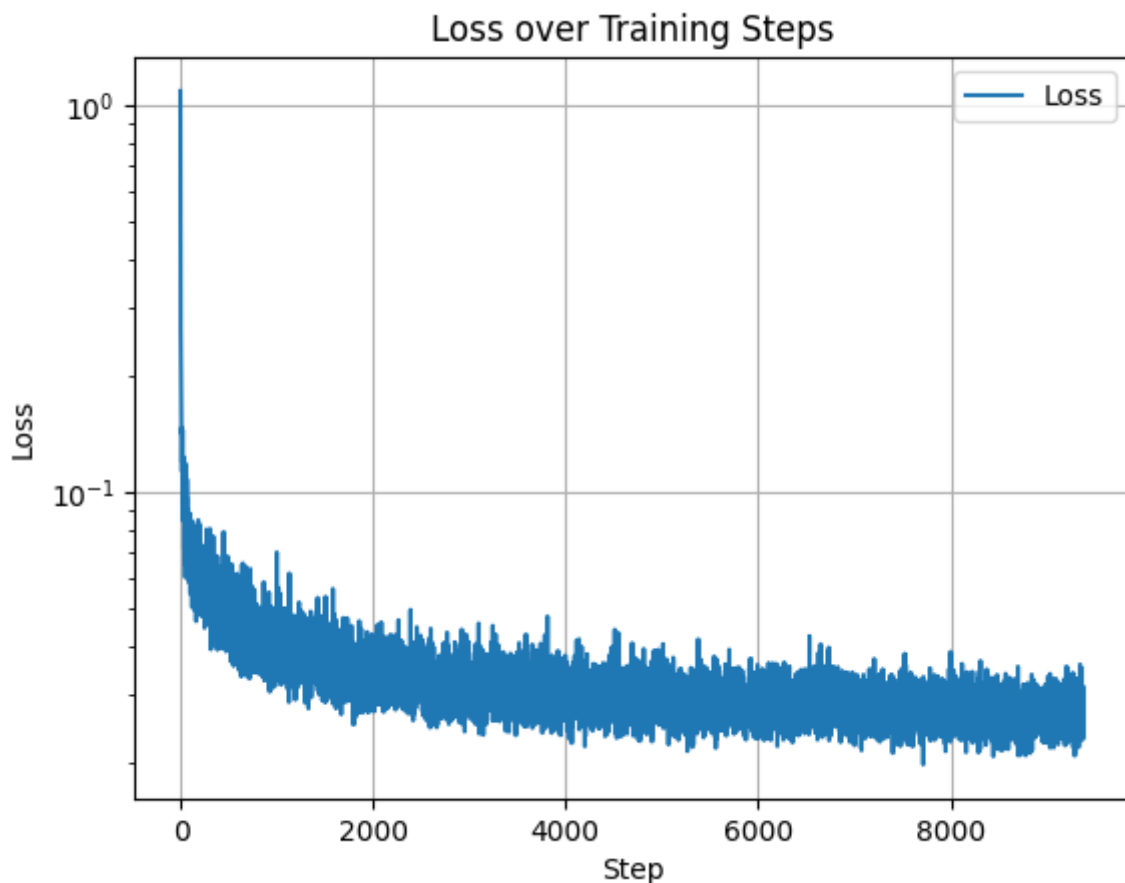
$$\beta_t = \beta_1 + t \cdot \frac{\beta_2 - \beta_1}{T}$$

- **Key Hyperparameters:**

- $T = 300$: Total diffusion steps.
 - $\beta_1 = 10^{-4}, \beta_2 = 0.02$: Noise schedule bounds.
 - Input size: $H = W = 28$, for MNIST images.
-

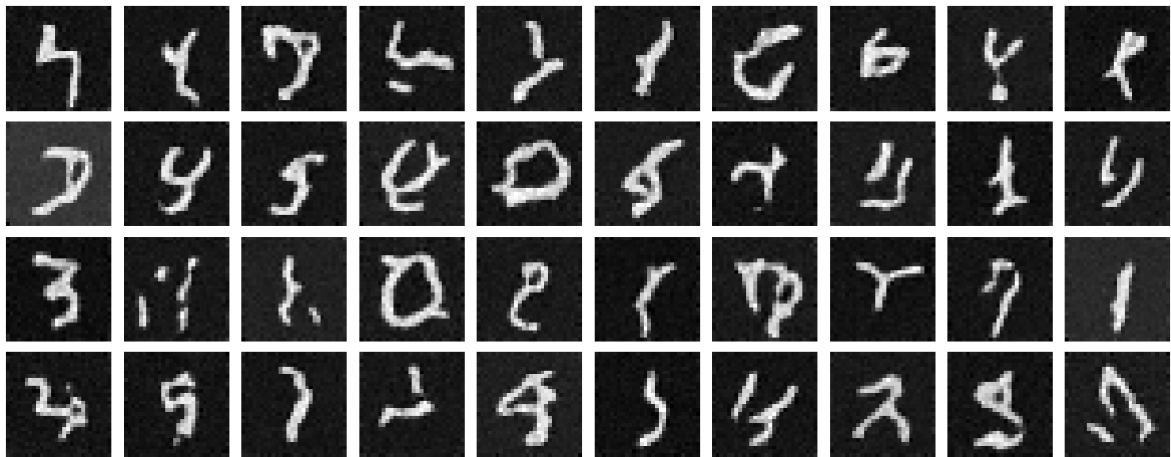
result:

Loss:

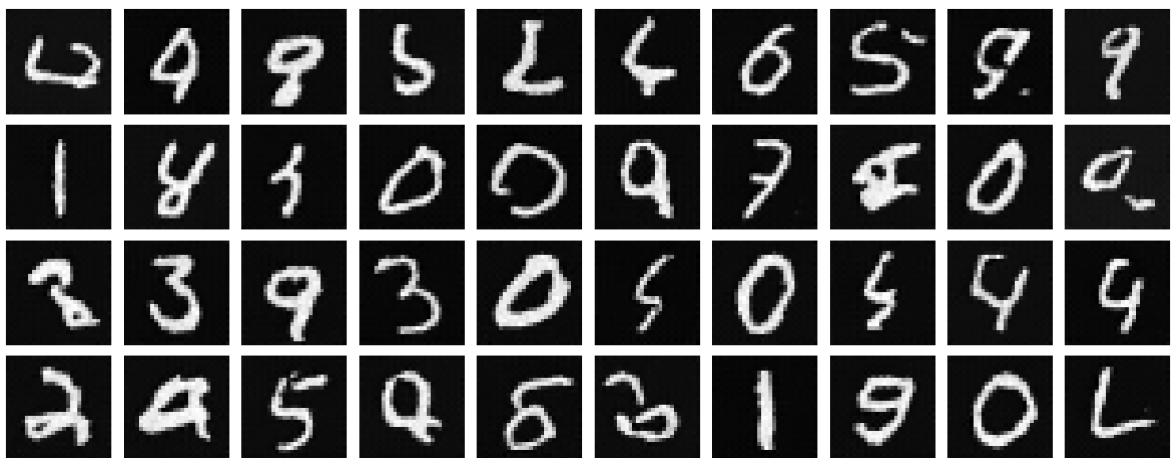


Sample:

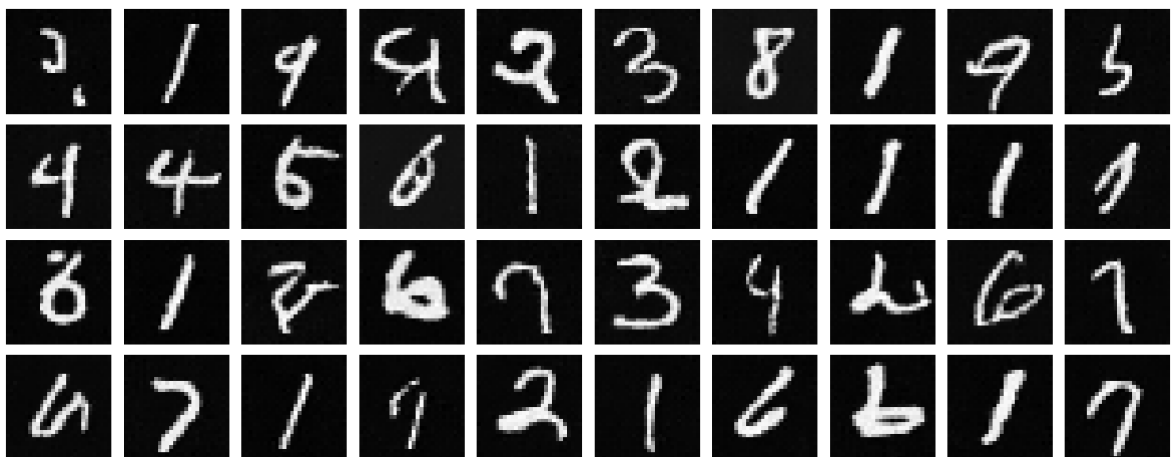
epoch = 1:



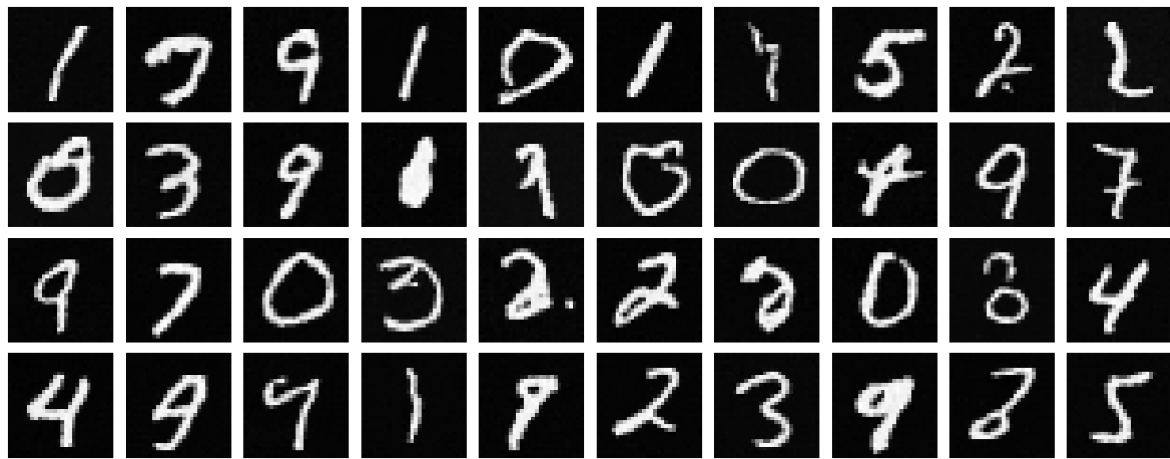
epoch = 5:



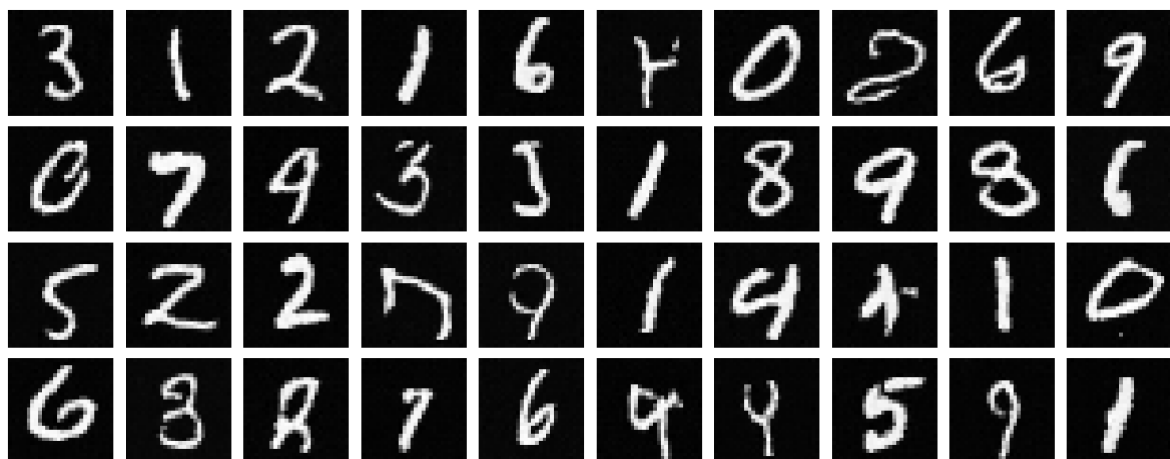
epoch = 10:



epoch = 15:



epoch = 20:



2. Adding Class-Conditioning to UNet

Key Features of the Algorithm

1. Class Conditional U-Net:

- The U-Net model predicts noise in the diffusion process while conditioning on both:
 - The timestep t , normalized to $[0, 1]$, which defines the progression of the diffusion process.
 - The class c , encoded as a one-hot vector, which incorporates semantic guidance into the generation.
- **Blocks of the U-Net:**
 - **Downsampling** extracts hierarchical features from the input image x .
 - **Flattening and Unflattening** adjust the feature dimensions for interaction with time and class conditions.
 - **Upsampling** reconstructs the image while combining features from lower resolutions through skip connections.
 - Time (ttt) and class (ccc) inputs are processed via fully connected blocks FCBlock and combined with intermediate U-Net features.

Training and Sampling

1. Training Phase:

- The model is trained with batches of images $x_{0 \times 0 \times 0}$ and their corresponding class labels ccc .

- Noise is added to x_0 to generate x_t , and the U-Net predicts $\epsilon_\theta(x_t, t, c)$.
- The loss function guides the model to correctly predict the noise.

2. Sampling Phase:

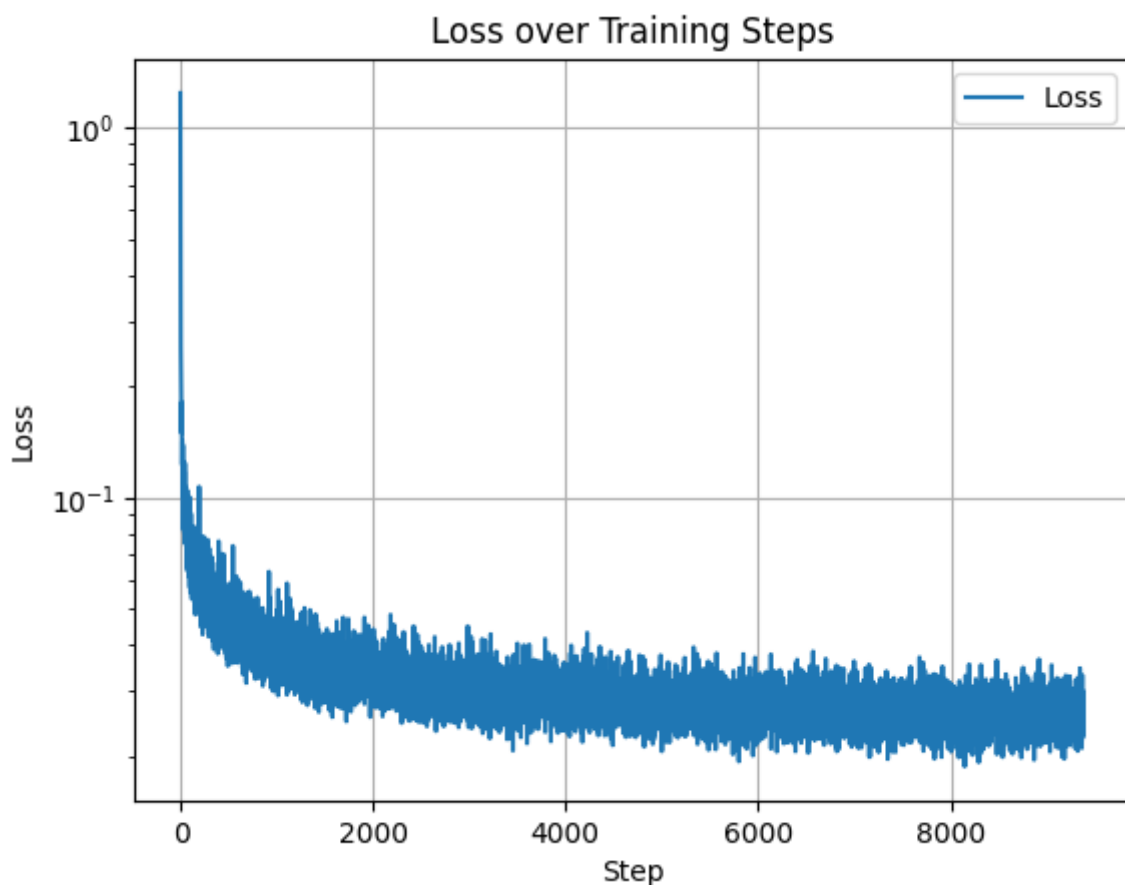
- Starts with random noise $x_T \sim \mathcal{N}(0, I)$.
- Class labels c are encoded as one-hot vectors to condition the sampling.
- Reverse diffusion iteratively refines x_t into the final image x_0 , applying classifier-free guidance for class-specific generation.

Applications

- **Class-Conditional Generation:** Enables generation of images corresponding to specific classes, such as MNIST digits.
- **Flexibility:** The model is adaptable to more complex datasets and larger architectures.

result:

Loss:

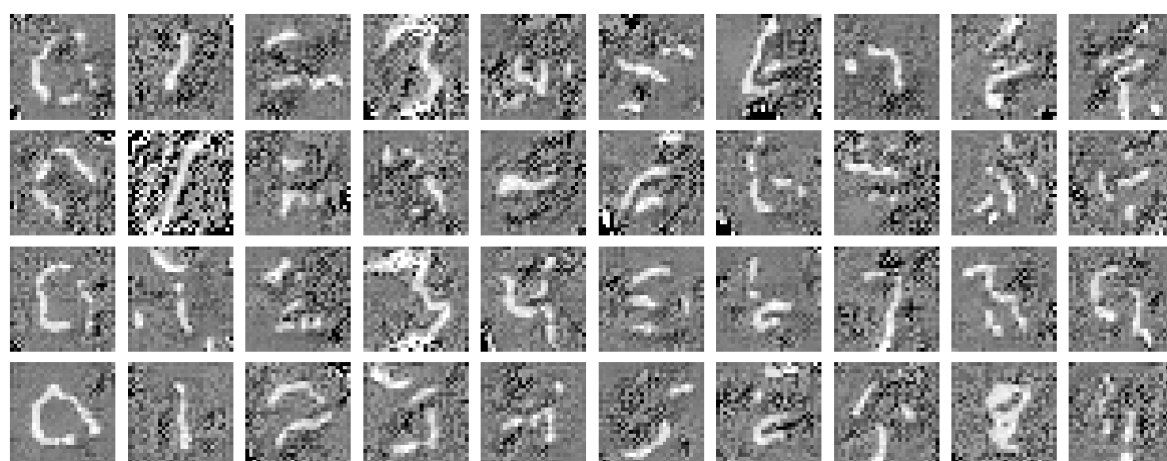


Sample:

epoch = 1:



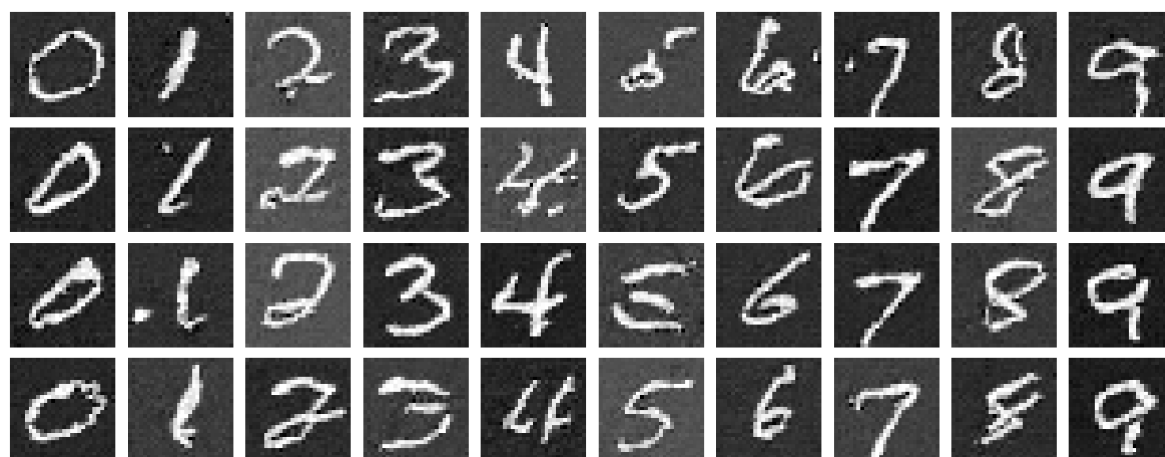
epoch = 5:



epoch = 10:



epoch = 15:



epoch = 20:

