Denoising Diffusion Probabilistic Models

Hoang Dung Vu Minh - Christopher Won

This version: December 18, 2025

1 Introduction

Denoising Diffusion Probabilistic Models (DDPMs) are a class of deep generative models—machine learning systems trained to synthesize new data that closely resembles their training distribution. Introduced by Sohl-Dickstein et al. (2015) and popularized by Ho, Jain, and Abbeel (2020), DDPMs have gained widespread attention for their stability during training and high-quality sample generation—often matching or surpassing GANs without adversarial optimization.

The method consists of two phases: (1) a forward diffusion process that progressively adds Gaussian noise to data over many timesteps until the signal becomes pure noise, and (2) a reverse process, learned by a neural network, that iteratively denoises random noise to generate new samples.

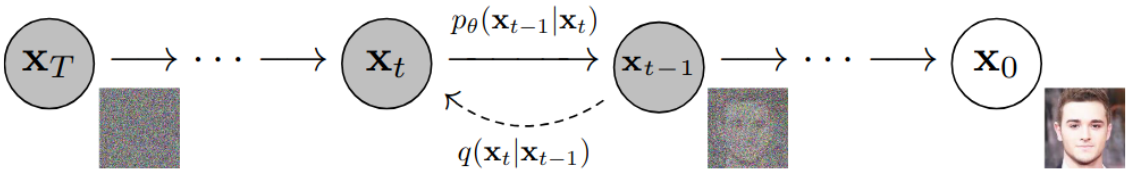

Figure 1: forward diffusion process and reverse process

Originally developed for image synthesis, DDPMs have since been applied across domains including audio generation, medical imaging, and time series modeling. In this work, we implement and evaluate DDPMs on two standard benchmarks: the MNIST dataset of handwritten digits and the CIFAR-10 dataset of natural images. Our experiments demonstrate the model's ability to generate diverse and coherent samples, highlighting both its strengths and challenges in terms of sample fidelity and computational cost.

2 Forward Diffusion

Given an image $\mathbf{x}_0 \in \mathbb{R}^d$, the *forward process* iteratively adds noise to create a sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$ via the recurrence for the measurable function f :

$$\mathbf{x}_t = \sqrt{1-\beta_t}\,\mathbf{x}_{t-1} + \sqrt{\beta_t}\,\boldsymbol{\epsilon}_t = f(\mathbf{x}_{t-1}, \boldsymbol{\epsilon}_t) \tag{1}$$

Here, each $\boldsymbol{\epsilon}_t \overset{i.i.d}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, and $\beta_t \in (0,1)$ is a pre-specified scalar called the variance of the added noise in the update at step t. The collection $\{\beta_t\}_{t=1}^T$ constitutes the *variance schedule*, which is chosen in advance as a hyperparameter of the model. For example, a common choice is the linearly

increasing values from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$ are used. Since $\{\beta_t\}_{t=1}^T$ here is increasing, more noise is added as the forward process evolves. At time zero we have an image and at time T we effectively have pure Gaussian noise.

Formally, given a data distribution $\mathbf{x}_0 \in \mathbb{R}^d \sim q(\mathbf{x}_0)$, the forward Markov process generates a sequence of random variables $\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_T$ with transition kernel $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$. The joint distribution of $\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_T$ conditioned on $\mathbf{x}_0$, denoted as $q(\mathbf{x}_1, \dots, \mathbf{x}_T \mid \mathbf{x}_0)$ is

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) := q(\mathbf{x}_1, \dots, \mathbf{x}_T \mid \mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \tag{2}$$

The forward process defines a discrete time Markov chain on an uncountable state space, and the associated transition density may be written

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\,\mathbf{x}_{t-1}, \beta_t \mathbf{I}). \tag{3}$$

$\mathcal{N}$ denotes a Gaussian distribution and we have parameterized the distribution with mean $\boldsymbol{\mu} = \sqrt{1-\beta_t}\,\mathbf{x}_{t-1}$ and covariance matrix $\boldsymbol{\Sigma} = \beta_t \mathbf{I}$. The noise variance β_t typically starts close to 0 for small t, meaning that the variance of the conditional distribution in Equation (3) is small, while $\sqrt{1-\beta_t}$ is close to 1, and thus the conditional mean is close to $\mathbf{x}_{t-1}$, the sample at the previous step.

The Gaussian transition kernel allows us to obtain the analytical form of $q(\mathbf{x}_t \mid \mathbf{x}_0)$ for all $t \in \{0, 1, \cdots, T\}$. Specifically, denoting $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=0}^{t} \alpha_s$, we have

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{4}$$

Given $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, we can easily obtain a sample of $\mathbf{x}_t$ by sampling a Gaussian vector $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and applying the transformation yields

$$\mathbf{x}_t \overset{(d)}{=} \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}^{[1]} \tag{5}$$

When $\bar{\alpha}_T \approx 0$, $\mathbf{x}_T$ is almost Gaussian in distribution, so we have $q(\mathbf{x}_T) := \int q(\mathbf{x}_T \mid \mathbf{x}_0) q(\mathbf{x}_0) d\mathbf{x}_0 \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$. It also follows that

$$q(\mathbf{x}_T) \approx p_{\text{prior}}(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}),$$

with $q(\mathbf{x}_T) = p_{\text{prior}}(\mathbf{x}_T)$ in the limit as $T \to \infty$. Here, p_{prior} denotes the *prior distribution* over the final latent variable $\mathbf{x}_T$, which is conventionally chosen to be a standard isotropic Gaussian.

Intuitively, the forward process gradually injects Gaussian noise into the data, progressively eroding its structure until, at the final step, only unstructured noise remains. For generating new data samples, DDPMs start by first generating an unstructured noise vector from the prior distribution (which is typically trivial to obtain), then gradually remove noise therein by running a learnable Markov chain in the reverse.

3 Reverse Process

Ideally, we would compute the reverse transition $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ and sample backward from $\mathbf{x}_T \sim p_{\text{prior}}$ to obtain $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, where $q(\mathbf{x}_0)$ is the (complex, unknown) data distribution. However, applying Bayes' rule,

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})\, q(\mathbf{x}_{t-1})}{q(\mathbf{x}_t)} = \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \int q(\mathbf{x}_{t-1} \mid \mathbf{x}_0) q(\mathbf{x}_0)\, d\mathbf{x}_0}{\int q(\mathbf{x}_t \mid \mathbf{x}_0) q(\mathbf{x}_0)\, d\mathbf{x}_0},$$

[1] (d) denotes equals in disitribution.

requires knowledge of $q(\mathbf{x}_0)$ (often denoted p_{complex})—which we do not have—and involves intractable integrals over the data distribution. Although the forward process only for $t > 0$ is Gaussian conditional on $\mathbf{x}_0$, the marginal $q(\mathbf{x}_0)$ is arbitrary and non-Gaussian. Consequently, $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ has no closed-form expression, and we must instead learn an approximation.

The idea is instead to find a suitable approximation to $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ and let that approximation define the reverse process. We construct a stepwise denoising process $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ that approximates the true reverse distribution $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$. Importantly, the reverse process of a Gaussian diffusion is itself Gaussian, so each reverse transition can be parameterized as a normal distribution. This means we only need to estimate its mean and variance.

We let a neural network with parameters θ, a parametric approximation of $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$, to predict these distributional parameters as functions of the current noisy sample $\mathbf{x}_t$ and the time step t, yielding

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\big(\mathbf{x}_{t-1};\, \mu_\theta(\mathbf{x}_t, t),\, \Sigma_\theta(\mathbf{x}_t, t)\big). \tag{6}$$

The inclusion of t as an input is essential: because the forward process uses a time-dependent variance schedule $\{\beta_t\}_{t=1}^T$, the amount and structure of noise present in $\mathbf{x}_t$ vary with t. The network must therefore adapt its denoising behavior accordingly at each step.

Like the forward process, the reverse process is a Markov chain, and we can write the joint probability of a sequence of samples as a product of conditionals and the marginal probability of $\mathbf{x}_T$,

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t). \tag{7}$$

Here, the distribution $p(\mathbf{x}_T)$ is the same as $q(\mathbf{x}_T)$, namely the pure noise distribution $p_{\text{prior}} = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$. This means that the generation process starts with Gaussian noise, followed by sampling from the learned individual steps of the reverse process.

With this reverse Markov chain, we can generate a data sample $\mathbf{x}_0$ by first sampling a noise vector $\mathbf{x}_T \sim p(\mathbf{x}_T)$, then iteratively sampling from the learnable transition kernel $\mathbf{x}_{t-1} \sim p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ until $t = 1$.

Key to the success of this sampling process is training the reverse Markov chain to match the actual time reversal of the forward Markov chain. That is, we have to adjust the parameter θ so that the joint distribution of the reverse Markov chain $p_\theta(\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_T) := p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ closely approximates that of the forward process $q(\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_T) := q(\mathbf{x}_0) \prod_{t=1}^T q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$. This is achieved by minimizing the Kullback-Leibler (KL) divergence between these two.

4 Methodology

4.1 DDPM's ELBO

We start off with a type of statistical distance: a measure of how much an approximating probability measures $\mathbb{Q}$ is different from a true probability measures $\mathbb{P}$.

Definition 4.1 (KL divergence). *Let $(\mathcal{X}, \mathcal{F})$ be a measurable space and $\mathbb{P}$ and $\mathbb{Q}$ are its probability measures. The Kullbach-Leibler (KL) divergence $\mathcal{D}_{KL}$ measures the distance between two probability measures defined by*

$$\mathcal{D}_{KL}\left(\mathbb{Q} \parallel \mathbb{P}\right) = \begin{cases} \int \log \frac{\mathbb{Q}(d\omega)}{\mathbb{P}(d\omega)} \mathbb{Q}(d\omega), & \mathbb{Q} \ll \mathbb{P} \\ \infty, & \textit{otherwise} \end{cases}$$

If both $\mathbb{P}$ and $\mathbb{Q}$ are both absolutely continuous with respect to the Lebesgue measure μ, then there exist densities p and q respectively such that

$$D_{KL}(\mathbb{Q} \parallel \mathbb{P}) = \int_{x \in \mathcal{X}} q(x) \log \frac{q(x)}{p(x)} \, \mu(dx), \quad \textit{when } \mathbb{Q} \ll \mathbb{P}$$

In Generative Modeling, we want to learn a model $p_\theta(\mathbf{x}_0)$ that approximates the real data complex distribution $q(\mathbf{x}_0)$, so that we can generate new samples of data. To do this, consider their respective KL divergence, which requires $q \ll p_\theta$:

$$\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_0) \mid p_\theta(\mathbf{x}_0)\right) = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_0)}{p_\theta(\mathbf{x}_0)}\right] = -\mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0)\right] + \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0)}\left[\log q(\mathbf{x}_0)\right]$$

Since only the first term of the expectation depends on θ, we observe that

$$\operatorname*{argmin}_\theta \mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_0) \mid p_\theta(\mathbf{x}_0)\right) = \operatorname*{argmax}_\theta \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0)\right]$$

This shows minizing the KL-divergence is equivalent to performing the maximum likelihood estimation (MLE):

$$\theta^* = \arg\max_\theta \mathbb{E}_{q(\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0)\right]$$

However, the marginal likelihood $p_\theta(\mathbf{x}_0)$ is defined as:

$$p_\theta(\mathbf{x}_0) = \int_{\mathbf{x}_{1:T}} p_\theta(\mathbf{x}_{0:T}) \, d\mathbf{x}_{1:T}.$$

where this integral is intractable because the latent trajectory $\mathbf{x}_{1:T}$ is high-dimensional and the reverse process $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is a neural network — so no closed-form integration. As a result we cannot compute $\log p_\theta(\mathbf{x}_0)$ directly. Instead, we can obtain a lower bound for it according to Theorem 4.1.

A central insight in DDPMs that resolves this intractability is to condition the reverse transition on the clean data sample $\mathbf{x}_0$. This conditioning renders the forward process posterior $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ analytically tractable, allowing the Kullback–Leibler (KL) divergence between the approximate reverse process $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ and the true posterior to be computed and minimized directly.

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0) q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} \overset{MP}{=} \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} \tag{8}$$

This tractability arises from two key properties of the forward process: its Markov property (MP), meaning $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0) = q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$, and the Gaussian nature of all involved distributions. As a result, $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ itself is Gaussian and admits a closed-form expression.

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0), \, \tilde{\beta}_t \mathbf{I})$$

where

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t, \quad \tilde{\beta}_t = \frac{(1 - \bar{\alpha}_{t-1})\beta_t}{1 - \bar{\alpha}_t}. \tag{9}$$

In addition, Eq. (8) also yields

$$\frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} = \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} \tag{10}$$

Theorem 4.1 (Evidence lower bound (ELBO) on log likelihood of DDPM).

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_0)\right]$$
$$\leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\mathcal{D}_{KL}\left(q(\mathbf{x}_T \mid \mathbf{x}_0)\|p(\mathbf{x}_T)\right) + \sum_{t=2}^{T}\mathcal{D}_{KL}\left(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\right) - \log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\right]$$

Proof. See Appendix 8.1.2 □

4.2 Evaluate the KL Divergence Analytically

We aim to compute

$$\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\right)$$

under the Gaussian assumptions of DDPM.

Under the assumption that

- True posterior:

  $$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0),\, \tilde{\beta}_t\mathbf{I})$$

  where

  $$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t, \quad \tilde{\beta}_t = \frac{(1-\bar{\alpha}_{t-1})\beta_t}{1-\bar{\alpha}_t}. \tag{11}$$

- Learned reverse process:

  $$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t),\, \sigma_t^2\mathbf{I}) \tag{12}$$

  Here, we set we set $\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2\mathbf{I}$ to untrained time dependent constant.

Definition 4.2. *For $q = \mathcal{N}(\mu_q, \Sigma_q)$ and $p = \mathcal{N}(\mu_p, \Sigma_p)$ in $\mathbb{R}^d$, we have the KL divergence between two Gaussian distributions as*

$$\mathcal{D}_{\mathrm{KL}}(q\|p) = \frac{1}{2}\left[\mathrm{tr}(\Sigma_p^{-1}\Sigma_q) + (\mu_p-\mu_q)^\top\Sigma_p^{-1}(\mu_p-\mu_q) - d + \log\frac{\det\Sigma_p}{\det\Sigma_q}\right].$$

After some manipulation we obtain the intermediate KL divergence

$$\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\right) = \underbrace{\frac{1}{2\sigma_t^2}\|\mu_\theta(\mathbf{x}_t, t) - \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)\|^2}_{\text{depends on } \theta} + \underbrace{\frac{d}{2}\left(\frac{\tilde{\beta}_t}{\sigma_t^2} - 1 + \log\frac{\sigma_t^2}{\tilde{\beta}_t}\right)}_{C,\ \text{constant w.r.t. } \theta} \tag{13}$$

4.3 Further simplifcation

Under the reparameterization $\mathbf{x}_t \overset{(d)}{=} \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$, we have

$$\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}\right)$$

Plugging in the true posterior mean in Eq. (11)

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}\right).$$

where ϵ_θ is the neural network predicted noise. We thus can parameterize the neural network model mean as

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right).$$

Therefore in Eq. (12) to sample $\mathbf{x}_{t-1} \sim p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t),\, \sigma_t^2\mathbf{I})$ is to compute

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$$

Then

$$\|\mu_\theta(\mathbf{x}_t, t) - \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)\|^2 = \frac{\beta_t^2}{\alpha_t(1-\bar{\alpha}_t)}\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\|^2.$$

Hence the intermediate KL divergence in Eq. (13) is simplified to

$$\mathcal{D}_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\big) = \frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2 + C$$

which is the standard DDPM training objective.

Then

$$\begin{aligned}
&\mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\Big[D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\big)\Big] \\
&= \mathbb{E}_{q(\mathbf{x}_0)}\left[\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\Big[D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\big)\Big]\right] \\
&= \mathbb{E}_{q(\mathbf{x}_0)}\left[\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right]\right] \\
&= \mathbb{E}_{q(\mathbf{x}_0)}\left[\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, t)\|^2\right]\right] \\
&= \mathbb{E}_{\mathbf{x}_0\sim q(\mathbf{x}_0)}\left[\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,\mathbf{I})}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, t)\|^2\right]\right]
\end{aligned}$$

where the last equality follows from Eq. (4) as there is a bijection (one-to-one mapping) between a draw $\mathbf{x}_t$ from $q(\mathbf{x}_t \mid \mathbf{x}_0)$, and a draw $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$ via the transformation in Eq. (5). Since this mapping is invertible, the two probability measures are identical.

To simplify the training loss, the model is trained with the simple loss function to minimize:

$$L_{\mathrm{simple}}(\theta) := \mathbb{E}_{t\sim\mathcal{U}[1,T],\mathbf{x}_0\sim q(\mathbf{x}_0),\boldsymbol{\epsilon}\sim\mathcal{N}(0,\mathbf{I})}\left[\left\|\epsilon - \boldsymbol{\epsilon}_\theta\Big(\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon},\, t\Big)\right\|^2\right] \tag{14}$$

To summarize, we can train the reverse process mean function approximator $\mu_\theta(\mathbf{x}_t, t)$ to predict $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$, or by modifying its parameterization, we can train it to predict $\boldsymbol{\epsilon}$ from $\boldsymbol{\epsilon}_\theta$.

5 Results

5.1 Results

We fix the forward process variances $\{\beta_t\}_{t=1}^T$ to constants that increase linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$ across all models in our experiments. The total number of timesteps T is varied between

the MNIST and CIFAR-10 datasets to study how the diffusion dynamics behave under different levels of noise scheduling. We used RTX 5090 GPU for all experiments.

The reverse (denoising) process is modeled using a U-Net architecture. We train all models with the Adam optimizer using a constant learning rate of 10^{-3}. It is important to note that in Figures 3, 5, 9, 11, and 13, the odd-numbered columns display reference (real) samples, while the even-numbered columns show corresponding generated samples.

5.2 MNIST dataset

5.2.1 Training Configuration

Both **Diffusion_model_1** and **Diffusion_model_2** were trained under identical hyperparameter conditions to ensure a fair architectural comparison:

- Batch size: 128
- Epochs: 50
- Timesteps: 500

5.2.2 Model Performance Comparison

The quality of generated samples differed significantly between the two models due to architectural complexity. **Diffusion_model_1**, which utilized a basic U-Net structure, failed to adequately capture the distribution of grayscale pixels in the 32×32 image grid, resulting in lower-quality outputs.

In contrast, **Diffusion_model_2** produced samples of notably higher quality—both clearer and more diverse. This improvement is attributed to its enhanced U-Net architecture, which incorporated:

- An additional encoder/decoder block
- A self-attention mechanism
- Transformer-based time embeddings

These additions improved the model's ability to capture both local details and global structural information.

To further analyze training behavior on **Diffusion_model_2**, we examined the average training and validation loss across epochs. The loss curves highlight the value of implementing an early-stopping mechanism (see Section 9) to prevent overfitting and illustrate the model's convergence rate. Additionally, we visualize the progressive denoising process at selected timesteps: $t = 0, 5, 10, 50, 100, 200, 499$, starting from initial Gaussian noise. By approximately $t = 100$, the generated images reveal recognizable structural outlines.

5.2.3 Architectural Insights

Training multiple U-Net variants on the MNIST dataset revealed that simply increasing model depth did not guarantee better performance. Instead, **Diffusion_model_2** demonstrated that a carefully balanced architecture—tailored to the dataset's characteristics—yielded the most promising results. This underscores the importance of architectural appropriateness over mere complexity in diffusion-based generative modeling.

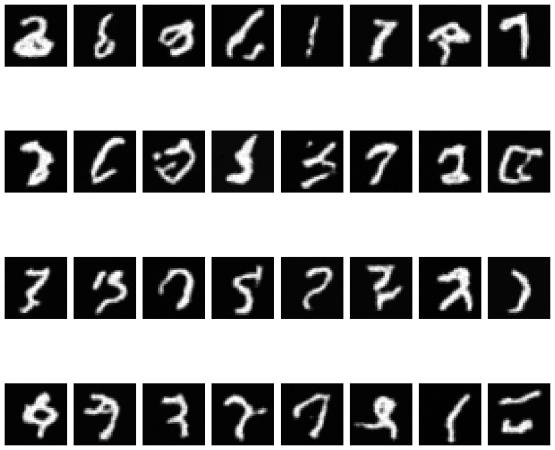

Figure 2: Diffusion_model_1 generated samples with $T = 500$.

Figure 3: Diffusion_model_1 comparison with real MNIST samples

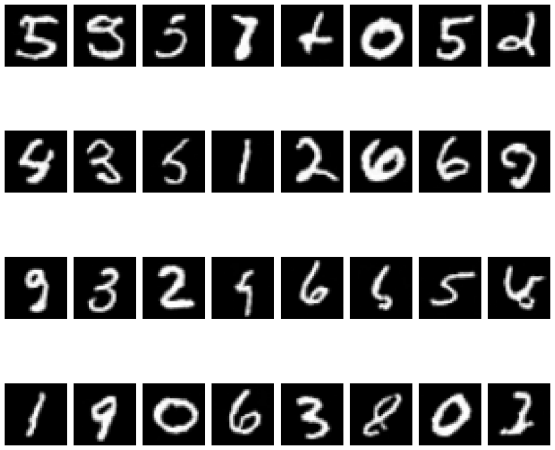

Figure 4: Diffusion_model_2 generated sample with $T = 500$.

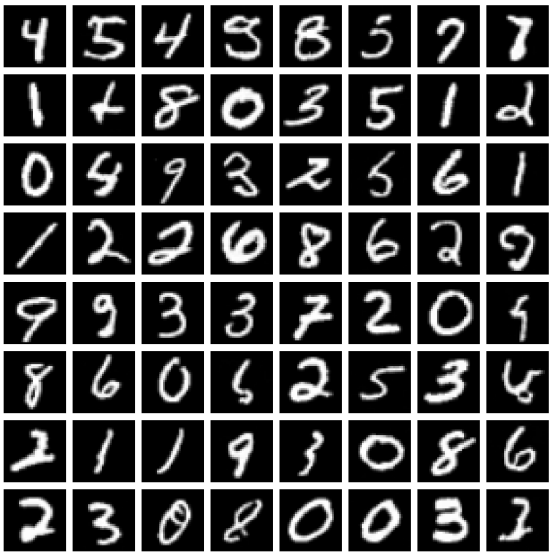

Figure 5: Diffusion_model_2 comparison with real MNIST samples

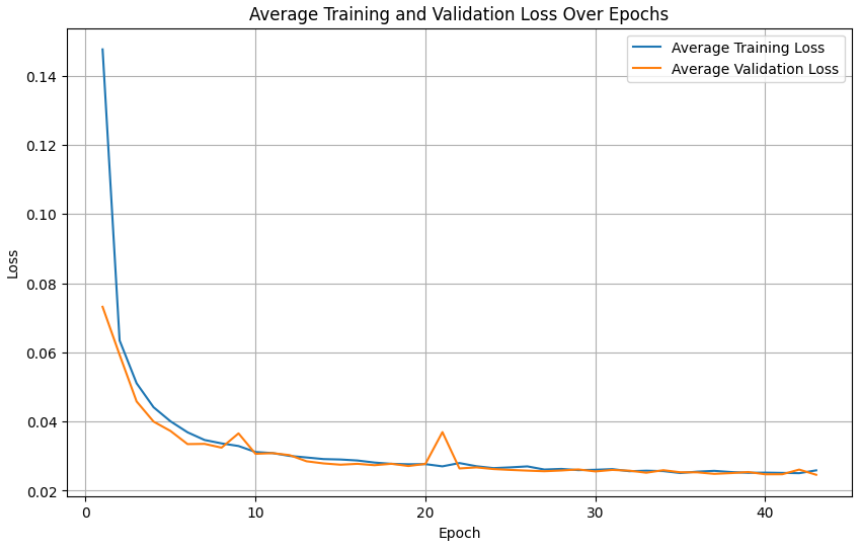

Figure 6: Diffusion_model_2 Loss over Epochs

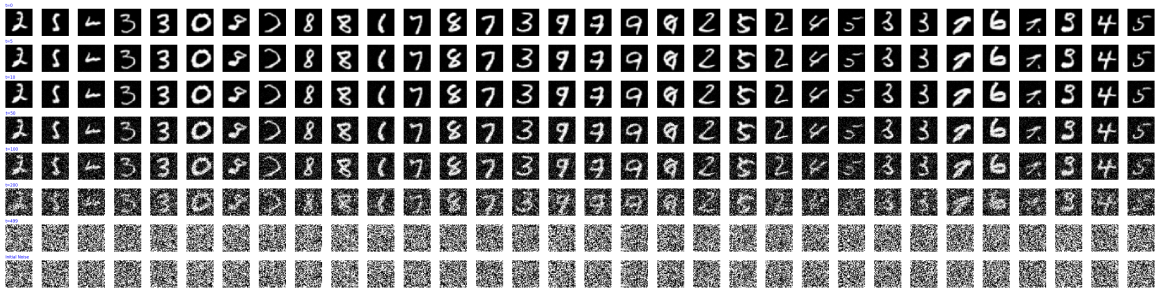

Figure 7: Diffusion_model_2 Progressive Generation

5.3 CIFAR-10 dataset

5.3.1 Training Configuration

CIFAR-10_diffusion_model_1 was trained with the following hyperparameters:

- Batch size: 128
- Epochs: 60
- Timesteps: 500

CIFAR-10_diffusion_model_2 and **CIFAR-10_diffusion_model_3** were trained under identical conditions for fair comparison:

- Batch size: 128
- Epochs: 60
- Timesteps: 1000

5.3.2 Architectural Variations and Outcomes

CIFAR-10_diffusion_model_1 adopted the same U-Net architecture as **MNIST_diffusion_model_2**, modified only for RGB input channels. This architecture proved insufficient for capturing the complexity of color images, resulting in generated samples that resembled pure Gaussian noise across RGB channels.

In response, **CIFAR-10_diffusion_model_2** introduced two architectural enhancements:

- Group Normalization in each encoder/decoder block
- Additional self-attention layers in the encoder pathway

These modifications yielded promising improvements, including clearer edge generation and increased overall brightness, indicating better capture of both local and structural image features.

CIFAR-10_diffusion_model_3 achieved the strongest qualitative alignment with reference CIFAR-10 samples. While individual images retain an abstract quality, this model demonstrates marked improvements in clarity, fine detail, and a more coherent and diverse color palette and edge structure compared to prior iterations due to further additions to the network (see more in Section **Implementation**).

5.3.3 Discussion

The progression from Model 1 to Model 3 illustrates the necessity of architectural scaling and normalization techniques when transitioning from grayscale to color image generation. Increased timesteps (from 500 to 1000), combined with enhanced normalization, attention mechanisms, and a greater number of model parameters, were critical in achieving perceptible improvements in sample quality.

Figure 8: CIFAR-10_diffusion_model_1 generated samples with $T = 500$.

Figure 9: Real vs Generated samples comparison from Figure 7

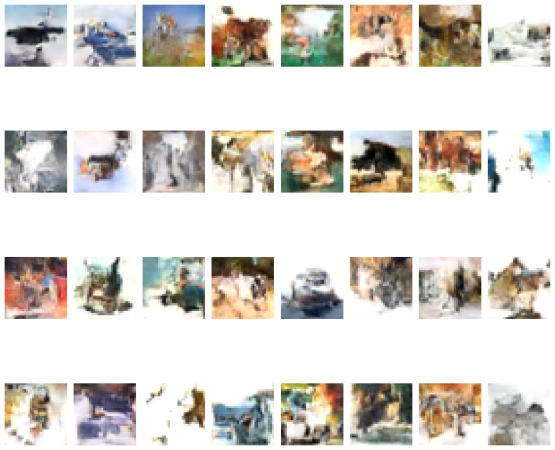

Figure 10: CIFAR-10_diffusion_model_2 generated samples with $T = 1000$.

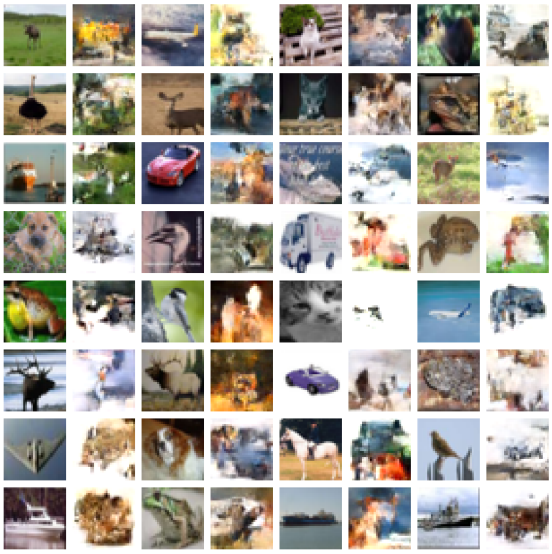

Figure 11: Real vs Generated samples comparison from Figure 9

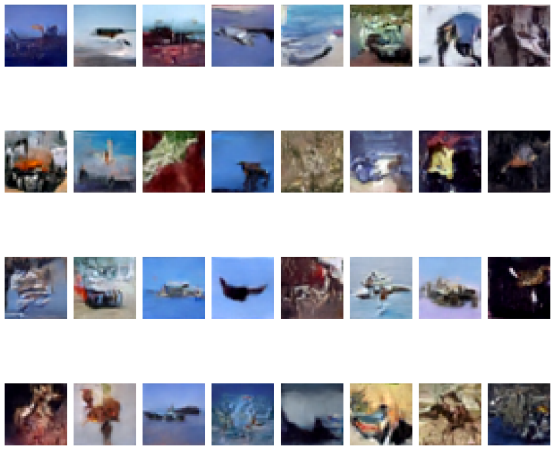

Figure 12: CIFAR-10_diffusion_model_3 generated samples with $T = 1000$.

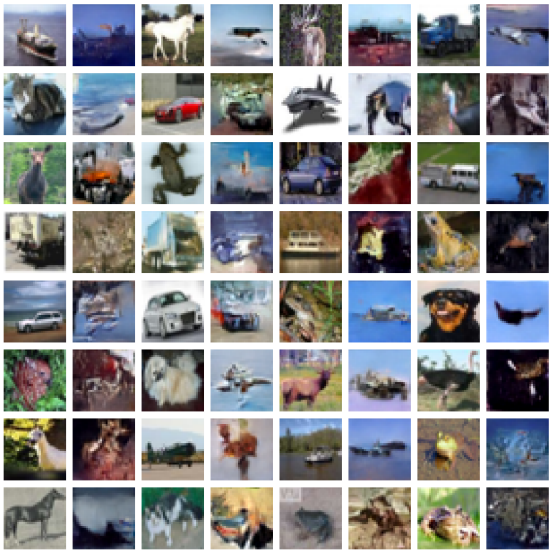

Figure 13: Real vs Generated samples comparison from Figure 11

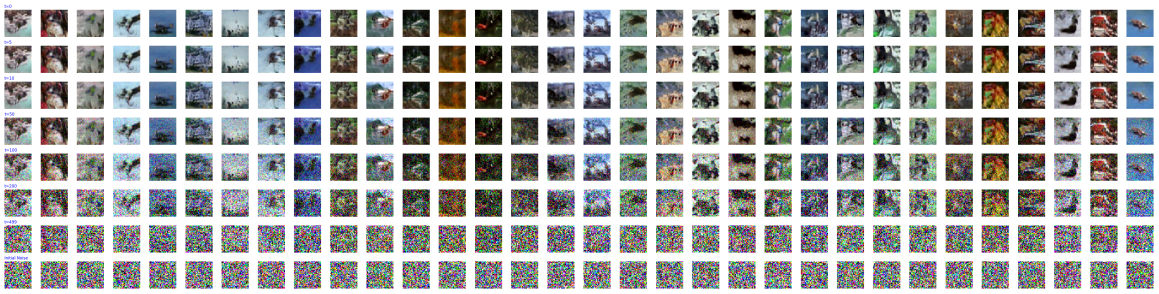

Figure 14: CIFAR-10_diffusion_model_3 Progressive Generation

6 Conclusion

In this work, we implemented and analyzed Denoising Diffusion Probabilistic Models (DDPMs) with a linear noise schedule and a U-Net backbone. Experiments on MNIST and CIFAR-10 demonstrate that the model successfully learns to generate coherent samples, with performance influenced by the number of diffusion steps and architectural choices. The results confirm the importance of the forward process design and highlight the U-Net's effectiveness in modeling the reverse denoising dynamics.

7 References

Jonathan Ho, Ajay Jain, and Pieter Abbeel. (2020). Denoising Diffusion Probabilistic Models. arXiv preprint arXiv:2006.11239

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. (2015). Deep Unsupervised Learning using Nonequilibrium Thermodynamics. arXiv preprint arXiv:1503.03585.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998–6008

Kapila, N., & collaborators. (2024). CNNtention: Can CNNs do better with Attention? arXiv preprint arXiv:2412.11657

APXML. (n.d.). U-Net attention mechanisms. In Advanced diffusion architectures: Chapter 2 – Advanced U-Net architectures.

Wang, Y., Chen, Y., Liu, X., & Zhao, L. (2024). Development of skip connection in deep neural networks for computer vision and medical image analysis: A survey. arXiv preprint arXiv:2405.01725

8 Appendix

8.1 Proof of Theorem 4.1

8.1.1 Expectation of a function of the subset of variables

Lemma 8.1. *The expectation of a function of the subset of variables* $\mathbf{x}_{a:b}$ *with* $1 \leq a \leq b \leq T$*, say* $f(\mathbf{x}_{a:b})$ *w.r.t. a distribution* $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$*, is given by taking the expectation only over the variables* $\mathbf{x}_{a:b}$*.*

$$\mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}[f(\mathbf{X}_{a:b})] = \mathbb{E}_{q(\mathbf{x}_{a:b}|\mathbf{x}_0)}[f(\mathbf{X}_{a:b})]$$

Proof. By definition we integrate the function times the density over all variables.

$$\begin{aligned}
\mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}[f(\mathbf{X}_{a:b})] &= \int_{\mathbf{x}_{1:T}} q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \cdot f(\mathbf{x}_{a:b})\, d\mathbf{x}_{1:T} \\
&= \int_{\mathbf{x}_{1:T}} q(\mathbf{x}_{a:b} \mid \mathbf{x}_0) \cdot q(\mathbf{x}_{1:a-1,b+1:T} \mid \mathbf{x}_0, \mathbf{x}_{a:b}) \cdot f(\mathbf{x}_{a:b})\, d\mathbf{x}_{1:T} \\
&= \int_{\mathbf{x}_{a:b}} q(\mathbf{x}_{a:b} \mid \mathbf{x}_0) f(\mathbf{x}_{a:b}) \left[\int_{\mathbf{x}_{1:a-1,b+1:T}} q(\mathbf{x}_{1:a-1,b+1:T} \mid \mathbf{x}_0, \mathbf{x}_{a:b})\, d\mathbf{x}_{1:a-1,b+1:T} \right] d\mathbf{x}_{a:b} \\
&= \int_{\mathbf{x}_{a:b}} q(\mathbf{x}_{a:b} \mid \mathbf{x}_0) f(\mathbf{x}_{a:b})\, d\mathbf{x}_{a:b} \\
&= \mathbb{E}_{q(\mathbf{x}_{a:b}|\mathbf{x}_0)}[f(\mathbf{X}_{a:b})]
\end{aligned}$$

Where we have used the following result in second equality. For any collection of random variables A and B, and conditioning event C, we have:

$$\mathbb{P}(A, B \mid C) = \mathbb{P}(A \mid C) \cdot \mathbb{P}(B \mid A, C)$$

with $A = \mathbf{x}_{a:b}$,$B = \mathbf{x}_{1:a-1,b+1:T}$, $C = \mathbf{x}_0$. □

8.1.2 Main proof

Proof. Training is performed by optimizing the usual variational bound on log likelihood

$$
\begin{aligned}
&\log p_{\boldsymbol{\theta}}(\mathbf{x}_0) \\
&= \log \int_{\mathbf{x}_{1:T}} p_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T} \\
&= \log \int_{\mathbf{x}_{1:T}} \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)} q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) d\mathbf{x}_{1:T} \\
&= \log \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)} \right] \\
&\geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)} \right] && \text{Jensen's inequality} \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{\prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right] && \text{Use Eqs (2) and (7)} \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log p(\mathbf{x}_T) + \sum_{t=1}^{T} \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log p(\mathbf{x}_T) + \sum_{t=2}^{T} \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} + \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_0 \mid \mathbf{x}_1)}{q(\mathbf{x}_1 \mid \mathbf{x}_0)} \right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log p(\mathbf{x}_T) + \sum_{t=2}^{T} \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} \cdot \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} + \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_0 \mid \mathbf{x}_1)}{q(\mathbf{x}_1 \mid \mathbf{x}_0)} \right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log p(\mathbf{x}_T) + \sum_{t=2}^{T} \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} \cdot \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} + \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_0 \mid \mathbf{x}_1)}{q(\mathbf{x}_1 \mid \mathbf{x}_0)} \right] && \text{Use Eq. (10)} \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log p(\mathbf{x}_T) + \sum_{t=2}^{T} \left(\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} + \log \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} \right) + \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_0 \mid \mathbf{x}_1)}{q(\mathbf{x}_1 \mid \mathbf{x}_0)} \right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log p(\mathbf{x}_T) + \sum_{t=2}^{T} \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} + \big(\log q(\mathbf{x}_1 \mid \mathbf{x}_0) - \log q(\mathbf{x}_T \mid \mathbf{x}_0) \big) + \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_0 \mid \mathbf{x}_1)}{q(\mathbf{x}_1 \mid \mathbf{x}_0)} \right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log p(\mathbf{x}_T) + \sum_{t=2}^{T} \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} - \log q(\mathbf{x}_T \mid \mathbf{x}_0) + \log p_{\boldsymbol{\theta}}(\mathbf{x}_0 \mid \mathbf{x}_1) \right] \\
&= \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T \mid \mathbf{x}_0)} + \sum_{t=2}^{T} \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} + \log p_{\boldsymbol{\theta}}(\mathbf{x}_0 \mid \mathbf{x}_1) \right] \\
&= \mathbb{E}_{q(\mathbf{x}_T|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T \mid \mathbf{x}_0)} \right] + \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_{t-1}, \mathbf{x}_t|\mathbf{x}_0)} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} \right] + \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} \log p_{\boldsymbol{\theta}}(\mathbf{x}_0 \mid \mathbf{x}_1)
\end{aligned}
$$

The last equality follows Lemma 8.1.1. In the end, we obtain that

$$
\log p_\theta(\mathbf{x}_0) \geq \mathbb{E}_{q(\mathbf{x}_T|\mathbf{x}_0)} \left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T \mid \mathbf{x}_0)} \right] + \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_{t-1}, \mathbf{x}_t|\mathbf{x}_0)} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} \right] + \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} \log p_{\boldsymbol{\theta}}(\mathbf{x}_0 \mid \mathbf{x}_1)
$$

Let us now focus on each term

- Term 1:

$$\mathbb{E}_{q(\mathbf{x}_T|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T \mid \mathbf{x}_0)}\right] = -\mathbb{E}_{q(\mathbf{x}_T|\mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_T \mid \mathbf{x}_0)}{p(\mathbf{x}_T)}\right] = -\mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_T \mid \mathbf{x}_0) \,||\, p(\mathbf{x}_T))$$

- Term 2: For any measureable function $f(\mathbf{x}_{t-1}, \mathbf{x}_t)$, we have

$$\begin{aligned}
\mathbb{E}_{q(\mathbf{x}_{t-1},\mathbf{x}_t|\mathbf{x}_0)}\left[f(\mathbf{x}_{t-1}, \mathbf{x}_t)\right] &= \iint f(\mathbf{x}_{t-1}, \mathbf{x}_t)\, q(\mathbf{x}_{t-1}, \mathbf{x}_t \mid \mathbf{x}_0)\, d\mathbf{x}_{t-1}\, d\mathbf{x}_t \\
&= \iint f(\mathbf{x}_{t-1}, \mathbf{x}_t)\, q(\mathbf{x}_{t-1} \mid \mathbf{x}_0, \mathbf{x}_t)\, q(\mathbf{x}_t \mid \mathbf{x}_0)\, d\mathbf{x}_{t-1}\, d\mathbf{x}_t \\
&= \int \left[\int f(\mathbf{x}_{t-1}, \mathbf{x}_t)\, q(\mathbf{x}_{t-1} \mid \mathbf{x}_0, \mathbf{x}_t)\, d\mathbf{x}_{t-1}\right] q(\mathbf{x}_t \mid \mathbf{x}_0)\, d\mathbf{x}_t \\
&= \int \mathbb{E}_{q(\mathbf{x}_{t-1}|\mathbf{x}_0,\mathbf{x}_t)}\left[f(\mathbf{x}_{t-1}, \mathbf{x}_t)\right] q(\mathbf{x}_t \mid \mathbf{x}_0)\, d\mathbf{x}_t \\
&= \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[\mathbb{E}_{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}\left[f(\mathbf{x}_{t-1}, \mathbf{x}_t)\right]\right]
\end{aligned}$$

  Then

$$\begin{aligned}
\mathbb{E}_{q(\mathbf{x}_{t-1},\mathbf{x}_t|\mathbf{x}_0)}\left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}\right] &= \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[\mathbb{E}_{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}\left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}\right]\right] \\
&= \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[-\mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))\right]
\end{aligned}$$

Hence, finally

$$\begin{aligned}
&\log p_\theta(\mathbf{x}_0) \geq \mathbb{E}_{q(\mathbf{x}_T|\mathbf{x}_0)}\left[\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T \mid \mathbf{x}_0)}\right] + \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_{t-1},\mathbf{x}_t|\mathbf{x}_0)}\left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)}\right] + \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} \log p_{\boldsymbol{\theta}}(\mathbf{x}_0 \mid \mathbf{x}_1) \\
&= -\mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_T \mid \mathbf{x}_0) \,||\, p(\mathbf{x}_T)) + \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[-\mathcal{D}_{\mathrm{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))\right] + \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} \log p_{\boldsymbol{\theta}}(\mathbf{x}_0 \mid \mathbf{x}_1)
\end{aligned}$$

Taking the expectation with respect to the inital data distribution yields

$$\begin{aligned}
&\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_0)\right] \\
&\leq \mathbb{E}_{q(\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_T \mid \mathbf{x}_0) \| p(\mathbf{x}_T)\right) + \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\right)\right] - \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}\left[\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\right]\right] \\
&= \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_T \mid \mathbf{x}_0) \| p(\mathbf{x}_T)\right) + \sum_{t=2}^{T} \mathcal{D}_{\mathrm{KL}}\left(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\right) - \log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\right]
\end{aligned}$$

Let us now show the last equality by each terms. By linearity of expectation, we obtain

- Term 1

$$
\begin{aligned}
&\mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[D_{\mathrm{KL}}\big(q(\mathbf{x}_T \mid \mathbf{x}_0) \,\|\, p(\mathbf{x}_T)\big)\right] \\
&= \int_{\mathbf{x}_0}\int_{\mathbf{x}_{1:T}} D_{\mathrm{KL}}\big(q(\mathbf{x}_T \mid \mathbf{x}_0) \,\|\, p(\mathbf{x}_T)\big) \cdot q(\mathbf{x}_0) \cdot q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)\, d\mathbf{x}_{1:T}\, d\mathbf{x}_0 \\
&= \int_{\mathbf{x}_0} D_{\mathrm{KL}}\big(q(\mathbf{x}_T \mid \mathbf{x}_0) \,\|\, p(\mathbf{x}_T)\big) \cdot q(\mathbf{x}_0) \left[\int_{\mathbf{x}_{1:T}} q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)\, d\mathbf{x}_{1:T}\right] d\mathbf{x}_0 \\
&= \int_{\mathbf{x}_0} D_{\mathrm{KL}}\big(q(\mathbf{x}_T \mid \mathbf{x}_0) \,\|\, p(\mathbf{x}_T)\big) \cdot q(\mathbf{x}_0)\, d\mathbf{x}_0 \\
&= \mathbb{E}_{q(\mathbf{x}_0)}\left[D_{\mathrm{KL}}\big(q(\mathbf{x}_T \mid \mathbf{x}_0) \,\|\, p(\mathbf{x}_T)\big)\right].
\end{aligned}
$$

- Term 2

$$
\begin{aligned}
&\mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\big)\right] \\
&= \int_{\mathbf{x}_0}\int_{\mathbf{x}_{1:T}} D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\big) \cdot q(\mathbf{x}_0) \cdot q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)\, d\mathbf{x}_{1:T}\, d\mathbf{x}_0 \\
&= \int_{\mathbf{x}_0}\int_{\mathbf{x}_t} D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\big) \cdot q(\mathbf{x}_0) \cdot \left[\int_{\mathbf{x}_{1:t-1}}\int_{\mathbf{x}_{t+1:T}} q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)\, d\mathbf{x}_{1:t-1}\, d\mathbf{x}_{t+1:T}\right] d\mathbf{x}_t\, d\mathbf{x}_0 \\
&= \int_{\mathbf{x}_0}\int_{\mathbf{x}_t} D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\big) \cdot q(\mathbf{x}_0) \cdot q(\mathbf{x}_t \mid \mathbf{x}_0)\, d\mathbf{x}_t\, d\mathbf{x}_0 \\
&= \mathbb{E}_{q(\mathbf{x}_0)}\left[\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\big)\right]\right]
\end{aligned}
$$

- Term 3

$$
\begin{aligned}
&\mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\right] \\
&= \int_{\mathbf{x}_0}\int_{\mathbf{x}_{1:T}} \big(-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\big) \cdot q(\mathbf{x}_0) \cdot q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)\, d\mathbf{x}_{1:T}\, d\mathbf{x}_0 \\
&= \int_{\mathbf{x}_0}\int_{\mathbf{x}_1} \big(-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\big) \cdot q(\mathbf{x}_0) \cdot \left[\int_{\mathbf{x}_{2:T}} q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)\, d\mathbf{x}_{2:T}\right] d\mathbf{x}_1\, d\mathbf{x}_0 \\
&= \int_{\mathbf{x}_0}\int_{\mathbf{x}_1} \big(-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\big) \cdot q(\mathbf{x}_0) \cdot q(\mathbf{x}_1 \mid \mathbf{x}_0)\, d\mathbf{x}_1\, d\mathbf{x}_0 \\
&= \mathbb{E}_{q(\mathbf{x}_0)}\left[\mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\right]\right].
\end{aligned}
$$

And we have proved the theorem, and the results macth the original expression in **?**. □

9 Implementation

9.1 UNet Architecture

The U-Net is a special type of Convolutional Neural Network (CNN). The U-Net is composed of two main components: a contracting path and an expanding path.

- Contracting path: aims to decrease the spatial dimensions of the image, while also capturing relevant information about the image.

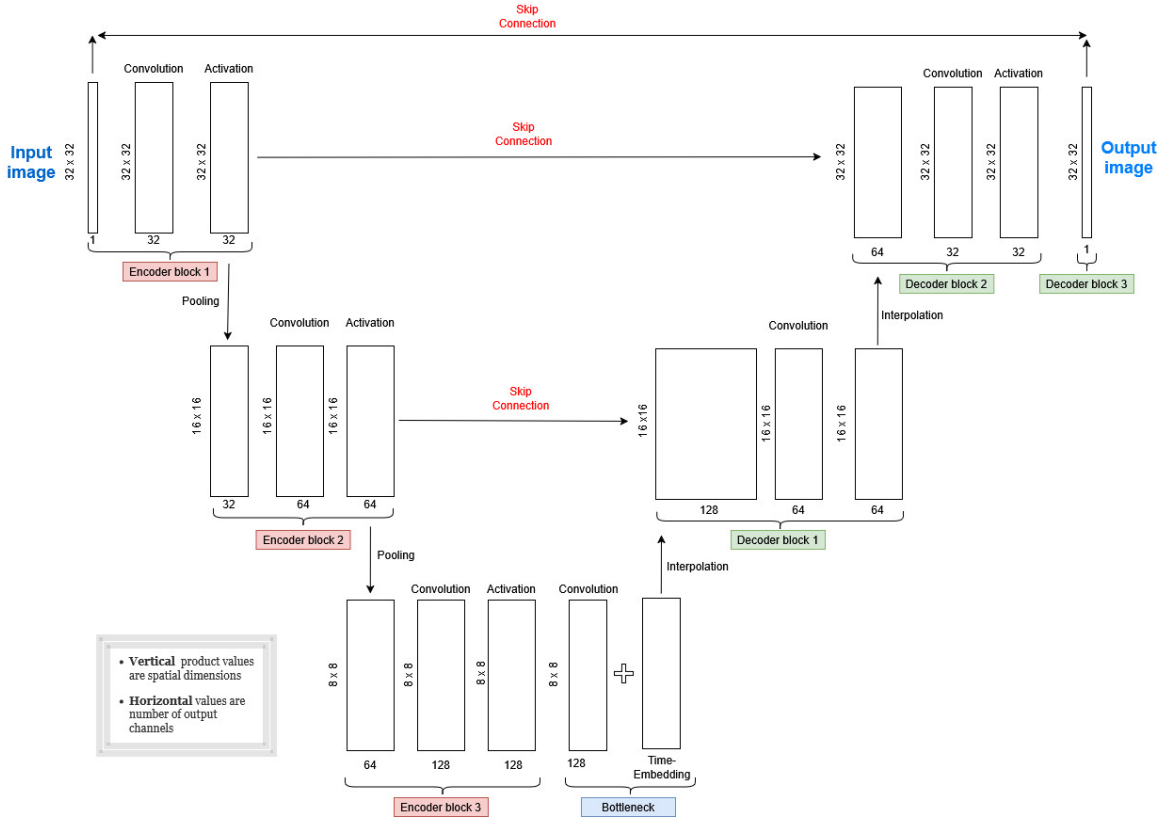

Figure 15: UNet: Diffusion_model_1 on MNIST

- Expanding path: aims to upsample the feature map and produce a relevant segmentation map using the patterns learnt in the contracting path.

The U-Net resembles an encoded-decoder architecture, which coincidentally makes a U shape.

We implement a custom U-Net model. During training (the forward diffusion process), we begin with a clean image $\mathbf{x}_0$, add Gaussian noise at a randomly chosen timestep t to obtain a noisy image $\mathbf{x}_t$, and train the U-Net to predict the noise that was added. The model is trained with the objective to minimize the simple loss function in Eq. (14) where $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ is the U-Net's predicted noise.

At sampling time (the reverse diffusion process), we start from pure Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$ and iteratively apply the trained U-Net to remove noise step by step. For each timestep t, we feed the current noisy sample $\mathbf{x}_t$ and the timestep t into the U-Net to obtain an estimate of the noise component $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$. Using this prediction together with the diffusion schedule, we compute a less noisy sample $\mathbf{x}_{t-1}$. Repeating this procedure till $t = 1$ gradually transforms random noise into a coherent, high-quality image sample.

To understand how a noisy image is refined into a high-quality sample, we examine the U-Net architecture.

9.2 Encoder's architecture Structure

The encoder in a U-Net is the contracting (downsampling) path that extracts hierarchical features from the input image by progressively reducing spatial resolution while increasing channel depth.

The encoder is typically a stack of blocks, each performing two key operations:

1. Convolutional Layers:
    - A 3×3 convolution block.
    - Activation function: ReLU/SiLU after each convolution.
    - Channel growth: Feature maps double in depth after each downsampling step (e.g., 32 → 64 → 128).
2. Downsampling
    - 2×2 average/max-pooling.
    - Spatial reduction: Halves height/width (e.g., 256×256 → 128×128). Preserves semantic content: Aggregates local information while discarding fine details.

9.3 Basic U-Net on MNIST/ Encoder

The simplest model was evaluated on the MNIST dataset, which consists of grayscale images of handwritten digits (0–9) with an original resolution of 28×28 pixels. Each pixel intensity is an integer in the range $[0, 255]$, where higher values correspond to brighter pixels. For compatibility with the network architecture, the images were resized to 32×32 pixels, resulting in an input tensor of shape $x_0 \in \mathbb{R}^{32\times32\times1}$.

The model was trained on 48,000 samples and validated on 12,000 samples, following a standard data split to ensure reliable evaluation of generalization performance.

Step 1: Convolutional Layer with Padding

The first convolutional layer applies a set of learnable 3×3 filters to the input. To preserve spatial dimensions after convolution, zero-padding of size 1 is applied, expanding the input from 32×32 to 34×34. This layer produces 32 feature maps, each of size 32×32, capturing distinct low-level patterns such as edges and textures in the input image.

Step 2: Non-Linear Activation

Following each convolution, a Rectified Linear Unit (ReLU) activation function is applied element-wise:

$$\text{ReLU}(x) = \max(0, x).$$

ReLU introduces non-linearity into the network, enabling it to model complex, non-linear relationships in the data while mitigating the vanishing gradient problem common in deep networks.

Step 3: Average Pooling

After activation, the feature maps are downsampled using average pooling with a 2×2 window and stride 2. This operation reduces the spatial dimensions by half, from 32×32 to 16×16, while retaining the most salient features. The reduction in spatial resolution is compensated by doubling the number of feature channels in the subsequent convolutional layer (e.g., from 32 to 64), following the common design principle in encoder architectures to increase representational capacity as spatial resolution decreases.

Steps 1 − 3 are repeated three times in the encoder (pooling is omitted in the third block). The encoder structure is consistent across models; increased complexity is achieved by adding an extra encoder/decoder block, complex time-embedding, self-attention mechanism, and group normalization.

Step 4: Bottleneck + Time-embedding

Given the input image has undergone three encoder block operations, its feature maps enter the bottleneck. This block consists of a 3×3 convolution that preserves the channel count, combined with a time-embedding layer. The bottleneck stores the most compressed and abstract information, serving as the bridge between the encoder and decoder pathways.

A scalar timestep t does not provide enough information for noise prediction. However, time-embedding also known as positional embedding, projects t into a higher-dimensional vector (here $\mathbb{R}^{128}$). This allows the network to treat all diffusion steps differently.

During training, a random t is sampled uniformly from $[0, t-1]$ for each image. The embedding block converts it into a 128-dimensional vector, uniquely representing the noise level at that step. The U-Net knows which stage of diffusion to address. Time-embedding t_{emb} is added to the 128 feature maps (each 8×8). For each channel, the same t_{emb} value is broadcast to every spatial location, injecting timestep information directly.

Now that the network understands when in the process it is operating (via the time-embedding), it uses this information to reconstruct the image. focusing on fine-grained details. This is the task of the decoder—the second half of the U-Net—which systematically reverses the encoder's downsampling. Guided by the injected time signal t_{emb}, it begins the work of building back detail and structure from the compressed representation.

9.4 Basic U-Net on MNIST/ Decoder

Step 5: Upsampling

The network begins by increasing the spatial dimensions through upsampling (interpolation). While the encoder downsamples via pooling, the decoder restores spatial dimensions through upsampling (interpolation). The spatial size is doubled : 8×8 to 16×16 after the first interpolation.

In our models, we implement nearest-neighbor interpolation where each new pixel copies the value of its nearest input pixel. Upsampling alone does not enhance detail; that role is fulfilled by skip connections and subsequent convolutional layers

Step 6 : Skip-Connections

Following from upsampling, the image undergoes its first skip-connections. These are concatenations with the corresponding feature maps from the encoding pathway with its respective decoder channel. They help the decoder recover fine-grained spatial information lost during the downsampling steps in the encoder.

A **long skip connection** crosses at least three convolution layers. Our simplest model contains two such connections, combining details from shallow features from encoder blocks 1 and 2 with its corresponding decoder block. This yields more representative feature maps and significantly improves sample quality; models without skip connections perform markedly worse.

As shown in Figure 1, our model uses three skip connections, each concatenating encoder features

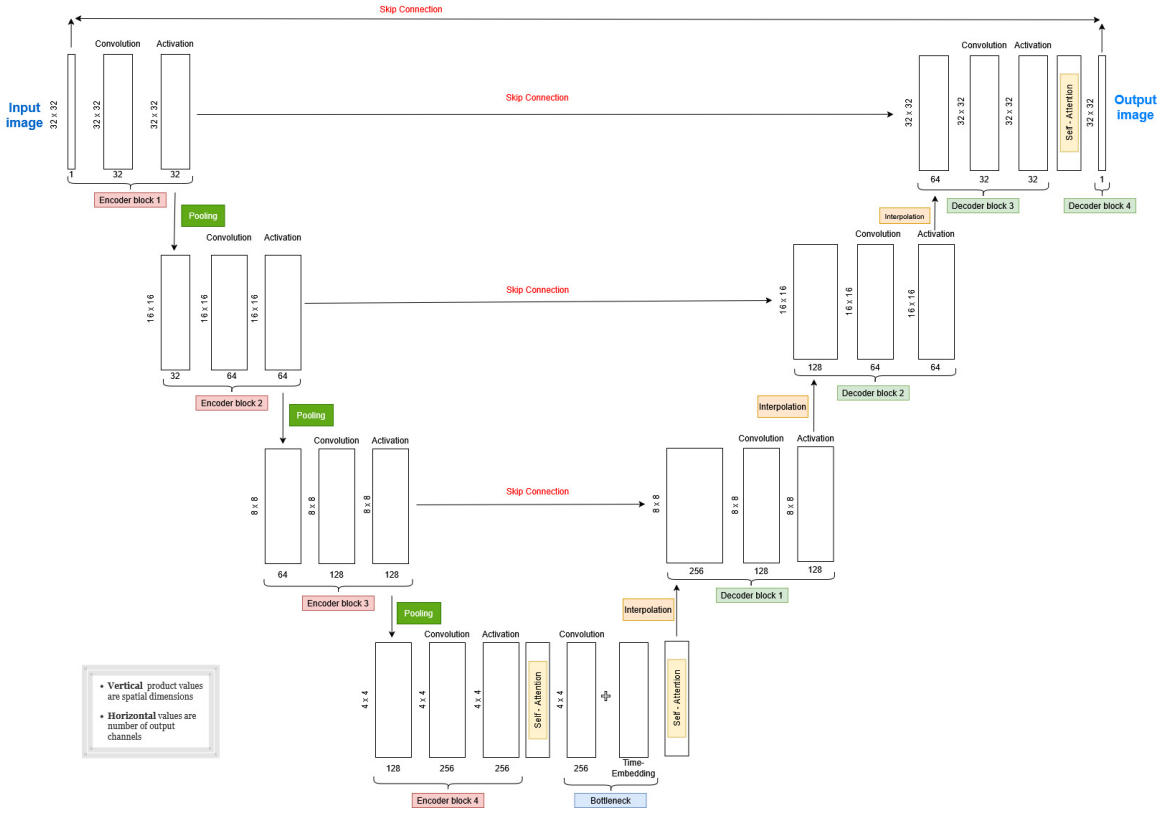

Figure 16: UNet: Diffusion_model_2 on MNIST

to the corresponding decoder layer along the channel dimension (dim = 1 in `torch.cat`).

Step 7 : Convolution

Similarly, a 3×3 convolution is applied after a skip-connection which is followed by an activiation function.

Steps 5 - 7 are repeated two more times, ending with a final convolutional layer immediately after the last activation. This final layer also receives a skip connection from the original input image x, helping preserve structural detail. The output is, theoretically, a high-quality generated sample.

This concludes the end-to-end process that the input image undergoes in our basic U-Net.

Results from this model are shown in Figure 3 and Figure 4. It was clear that **Diffusion_model_1** required adjustments to better capture the complexity of the data. **Diffusion_model_2** was developed as a result of this analysis.

9.5 Advanced U-Net on MNIST

Aligns similarly with the basic U-Net architecture, but with increased complexity. We introduce a model **Diffusion_model_2** with 1.8 million parameters with the the following additions:

- Three self-attention layers
- Sinusoidal positional embedding followed by an MLP for timestep embedding. This used in Transformers followed by Multi-Layer Perceptron (Linear $\rightarrow$ SiLU $\rightarrow$ Linear).

- An extra encoder block: 3×3 convolution, SiLU (Sigmoid Linear Unit) activation, and max-pooling (outputs the maximum value from each 2×2 pooling region)
- A corresponding extra decoder block with its skip-connection

It is important we understand, in particular, what self-attention is and its importance to our network.

9.6 Self-Attention

Self-Attention allows the network to understand relationships between distant parts of an image. Think of it as the network's way of "looking around" and deciding which areas are most relevant to each other.

9.6.1 How Self-Attention Works

The process starts with a feature map $x \in \mathbb{R}^{H \times W \times C}$, which contains spatial information at different locations. This map is transformed into three representations using 1×1 convolutional layers:

$$Q = W_Q x \quad \text{(Query: "What am I looking for?")} \tag{15}$$
$$K = W_K x \quad \text{(Key: "What do I contain?")} \tag{16}$$
$$V = W_V x \quad \text{(Value: "What information do I have?")} \tag{17}$$

Here, W_Q, W_K, W_V are learnable weight matrices updated during training.

9.6.2 The Attention Mechanism

The core operation computes attention scores between all spatial locations:

$$\text{Attention}(Q, K) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)$$

, where d_k is the dimension of queries and keys (for scaling stability). This creates an attention map that answers: "How much should each location focus on every other location?"

The final output is a weighted combination of value vectors:

$$\text{Attention}(Q, K, V) = \text{Attention}(Q, K) \cdot V$$

This produces a new feature map where each position now contains information from all relevant pixels across the image.

9.6.3 Integrating Attention Back into the Network

The attention mechanism produces a refined feature map, but we do not want to discard the original information. We combine them using a form of **learned blending**:

$$\text{Output} = x + \gamma \cdot \text{Attention}(Q, K, V)$$

where:

- x is the original input feature map.
- Attention(Q, K, V) is the new, context-aware feature map from the self-attention layer.
- γ is a **single, shared learnable parameter** (a scalar, not a matrix).

Why γ?

- It starts training initialized to **zero**. This means the initial output is simply $x + 0 \cdot Attention(Q, K, V) = x$. The attention layer is effectively "turned off."
- During training, if the self-attention mechanism helps the model make better predictions, the gradient descent process will gradually **increase the value of** γ.
- This allows the network to *learn how important* the attention features are. If they are not useful, γ will stay near zero.

Intuition: Think of γ as a volume knob. Initially, the "attention signal" is muted ($\gamma = 0$). As the network learns that paying attention to long-range relationships improves its performance, it slowly turns up the volume, blending in more of the attention output. This mechanism ensures a stable learning process, where complex relationships are introduced gradually.

The result is an output feature map with the same spatial dimensions ($H \times W \times C$), but where each location now contains globally-informed context.

9.6.4 Why Self-Attention?

Convolutional layers capture local patterns but have limited receptive field growth, restricting long-range dependency modeling.

Self-attention overcomes this by making each pixel's representation a weighted blend of information from the entire feature map. This provides:

- **Global context:** Connections between all spatial positions.
- **Adaptive receptive field:** The model learns which regions matter for each pixel.
- **Improved coherence:** Enables consistency in color and structure across distant regions.

Self-attention is typically placed in the bottleneck or deeper layers, where feature maps are spatially smaller (reducing computational cost) and represent higher-level semantics. This allows the U-Net to learn global dependencies through query–key similarity computations across all positions.

In summary, **Diffusion_model_2** is structured with 4 encoder–decoder blocks, 3 intermediate self-attention layers, and 4 corresponding skip connections—an architecture scaled to capture the complexity of the MNIST dataset.

9.6.5 Early Stopping: Preventing Overfitting

Having introduced a more complex architecture with self-attention, the risk of overfitting increases. To mitigate this, we employ **early stopping**. This technique halts training when the validation loss no longer improves, ensuring the model does not memorize training noise.

Two key parameters control early stopping:

- **patience**: The number of epochs to wait for an improvement in validation loss before stopping.
- **delta**: The minimum change in validation loss required to qualify as an improvement.

When early stopping is triggered, training ends early and the model weights from the best validation epoch (`best_model_state`) are restored. This guarantees that sample generation uses the most generalizable version of the model, even if performance degraded in later epochs.

For `Diffusion_model_1` trained on MNIST, early stopping was found to be unnecessary, as full training consistently produced higher-quality samples. In contrast, early stopping was implemented in `Diffusion_model_2`. Although no significant performance difference was observed without this regularization technique, it was retained as a safeguard against potential validation instability or unexpected training divergence in alternative experimental settings.

9.7 Advanced models on CIFAR-10

Figures 7–9 present the results of applying three distinct U-Net architectures to the CIFAR-10 dataset, which consists of everyday 32×32 color images.

The model was trained on 40,000 samples and validated on 10,000 samples.

- **CIFAR-10_diffusion_model_1** – Follows the same architecture as `MNIST_Diffusion_model_2`.
- **CIFAR-10_diffusion_model_2** – An enhanced version with the following modifications:
  - Group Normalization (GN) [normalizing features within grouped channels, independent of batch size] added after each 3×3 convolution layer, preceding activation throughout both the encoder and decoder pathways.
  - Two additional Self-Attention layers in the encoder pathway.
  - Timestep increased from 500 to 1000 [to generate higher-quality samples through a more gradual and controlled denoising process].
  - Early stopping integrated (patience = 10, $\Delta = 0.001$).
  - Total parameters: 1,852,657 (slightly more than Model 1).
- **CIFAR-10_diffusion_model_3** – The most complex model, with approximately 66 million parameters. Built upon Model 2, it includes:
  - Output channels of all convolution layers multiplied by 3 (e.g., $32 \rightarrow 96$).
  - An extra encoder-decoder block before and after the bottleneck layer, yielding 1536 output channels at the deepest encoder stage.
  - Adjusted early stopping (patience = 13, $\Delta = 0.001$) to permit longer training and avoid premature convergence.