# Denoising Diffusion Probabilistic Models

**Kshitij Ambilduke** **Théo Basséras** **Nemanja Vujadinović**
ENS Paris-Saclay, Gif-sur-Yvette

## Abstract

In this work, we present a unified framework for Denoising Diffusion Probabilistic Models (DDPMs). We interpret DDPMs through four theoretically equivalent loss functions: predicting the means of the reverse process ($\mu_q$), the original data ($x_0$), the noise ($\epsilon$), or the score ($s_\theta$), and adapt the sampling processes accordingly. Our empirical analysis reveals that the simplified noise-prediction objective ($L_\epsilon$) significantly outperforms its equivalent score-prediction objective ($L_{s_\theta}$) in sample quality. We attribute this to the advantageous weighting of noise levels in the simplified loss. Furthermore, we challenge the fixed covariance assumption by implementing a learned variance objective, which we find improves Negative Log-Likelihood (NLL). Finally, we evaluate classifier-free guidance, demonstrating that lower guidance scales yield superior FID scores by preserving diversity compared to strong guidance. All code for equivalent loss functions, trainable covariance, and conditional generation is available on *Github*[1]

## 1 Introduction

Generative modeling aims to approximate an unknown probability distribution $p_{\text{data}}(\mathbf{x})$ to synthesize high-fidelity data. While traditional methods like VAEs and GANs have seen success, a new paradigm known as *score-based generative modeling* has emerged. This framework avoids direct likelihood estimation by instead learning the score function, $\nabla_x \log p(x)$, allowing iterative refinement of noise into valid data samples. Originally, *score matching* was proposed to estimate non-normalized densities $\tilde{p}_\theta(x) = e^{-E_\theta(x)}$ by minimizing the expected squared error between the model score and data score.

$$J(\theta) = \frac{1}{2}\mathbb{E}_{p_{\text{data}}}\left[\|s_\theta(x) - \nabla_x \log p_{\text{data}}(x)\|_2^2\right]$$

Since, the true score depends on the data density which is unknown, under a few regularity conditions, Hyvärinen [2005] showed that optimizing $J(\theta)$ is equivalent to optimizing the following objective.

$$J_2(\theta) = \mathbb{E}_{x\sim p_{data}}\left[\text{tr}(\nabla_x s_\theta(x)) + \frac{1}{2}\|s_\theta(x)\|_2^2\right]$$

While theoretically sound and implementable, this objective is computationally difficult to scale. A key insight linking theory to practice is provided by Vincent [2011]. They demonstrated that *denoising* is equivalent to score estimation. Specifically, training a denoising autoencoder $f_\theta$ to reconstruct clean data $x$ from inputs perturbed by Gaussian noise ($\tilde{x} = x + \sigma\epsilon$) implicitly estimates the score of the *smoothed* distribution $p_\sigma$. In the limit of small noise $\sigma$, the optimal denoiser satisfies $f_\theta(\tilde{x}) - \tilde{x} \propto \nabla_{\tilde{x}} \log p_\sigma(\tilde{x})$. This relationship allows neural networks to learn score functions simply by minimizing a reconstruction objective, bypassing the need for complex trace estimators Song et al. [2019].

Once the score network $s_\theta(x, \sigma)$ is trained, we can generate novel samples using *Annealed Langevin Dynamics* Song and Ermon [2019]. To ensure the model covers the entire data manifold, the network is conditioned on a sequence of noise levels $\sigma \in \{\sigma_1, \ldots, \sigma_L\}$. Sampling proceeds by iteratively updating the state $x$ using the estimated score and a step size $\alpha$ depending on the noise scale. By annealing $\sigma$ down to zero, the process refines coarse, noisy samples into sharp data points lying on the data manifold.

$$x \leftarrow x + \frac{\alpha}{2}s_\theta(x, \sigma) + \sqrt{\alpha}\, z, \quad z \sim \mathcal{N}(0, I)$$

This score-based perspective provides the theoretical underpinning for DDPMs whose denoising can be seen as a specific parameterization of the Annealed Langevin Dynamics. Although DDPMs are often parameterized to predict the noise $\epsilon$ rather than the score directly, this objective is mathematically equivalent to weighted denoising score matching.

In the following sections, we detail a unified framework for DDPMs. We start by formulating the forward and reverse process. Then, we analyze equivalent objective functions, showing that parameterizing the model to predict noise $\epsilon$, data $x_0$, or the score are equivalent. Finally, we layout the framework for learned variance and conditional generation.

[1] https://github.com/vujadinovicn/text2image-diffusion/

## 2 Formulation of DDPM

### 2.1 Forward Process

We define the data variable as $x_0$ and the latent variables as a sequence $x_1, x_2, \ldots, x_T$, where $T$ is a hyperparameter representing the total number of timesteps. The Encoding (or Forward) Process transforms the data $x_0$ point into a particular latent vector $x_t$. The forward process is defined as a first-order Markov chain. For each step $t$, the transition is parameterized by a fixed variance schedule of scalars $\alpha_1, \ldots, \alpha_T \in [0, 1]$

$$x_{t+1} = \sqrt{\alpha_t} x_t + \sqrt{1 - \alpha_t} \epsilon_t$$

Here, $\epsilon_t \sim \mathcal{N}(0, I)$. Based on this recursive formulation, we can define a series of conditional forward distributions on the latent variables.

$$q(x_t \mid x_{t-1}) = \mathcal{N}\Big(x_t; \sqrt{\alpha_t} x_{t-1}, (1 - \alpha_t) I\Big) \tag{1}$$

If we define $q(x_t \mid x_{t-1})$ in this manner, the stationary distribution of the Markov chain approaches a standard normal distribution as $T \to \infty$ i.e. $q(x_T) \approx \mathcal{N}(x_T; 0, I)$. This serves as a starting point for the reverse process.

### 2.2 Reverse Process

We define the model as a decoding distribution $p_\theta(x_0, x_1, \ldots, x_T)$ which represents the joint distribution between the latent variables and the data variables. The joint distribution is decomposed using the chain rule and, similar to the forward process, we assume a first-order Markov chain for the decoding steps:

$$p_\theta(x_0, x_1, \ldots, x_T) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)$$

Here, owing to the forward process formulation, the transition probability $p_\theta(x_{t-1} \mid x_t)$ is parameterized as a Gaussian distribution where the mean and variance are learned functions:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\Big(x_{t-1}; \mu_\theta(x_t), \Sigma_\theta(x_t)\Big) \tag{2}$$

In this formulation, $\mu_\theta$ and $\Sigma_\theta$ are neural networks (or functions parametrized by $\theta$) that take $x_t$ as input to predict the parameters of the distribution for the previous step $x_{t-1}$.

### 2.3 Loss function

To learn the parameters of the decoding distribution, we rely on optimizing the ELBO. Following the derivation outlined by Ho et al. [2020] (maximizing $J_\theta(q)$ is equivalent to maximizing a lower bound of the log likelihood), the ELBO in case of DDPMs takes the following form:

$$\begin{aligned} J_\theta(q) &= \mathbb{E}_{q(x_1,\cdots,x_T|x_0)}\left[\log \frac{p_\theta(x_0, x_1, \cdots, x_T)}{q(x_1, \cdots, x_T|x_0)}\right] \\ &= \underbrace{\mathbb{E}_{q(x_1|x_0)}[\log p_\theta(x_0|x_1)]}_{\text{Reconstruction term}} - \underbrace{\mathbb{D}_{KL}[q(x_T|x_0) \| p(x_T)]}_{\text{Prior matching term}} - \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(x_t|x_0)}\left[\mathbb{D}_{KL}\left(q(x_{t-1}|x_t, x_0) \| p_\theta(x_{t-1}|x_t)\right)\right]}_{\text{Denoising term}} \end{aligned} \tag{3}$$

Here, the Prior-matching term does not depend on $\theta$ and is hence omitted during optimization. Furthermore, since we have a fixed forward process, $q(x_{t-1}|x_t, x_0)$ can be achieved in closed form using Eq.1 and bayes rule.

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}\big(x_{t-1}; \mu_q(x_t, x_0), \Sigma_q\big)$$

$$\mu_q(x_t, x_0) = \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1}) x_t + \sqrt{\bar{\alpha}_{t-1}}(1 - \alpha_t) x_0}{1 - \bar{\alpha}_t} \qquad \Sigma_q = \frac{(1 - \alpha_t)(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} I \tag{4}$$

Here, $\bar{\alpha} = \prod_{i=1}^{T} \alpha_i$. Using this and the formulation of the reverse process as seen in Eq. 2, the denoising term in ELBO optimization becomes tractable. Owing to training stability, as a design choice, Ho et al. [2020] proposed to only learn the means of the reverse process, setting the covariance matrix of the reverse process to $\Sigma_q$. Although we challenge this choice in later sections, for brevity, we show the derivations using fixed covariance. Hence, considering this, the denoising term takes the following form:

$$
\begin{aligned}
\mathbb{D}_{\text{KL}}\Big[q(x_{t-1}|x_t,x_0) \parallel p_\theta(x_{t-1}|x_t)\Big] &= \mathbb{D}_{\text{KL}}\Big[\mathcal{N}(x_{t-1};\mu_q,\Sigma_q) \parallel \mathcal{N}(x_{t-1};\mu_\theta,\Sigma_q)\Big] \\
&= \frac{1}{2}\left[\log\frac{|\Sigma_q|}{|\Sigma_q|} - d + \text{tr}(\Sigma_q^{-1}\Sigma_q) + (\mu_\theta-\mu_q)^T\Sigma_q^{-1}(\mu_\theta-\mu_q)\right] \\
&= \frac{1}{2\sigma_q^2}\|\mu_\theta-\mu_q\|_2^2
\end{aligned}
$$

Where, $\sigma_q^2$ refers to the scalar multiplied to $I$ in Eq. 4. Furthermore, it should be noted that in Eq. 3 summing over $t>2$ is equivalent, up to a constant factor $T$, to taking the expectation over $t\sim\mathcal{U}(2,T)$. Since the reverse process is also parameterized using a Gaussian, we can write the overall ELBO and the corresponding loss $L$ as follows:

$$J_\theta(q) = \mathbb{E}_{q(x_1|x_0)}\left[\frac{-1}{2\sigma_q^2}\|x_0-\mu_\theta(x_1)\|_2^2\right] - \mathbb{E}_{q(x_t|x_0),t\sim\mathcal{U}(2,T)}\left[\frac{1}{2\sigma_q^2}\|\mu_\theta-\mu_q\|_2^2\right] \tag{5}$$

$$L_{\mu_\theta} = \mathbb{E}_{q(x_1|x_0)}\left[\frac{1}{2\sigma_q^2}\|x_0-\mu_\theta(x_1)\|_2^2\right] + \mathbb{E}_{q(x_t|x_0),t\sim\mathcal{U}(2,T)}\left[\frac{1}{2\sigma_q^2}\|\mu_\theta(x_t,t)-\mu_q(x_t,x_0)\|_2^2\right] \tag{6}$$

This is broadly regarded as the loss corresponding to the Variational Lower Bound and DDPMs can be seen as regressors over the $\mu_q$ with an additional term corresponding to reconstruction. Instead of regressing over $\mu_q$, the loss function shown in Eq. 6 can be shown to be equivalent to regressing over $x_0$ instead (Appendix A). The main advantage of doing so is that instead having two separate terms in the loss function, we can combine both into a single loss $L_{x_0}$.

$$L_{x_0} = \mathbb{E}_{q(x_t|x_0),t\sim\mathcal{U}(1,t)}\left[\frac{1}{2\sigma_q^2}\frac{(1-\alpha_t)^2}{(1-\bar{\alpha}_t)^2}\bar{\alpha}_{t-1}\,\|\hat{x}_\theta(x_t)-x_0\|_2^2\right] \tag{7}$$

Similarly, we can see DDPMs as a regressor over the added noise with corresponding loss $L_\epsilon$(Appendix B) and as score-predictors with loss $L_{s_\theta}$ (Appendix C). In all of these, the reconstruction term and the denoising term can be combined into a single loss term.

$$L_\epsilon = \mathbb{E}_{q(x_t|x_0),t\sim\mathcal{U}(1,t)}\left[\frac{1}{2\sigma_q^2}\frac{(1-\alpha_t)^2}{(1-\bar{\alpha}_t)\alpha_t}\|\epsilon_t-\hat{\epsilon}_\theta(x_t)\|_2^2\right] \tag{8}$$

$$L_{s_\theta} = \mathbb{E}_{q(x_t|x_0),t\sim\mathcal{U}(1,t)}\left[\frac{1}{2\sigma_q^2}\frac{(1-\alpha_t)^2}{\alpha_t}\|\nabla_{x_t}\log p(x_t)-s_\theta(x_t)\|_2^2\right] \tag{9}$$

## 2.4 Sampling

Depending on the realization of the loss function used to train DDPM, the sampling changes accordingly. The central idea is to iteratively sample from the reverse process $p(x_{t-1}|x_t)$. The overall process of sampling relies on starting from a purely noisy image $x_t$ where $x_t\sim\mathcal{N}(0,\sigma_q^2 I)$. Then iteratively, this noisy image is denoised traversing along the chain $x_T\to x_{T-1}\cdots\to x_0$. Depending on the formulation of loss, the mean of each step in the reverse process can be expressed in terms of the output of the DDPM model. Following are some equivalent ways.

$$[\mu_\theta] \qquad x_{t-1} = \mu_\theta(x_t,t)+\sigma_t z \tag{10}$$

$$[x_0] \qquad x_{t-1} = \left(\frac{(1-\bar{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\bar{\alpha}_t}\right)x_t + \left[\frac{(1-\alpha_t)\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_t}\right]\hat{x}_\theta(x_t,t)+\sigma_t z \tag{11}$$

$$[\epsilon_\theta] \qquad x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t,t)\right)+\sigma_t z \tag{12}$$

$$[s_\theta] \qquad x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t+(1-\alpha_t)s_\theta(x_t,t)\right)+\sigma_t z \tag{13}$$

Where, $z\sim\mathcal{N}(0,I)$, is the diffusion term which encourages diversity in the generated data. Repeating this process until $t=0$ yields $x_0$, the new generated sample. It is worth noticing that Equation 13 can be interpreted as a specific parameterization of the Annealed Langevin Dynamics Song and Ermon [2019].

### 2.5 Beta schedules and variance

Note that in our formulation, we have chosen to stick with the notation $1 - \alpha_t$ which is equivalentely denoted as $\beta_t$ in the original work of DDPMs Ho et al. [2020]. The choice of schedule for $\alpha_t$ (linear, cosine, or learned) affects sampling quality and convergence. Adhering to the original work, we have used linear schedule for $\alpha_t$. Nichol and Dhariwal [2021] show that learning reverse variances improves likelihood and sample quality and can reduce the number of required sampling steps substantially. Taking inspiration from this, we have also implemented DDPM model which predicts both $\mu_\theta(x_t)$ and $\Sigma_\theta(x_t)$.

## 3 Conditional generation: Guided diffusion

Consider $x_0$ as the data variable and $y$ as the label of this data variable. Hence, considering the score-prediction instantiation of diffusion model, in unconditional generation, we regressed over the score $\nabla_{x_t} \log p(x_t)$ equivalently, we now need to regress over $\nabla_{x_t} \log p(x_t|y)$ for conditional generation. Using Bayes rule, we can simplify the conditional score $\nabla_{x_t} \log p(x_t|y)$ as follows:

$$\begin{aligned}\nabla_{x_t} \log p(x_t|y) &= \nabla_{x_t} \log p(x_t) + \nabla_{x_t} \log p(y|x_t) + \nabla_{x_t} \log p(y) \\ &= \nabla_{x_t} \log p(x_t) + \nabla_{x_t} \log p(y|x_t)\end{aligned} \tag{14}$$

Notably, we see that the conditional score is the sum of the unconditional score $\nabla_{x_t} \log p(x_t)$ and the classifier score $\nabla_{x_t} \log p(y|x_t)$. We already saw how we can estimate the unconditional score by parameterizing it using a neural network $\nabla_{x_t} \log p_\theta(x_t)$ whereas, the later term, also known as classifier guidance Dhariwal and Nichol [2021], can be parameterized using any classifier trained to predict $p_\phi(y|x_t)$. This can be trivially achieved by backpropagating the gradients upto the input $x_t$ of the classifer to get $\nabla_{x_t} \log p_\phi(y|x_t)$. Once we have this, we can steer the original sampling process towards a particular class by simply adding the classifier score to the unconditional score during sampling.

An alternate approach to arrive at the same conclusion uses the discrete transition probabilities. To achieve conditional generation, we first define a conditional reverse process $\hat{q}(x_t|x_{t+1}, y)$. As shown by Dhariwal and Nichol [2021], this conditional transition can be approximated as the product of the unconditional transition and the classifier probability: $\hat{q} \propto q(x_t|x_{t+1})p(y|x_t)$. In diffusion model, we want $q(x_t|x_{t+1}) \approx p(x_t|x_{t+1})$, replacing this and taking the gradient with respect to $x$ on both sides, we recover Equation 14.

While this method provides an off-the-shelf framework to generate images conditioned on labels, the main disadvantage of this method is its reliance on a classifier to get the classifier score. For reliably predicting the classifier score, we would need to have a model that correctly predicts an image at all noise levels. This is very hard to do in practice. Hence, Ho and Salimans [2022] suggested a simpler way to introduce guidance. Particularly, they added the unconditional score $\nabla_{x_t} \log p(x_t)$ with the a weighted conditional score $w\nabla_{x_t} \log p(y|x_t)$ to get the guided score $\nabla_{x_t} \log p(x_t|y)$. This is equivalent to saying $\tilde{p}_\theta(x_t|y) \propto p_\theta(x_t) \cdot p_\theta(y|x_t)^{(1+w)}$ is the conditional distribution used for generating the conditional images. Taking score with respect to $x$ on both sides, we get:

$$\begin{aligned}\nabla_{x_t} \log \tilde{p}(x_t|y) &= \nabla_{x_t} \log p(x_t|y) + w\nabla_{x_t} \log p(y|x_t) \\ &= \nabla_{x_t} \log p(x_t|y) + w\left(\nabla_{x_t} \log p(x_t|y) - \nabla_{x_t} \log p(x_t)\right) \\ &= (1+w)\nabla_{x_t} \log p(x_t|y) - w\nabla_{x_t} \log p(x_t)\end{aligned}$$

This completely eliminates the need of having an external model to generate the classifier guidance and we can essentially train a single model to output $\nabla_{x_t} \log p(x_t|y)$ by passing the true label along with $x_t$ and to generate $\nabla_{x_t} \log p(x_t)$ by passing a null label along with $x_t$. The sampling in this case is again straight forward, we just replaces the unconditional score by the conditional score and use Equation 13 to generate conditional images.

## 4 Implementation Details

### 4.1 Model and training

**Loss functions** Following Ho et al. [2020], instead of optimizing for the complete Equation 8, we only optimize the norm difference i.e. $L_\epsilon = \|\epsilon_t - \hat{\epsilon}_\theta(x_t)\|_2^2$. Using this, we find the equivalent loss function for $L_{s_\theta}$ by substituting $\epsilon_t = -\nabla_{x_t} \log p(x_t|x_0)\sqrt{1-\bar{\alpha}_t}$. This comes out to be $L_{s_\theta} = (1-\bar{\alpha}_t)\|\nabla_{x_t} \log p(x_t|x_0) - s_\theta(x_t)\|_2^2$. Note that we have replaced $\nabla_{x_t} \log p(x_t)$ in Equation 9 with $\nabla_{x_t} \log p(x_t|x_0)$ using the result from denoising score matching Vincent [2011]. However, we see suboptimal performance in terms of FID score by deriving the weighting for $L_{s_\theta}$ directly from $L_\epsilon$. Hence, for $L_{x_0}$, we stick to the intuition of disregarding the constants accompanied with the norm difference and optimize $L_{x_0} = \|x_0 - \hat{x}_\theta(x_t)\|_2^2$. Besides these, we found that empirically optimizing $L_{\mu_\theta}$ to be extremely difficult when we have a fixed variance schedule for the reverse process, however, it works well in the case of learned variance.

**Model architecture** We adopt a U-Net architecture leveraging a PixelCNN++ Salimans et al. [2017] backbone to generate outputs based on the loss function defined in Section 2.3. Input images are resized to a spatial resolution of $32 \times 32$ before entering the network. The encoder progressively downsamples the feature maps in the sequence $32 \to 16 \to 8 \to 4$.Given the single-channel nature of the MNIST dataset Deng [2012], we apply an initial convolution layer to project the input into 64 feature channels. Subsequently, channel depth is increased at each downsampling stage using multipliers of 1, 2, 4, and 8, respectively. Additionally, we use attention mechanisms at the $16 \times 16$ resolution and utilize Group Normalization Wu and He [2018] throughout.

**Other hyperparameters and data** Following the original work, we train a single network for estimating the required output across all time-steps $t \in \{1, 2, \cdots, T\}$ where $T = 1000$. For passing the information about the time-steps, we utilize sinusoidal embeddings as proposed originally for Transformers Vaswani et al. [2017]. For the model trained for conditional generation, we use trainable embedding matrix for each class of label, these embeddings are added to the sinusoidal embeddings corresponding to time-steps and passed into the network. Furthermore, we use $\beta_1 = 0.001$ and $\beta_T = 0.02$ while the intermediate values are interpolated according to a linear schedule. For the optimizer, we use Adam Kingma and Ba [2014] with a learning of $1e-4$. Unlike the original work, we do not update the network using EMA. Due to computational constraints limited to a single T4 GPU provided free on `Google Colab`, we limited the input classes for MNIST. Specifically, for MNIST we trained on digits $\{1, 2, 3\}$ which correspond to the same class labels. We train our model on 50 epochs using a batch size of 32.

## 4.2 Learned variance

As mentioned in Section 2.5, we also train our network to learn the reverse variance along with the mean. To this end, we adjust the final layer of U-Net so that it outputs an additional vector $v$. Following Nichol and Dhariwal [2021], we compute the variance from $v$ by interpolating between $\beta_t$ and $\tilde{\beta}_t$ in the log domain, i.e., $\Sigma_\theta(x_t, t) = \exp(v \log \beta_t + (1-v) \log \tilde{\beta}_t)$. The remaining part of the network, together with all the parameters, stays unchanged. During training, $\mu_\theta(x_t)$ is supervised using a noise prediction objective in Eq. 8, while $\Sigma_\theta(x_t, t)$ is supervised using the VLB objective derived from the ELBO in Eq. 3. Note that during the computation of the VLB term, the mean prediction is detached so that the computed gradients only update the variance head. The final learned variance loss is then computed as $L_{\text{learned}} = L_\epsilon + \lambda L_{\text{VLB}}$, with $\lambda = 1000$. The value of $\lambda$ was determined empirically and has been proven to be stable under the noise prediction objective. As for sampling, we compute the learned variance in the same way as during training, and replace the fixed variance $\sigma_t$ in Eq. 10 with it.

## 4.3 Guided diffusion

As seen in Section 3, we need to condition the model on both, the true labels $y$ and a null label $\phi$ to generate the conditional and the unconditional score respectively. During training, we do this by randomly replacing the true labels with $\phi$ with a probability $p_\phi$. The model is trained using a score matching objective as discussed in Equation 9. This approach forces the network to learn both the unconditional score and the conditional score within a single model.

In practice, we set $p_\phi = 0.2$. Furthermore, we introduce a trainable embedding matrix to encode the classes into vectors. In particular, since we use 3-classes, we would need a total of 4 embeddings (3 corresponding to classes and 1 corresponding to the null label). We add this embedding to the sinusoidal embedding for the time-step. We keep rest of the settings related to the model and data identical to unconditional case.

| **Loss type** | **FID** $\downarrow$ | **FID Binarized** $\downarrow$ | **NLL** $\downarrow$ |
|---|---|---|---|
| MNIST split topline | 1.12 | 0.74 | – |
| $x_0$ prediction | 49.10 | 9.89 | – |
| $\epsilon$ prediction | 15.75 | 10.13 | 1.21 |
| $s_\theta(x_t)$ prediction | 78.83 | 17.36 | – |
| $s_\theta(x_t\|y)$ prediction ($w = 5$) | 74.21 | 25.16 | – |
| $s_\theta(x_t\|y)$ prediction ($w = 1$) | 61.91 | 15.69 | – |
| Learned variance | 58.89 | 14.45 | 1.1 |

Table 1: FID and NLL comparison of different loss functions. For the topline, we split our MNIST subset into two non-overlapping subsets and compute the FID score between them. NLL is given in bits/dim.

## 5 Results and Discussion

**Experimental Setup and Evaluation Metrics** To evaluate the generation quality of different loss functions, we utilize the Fréchet Inception Distance (FID) to compare generated samples against the MNIST training set. In particular we use the Pytorch-FID library Seitzer [2020] for getting the FID scores. Restricting our domain to digits $\{1, 2, 3\}$, we utilize a subset of the training data containing approximately 18k images, evenly distributed across classes. For the open-ended generations corresponding to $L_{x_0}$, $L_\epsilon$, $L_{s_\theta}$, and $L_{\text{learned}}$, we generate 5,000 samples each. For conditional generation, we generate an equal number of images per class, totaling 5,000. We further evaluate the binarized versions of our generated images against the binarized MNIST training set. Finally, following Nichol and Dhariwal [2021], we compare the Negative Log-Likelihood (NLL) of images generated from $L_{\text{learned}}$ against $L_\epsilon$. Generated samples are visualized in Figure 1, with binarized versions in Figure 2 (Appendix).

**Grayscale vs. Binarized generations** The discrepancy in FID scores between grayscale and binarized images suggests that while the generated grayscale images appear plausible, they contain high-frequency noise or minor artifacts. These fluctuations inflate the standard FID score but are effectively filtered out during binarization. Notably, the simplified $L_\epsilon$ objective performs exceptionally well in the grayscale domain, achieving an FID of $15.75$. Although we achieve plausible FID scores, we admit that we did not manage to match the score of the topline as seen in Table 1.

**Analysis of weighting in score prediction loss** Upon binarization, both $L_\epsilon$ and $L_{x_0}$ achieve comparable performance, however, the model trained on $L_{s_\theta}$ exhibits a significantly higher FID. We attribute this to the weighting term in the loss implementation discussed in Section 4.1. Specifically, the multiplier $(1 - \bar{\alpha}_t)$ in $L_{s_\theta} = (1 - \bar{\alpha}_t)\|\nabla_{x_t} \log p(x_t|x_0) - s_\theta(x_t)\|_2^2$ varies drastically across $t$, effectively penalizing the model less for errors at higher noise levels (large $t$). In contrast, the simplified $L_{x_0}$ and $L_\epsilon$ objectives optimize the norm difference directly, penalizing errors equally across all time steps. This suggests that disregarding the variational lower bound weighting terms in favor of uniform weighting is beneficial for sample quality in DDPMs.

**Trade-off between classifier guidance and diversity** As the conditional model was trained using the $L_{s_\theta}$ formulation, its scores remain in a similar range to the unconditional $L_{s_\theta}$ model. Table 1 highlights the influence of the guidance scale $w$ on FID. A higher $w$ forces the model to generate the most probable images for a specific class, compromising intra-class diversity and degrading the FID score. Conversely, decreasing $w$ improves the score. Interestingly, the conditional model with $w = 1$ outperforms the unconditional model. We hypothesize this is because conditional generation ensures balanced sampling across classes, whereas the unconditional model may suffer from slight class imbalance.

**Learned variance** Additionally, we observe that training the model with learned variance improves the log-likelihood while keeping the FID Binarized score close to the baseline method that predicts only the mean value. As noted by Nichol and Dhariwal [2021], while DDPMs are generally known for producing high-quality samples, they struggle to achieve the competitive log-likelihoods with other likelihood-based models (e.g., VAE). Our experiment shows that incorporating a learned variance objective can partially alleviate this limitation without modifying the model architecture or substantially degrading the sample quality. As a result, DDPMs can benefit from improved likelihood-based modeling, which Razavi et al. [2019] have associated with better coverage of the data distribution.

## 6 Conclusion

In this work, we presented a unified framework for Denoising Diffusion Probabilistic Models (DDPMs), interpreting them through four theoretically equivalent loss functions: predicting the mean, original data, noise, or score. Our empirical analysis reveals that despite this theoretical equivalence, the simplified noise-prediction objective ($L_\epsilon$) significantly outperforms the score-prediction objective ($L_{s_\theta}$) in terms of sample quality. We attribute this divergence to the advantageous weighting of noise levels in the simplified loss, where disregarding the variational lower bound weighting terms in favor of uniform weighting proves beneficial for generation performance. Furthermore, we demonstrated that incorporating a learned variance objective improves Negative Log-Likelihood (NLL) without degrading sample quality, allowing DDPMs to achieve better likelihood-based modeling. Additionally, our evaluation of classifier-free guidance indicates a critical trade-off between guidance strength and diversity. Specifically, lower guidance scales yield superior FID scores by avoiding the collapse of intra-class diversity observed with strong guidance.

## Contributions

**Kshitij Ambilduke:** Implementation of model, equivalent losses and conditional generation. Report and poster.
**Nemanja Vujadinović:** Implementation of the model and learned variance. Report and poster.
**Théo Basséras:** Evaluation pipeline (FID/IS), visualizations, parts of the report and experiments.

## References

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. doi: 10.1109/MSP.2012.2211477.

Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Bradley Efron. Tweedie's formula and selection bias. *Journal of the American Statistical Association*, 106:1602 – 1614, 2011. URL https://api.semanticscholar.org/CorpusID:23284154.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022. URL https://arxiv.org/abs/2207.12598.

Jonathan Ho, Ajay Jain, and P. Abbeel. Denoising diffusion probabilistic models. *ArXiv*, abs/2006.11239, 2020. URL https://api.semanticscholar.org/CorpusID:219955663.

Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005. URL http://jmlr.org/papers/v6/hyvarinen05a.html.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL https://api.semanticscholar.org/CorpusID:6628106.

Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021. URL https://arxiv.org/abs/2102.09672.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with VQ-VAE-2. *Advances in Neural Information Processing Systems*, 32, 2019.

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: A pixelcnn implementation with discretized logistic mixture likelihood and other modifications. In *ICLR*, 2017.

Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. https://github.com/mseitzer/pytorch-fid, August 2020. Version 0.3.0.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/3001ef257407d5a371a96dcd947c7d93-Paper.pdf.

Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation, 2019. URL https://arxiv.org/abs/1905.07088.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Comput.*, 23(7):1661–1674, July 2011. ISSN 0899-7667. doi: 10.1162/NECO_a_00142. URL https://doi.org/10.1162/NECO_a_00142.

Yuxin Wu and Kaiming He. Group normalization. In *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, page 3–19, Berlin, Heidelberg, 2018. Springer-Verlag. ISBN 978-3-030-01260-1. doi: 10.1007/978-3-030-01261-8_1. URL https://doi.org/10.1007/978-3-030-01261-8_1.

## Appendix

## A Loss parameterization 1: $x_0$-prediction

To derive a equivalent objective, we analyze the loss between the model mean $\mu_\theta$ and the true posterior mean $\mu_q(x_t, x_0)$ for the following term in the Evidence Lower Bound (ELBO) loss:

$$\mathcal{L} \propto \frac{1}{2\sigma_q^2} \left\| \mu_\theta(x_t) - \mu_q(x_t, x_0) \right\|_2^2$$

The true posterior mean $\mu_q$ can be expressed as a linear combination of $x_t$ and $x_0$:

$$\mu_q(x_t, x_0) = \underbrace{\left[ \frac{(1-\bar{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\bar{\alpha}_t} \right]}_{C_1} x_t + \underbrace{\left[ \frac{(1-\alpha_t)\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_t} \right]}_{C_2} x_0$$

We can reparameterize $\mu_\theta$ to match the structure of $\mu_q$. By doing so, instead of predicting the mean directly, the neural network predicts an estimate of the data, $\hat{x}_\theta(x_t)$, effectively acting as a scaling and shifting operation:

$$\mu_\theta(x_t) = C_1 x_t + C_2 \hat{x}_\theta(x_t)$$

Substituting this formulation into the loss function simplifies the objective to a direct reconstruction loss on the data variable: $L_{x_0} \propto \left\| \hat{x}_\theta(x_t) - x_0 \right\|_2^2$

## B Loss parameterization 2: Regression over Added Noise ($\epsilon$)

Alternatively, we can express $x_0$ in terms of the noise $\epsilon_t$ added during the forward process. In particular, $x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_t)$. Substituting this into Eq. 4 yields:

$$\mu_q(x_t, \epsilon_t) = \frac{1}{\sqrt{\alpha_t}} x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}} \epsilon_t$$

We can again reparameterize the model mean $\mu_\theta$ similarly, using a network $\hat{\epsilon}_\theta(x_t)$ to predict the noise:

$$\mu_\theta(x_t) = \frac{1}{\sqrt{\alpha_t}} x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}} \hat{\epsilon}_\theta(x_t)$$

This simplifies the consistency term to a regression over the added noise $L_\epsilon \propto \left\| \epsilon_t - \hat{\epsilon}_\theta(x_t) \right\|_2^2$

## C Loss parameterization 3: Score Matching

Finally, we can interpret DDPMs as score predictors by utilizing Tweedie's Formula Efron [2011]. For a variable $x_t \sim \mathcal{N}(\mu_{x_t}, \Sigma_{x_t})$, the best estimate of the mean given the observation is:

$$\mathbb{E}[\mu_{x_t} \mid x_t] = x_t + \Sigma_{x_t} \nabla_{x_t} \log p(x_t)$$

In the context of the DDPM forward process, where $q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$, we can estimate the "true mean" component $\sqrt{\bar{\alpha}_t}x_0$ as:

$$\sqrt{\bar{\alpha}_t}x_0 = x_t + (1-\bar{\alpha}_t)\nabla_{x_t} \log p(x_t) \tag{15}$$

$$\therefore x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} x_t + \frac{1-\bar{\alpha}_t}{\sqrt{\bar{\alpha}_t}} \nabla_{x_t} \log p(x_t) \tag{16}$$

Substituting this expression for $x_0$ back into the true posterior mean $\mu_q(x_t, x_0)$ derived in Eq. 4, the formulation simplifies to:

$$\mu_q(x_t) = \frac{1}{\sqrt{\alpha_t}} x_t + \frac{1-\alpha_t}{\sqrt{\alpha_t}} \nabla_{x_t} \log p(x_t)$$

To match this structure, we reparameterize the model mean $\mu_\theta$ to predict the score using a network $S_\theta(x_t)$:

$$\mu_\theta(x_t) = \frac{1}{\sqrt{\alpha_t}} x_t + \frac{1-\alpha_t}{\sqrt{\alpha_t}} S_\theta(x_t)$$

Consequently, the consistency loss becomes equivalent to explicit score matching $L_{s_\theta} \propto \left\| \nabla_{x_t} \log p(x_t) - S_\theta(x_t) \right\|_2^2$

## D Training and sampling algorithms

**Algorithm 1** DDPM Training with Learned Variance

```
1:  Precompute $\alpha_t$, $\bar{\alpha}_t$, $\bar{\alpha}_{t-1}$, $\sigma_t^2$ for $t = 1, \dots, T$
2:  for epoch $= 1$ to num_epochs do
3:      for each batch $(x_0, y)$ in $\mathcal{D}$ do
4:          Sample $t \sim$ Uniform$\{0, \dots, T-1\}$
5:          Sample $\epsilon \sim \mathcal{N}(0, I)$
6:          $x_t \leftarrow \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} z$
7:          Predict $\mu_\theta$ and $v$ from the model
8:          $\Sigma_\theta \leftarrow \exp(v \log \beta_t + (1 - v) \log \tilde{\beta}_t)$.
9:          Compute mean loss $\mathcal{L}_{mean}(\theta)$ using input $(x_t)$ as per Eq. 8
10:         $\mu_q \leftarrow \frac{1}{1-\bar{\alpha}_t} (\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1}) x_t + \sqrt{\bar{\alpha}_{t-1}}(1 - \alpha_t) x_0)$
11:         Compute variance loss $\mathcal{L}_{var}(\theta)$ using input $(\Sigma_\theta, \mu_q)$ as per Eq. 3
12:         $\mathcal{L} \leftarrow \mathcal{L}_{\text{mean}} + \lambda \mathcal{L}_{\text{var}}$
13:         Update $\theta$ via Adam optimizer to minimize $\mathcal{L}$
14:     end for
15: end for
```

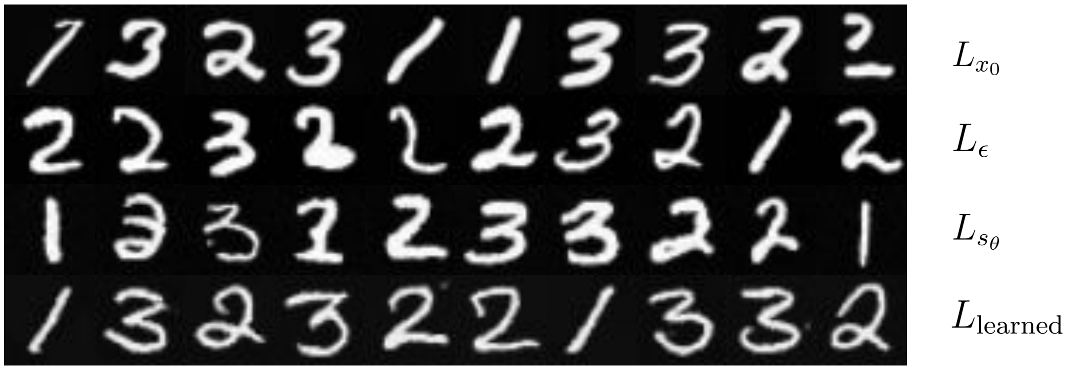
**Algorithm 2** Conditional DDPM Training with Classifier-Free Guidance

```
Require: Training data $\mathcal{D}$, Score model $s_\theta(x, t, c)$, $p_\phi = 0.2$
1:  Precompute $\alpha_t, \sigma_t^2$ schedules
2:  for epoch $= 1$ to num_epochs do
3:      for each batch $(x_0, y)$ in $\mathcal{D}$ do
4:          Sample $t \sim$ Uniform$\{0, \dots, T-1\}$
5:          Sample noise $z \sim \mathcal{N}(0, I)$
6:          $x_t \leftarrow \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} z$
7:          $r \sim$ Uniform$(0, 1)$                          ▷ Sample dropout probability
8:          if $r < p_\phi$ then
9:              $c \leftarrow \phi$                            ▷ Use null label for unconditional learning
10:         else
11:             $c \leftarrow y$                               ▷ Use true label for conditional learning
12:         end if
13:         Compute loss $\mathcal{L}(\theta)$ using input $(x_t, t, c)$ as per Equation 9
14:         Update $\theta$ via Adam optimizer to minimize $\mathcal{L}$
15:     end for
16: end for
```

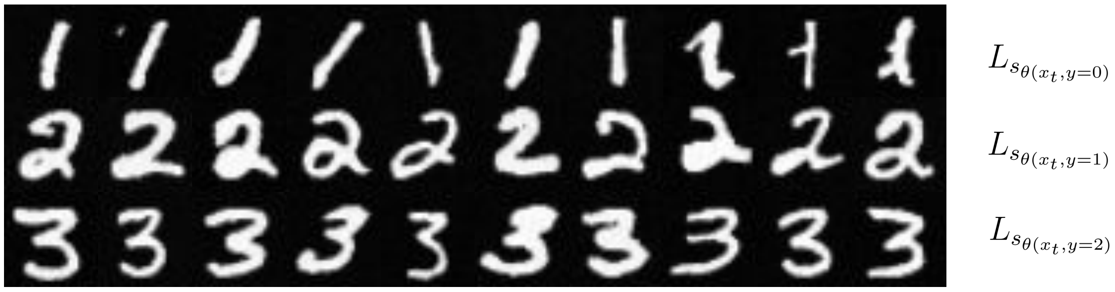
**Algorithm 3** Conditional DDPM Sampling

```
Require: Score model $s_\theta(x, t, y)$, guidance scale $w$, target class $y$, null class $y_\emptyset$
Require: Noise schedule $\alpha_t$, variance schedule $\sigma_t^2$
1:  $x \sim \mathcal{N}(0, I)$
2:  for $t = T - 1, \dots, 0$ do
3:      $s_{\text{cond}} \leftarrow s_\theta(x, t, y)$
4:      $s_{\text{uncond}} \leftarrow s_\theta(x, t, y_\emptyset)$
5:      $\tilde{s}_\theta \leftarrow (1 + w) s_{\text{cond}} - w s_{\text{uncond}}$
6:      $\mu_t \leftarrow \frac{1}{\sqrt{\alpha_t}} (x + (1 - \alpha_t) \tilde{s}_\theta)$
7:      if $t > 0$ then
8:          $z \sim \mathcal{N}(0, I)$
9:          $x \leftarrow \mu_t + \sqrt{\sigma_t^2} z$
10:     else
11:         $x \leftarrow \mu_t$
12:     end if
13: end for
14: return $x$
```

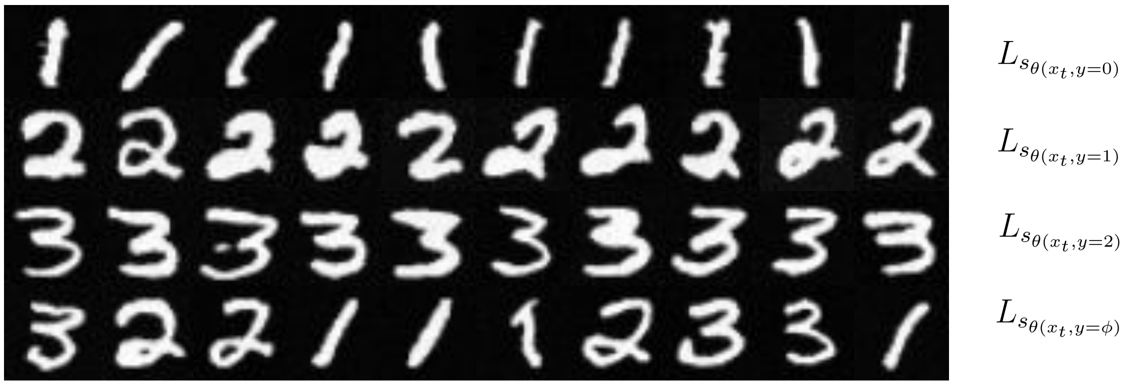
## E Visualizations



(a) Model outputs across different loss functions.

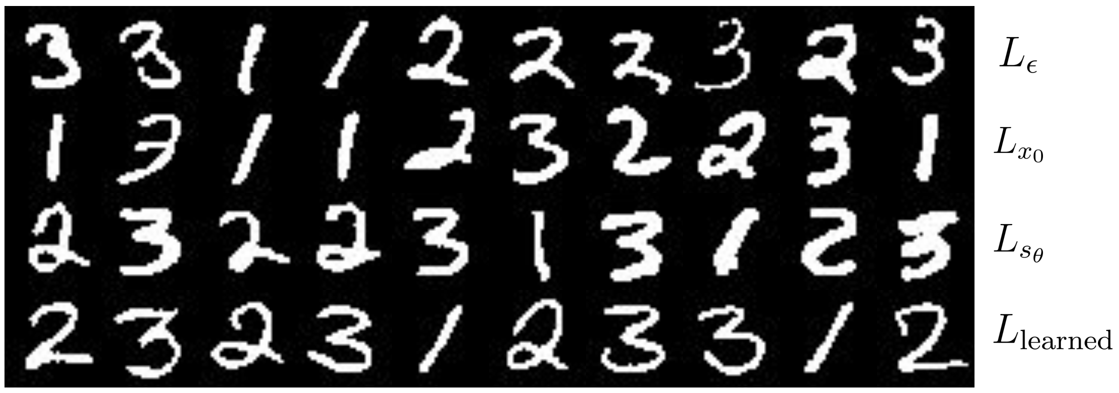

$L_{s_{\theta(x_t, y=0)}}$
$L_{s_{\theta(x_t, y=1)}}$
$L_{s_{\theta(x_t, y=2)}}$

(b) Conditional generation $(w = 1)$

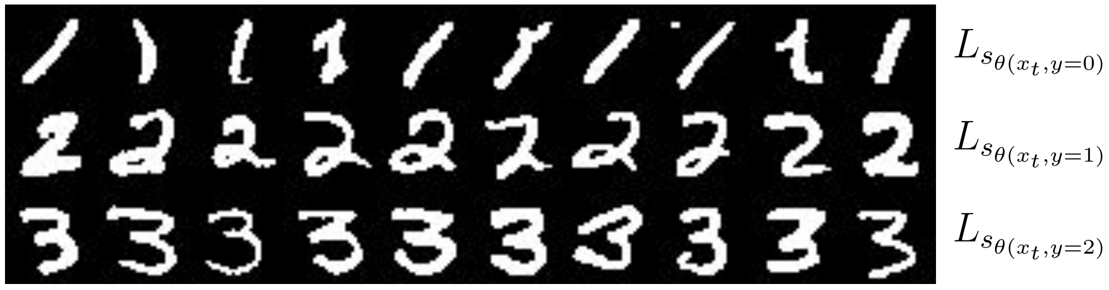

$L_{s_{\theta(x_t, y=0)}}$
$L_{s_{\theta(x_t, y=1)}}$
$L_{s_{\theta(x_t, y=2)}}$
$L_{s_{\theta(x_t, y=\phi)}}$

(c) Conditional generation $(w = 5)$

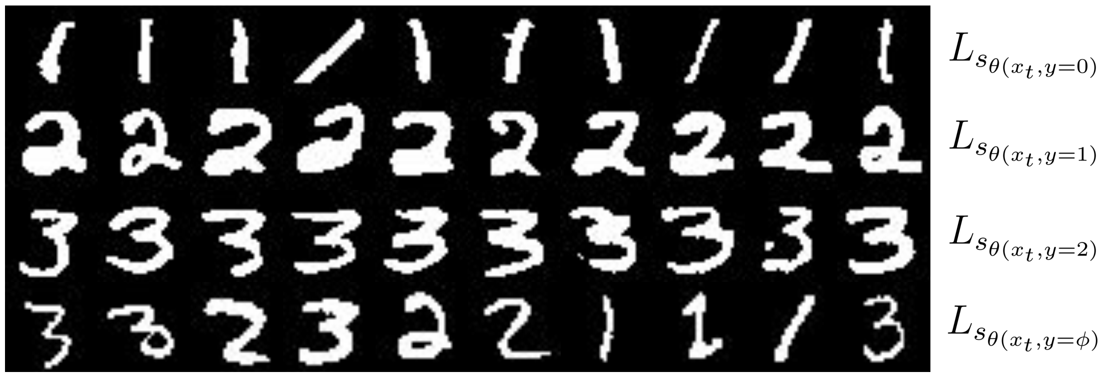
Figure 1: Generated Grayscale images



(a) Model outputs across different loss functions.


$L_{s_{\theta(x_t, y=0)}}$
$L_{s_{\theta(x_t, y=1)}}$
$L_{s_{\theta(x_t, y=2)}}$

(b) Conditional generation $(w = 1)$


$L_{s_{\theta(x_t, y=0)}}$
$L_{s_{\theta(x_t, y=1)}}$
$L_{s_{\theta(x_t, y=2)}}$
$L_{s_{\theta(x_t, y=\phi)}}$

(c) Conditional generation $(w = 5)$

Figure 2: Generated images after binarization.