DENOISING DIFFUSION PROBABILISTIC MODELS (DDPMS)

Samer Bujana (s240033), Rasmus Clausen (s203279),
Alba Gonzalo (s243343), Leonardo Rodovero (s240095).

https://github.com/RCDane/DDPM-DL

ABSTRACT

This work focuses on understanding and implementing DDPMs on the MNIST dataset, highlighting how dynamic schedulers and conditional training with Free Classifier Guidance enhance image quality and generation control.

1. INTRODUCTION

Denoising Diffusion Probabilistic Models (DDPMs) are a class of generative models that have garnered significant attention for their ability to produce high-quality data, particularly in image generation. This project aims to explore the principles underpinning DDPMs and implement them on the MNIST dataset. The report begins by examining the operational mechanisms and mathematical foundations of DDPMs.

Subsequently, to deepen our understanding, we analyze the impact of different scheduling strategies and the number of timesteps on model performance. Through an ablation study, we demonstrate that dynamic schedulers, such as cosine and sigmoid, outperform the linear scheduler in generating high-quality images, as evidenced by FID scores.

Lastly, we improve our implementation by incorporating conditional training through the Free Classifier Guidance, which enables controlled, class-specific image generation.

2. UNDERSTANDING THE PRINCIPLES OF DDPMS

DDPMs generate data by simulating a process that gradually adds noise, called the forward or diffusion process, to an image and then learning how to reverse that process, known as the reverse denoising process. These models are built upon these two main processes, along with the training objective. Furthermore, model performance is evaluated based on metrics such as FID scores. This section provides a comprehensive framework for understanding the key elements that define how DDPMs function.

2.1. Forward Process

The forward or diffusion process can be understood as a fixed Markov chain in which Gaussian noise is gradually added to a sample x_0 according to a variance schedule $\{\beta_t\}$. This schedule determines the amount of noise added in each timestep t, and can be fixed or learned. Using the notation $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, the forward process allows direct sampling of x_t at any timestep t without requiring iterative computations. The closed-form expression is:

$$q(x_t|x_0) = \mathcal{N}\left(x_t; \sqrt{\overline{\alpha}_t}x_0, (1-\overline{\alpha}_t)I\right), \tag{1}$$

where I is the identity matrix. This property enables efficient training by avoiding the need to compute all intermediate steps sequentially.

2.2. Reverse Process

The reverse process is modeled as another Markov chain, which seeks to recover the original sample x_0 by progressively removing the noise from x_T, a pure Gaussian sample. Unlike the forward process, we cannot sample directly from the noised image to the original one, as this transition is not tractable. Instead, the reverse process gradually denoises the sample through a series of steps, each of which involves predicting the added noise and updating the sample.

This process is parameterized as,

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

where μ_θ and Σ_θ are the mean and variance to be learned, respectively. We can set Σ_θ fixed and our task will then be to predict μ_θ at each timestep.

Therefore, the reverse process relies on predicting $\mu_\theta(x_t, t)$. However, a common parameterization shows us that we can express μ_θ in terms of the the added noise at the image at timestep t, $\epsilon_\theta(x_t, t)$,

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}}\epsilon_\theta(x_t, t)\right). \tag{3}$$

The reverse process relies then on predicting either $\mu_\theta(x_t, t)$ or $\epsilon_\theta(x_t, t)$. However, predicting the noise is preferred, as it simplifies the denoising task.

Now, using the properties of the forward process, the posterior $q(x_{t-1}|x_t, x_0)$ can be computed in closed form as,

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I\right), \tag{4}$$

where, $\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\overline{\alpha}_{t-1}}\beta_t}{1-\overline{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1-\overline{\alpha}_{t-1})}{1-\overline{\alpha}_t}x_t$, and $\tilde{\beta}_t = \frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t}\beta_t$.

This formulation connects the reverse process directly with denoising score matching. By substituting $\mu_\theta(x_t, t)$ into the sampling expression, the final update equation for the reverse process becomes,

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sqrt{\beta_t}z, \quad (5)$$

where $z \sim \mathcal{N}(0, I)$ is added to introduce a small amount of noise. This noise term is necessary to prevent the model from perfectly denoising every sample, allowing for a stochastic sampling process. This approach helps preserve the diversity of the generated samples while ensuring that the reverse process still follows the distribution of the original data.

2.3. Training Objective

All of this set, we can train our model to predict the noise at each timestep by optimizing the variational lower bound on the negative log-likelihood of $p_\theta(x_0)$. Using the property of the forward process, this objective decomposes into three terms:

$$\begin{aligned}\mathcal{L} =& \mathbb{E}_q\left[\mathrm{KL}(q(x_T|x_0)\|p(x_T))\right] \\ &+ \sum_{t=2}^{T}\mathrm{KL}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t)) \\ &- \log p_\theta(x_0|x_1).\end{aligned} \quad (6)$$

In practice, the KL divergences between Gaussian distributions simplify the computations. Thus, the final objective used for training can also be expressed as the Mean Squared Error (MSE) between the true noise ϵ and its predicted counterpart $\epsilon_\theta(x_t, t)$:

$$\mathcal{L} = \mathbb{E}_{x_0,\epsilon,t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]. \quad (7)$$

2.4. Model Evaluation using FID

To evaluate the effectiveness of our image generation model, we use the **Frechet Inception Distance (FID)**. FID measures the distance between the feature distributions of real and generated images, providing a quantitative assessment of how similar the generated images are to real ones. The FID score is calculated by comparing the mean and covariance of feature vectors extracted from a network. A lower FID score indicates that the generated images are more similar to the real images. The mathematical formulation of the FID score is provided in the appendix A.

3. U-NET ARCHITECTURE FOR DDPM

The U-Net architecture for DDPMs predicts the noise component at each timestep during the reverse diffusion process. It follows a U-shaped structure consisting of a contracting path (downsampling), a bottleneck (latent space), and an expanding path (upsampling), as shown in Fig. 1. Skip connections between corresponding levels of the contracting and expanding paths preserve spatial details and enhance reconstruction quality. Key components include Residual Blocks, Time Embedding, and Self-Attention Blocks.

3.1. Residual Blocks

Residual Blocks (ResBlocks) are central to the U-Net, stabilizing training with residual connections and enabling efficient gradient flow. These blocks are conditioned on the timestep embedding, which adapts their operations to the specific noise level at a given timestep. For detailed mathematical formulations, see Appendix B.1.

3.2. Time Embedding

Time embeddings condition the U-Net on the timestep t, incorporating temporal information into the feature maps. These embeddings are generated using sinusoidal positional encoding and projected into the feature space. The mathematical details are provided in Appendix B.2.

3.3. Self-Attention Blocks

Self-Attention Blocks capture global spatial dependencies by computing relationships between all spatial positions in the feature map, complementing the local feature extraction of convolutions. This mechanism enhances the model's ability to process both global structures and fine-grained details. The mathematical formulation is detailed in Appendix B.3.

3.4. Overall Design

The downsampling path reduces spatial resolution through DownBlocks, which include ResBlocks and Self-Attention Blocks. The bottleneck applies stacked ResBlocks conditioned on the timestep embedding, enabling abstract feature extraction. The upsampling path mirrors the downsampling path, increasing resolution through UpBlocks and utilizing skip connections to retain fine-grained spatial information.

4. EXPERIMENTAL SETUP AND TRAINING PROCEDURE

Our experiments were conducted on the MNIST dataset, which provides 60,000 training images and 10,000 test images. Each image was resized to 32×32 pixels and normalized for consistency. We separated the original training set into an $80\% - 20\%$ split, using 48,000 images for training and 12,000 for validation. This partition facilitated both hyperparameter tuning and performance monitoring throughout training.

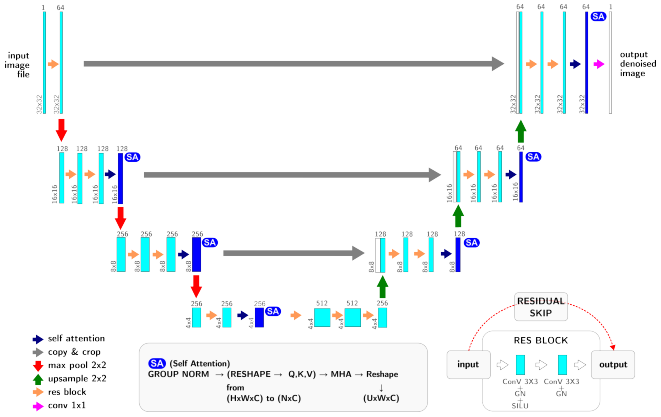

Fig. 1. U-Net architecture employed in the DDPM implementation.

Specifically, the U-Net architecture (initialized with a time-embedding dimension of 256) was optimized via the loss of mean squared error (MSE) between the true noise and the predicted noise of the model at each time step. We trained for $n = 20$ epochs using Adam with a learning rate of 0.0001 and batch size 64. During each iteration, a random timestep $t \in \{1, \ldots, T\}$ was selected, and Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$ was added to the input image to produce x_t. The U-Net then attempted to predict ϵ, and the resulting MSE was backpropagated to update model parameters. Validation-loss measurements at each epoch guided the monitoring of overfitting and generalization.

5. IMPACT OF SCHEDULERS AND TIMESTEP SIZE

Schedulers and the number of timesteps selected for the process play a critical role in training the model by defining how noise is interpolated into the images during the denoising process. As discussed earlier, the scheduling strategy significantly influences the performance of the model, particularly for datasets with varying resolutions.

To understand their impact on image generation, we employed different numbers of time steps and variance schedules in the DDPMs. Initially, we adopted the linear scheduler as described in [1]. Subsequently, we tested the cosine scheduler, inspired by the advancements in [2], which highlight that the linear scheduler is less effective for generating lower-resolution images.

Finally, we also explored the sigmoid scheduler, introduced in [3] and [4]. This scheduler offers smoother noise interpolation compared to the abrupt changes of the linear scheduler, and its variance curve adapts more flexibly than that of the cosine scheduler in certain scenarios. This makes it particularly advantageous for datasets with complex patterns or higher variability in resolution, where balancing the noise distribution across timesteps is important.

Figure 2 depicts how the noise is interpolated using the linear, cosine, and sigmoid schedulers, respectively. Here we can see how the linear scheduler introduces noise abruptly, especially towards the end of the forward noising process, where it becomes excessive and contributes little to sample quality. In contrast, both the cosine and sigmoid schedulers add noise more progressively, which will allow the model to adapt more effectively and achieve smoother denoising during the reverse process, thereby improving overall image quality.

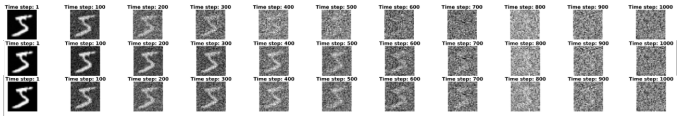

Fig. 2. Forward Process for Different Schedulers.

5.1. Results and Analysis

We trained nine different models, one for each scheduler with timestep size of 100, 500, and 1000. Then, we calculated the FID scores of each to compare them and see their different performances and results, see Table 1, for a more visual understanding Figure 5 in the Appendix C.2 can be seen.

Timesteps	100	500	1000
Linear Scheduler	28.247	9.239	**7.823**
Cosine Scheduler	9.207	7.023	**5.735**
Sigmoid Scheduler	7.734	5.773	**5.039**

Table 1. FID scores for the different schedulers and timesteps, training for 20 epochs.

As expected, we found that both the cosine and sigmoid schedulers outperformed the linear scheduler, with the sigmoid showing a slight advantage in performance. This suggests that more dynamic scheduling approaches are better suited for the task compared to a constant rate of change. The progressive addition of noise in these schedulers likely facilitates more effective learning, allowing the model to adapt gradually and stabilize during training.

5.1.1. Generated Images

We took the best models for each scheduler, which, coincidentally, for all of them it was when they had a timestep size of a 1000. As the timestep size increases, the noise added during each diffusion step and removed during each reverse step decreases. This likely explains why larger timesteps improve the model's performance. However, there may be an ideal timestep size, as too little noise could make it harder for the model to learn effectively.

Figure 3 presents the results of the generated images for each model. Overall, the results are satisfactory, even though the readability in certain cases might be questionable.

Fig. 3. Generated images for the linear, cosine, and sigmoid schedulers with a 1000 timesteps, respectively.

6. CLASSIFIER FREE GUIDANCE

With a standard DDPM model, it is possible to sample generated images from the learned distribution, in this case MNIST digits. However, with this, it is not possible to choose which class of digits to generate or control the quality of the samples. Here we implement Classifier Free Guidance which makes it possible to condition the model to generate specific digits and allows some control of the quality of the final images.

6.1. Conditional training and guidance

Classifier Free Guidance works by training a DDPM model both conditionally and unconditionally. What makes the model conditional is that during training, it is given a learnable embedding representing the class label of the image that it is predicting noise from. For our model, we concatenate the class embedding onto the feature maps before all **UpBlocks**, **DownBlocks** and the **ResBlocks** in the latent space of the model. To also train it unconditionally, we with some probability $p < 1$ instead of the class embedding it is given a zero embedding $\emptyset$, which is the embedding filled with zeros. This way, the model learns to predict noise both conditionally and unconditionally.

What makes this method guided, is that we then, during the backwards process, can use both the conditional and unconditional parts of the model to guide our noise predictions. This is done using the equation 8.

$$\boldsymbol{\epsilon}_{\text{CFG}} = (1 + w) \cdot \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - w \cdot \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \tag{8}$$

Where $e_\theta(x_t, t, y)$ is the predicted conditional noise given the class label y, $e_\theta(x_t, t)$ is the unconditional noise prediction and w is the guidance weight. If we think of the two noise predictions as vectors, then we take the vector from the unconditional noise to the conditional one and scale it by w. for $0 < w < 1$ this is an interpolation between the noises, while for $w > 1$ it works as an extrapolation.

For our this experiment we used the same setup as for the basic DDPM model with a Linear Scheduler, with a few changes. We trained the model for 30 epochs, with a propablility $\mathbf{p} = 0.2$ of training unconditionally. The higher amount of epochs was to ensure that the model had time to learn both the conditional and unconditional task.

6.2. Classifier Free Guidance Results

To evaluate our CFG DDPM model, calculated the FID score between the MNIST test set and a sampled set of digits from our model. We chose to generate 200 images from each digit class. The resulting FID scores can be seen on Table 2. With a guidance weight $w = 0$, we have a FID score of 1.792, which is better than for our earlier results. this is the same as sampling the model conditionally, without guidance. So the improved FID makes sense, since the model has more class information. With $w = 1$ we get the best results, and as w increases the FID score gets worse. The reason for this degradation as w increases is that the generated samples lose variety, which can be seen on 4 thus making the generated images more alike. This means that the distribution of the generated images is not as varied as the original MNIST dataset, which results in a worse FID.

weight(w)	0	0.1	0.2	0.5	1	2	10	50
FID	1.792	1.113	1.14	0.608	0.464	0.578	1.839	14.669

Table 2. FIDs for different guidance values w.

Fig. 4. Visualization of digits sampled using CFG with different classification weights w.

7. CONCLUSION

In summary, our findings indicate that the **cosine** and **sigmoid** schedulers consistently outperform the linear baseline in FID-based evaluations, suggesting that smoother noise interpolation enhances the quality of generated digits. Furthermore, the optimal number of timesteps was identified as $T = 1000$.

Additionally, the integration of **Classifier-Free Guidance** enables class-conditional sampling, offering a well-balanced trade-off between fidelity and diversity. In particular, a guidance weight of $w = 1$ resulted in the best FID score among the variants tested. These results emphasize the key role noise scheduling, timesteps, and guidance strategies in improving both the realism and controllability of DDPM-generated images.

8. REFERENCES

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[2] Alexander Quinn Nichol and Prafulla Dhariwal, "Improved denoising diffusion probabilistic models," *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[3] Tianjun Chen et al., "On the importance of noise scheduling for diffusion models," *arXiv preprint arXiv:2301.10972*, 2023.

[4] Allan Jabri, David J. Fleet, and Ting Chen, "Scalable adaptive computation for iterative generation," *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

9. APPENDIX

In this appendix, we present additional developments and details that, while not central to the main body of the work, may provide valuable insights for readers interested in exploring further aspects of the study. These supplementary materials include technical implementations and extended experiments that could enhance the understanding of the methodologies and results discussed in the main text.

A. FID SCORE MATHEMATICAL FORMULATION

The FID score measures the similarity between the feature distributions of real images and generated images using the following formula:

$$\text{FID}(P, Q) = \|\mu_P - \mu_Q\|_2^2 + \text{Tr}(C_P + C_Q - 2\sqrt{C_P C_Q}) \quad (9)$$

Where,

- μ_P and μ_Q are the means of the feature distributions for the real (P) and generated (Q) images.
- C_P and C_Q are the covariances of these feature distributions.
- $\| \cdot \|_2$ denotes the L_2 norm.
- $\text{Tr}(\cdot)$ refers to the trace of the matrix.

B. MATHEMATICAL FORMULATIONS OF U-NET COMPONENTS

B.1. Residual Block Formulation

A Residual Block applies the following sequence of operations:

$$f_{\text{ResBlock}}(x) = \text{GN}(\text{Conv}(\text{SiLU}(\text{GN}(\text{Conv}(x))))), \quad (10)$$

where Conv is a 3×3 convolution, GN denotes Group Normalization, and $\text{SiLU}(x) = x \cdot \sigma(x)$ is the activation function. The defining feature of the ResBlock is its residual connection:

$$\text{Output} = x + f_{\text{ResBlock}}(x), \quad (11)$$

which stabilizes gradient flow and ensures efficient training.

B.2. Time Embedding Formulation

The time embedding is computed using sinusoidal positional encoding:

$$\text{TE}(t) = \text{Concat}\left(\sin(t \cdot \omega_i), \cos(t \cdot \omega_i)\right)_{i=1}^{d}, \quad (12)$$

where $\omega_i = \frac{1}{10000^{2i/d}}$ and d is the embedding dimension. The resulting embedding is projected into the feature space:

$$\text{TE}_{\text{proj}}(t) = W_t \cdot \text{TE}(t) + b_t, \quad (13)$$

where W_t and b_t are learnable parameters. The embedding is then added to the feature maps:

$$x_{\text{updated}} = x + \text{TE}_{\text{proj}}(t). \quad (14)$$

B.3. Self-Attention Mathematical Formulation

Given a feature map $x \in \mathbb{R}^{B \times C \times H \times W}$, where B is the batch size, C is the number of channels, and H, W are spatial dimensions, the Self-Attention mechanism first reshapes it into a sequence of tokens $X \in \mathbb{R}^{B \times N \times C}$, where $N = H \cdot W$.

Three projections, Queries (Q), Keys (K), and Values (V), are computed as:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V \quad (15)$$

where $W_Q, W_K, W_V \in \mathbb{R}^{C \times d}$ are learnable weight matrices, and d is the attention dimension.

The attention scores are calculated as:

$$\text{Attention Scores} = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) \quad (16)$$

These scores determine how much each token (spatial position) attends to every other token. The output of the Self-Attention block is then computed as:

$$\text{Output} = \text{Attention Scores} \cdot V \quad (17)$$

Finally, the output is reshaped back to $\mathbb{R}^{B \times C \times H \times W}$ and combined with the input via a residual connection:

$$x_{\text{updated}} = x + \text{Output} \quad (18)$$

This formulation allows the model to aggregate information across all spatial positions, enabling it to model both global and local features effectively.

C. SCHEDULERS AND TIME STEPS

C.1. Schedulers formulations

The mathematical definitions for the schedulers employed in the report are presented below.

- *Linear*:

$$\boldsymbol{\beta}_t = \beta_{start} - t \cdot \frac{\beta_{end} - \beta_{start}}{T} \tag{19}$$

  Where T is the total number of timesteps.

- *Cosine*:

$$\overline{\boldsymbol{\alpha}}_t = \frac{f(t)}{f(0)}, \quad \boldsymbol{f}_t = \cos^2\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right) \tag{20}$$

  Where s controls the offset for smoother transitions and $\overline{\alpha_t}$ is the mean of the cumulative product of $1 - \beta_t$.

- *Sigmoid*:

$$\boldsymbol{\alpha}_t = \frac{1}{1 + e^{-a \cdot (t/T - b)}} = \sigma(a \cdot (t/T + b)). \tag{21}$$

  Where a controls the steepness of the curve and b shifts the curve along the timeline.

C.2. Further Results and Analysis

We include a graph showing the FID scores and results obtained in section 5.1.

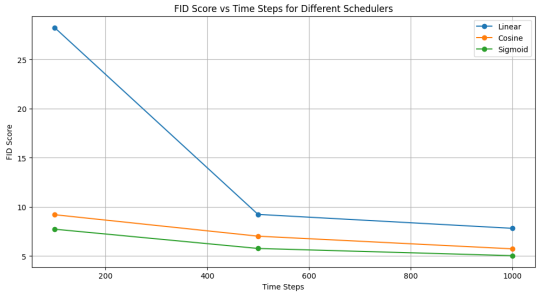

Fig. 5. FID Scores Over Timesteps.