CS726 : Advanced Machine Learning

Indian Institute Of Technology, Bombay

Programming Assignment 2

Author:
Arijeet Mondal : 24m0812
Daksh Goyal : 24m0756
Rahul Choudhary : 200070065

Instructor:
Prof. Sunita Sarawagi

March 17, 2025

Contents

1 Denoising Diffusion Probabilistic Models

1.1 Implementation

Algorithm 1 Training Algorithm for DDPM

Require: Model θ, NoiseScheduler S, Dataset D, Learning Rate η, Epochs E

for epoch = 1 to E **do**
  for mini-batch (x_0, y) in D **do**
    Sample $t \sim U(\{1, \ldots, T\})$ ▷ Uniform timestep sampling
    Sample $\epsilon \sim \mathcal{N}(0, I)$ ▷ Gaussian noise
    Compute $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$ ▷ Noisy sample generation
    Predict noise: $\hat{\epsilon} = \theta(x_t, t)$
    Compute loss: $L = \|\epsilon - \hat{\epsilon}\|^2$ ▷ Mean Squared Error Loss
    Update model parameters: $\theta \leftarrow \theta - \eta \nabla_\theta L$
  end for
end for

Algorithm 2 Sampling Algorithm for DDPM

Require: Model θ, NoiseScheduler S, Number of Samples N

Initialize $x_T \sim \mathcal{N}(0, I)$ ▷ Start from Gaussian noise
for $t = T$ to 1 **do**
  Predict noise: $\hat{\epsilon} = \theta(x_t, t)$
  Compute $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\hat{\epsilon}\right) + \sigma_t z$ ▷ Reverse process
  Sample $z \sim \mathcal{N}(0, I)$ if $t > 1$ else $z = 0$
end for
Return x_0

Model Architecture Description

Component	Description
Input Dimension	n_{dim}
Time Embedding	Embedding Layer → Linear(256) → ReLU → Linear(256)
Noise Prediction Model	Linear(n_{dim} + 256, 512) LayerNorm(512) ResidualBlock(512, 512) ResidualBlock(512, 512) Linear(512, n_{dim})
Optimizer	Adam (η = 1e-4)
Loss Function	Mean Squared Error (MSE)
Scheduler Types	Linear, Cosine, Sigmoid

Table 1: Description of DDPM Model Architecture

1.2 Effects of Hyper Parameter: Number of diffusion steps (T)

For this, we use the linear noise scheduler. The linear noise schedule in DDPM defines a sequence of noise variance values (β_t) that increase linearly over time. It is formulated as:

$$\beta_t = \beta_{\text{start}} + \frac{t}{T}(\beta_{\text{end}} - \beta_{\text{start}}), \quad t \in \{0, 1, ..., T-1\} \tag{1}$$

where:

- β_{start} is the initial noise variance.
- β_{end} is the final noise variance.
- T is the total number of timesteps.
- t is the current timestep.

This schedule ensures a gradual increase in noise, allowing for a smoother denoising process during inference. The corresponding α_t and cumulative product $\bar{\alpha}_t$ values are computed as:

$$\alpha_t = 1 - \beta_t \tag{2}$$

$$\bar{\alpha}_t = \prod_{i=0}^{t} \alpha_i \tag{3}$$

These values control how much information from the original data is preserved as noise is added during the forward diffusion process.
The following hyperparamaters are kept fixed during training:

Parameter	**Value**
β_{start}	0.00001
β_{end}	0.02
Epochs	2500
Batch Size	128
Learning Rate (η)	0.0001

Table 2: Fixed Hyperparameters for the Model

We get 8000 samples from the trained model and use Earth Mover Distance (EMD) with subsample size as 600, sampled over 5 iterations as evaluation metric along with Negative Log-Likelihood (NLL).

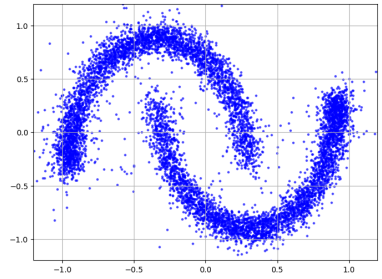
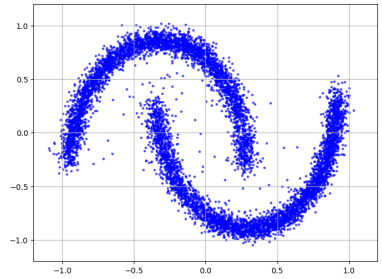
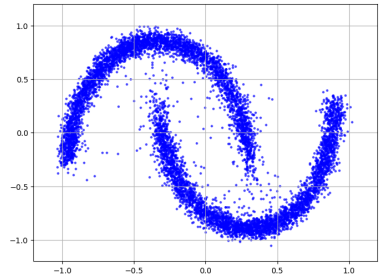
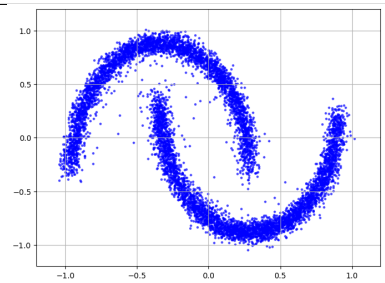
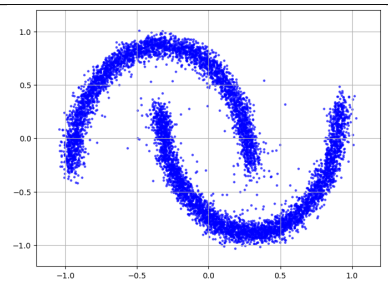
T	EMD	NLL	Image
10	79.732 ± 6.477	1.018	
50	60.756 ± 10.643	0.966	
100	58.621 ± 11.631	0.960	
150	62.365 ± 18.658	0.947	
200	68.356 ± 17.349	0.956	

Table 3: EMD and NLL values for the Moons dataset.

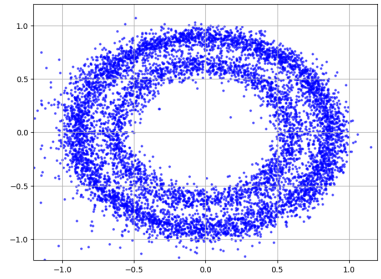
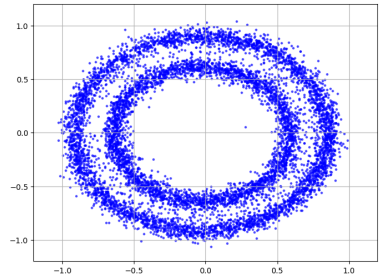
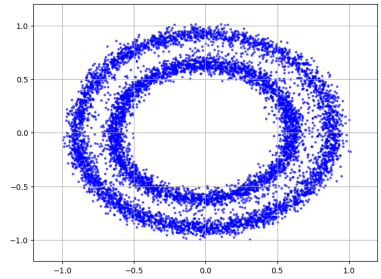
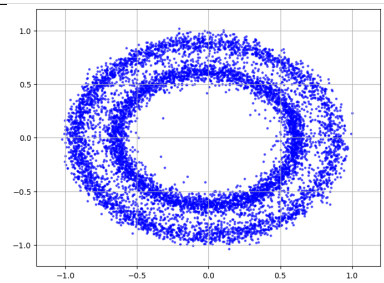
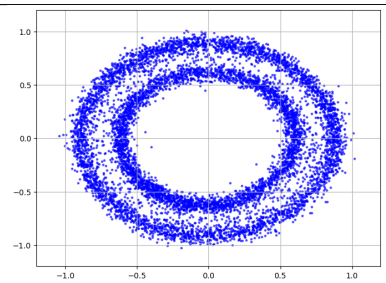
T	EMD	NLL	Image
10	61.116 ± 12.305	1.035	
50	56.599 ± 11.507	1.004	
100	53.169 ± 8.754	0.999	
150	62.565 ± 8.682	0.989	
200	52.033 ± 7.443	0.993	

Table 4: EMD and NLL values for the Circles dataset.

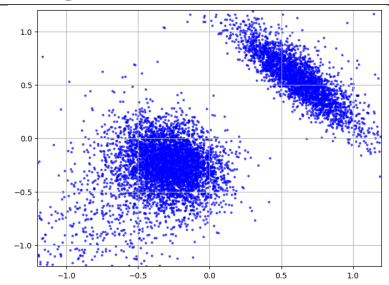
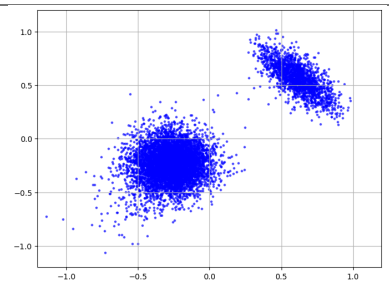
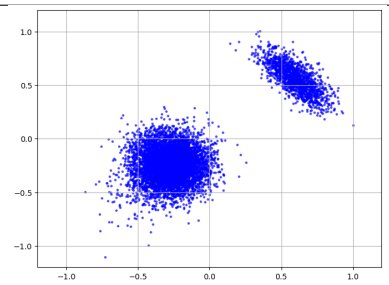
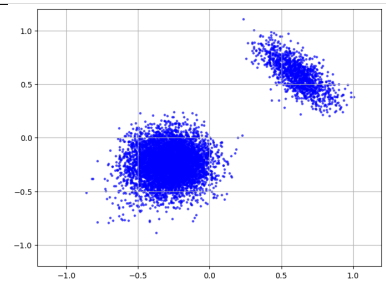
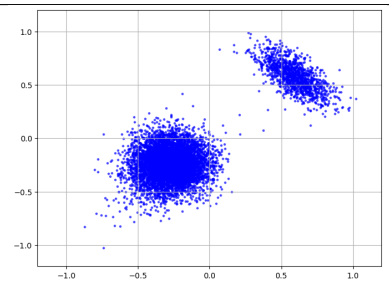
T	EMD	NLL	Image
10	188.587 ± 5.196	0.347	
50	54.715 ± 4.612	0.078	
100	27.214 ± 4.273	0.035	
150	23.273 ± 3.269	0.020	
200	24.324 ± 3.639	0.014	

Table 5: EMD and NLL values for the Blobs dataset.

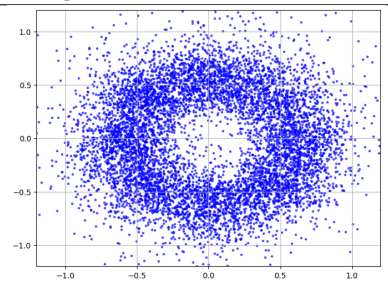
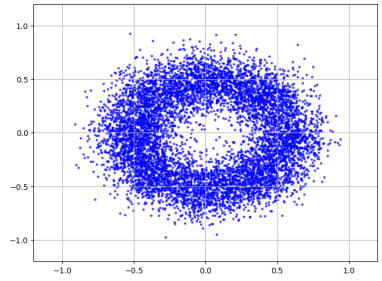
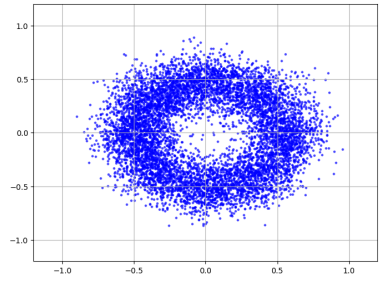
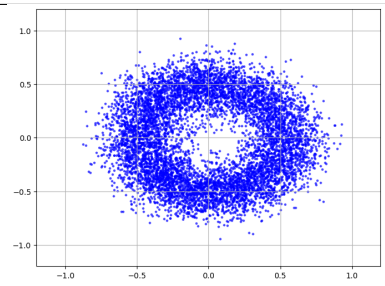
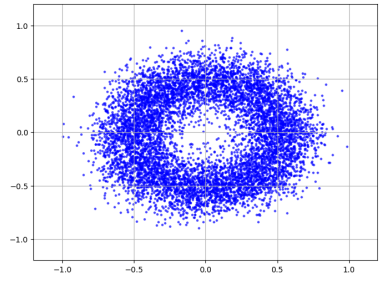
T	EMD	NLL	Image
10	83.919 ± 7.517	0.706	
50	34.212 ± 4.296	0.554	
100	39.778 ± 5.434	0.534	
150	40.291 ± 4.855	0.535	
200	36.824 ± 3.988	0.551	

Table 6: EMD and NLL values for the Manycircles dataset.

1.3 Effects of Hyper Parameter: Noise schedule

1.3.1 Linear noise schedule

In this evaluation all are hyperparameters other than noise schedule parameters are kept same and the timesteps is takes as 50.

β_{start}	β_{end}	**EMD**	**NLL**
0.00001	0.02	60.756 ± 10.643	0.966
0.00001	0.05	62.485 ± 12.627	0.961
0.00001	0.01	58.613 ± 8.736	0.984
0.00005	0.02	61.183 ± 10.650	0.966
0.0001	0.02	60.614 ± 11.222	0.962

Table 7: Earth Mover Distance and Negative Log Likelihood values for the Moons dataset.

β_{start}	β_{end}	**EMD**	**NLL**
0.00001	0.02	56.599 ± 11.507	1.004
0.00001	0.05	53.415 ± 5.964	0.994
0.00001	0.01	55.351 ± 12.057	1.012
0.00005	0.02	56.740 ± 11.671	1.004
0.0001	0.02	55.507 ± 11.315	1.002

Table 8: Earth Mover Distance and Negative Log Likelihood values for the Circles dataset.

β_{start}	β_{end}	**EMD**	**NLL**
0.00001	0.02	54.715 ± 4.612	0.078
0.00001	0.05	25.121 ± 4.774	0.022
0.00001	0.01	111.235 ± 9.246	0.172
0.00005	0.02	54.217 ± 4.806	0.078
0.0001	0.02	53.834 ± 4.857	0.077

Table 9: Earth Mover Distance and Negative Log Likelihood values for the Blobs dataset.

β_{start}	β_{end}	**EMD**	**NLL**
0.00001	0.02	34.212 ± 4.296	0.554
0.00001	0.05	35.682 ± 4.331	0.548
0.00001	0.01	37.433 ± 3.221	0.584
0.00005	0.02	34.246 ± 4.094	0.555
0.0001	0.02	34.013 ± 4.158	0.555

Table 10: Earth Mover Distance and Negative Log Likelihood values for the Manycircles dataset.

1.3.2 Cosine noise schedule

The cosine noise schedule is designed to smoothly control noise levels during the forward diffusion process. It defines the cumulative product of the noise scales $\bar{\alpha}_t$ as follows:

$$\bar{\alpha}_t = \frac{\cos^2\left(\frac{t/T+s}{1+s} \cdot \frac{\pi}{2}\right)}{\cos^2\left(\frac{s}{1+s} \cdot \frac{\pi}{2}\right)} \tag{4}$$

where:

- t is the current timestep.
- T is the total number of timesteps.
- s is a small offset (commonly $s = 0.008$) to ensure $\bar{\alpha}_0 \approx 1$.

From the cumulative product of alphas, the noise variance β_t is derived as:

$$\beta_t = 1 - \frac{\bar{\alpha}_{t+1}}{\bar{\alpha}_t} \tag{5}$$

This formulation ensures that noise variance remains positive and is typically clamped within the range $[0, 0.999]$ for numerical stability.

The individual alpha values are computed as:

$$\alpha_t = 1 - \beta_t \tag{6}$$

The cumulative product of alphas, $\bar{\alpha}_t$, follows:

$$\bar{\alpha}_t = \prod_{i=0}^{t} \alpha_i \tag{7}$$

In this evaluation all hyperparameters other than noise schedule parameters are kept same and the timesteps are taken as 50. We took s = 0.08 for the following evaluation.

Dataset	**EMD**	**NLL**
Moons	65.867 ± 14.765	0.960
Circles	54.496 ± 10.854	0.985
Blobs	57.278 ± 33.854	0.013
Manycircles	49.991 ± 27.414	0.551

Table 11: Comparison of EMD and NLL for Different Datasets For Cosine Noise Schedule

1.3.3 Sigmoid noise schedule

The sigmoid noise schedule is designed to control the noise variance in a smooth, non-linear fashion during the diffusion process. It leverages the sigmoid function to gradually transition the noise variance over time.

The sigmoid noise schedule maps the time step t to a cumulative noise level using:

$$\gamma_t = \frac{v_{\text{end}} - \sigma\left(\frac{t \cdot (\text{end} - \text{start}) + \text{start}}{\tau}\right)}{v_{\text{end}} - v_{\text{start}}} \tag{8}$$

Where:

- γ_t is the cumulative product of α values (also known as α_{bar}).
- $v_{\text{start}} = \sigma\left(-\frac{\text{start}}{\tau}\right)$
- $v_{\text{end}} = \sigma\left(-\frac{\text{end}}{\tau}\right)$

- start, end, τ are hyperparameters that control the curve's shape and transition speed.

The individual alpha values are computed as:

$$\alpha_t = \frac{\gamma_t}{\max(\gamma_{t-1}, \epsilon)} \tag{9}$$

Where ϵ is a small value (e.g., 10^{-9}) to avoid division by zero.
The noise variance β_t is derived as:

$$\beta_t = 1 - \alpha_t \tag{10}$$

Dataset	**EMD**	**NLL**
Moons	66.100 ± 8.540	0.909
Circles	70.310 ± 9.389	0.952
Blobs	49.914 ± 7.799	-0.055
Manycircles	54.099 ± 3.601	0.480

Table 12: Comparison of EMD and NLL for Different Datasets for Sigmoid Noise Schedule

1.4 Hyperparameters for Albatross Dataset

The following table summarizes the hyperparameters used for the Albatross dataset (linear noise schedule is used):

Hyperparameter	**Value**
T	200
β_{start}	0.00001
β_{end}	0.02
Epochs	2500
Batch Size	128
Learning Rate (η)	0.0001

Table 13: Hyperparameters for Albatross Dataset

2 Classifier-Free Guidance

2.1 Guided Sampling v/s Conditional Sampling

Conditional Sampling: Conditional sampling is a technique where the model is trained to predict noise while directly conditioning on class labels. This approach requires labeled data during training and leverages the provided labels to guide the sampling process. The noise prediction network is defined as follows:

$$\epsilon_\theta(x_t, y) \tag{11}$$

Algorithm 3 Conditional Sampling

Require: $x_T \sim \mathcal{N}(0, I)$, NoiseScheduler β_t, Model $\epsilon_\theta(x_t, y)$
1: **for** $t = T$ to 1 **do**
2: Sample $z \sim \mathcal{N}(0, I)$ if $t > 1$ else $z = 0$
3: $x_{t-1} = \frac{1}{\sqrt{1-\beta_t}} (x_t - \beta_t \epsilon_\theta(x_t, y)) + \sqrt{\beta_t} z$
4: **end for**
5: **return** x_0

In this method, the network's ability to predict the noise conditioned on the class label ensures the generated sample aligns with the desired class during inference.

Guided Sampling (Classifier-Free Guidance): Guided sampling using Classifier-Free Guidance enhances sample quality by combining predictions from both a class-conditional and an unconditional noise prediction model. This method is designed to improve control over sample diversity and fidelity using a guidance scale parameter, w. The noise prediction is defined as follows:

$$\hat{\epsilon}_\theta(x_t, y) = (1 + w)\epsilon_\theta(x_t, y) - w\epsilon_\theta(x_t) \tag{12}$$

Here, $\epsilon_\theta(x_t, y)$ is the class-conditional noise prediction, while $\epsilon_\theta(x_t)$ is the unconditional prediction. The guidance scale w controls the trade-off between sample fidelity and diversity. Larger values of w result in stronger conditioning, improving class alignment but potentially reducing sample diversity.

The guided sampling procedure follows this algorithm:

Algorithm 4 Guided Sampling (Classifier-Free Guidance)

Require: $x_T \sim \mathcal{N}(0, I)$, NoiseScheduler β_t, Model $\epsilon_\theta(x_t, y)$ and $\epsilon_\theta(x_t)$, Guidance scale w

1: **for** $t = T$ to 1 **do**
2: Sample $z \sim \mathcal{N}(0, I)$ if $t > 1$ else $z = 0$
3: $\hat{\epsilon}_\theta(x_t, y) = (1 + w)\epsilon_\theta(x_t, y) - w\epsilon_\theta(x_t)$
4: $x_{t-1} = \frac{1}{\sqrt{1-\beta_t}} \left(x_t - \beta_t \hat{\epsilon}_\theta(x_t, y)\right) + \sqrt{\beta_t} z$
5: **end for**
6: **return** x_0

By adjusting the guidance scale w, we can effectively control the strength of conditional guidance during sampling, balancing between sample diversity and the quality of generated outputs.

The following tables show the comparison between Conditional Sampling and Guided Sampling. In Guided Sampling we trained with 10% of the labels being masked and the guidance scale as 2.0.

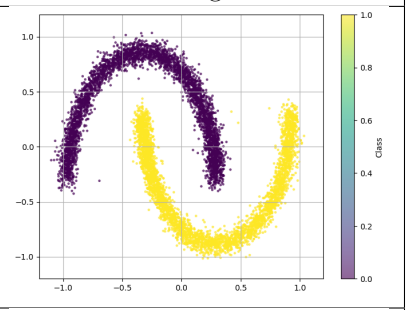
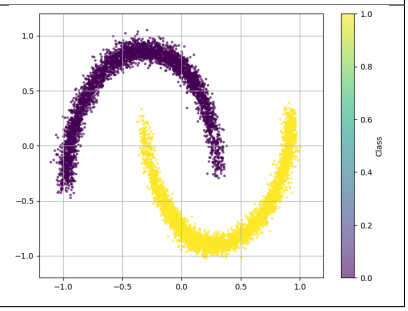
	EMD	NLL	Image
Conditional Sampling	55.704 ± 3.335	0.949	
Guided Sampling	69.427 ± 11.104	0.981	

Table 14: Comparison of Sampling on Moons dataset

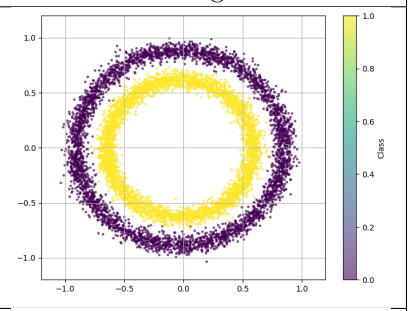
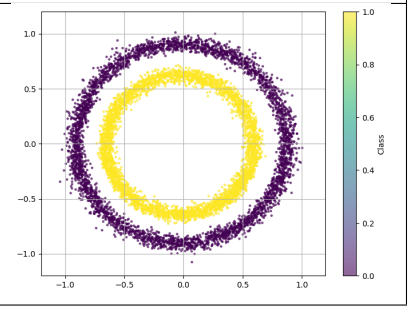
	EMD	NLL	Image
Conditional Sampling	50.783 ± 9.094	0.986	
Guided Sampling	54.878 ± 11.119	0.998	

Table 15: Comparison of Sampling on Circles dataset

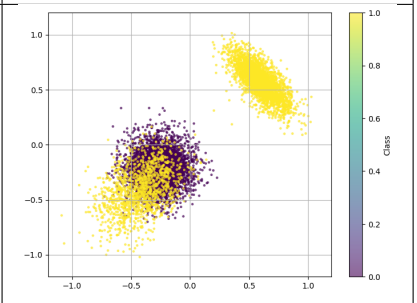
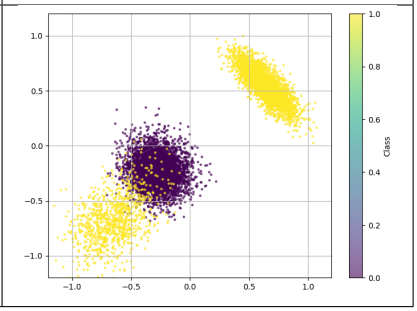
	EMD	NLL	Image
Conditional Sampling	129.315 ± 12.943	0.210	
Guided Sampling	176.761 ± 15.484	0.306	

Table 16: Comparison of Sampling on Blobs dataset

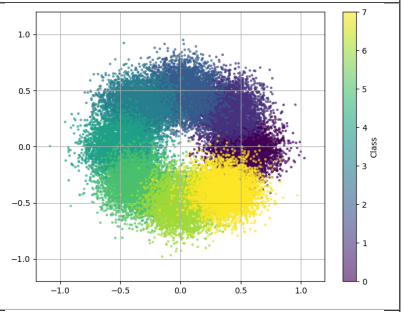
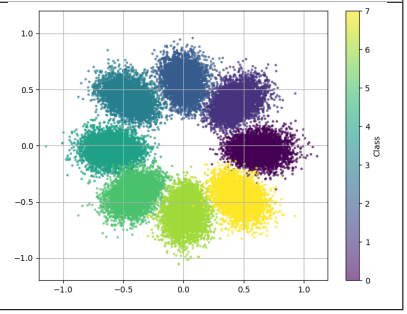
	EMD	NLL	Image
Conditional Sampling	36.668 ± 1.858	0.536	
Guided Sampling	57.367 ± 3.871	0.635	

Table 17: Comparison of Sampling on Manycircles dataset

2.2 Effect of Guidance Scale

This evaluation method involves training a classifier on real data and then evaluating its performance on samples generated by the diffusion model.

A multi-layer perceptron (MLP) classifier is trained on real data. The dataset is split into training and test sets. The classifier's overall accuracy and per-class accuracies on the test set are recorded as benchmarks.

For each guidance scale w, the diffusion model generates samples for each class. These samples are then fed into the pre-trained classifier. The classifier's predictions are compared with the intended class labels:

- Overall Accuracy: The proportion of all generated samples that the classifier correctly classifies.
- Per-Class Accuracy: The fraction of samples for each individual class that are correctly classified.

The following images show the effect of the guidance scale on the quality of the generated data. The guidance scale used are 0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 2.0, 3.0.

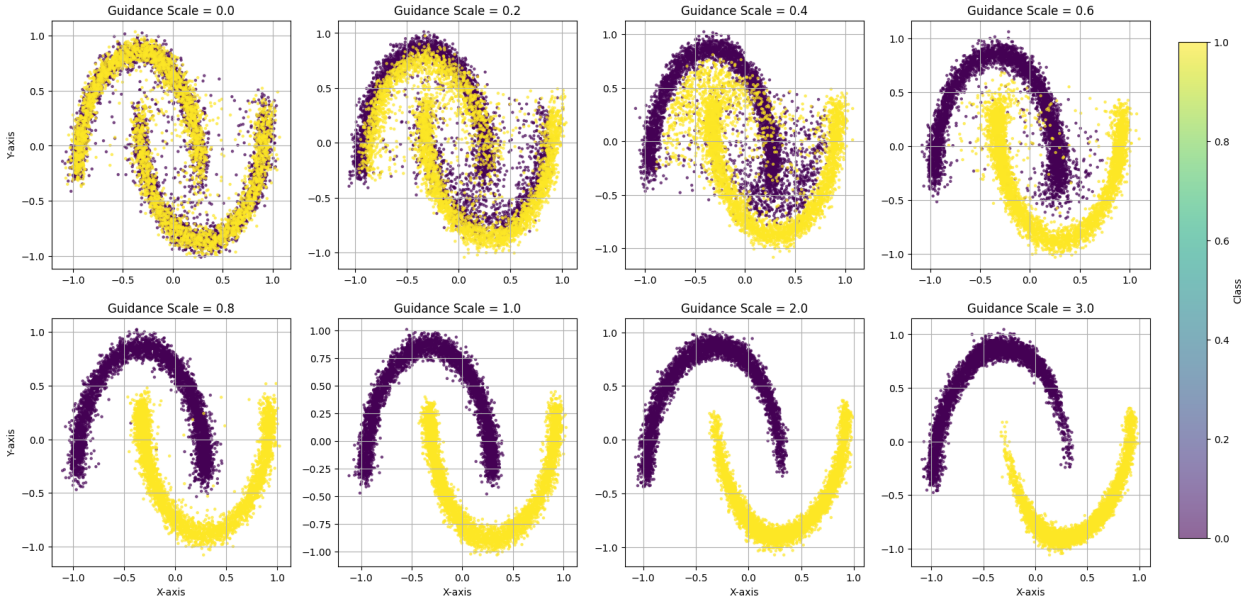


Figure 1: Effect of guidance scales on the generated data for moons dataset

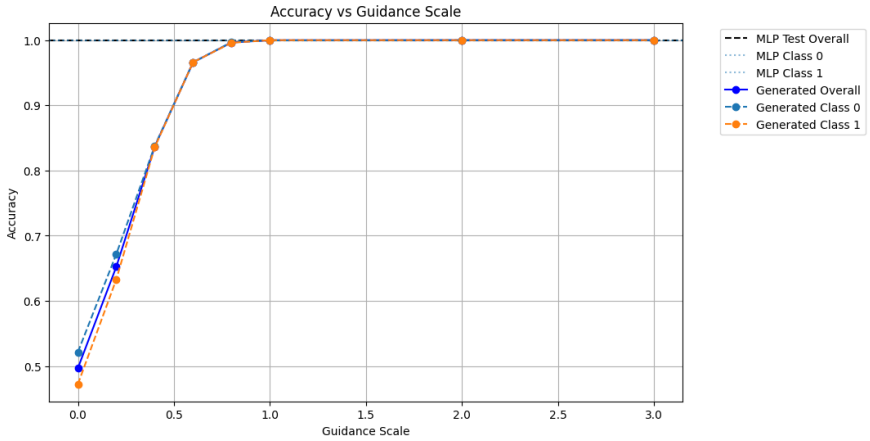


Figure 2: Accuracy of generated data for moons dataset

Fig 1 and Fig 2 shows the effect of the guidance scales on the moon dataset.

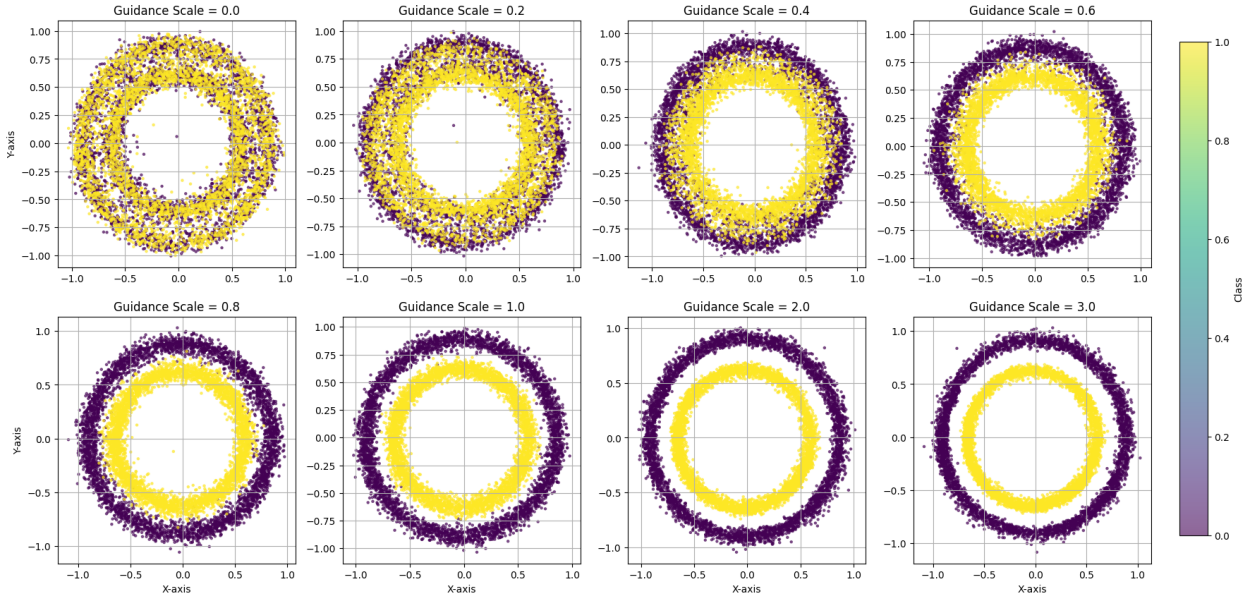


Figure 3: Effect of guidance scales on the generated data for circles dataset

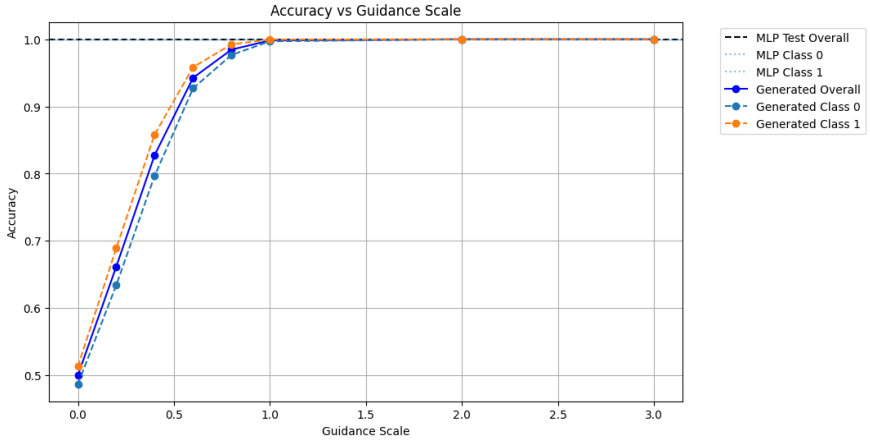


Figure 4: Accuracy of generated data for circles dataset

Fig 3 and Fig 4 shows the effect of the guidance scales on the circles dataset.

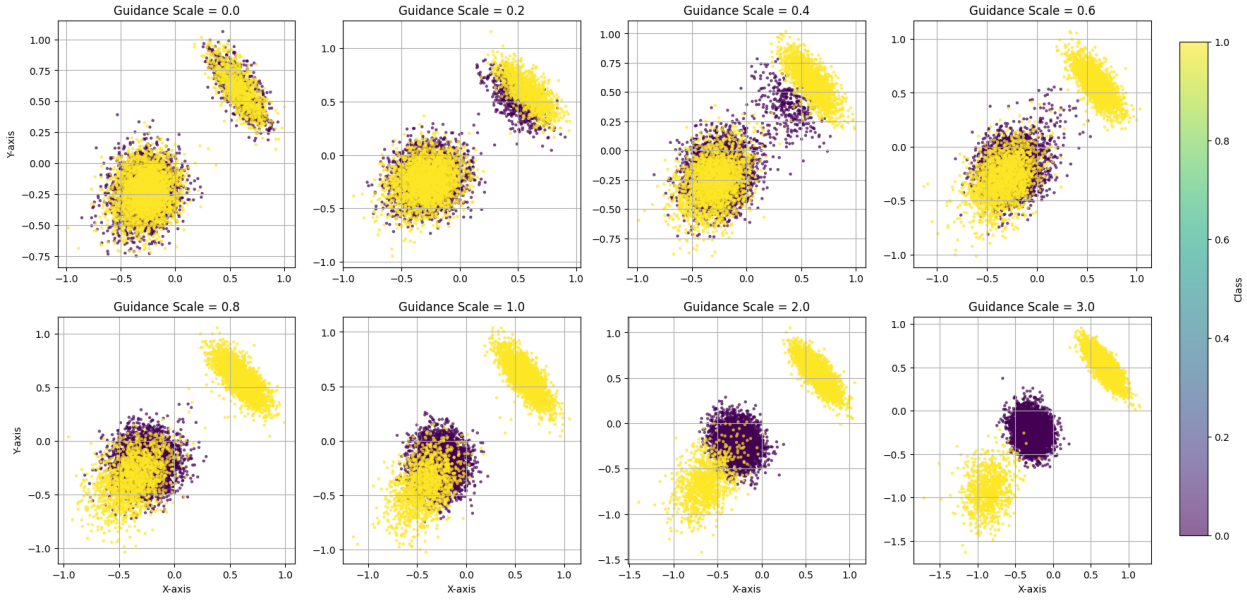


Figure 5: Effect of guidance scales on the generated data for blobs dataset

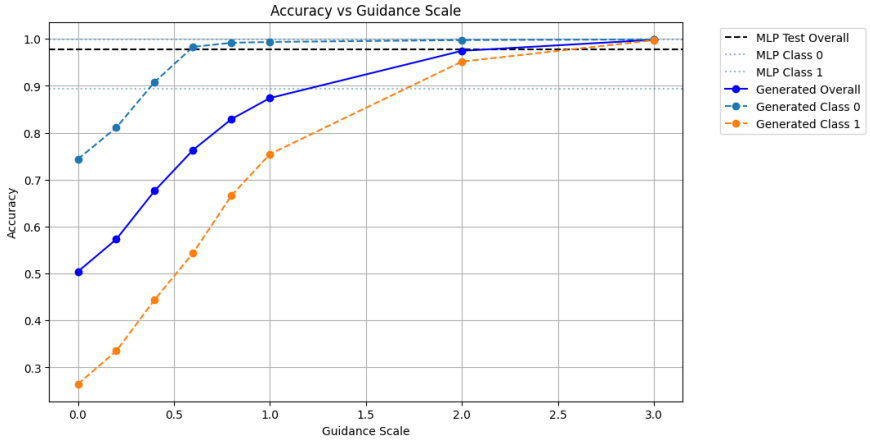


Figure 6: Accuracy of generated data for blobs dataset

Fig 5 and Fig 6 shows the effect of the guidance scales on the blobs dataset.

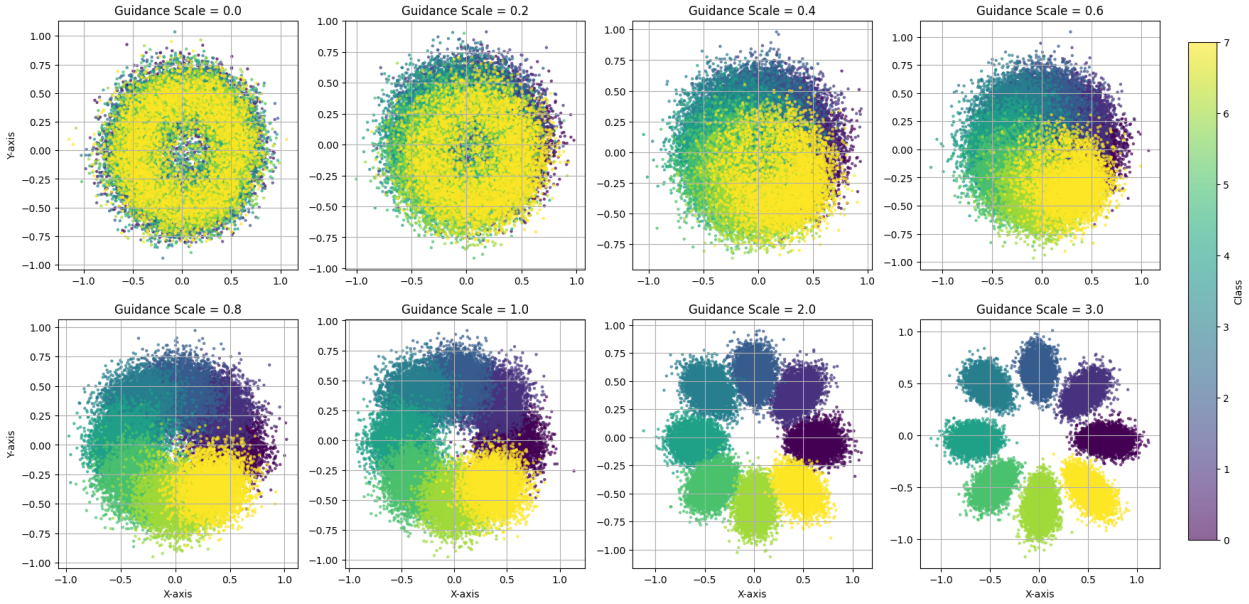


Figure 7: Effect of guidance scales on the generated data for manycircles dataset

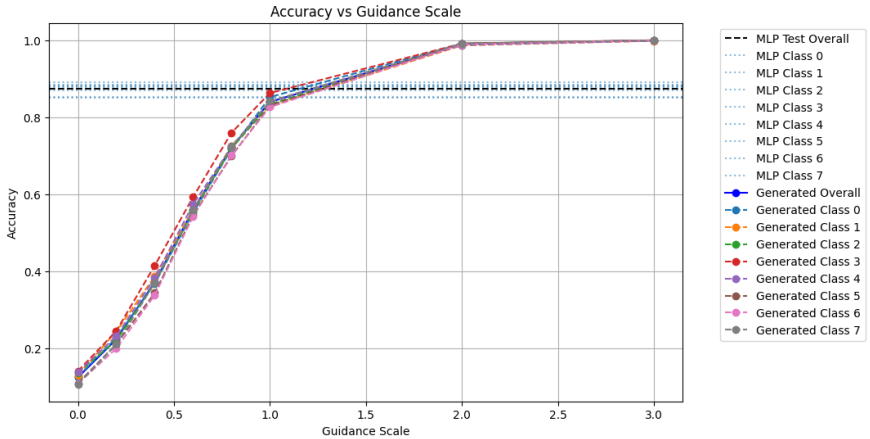


Figure 8: Accuracy of generated data for manycircles dataset

Fig 7 and Fig 8 shows the effect of the guidance scales on the manycircles dataset.

As we can observe that low guidance scales blur distinctions, optimal scales align with MLP accuracy, whereas in high guidance scales there is over-regularization.

2.3 Classifier Comparison

Here, we derive and explain the motivation behind the proposed training-free method for classifying inputs using the conditional diffusion model trained for Classifier-Free Guidance (CFG).

The core idea behind this method is to leverage the diffusion model's inherent noise estimation capability to determine class probabilities without requiring a separate classifier. The diffusion model trained with

class embeddings already learns conditional dependencies, which can be exploited for classification.

Given an input x, our goal is to predict the class label y. The diffusion model predicts noise values conditioned on the input x, time step t, and class label y. The objective is to minimize the Mean Squared Error (MSE) between predicted and true noise:

$$\mathcal{L}(\theta) = \mathbb{E}_{x_0,\epsilon,t}\left[\|\epsilon - \epsilon_\theta(x_t, t, y)\|^2\right] \tag{13}$$

Where:

- $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$
- ϵ is the noise sampled from a normal distribution.
- $\bar{\alpha}_t$ is the cumulative product of alpha values in the noise schedule.

To predict the class probabilities, we compute the expected MSE for each class. The predicted noise for class c is given by:

$$\epsilon_\theta(x_t, t, c) \tag{14}$$

We accumulate the MSE for each class and compute probabilities using the softmax function:

$$p(y = c|x) = \frac{\exp(-\text{MSE}(\epsilon_\theta(x_t, t, c), \epsilon))}{\sum_{c'} \exp(-\text{MSE}(\epsilon_\theta(x_t, t, c'), \epsilon))} \tag{15}$$

Since smaller MSE values indicate a better match, the softmax over the negative MSE values effectively models the posterior probabilities.

This derivation demonstrates how the noise prediction model can be used directly for classification without training a dedicated classifier.

The following are comparisons between the MLP classifier and the Diffusion classifier

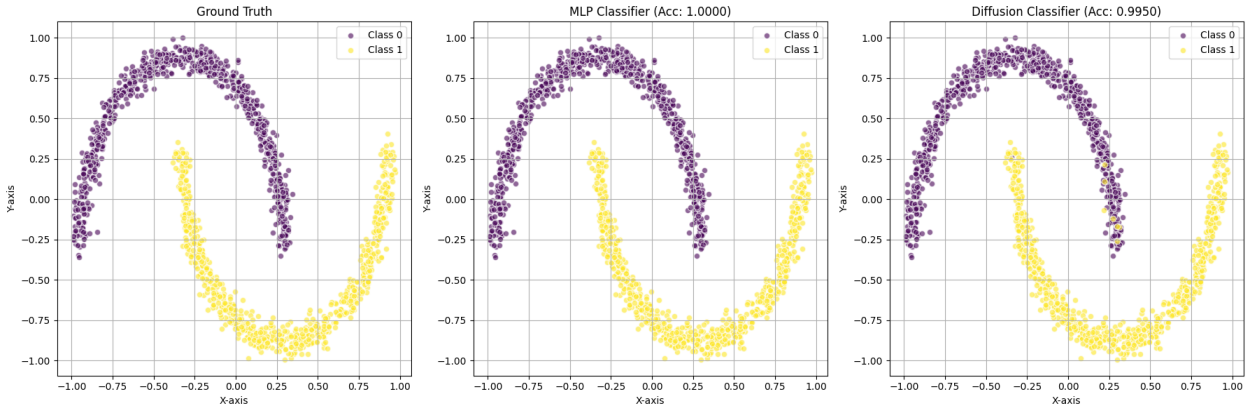


Figure 9: Comparison of MLP and Diffusion Classifier on Moon Dataset

Metric	MLP	Diffusion
Overall Accuracy	1.0000	0.9950
Class 0 Accuracy	1.0000	0.9913
Class 1 Accuracy	1.0000	0.9987

Table 18: Moons Dataset Accuracy Metrics Comparision

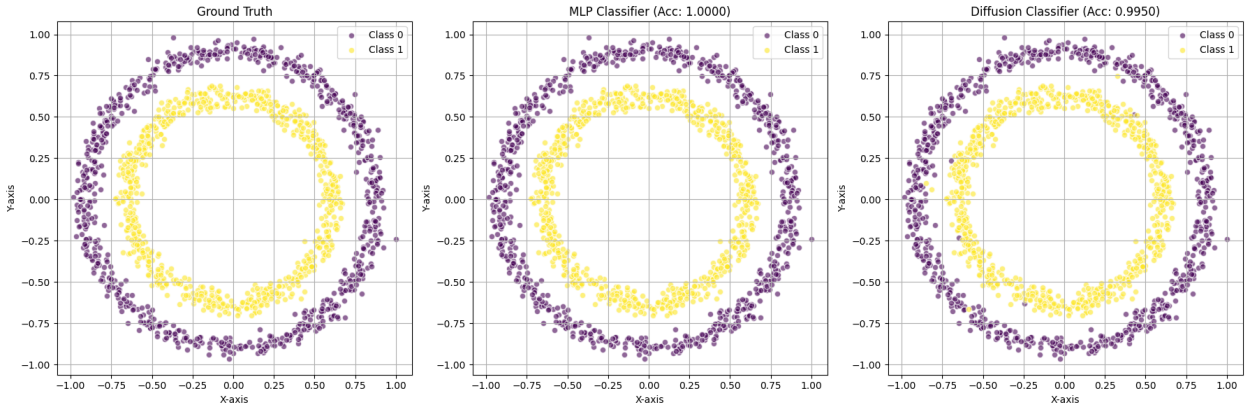


Figure 10: Comparison of MLP and Diffusion Classifier on Circles Dataset

Metric	MLP	Diffusion
Overall Accuracy	1.0000	0.9950
Class 0 Accuracy	1.0000	0.9950
Class 1 Accuracy	1.0000	0.9950

Table 19: Circles Dataset Accuracy Metrics Comparison

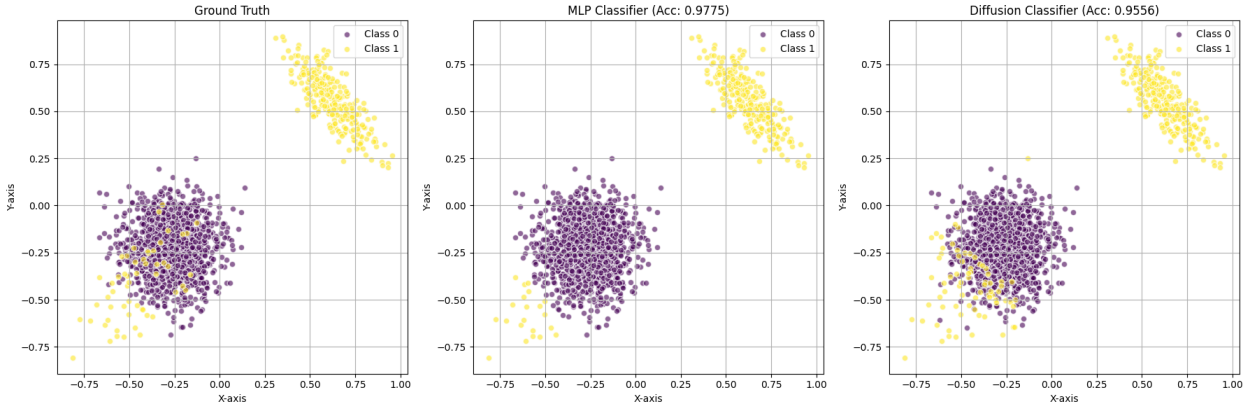


Figure 11: Comparison of MLP and Diffusion Classifier on Blob Dataset

Metric	MLP	Diffusion
Overall Accuracy	0.9775	0.9556
Class 0 Accuracy	0.9992	0.9653
Class 1 Accuracy	0.8943	0.9184

Table 20: Blobs Dataset Accuracy Metrics Comparison

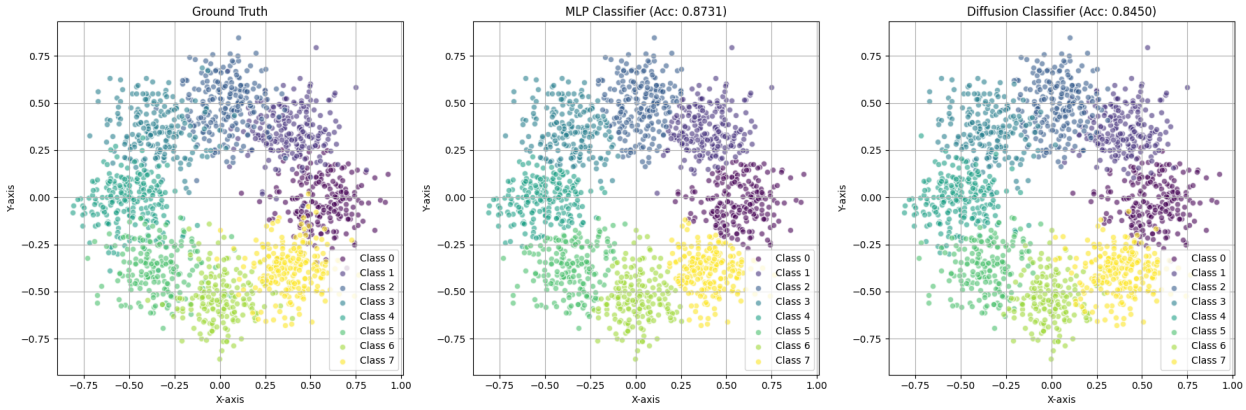


Figure 12: Comparison of MLP and Diffusion Classifier on Many Circles Dataset

Metric	MLP	Diffusion
Overall Accuracy	0.8731	0.8450
Class 0 Accuracy	0.8905	0.8955
Class 1 Accuracy	0.8798	0.8317
Class 2 Accuracy	0.8841	0.8406
Class 3 Accuracy	0.8827	0.8265
Class 4 Accuracy	0.8498	0.8498
Class 5 Accuracy	0.8525	0.8251
Class 6 Accuracy	0.8772	0.8070
Class 7 Accuracy	0.8688	0.8733

Table 21: Manycircles Dataset Accuracy Metrics Comparison

3 Reward Guidance

3.1 Implementation of SVDD

This posterior mean approximation is achieved by leveraging the learned noise prediction model to reconstruct the original data efficiently. The scaling factor $\frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}$ effectively separates noise from the true data distribution, ensuring stable sampling during the reverse diffusion process.

In Denoising Diffusion Probabilistic Models (DDPM), the posterior mean approximation is derived to compute the reverse diffusion process efficiently. The diffusion process introduces noise to the data through a fixed forward process, while the model learns the reverse process to recover the original data distribution.

In the forward process, noise is gradually added to the data over T timesteps according to:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, \beta_t I) \tag{16}$$

By recursively applying the forward process, we obtain:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) \tag{17}$$

where $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ is the cumulative product of noise scalings.

The reverse process is modeled as:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I) \tag{18}$$

To approximate the posterior mean $\mu_\theta(x_t, t)$, we use Bayes' theorem:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}t}} \epsilon_\theta(x_t, t) \right) \tag{19}$$

where ϵ_θ is the predicted noise obtained from the model.

In Soft Value-Based Decoding (SVDD), the goal is to sample from an unconditional model while incorporating a reward function to guide the sample toward high-reward outcomes. The posterior mean approximation is central to this as it allows for efficient sampling by modifying the estimated mean.

In SVDD-PM the predicted noise model $\epsilon_\theta(x_t, t)$ is combined with the reward gradient to refine the denoising step. Then guidance effectively shifts the posterior mean estimate towards high-reward samples without requiring explicit conditioning in the model.

By leveraging the posterior mean approximation, SVDD-PM aligns the denoising step with the reward landscape, ensuring generated samples are both realistic and optimized for desirable outcomes. This is achieved by adjusting the predicted noise in the sampling process based on the gradient of the reward function.

Algorithm 5 Soft Value-Based Decoding with Posterior Mean (SVDD-PM)

Require: Pre-trained noise model ϵ_θ, reward function $R(x)$, noise schedule $\{\beta_t\}$, number of timesteps T
Ensure: Sample x_0 guided by reward function

1: Sample $x_T \sim \mathcal{N}(0, I)$ ▷ Start from Gaussian noise
2: **for** $t = T, T-1, \ldots, 1$ **do**
3: Compute predicted noise: $\hat{\epsilon} = \epsilon_\theta(x_t, t)$
4: Compute posterior mean approximation:

$$\hat{\mu}_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \hat{\epsilon} \right)$$

5: Compute reward gradient: $\nabla_{x_t} R(x_t)$
6: Compute SVDD-PM update step:

$$x_{t-1} \leftarrow \hat{\mu}_\theta(x_t, t) + \gamma_t \nabla_{x_t} R(x_t) + \sigma_t z \quad \text{where } z \sim \mathcal{N}(0, I)$$

7: **end for**
8: **return** x_0