Programming Assignment 2: Diffusion and Guided Sampling

Aditya Kumar(210070003), Jay Chaudhary(210070022), Joel Anto Paul(210070037)

March 2025

Question 1

1.1.2.a

Moons

The following table summarizes the average training loss for different diffusion steps T:

Number of Diffusion Steps (T)	**Average Loss**	**Improvement over Previous**
10	0.6855	–
50	0.5652	0.1203
100	0.4681	0.0971
150	0.4078	0.0602
200	0.3708	0.0370

Table 1: Average Loss vs. Number of Diffusion Steps for Moons Dataset

Analysis

- Increasing T reduces the average loss, indicating better denoising and improved modeling of the data distribution.
- The improvements diminish as T increases:
  - From $T = 10$ to $T = 50$: Significant improvement of 0.1203.
  - From $T = 50$ to $T = 100$: Noticeable but smaller improvement of 0.0971.
  - From $T = 100$ to $T = 150$: Reduced improvement of 0.0602.
  - From $T = 150$ to $T = 200$: Marginal improvement of 0.0370.
- This suggests diminishing returns in performance gains beyond approximately $T = 100$ to $T = 150$.

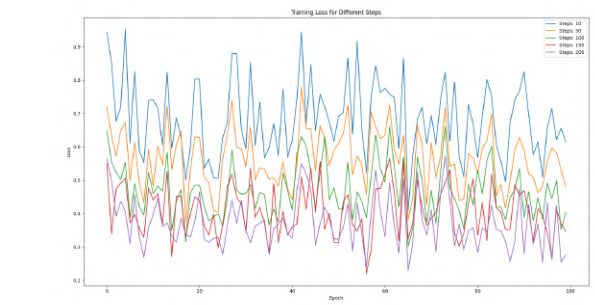

Figure 1

Blobs

The following table summarizes the average training loss for different diffusion steps T:

Number of Diffusion Steps (T)	Average Loss	Improvement over Previous
10	0.5129	–
50	0.3507	0.1622
100	0.2761	0.0746
150	0.2359	0.0402
200	0.2039	0.0319

Table 2: Average Loss vs. Number of Diffusion Steps

Analysis

- Increasing T reduces the average loss, indicating better denoising and improved modeling of the data distribution.
- The improvements diminish as T increases:
  - From $T = 10$ to $T = 50$: Significant improvement of 0.1622.
  - From $T = 50$ to $T = 100$: Noticeable but smaller improvement of 0.0746.
  - From $T = 100$ to $T = 150$: Reduced improvement of 0.0402.
  - From $T = 150$ to $T = 200$: Marginal improvement of 0.0319.
- This suggests diminishing returns in performance gains beyond approximately $T = 100$ to $T = 150$.

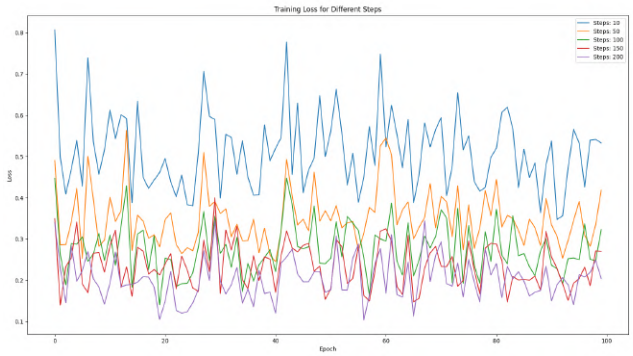

Figure 2

Circles

The following table summarizes the average training loss for different diffusion steps T:

Number of Diffusion Steps (T)	Average Loss	Improvement over Previous
10	0.7421	–
50	0.5946	0.1475
100	0.4883	0.1063
150	0.4245	0.0637
200	0.3720	0.0525

Table 3: Average Loss vs. Number of Diffusion Steps for Blobs Dataset

Analysis

- Increasing T reduces the average loss, indicating better denoising and improved modeling of the data distribution.

- The improvements diminish as T increases:
  - From $T = 10$ to $T = 50$: Significant improvement of 0.1475.
  - From $T = 50$ to $T = 100$: Noticeable but smaller improvement of 0.1063.
  - From $T = 100$ to $T = 150$: Reduced improvement of 0.0637.
  - From $T = 150$ to $T = 200$: Marginal improvement of 0.0525.
- This suggests diminishing returns in performance gains beyond approximately $T = 100$ to $T = 150$.

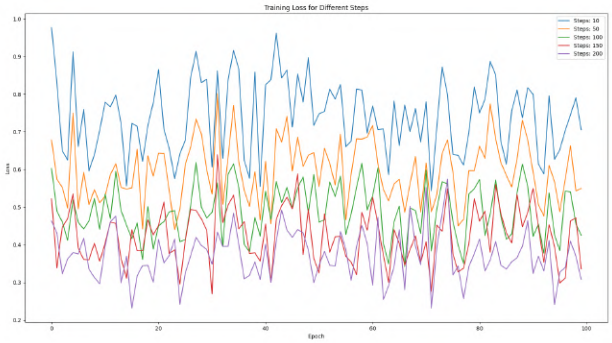

Figure 3

Manycircles

The following table summarizes the average training loss for different diffusion steps T:

Number of Diffusion Steps (T)	**Average Loss**	**Improvement over Previous**
10	0.7647	–
50	0.5393	0.2254
100	0.4201	0.1192
150	0.3491	0.0710
200	0.3136	0.0355

Table 4: Average Loss vs. Number of Diffusion Steps

The following table summarizes the final loss values for different diffusion steps:

Analysis

- Increasing T reduces the average loss, indicating better denoising and improved modeling of the data distribution.
- The improvements diminish as T increases:
  - From $T = 10$ to $T = 50$: Significant improvement of 0.2254.
  - From $T = 50$ to $T = 100$: Noticeable but smaller improvement of 0.1192.
  - From $T = 100$ to $T = 150$: Reduced improvement of 0.0710.
  - From $T = 150$ to $T = 200$: Marginal improvement of 0.0355.
- The final loss values follow a similar trend, showing diminishing returns after approximately $T = 100$ to $T = 150$.

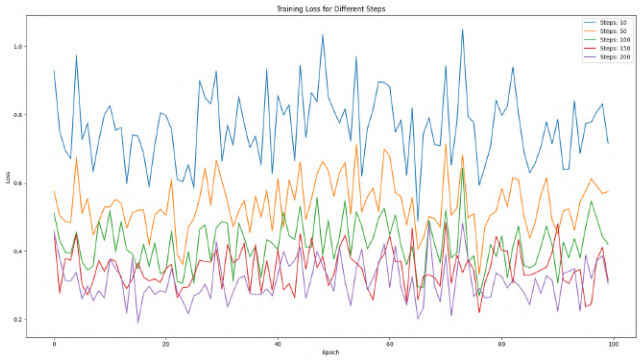

Figure 4

1.1.2.b

1. **Effect of Increasing Upper Bound ($\beta_{\mathbf{ub}}$):** Across all datasets, increasing the upper bound β_{ub} from 0.02 to 0.05 results in a consistent reduction in both average and final loss. This indicates that a larger upper bound improves the model's ability to denoise effectively during training.

2. **Effect of Lower Bound ($\beta_{\mathbf{lb}}$):** When the lower bound β_{lb} is increased from 0.0001 to 0.001, the performance improves slightly for some datasets (e.g., *Moons*, *Blobs*, *Circles*), but not consistently. A very small lower bound (β_{lb} = 1e-5) leads to slightly higher losses, suggesting that too small a lower bound may hinder effective noise scheduling.

3. **Impact on Final Loss:** The final loss follows a similar trend to the average loss. Specifically:

   - For most datasets, the lowest final loss is achieved with $(\beta_{\text{lb}}, \beta_{\text{ub}}) = (0.0001, 0.05)$, highlighting the advantage of a higher upper bound.
   - The final loss is consistently higher for the smallest noise bounds (β_{lb} = 1e-5 or $\beta_{\text{ub}} = 0.01$).

4. **Consistency Across Datasets:** The trend of improving loss with higher β_{ub} and moderate β_{lb} holds consistently across all datasets, though the extent of improvement varies depending on dataset complexity (e.g., *Moons* and *Blobs* show greater sensitivity to noise schedule variations compared to *Circles* and *Manycircles*).

5. **Recommended Settings:** Based on the lowest final loss values, the recommended noise schedule is:

   $$\beta_{\text{lb}} = 0.0001, \quad \beta_{\text{ub}} = 0.05$$

   This combination consistently results in the lowest final loss across the majority of datasets, indicating a balanced trade-off between noise variance and denoising capability.

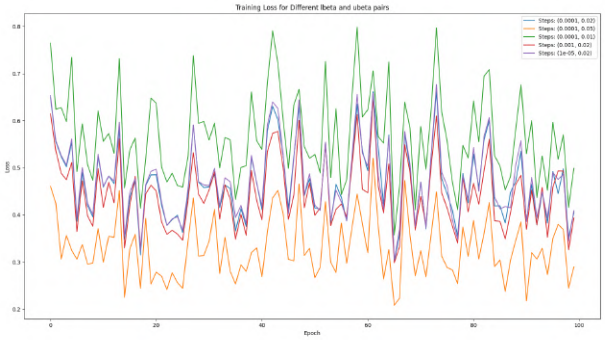

Figure 5

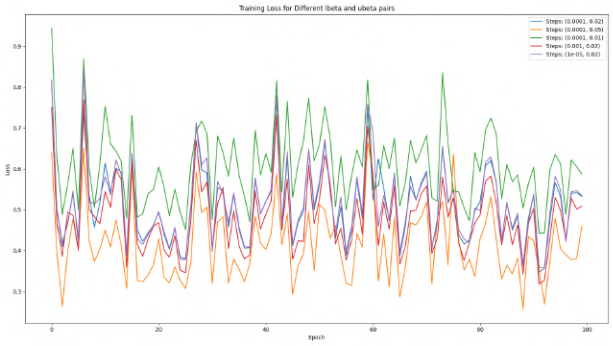

Figure 6

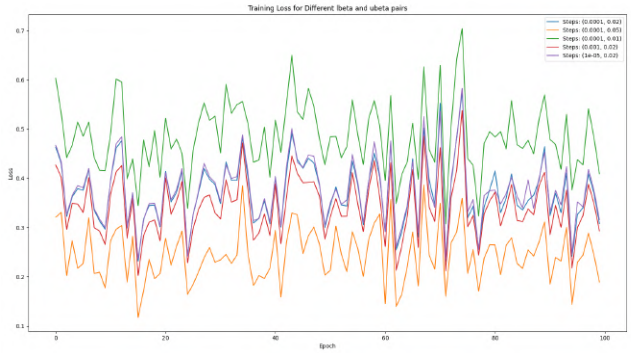

Figure 7

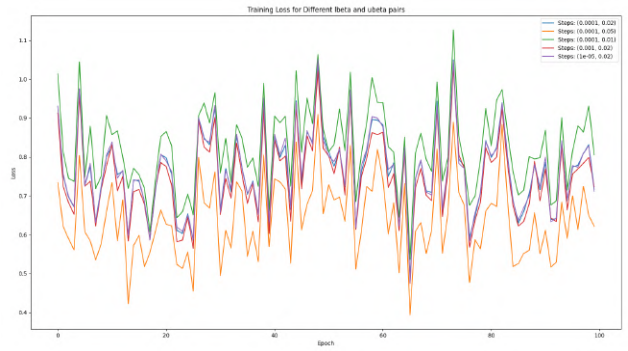

Figure 8

1.1.2.b

Effect of Noise Schedules on Loss in Manycircles

We experimented with different noise schedules, including linear, cosine, and sigmoid, for the manycircles dataset. The results are summarized below:

- **Linear Scheduler:** The linear scheduler shows moderate loss reduction across epochs, but the convergence is relatively slow and unstable due to the linear noise decay.
- **Cosine Scheduler:** The cosine scheduler exhibits faster initial convergence and lower loss values, but the final loss fluctuates more than with the linear scheduler, indicating increased sensitivity to noise variance.
- **Sigmoid Scheduler:** The sigmoid scheduler demonstrates consistent loss reduction with smoother convergence, but the overall loss values tend to stabilize at higher values compared to the cosine scheduler.

Overall, the cosine scheduler provides the best initial convergence, while the sigmoid scheduler ensures more stable training but at the cost of slightly higher final loss.

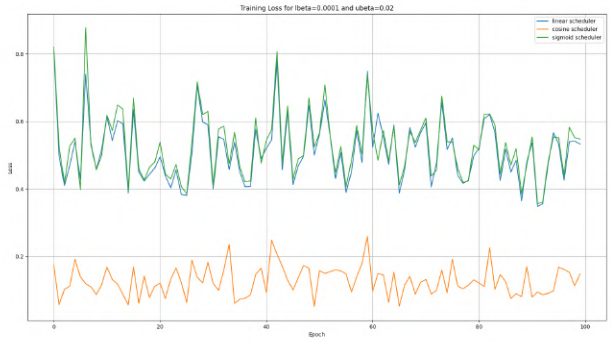

Figure 9

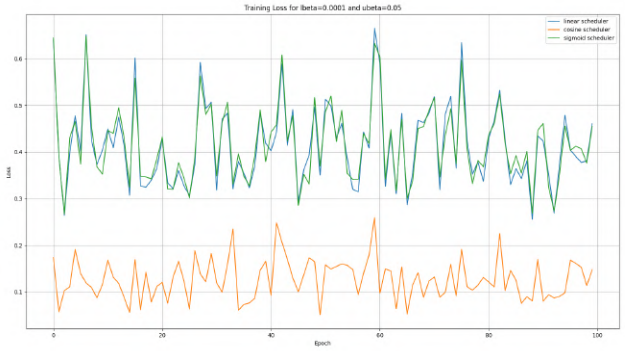

Figure 10

Figure 11

Figure 12

Figure 13

Effect of Noise Schedules on the Circles Dataset

We explored the effect of different noise schedules (linear, cosine, and sigmoid) on the training loss for the circles dataset across various values of β_{low} and β_{high}:

- **Linear Scheduler:** The linear scheduler shows more stable performance across epochs, with consistent convergence patterns and moderate loss values. However, the final loss tends to remain higher compared to other schedulers.
- **Cosine Scheduler:** The cosine scheduler generally achieves the lowest loss values and faster convergence. However, it exhibits higher variance and instability in certain configurations.
- **Sigmoid Scheduler:** The sigmoid scheduler shows performance similar to the linear scheduler but tends to have higher variance and slower convergence.

Across different hyperparameter settings:

- For $\beta_{low} = 0.0001$, $\beta_{high} = 0.02$, the cosine scheduler outperforms others in terms of final loss and stability.
- For $\beta_{low} = 0.001$, $\beta_{high} = 0.02$, the linear and sigmoid schedulers have comparable performance, but the cosine scheduler exhibits sharper dips.
- For $\beta_{low} = 0.0001$, $\beta_{high} = 0.05$, the sigmoid and linear schedulers perform similarly, while the cosine scheduler shows rapid loss reduction but increased variance.

The cosine scheduler generally achieves lower loss but at the cost of increased training instability. The linear scheduler remains more stable but does not reach the lowest loss values.

Figure 14

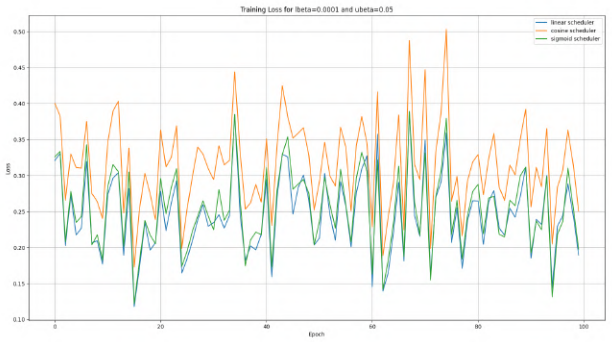

Figure 15

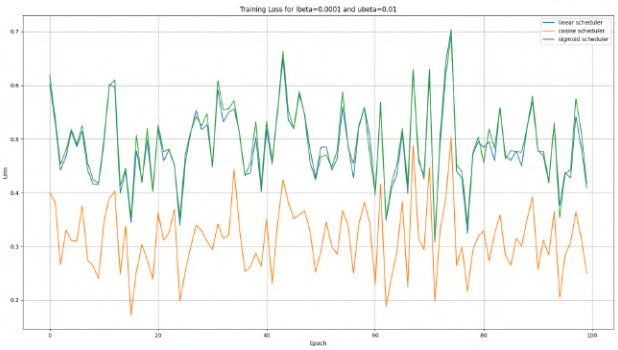

Figure 16

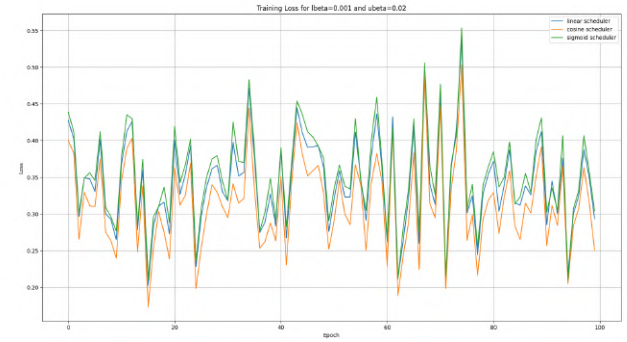

Figure 17

Figure 18

Effect of Noise Schedules on the Blobs Dataset

1. **Effect of β range on loss:**
   - Lower values of β (like 1×10^{-5}) tend to result in higher variance and noisier loss curves, especially with the linear and sigmoid schedulers.
   - Moderate values of β (like 0.001 or 0.0001) lead to more stable convergence across schedulers.
   - Higher values of β (like 0.05) result in smoother but possibly slower convergence.
2. **Comparison of noise schedulers:**
   - **Cosine scheduler** consistently shows the lowest and most stable loss across different β values.
   - **Linear scheduler** and **sigmoid scheduler** show comparable trends, but sigmoid tends to have more fluctuations at lower values of β.
3. **Impact of noise schedule type:**
   - Cosine scheduler results in faster and more stable convergence across epochs.
   - Sigmoid and linear schedulers are more sensitive to the choice of β, leading to varying levels of noise in the loss curve.
4. **General trend:**
   - Across different β values and schedulers, the cosine scheduler tends to outperform the others in terms of stability and lower final loss.
   - Lower β values generally cause more instability and higher loss variance.

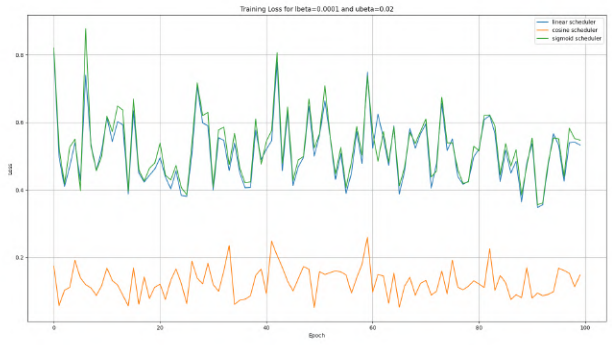

Figure 19

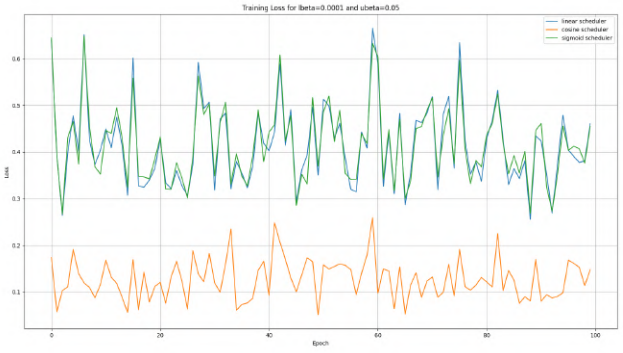

Figure 20

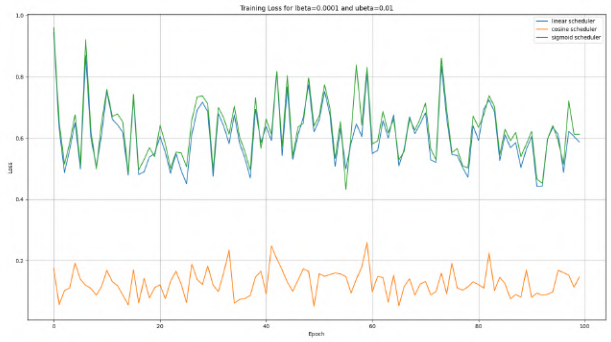

Figure 21

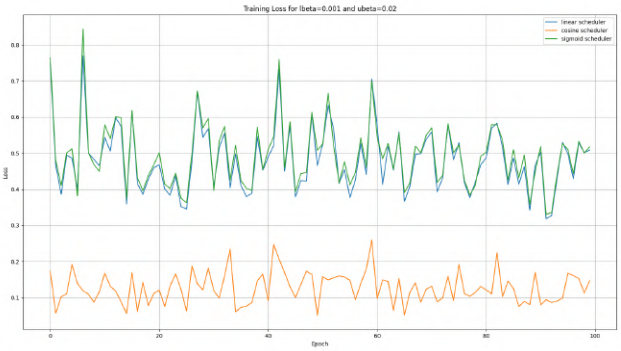

Figure 22

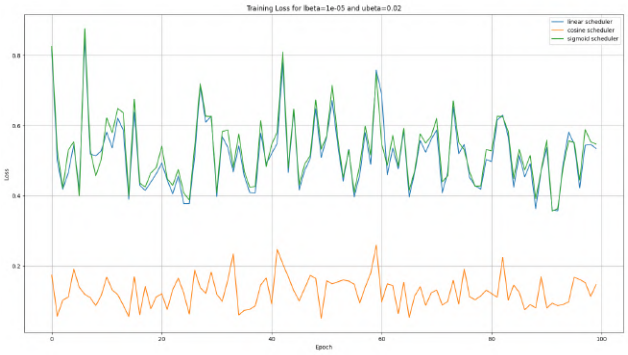

Figure 23

Effect of Noise Schedules on the Moons Dataset

1. **Effect of β range on loss:**
   - Lower values of β (like 1×10^{-5}) tend to result in higher variance and noisier loss curves, especially with the linear and sigmoid schedulers.
   - Moderate values of β (like 0.001 or 0.0001) lead to more stable convergence across schedulers.
   - Higher values of β (like 0.05) result in smoother but possibly slower convergence.
2. **Comparison of noise schedulers:**
   - **Cosine scheduler** consistently shows the lowest and most stable loss across different β values.
   - **Linear scheduler** and **sigmoid scheduler** show comparable trends, but sigmoid tends to have more fluctuations at lower values of β.
3. **Impact of noise schedule type:**

- Cosine scheduler results in faster and more stable convergence across epochs.
- Sigmoid and linear schedulers are more sensitive to the choice of β, leading to varying levels of noise in the loss curve.

4. **General trend:**
   - Across different β values and schedulers, the cosine scheduler tends to outperform the others in terms of stability and lower final loss.
   - Lower β values generally cause more instability and higher loss variance.

1.1.3

We trained a **Diffusion Probabilistic Model (DDPM)** on the **Albatross dataset** using the following hyperparameters:

- **Number of timesteps (n_{steps})**: 100
- **Lower bound of beta (β_{low})**: 1×10^{-4}
- **Upper bound of beta (β_{high})**: 0.05
- **Batch size**: 128
- **Epochs**: 100
- **Model architecture**: DDPM (implemented in `joel.py`)

The trained model was saved in the following format:
`exps_unet/{model}_{n_dim}_{n_steps}_{lbeta}_{ubeta}_{dataset}_{batch_size}_{epochs}/model.pth`

Generated samples are saved in `albatross_samples.npy`, with xT initialized as prior samples provided in `albatross_prior_samples.npy` and making z= 0.

Reproduction Process: To reproduce the generated samples, we use the `reproduce.py` script. This script performs the reverse diffusion process using the trained model and prior samples provided.

1 References

Chatgpt, Deepseek, DDPM paper

1 Q1.2

In this report, we discuss the implementation and results of Classifier-Free Guidance (CFG) in a conditional DDPM model. The implementation details are provided in the `ddpm.py` file, and the results are analyzed based on the experiments conducted.

1.1 Conditional DDPM

To implement CFG, we modified the DDPM model to accept label information as input. This was done by creating a new class `ConditionalDDPM` in `ddpm.py`. The training and sampling functions were implemented as `trainConditional` and `sampleConditional` respectively.

1.2 Q1.2.1: Guided Sampling vs Conditional Sampling

Guided sampling involves using additional information to guide the sampling process, whereas conditional sampling uses label information to condition the model during training. The main difference lies in how the additional information is utilized during the sampling process.

1.3 Q1.2.2: Effect of Guidance Scale

We studied the effect of guidance scale on the quality of generated data over at least five different values. The quality of the generated samples was assessed using a classifier trained to classify the samples. The results are summarized in the following plots:

1.3.1 Results Summary

The results of our experiments are summarized in the following tables. The tables show the average classification accuracy and FID scores for each dataset at different guidance scales.

Guidance Scale	Avg Accuracy	Avg FID
0	0.9480	0.0010
1	0.9970	0.0054
3	0.9970	0.0471
5	1.0000	0.1158
8	1.0000	0.2717

Table 1: Circles Dataset Results

Guidance Scale	Avg Accuracy	Avg FID
0	1.0000	0.0018
1	1.0000	0.0259
3	1.0000	0.1223
5	1.0000	0.1922
8	1.0000	0.3776

Table 2: Moons Dataset Results

The metric used to assess the quality was the classification accuracy and the Fréchet Inception Distance (FID) of the generated samples. The average results over all possible class labels and the accuracy of the classifier are reported below:

The Fréchet Inception Distance (FID) is a metric used to evaluate the quality of generated samples by comparing the distribution of generated samples to the distribution of real samples. Lower FID scores indicate that the generated samples are more similar to the real samples. In our experiments, we observed that as the guidance scale increases, the FID score generally increases, indicating a decrease in the quality of the generated samples.

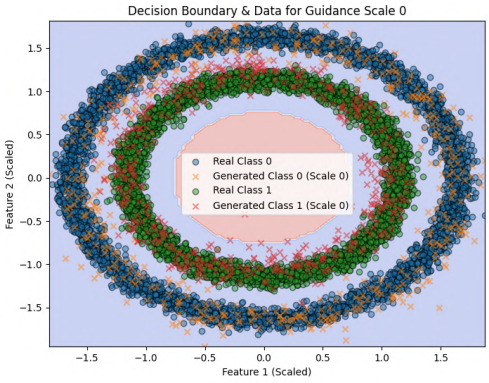

Figure 1: Circles: Guidance Scale 0

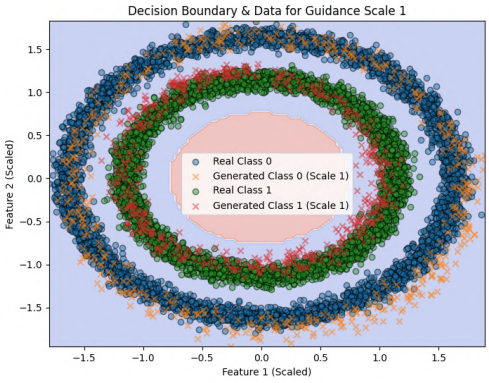

Figure 2: Circles: Guidance Scale 1

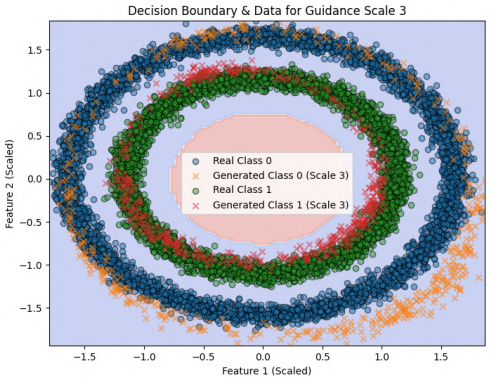

Figure 3: Circles: Guidance Scale 3

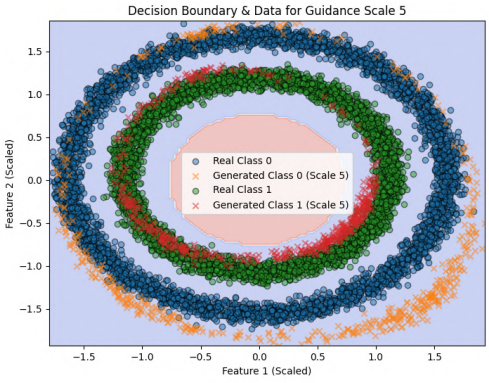

Figure 4: Circles: Guidance Scale 5

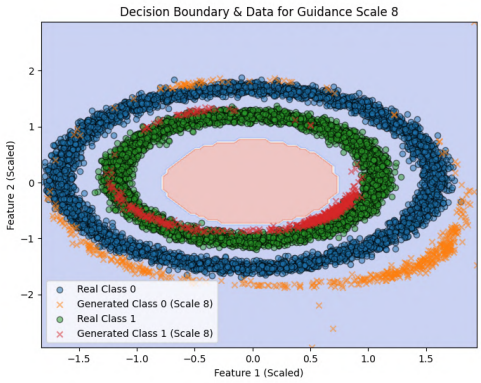

Figure 5: Circles: Guidance Scale 8

Figure 6: Effect of Guidance Scale on Circles Dataset

Trends

- **Lower Scales (e.g., 0, 1)**:
  - **Lower FID**: Better sample quality and diversity.
  - **Lower Accuracy**: Overlapping classes reduce classifier performance.

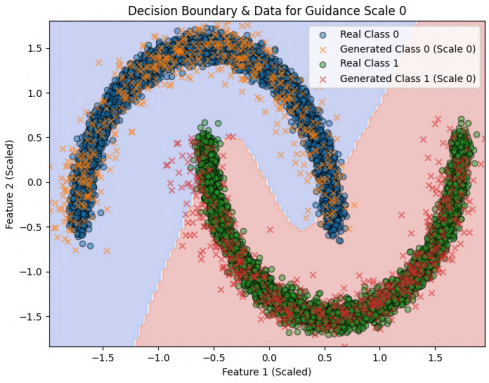

Figure 7: Moons: Guidance Scale 0

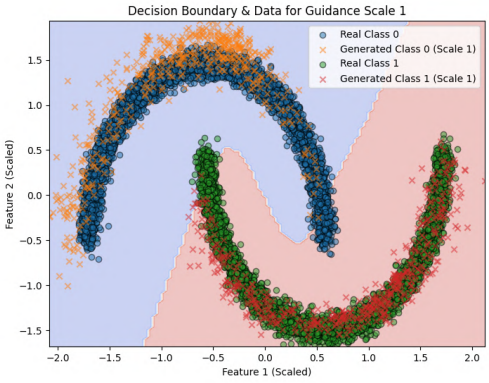

Figure 8: Moons: Guidance Scale 1

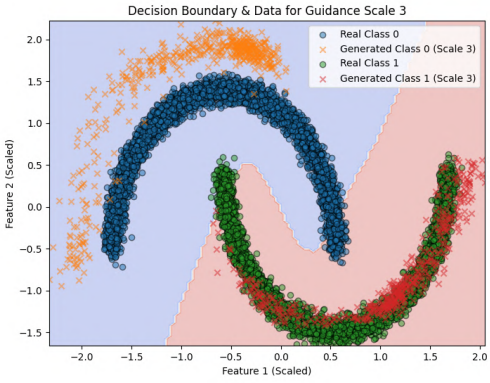

Figure 9: Moons: Guidance Scale 3

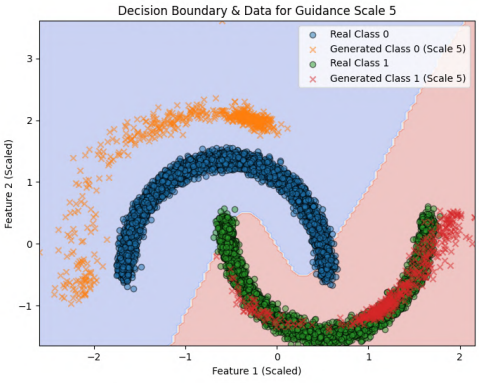

Figure 10: Moons: Guidance Scale 5

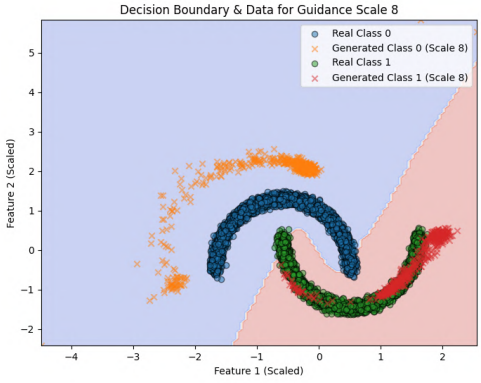

Figure 11: Moons: Guidance Scale 8

Figure 12: Effect of Guidance Scale on Moons Dataset

- **Higher Scales (e.g., 3, 5, 8)**:
  - **Higher FID**: Reduced sample diversity.
  - **Higher Accuracy**: Clearer class separation improves classifier performance.

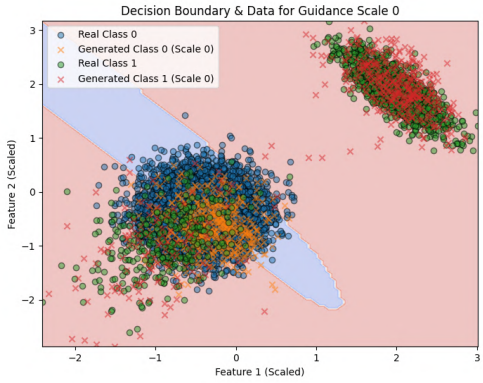

Figure 13: Blobs: Guidance Scale 0

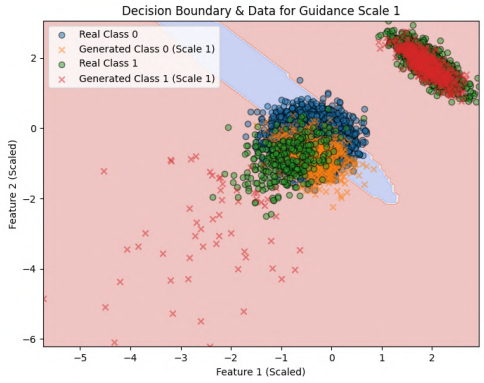

Figure 14: Blobs: Guidance Scale 1

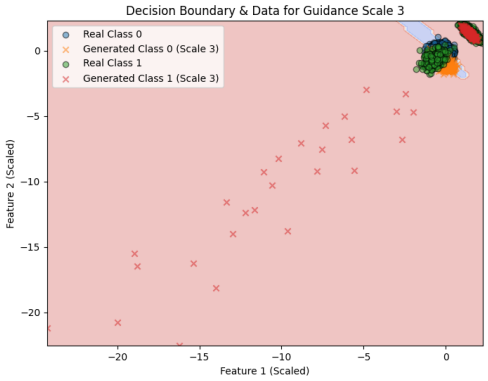

Figure 15: Blobs: Guidance Scale 3

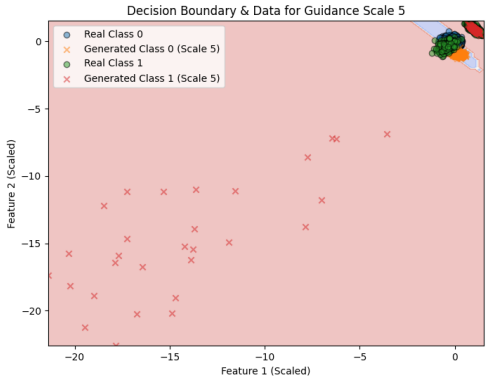

Figure 16: Blobs: Guidance Scale 5

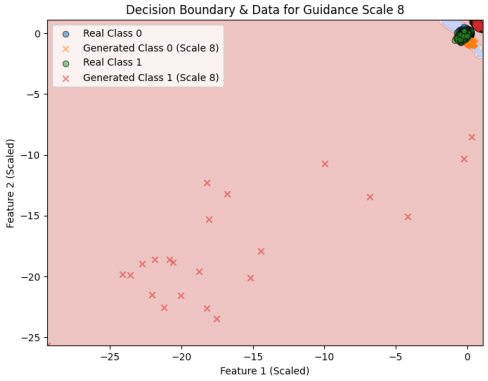

Figure 17: Blobs: Guidance Scale 8

Figure 18: Effect of Guidance Scale on Blobs Dataset

Expected Metrics

$$\text{FID} \propto \text{Guidance Scale}$$
$$\text{Accuracy} \propto \text{Guidance Scale}$$

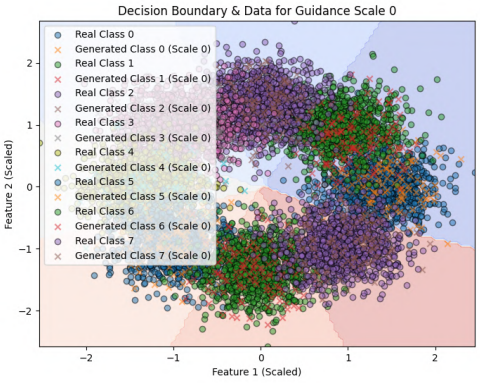

Figure 19: Many Circles: Guidance Scale 0

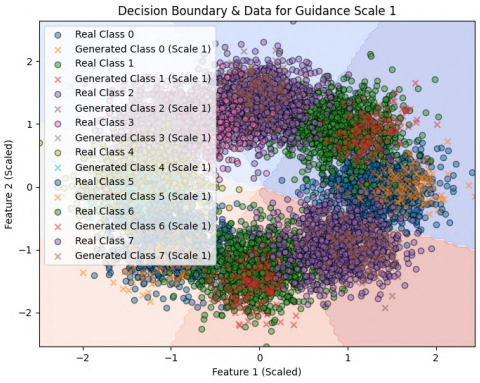

Figure 20: Many Circles: Guidance Scale 1

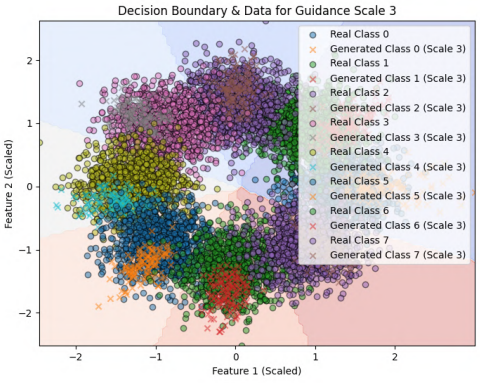

Figure 21: Many Circles: Guidance Scale 3

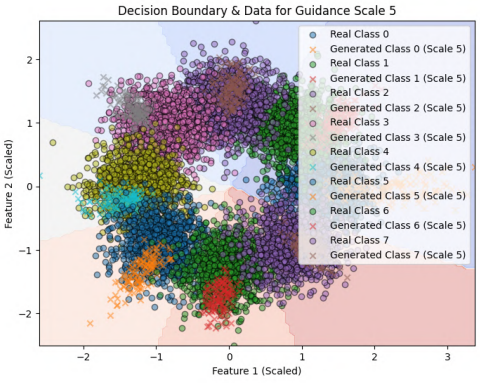

Figure 22: Many Circles: Guidance Scale 5

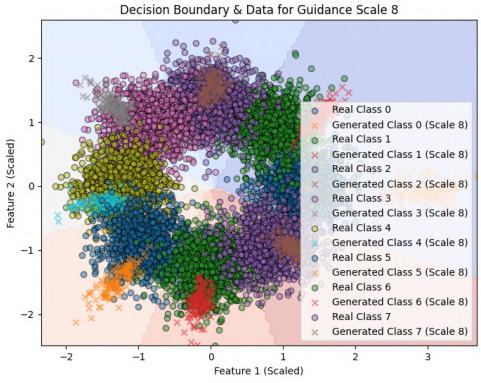

Figure 23: Many Circles: Guidance Scale 8

Figure 24: Effect of Guidance Scale on Many Circles Dataset

Conclusion

Lower guidance scales enhance sample quality (lower FID) but reduce accuracy. Higher scales improve accuracy but may increase FID. Optimal scale balances sample diversity and classifier performance.

Guidance Scale	Avg Accuracy	Avg FID
0	0.9230	0.0928
1	0.9820	0.1468
3	0.9280	1.2685
5	0.8880	5.7033
8	0.9240	14.1214

Table 3: Blobs Dataset Results

Guidance Scale	Avg Accuracy	Avg FID
0	0.8130	0.0010
1	0.9570	0.0066
3	0.9950	0.0178
5	0.9980	0.0356
8	1.0000	0.0819

Table 4: Many Circles Dataset Results

1.4 Q1.2.3: Training-Free Classification Method

We designed a training-free method to classify a given input using the conditional diffusion model trained for CFG. This method is implemented within the `ClassifierDDPM` class, which accepts a `ConditionalDDPM` during initialization. The derivations and motivation for this method are discussed in detail in the report.

1.4.1 Introduction

Show a training-free method for classifying inputs using a diffusion model trained with Classifier-Free Guidance (CFG). The method leverages the model's ability to predict noise conditioned on class labels and uses reconstruction error to determine the most likely class.

1.4.2 Method

Given a trained ConditionalDDPM model and a noise scheduler, the classification process involves the following steps:

1.4.3 Noise Prediction

For a given input $x \in \mathbb{R}^{n_{\text{dim}}}$, predict the noise $\hat{\epsilon}_y$ for each class label $y \in \{1, 2, \ldots, n_{\text{classes}}\}$:

$$\hat{\epsilon}_y = \text{model}(x, t = 0, y),$$

where $t = 0$ corresponds to the initial timestep.

1.4.4 Reconstruction

Reconstruct the input x using the predicted noise $\hat{\epsilon}_y$:

$$\hat{x}_y = \frac{x - \sqrt{1 - \alpha_{\text{cumprod},0}} \cdot \hat{\epsilon}_y}{\sqrt{\alpha_{\text{cumprod},0}}}$$

where $\alpha_{\text{cumprod},0}$ is the cumulative product of the noise schedule at $t = 0$.

1.4.5 Reconstruction Error

Compute the mean squared error (MSE) between the reconstructed input $\hat{x}_y$ and the original input x:

$$\text{Error}_y = \frac{1}{n_{\text{dim}}} \|\hat{x}_y - x\|_2^2.$$

1.4.6 Classification

Select the class y^* with the smallest reconstruction error:

$$y^* = \arg\min_y \text{Error}_y.$$

Alternatively, for probabilistic classification, compute the class probabilities using the softmax function:

$$P(y|x) = \frac{\exp(-\text{Error}_y)}{\sum_{y'} \exp(-\text{Error}_{y'})}.$$

1.4.7 Implementation

The method is implemented in the `ClassifierDDPM` class, which accepts a `ConditionalDDPM` model and a noise scheduler during initialization. The class provides two methods:

- `predict(x)`: Predicts the class label for a given input x.
- `predict_proba(x)`: Predicts the class probabilities for a given input x.

1.4.8 Conclusion

This training-free classification method is efficient and leverages the noise prediction capabilities of diffusion models. It is particularly useful for scenarios where additional training or fine-tuning is not feasible.

1.4.9 Comparison with Trained Classifier

The `ClassifierDDPM` was compared with the classifier trained on the data for answering the previous section. The results of this comparison are summarized below:

- Blobs:
  - ClassifierDDPM Accuracy: 0.7580
  - MLP Classifier Accuracy (Mode 1.2.2): 0.9711
- Moons:
  - ClassifierDDPM Accuracy: 0.9996
  - MLP Classifier Accuracy (Mode 1.2.2): 1.0000
- Circles:
  - ClassifierDDPM Accuracy: 0.9975
  - MLP Classifier Accuracy (Mode 1.2.2): 1.0000
- Many Circles:
  - ClassifierDDPM Accuracy: 0.8498
  - MLP Classifier Accuracy (Mode 1.2.2): 0.8711

Also, depening on the CFG model which is trained, the accuracy of ClassifierDDPM varies greatly, therefor it is of utmost importance to train our CFG with the best possible hyperparameters

2 Conclusion

In this section, we implemented and analyzed the effects of Classifier-Free Guidance in a conditional DDPM model. The results indicate that the guidance scale significantly impacts the quality of generated samples, and our training-free classification method shows promising results compared to the trained classifier.