CS726: Advanced Machine Learning **Jan-May 2025**

Programming Assignment 2: Denoising Diffusion Probabilistic Models

Team Name: Smart Researchers Team Members: **24M2136**, **24M2119**, **21D070001**

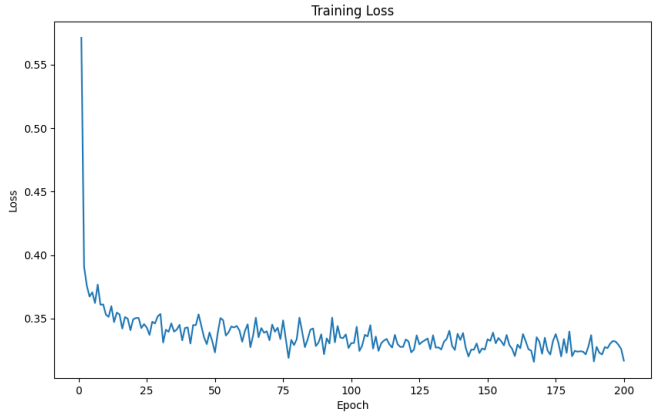
1 Problem Statement

1.1 Denoising Diffusion Probabilistic Models

1.1.1 Implement Unconditional DDPM

The Unconditional DDPM implemented in DDPM class. Below are the results we get on different datasets.

1. Moons



Evaluation Metrics:

- **EMD**: 63.1243
- **NLL**: 0.9436

Figure 1: Training Loss and Evaluation Metrics

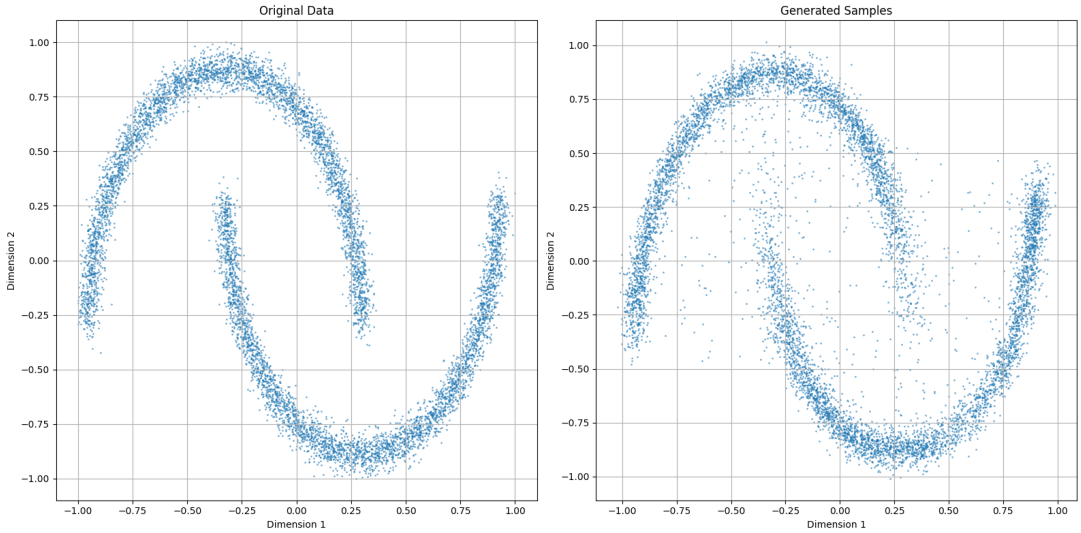
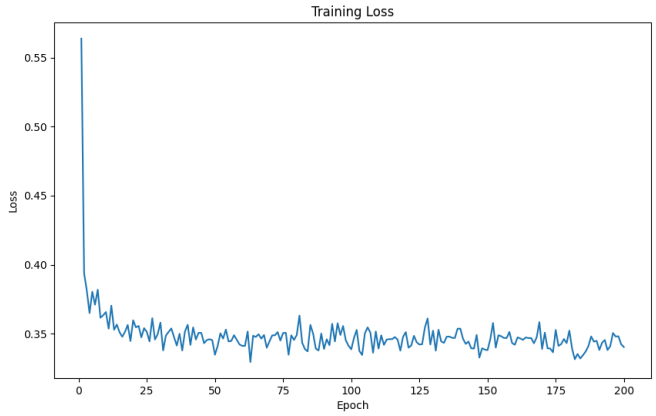


Figure 2: Comparison of Original Data and Generated Samples

2. Circles



Evaluation Metrics:

- **EMD**: 95.5196
- **NLL**: 0.9881

Figure 3: Training Loss and Evaluation Metrics

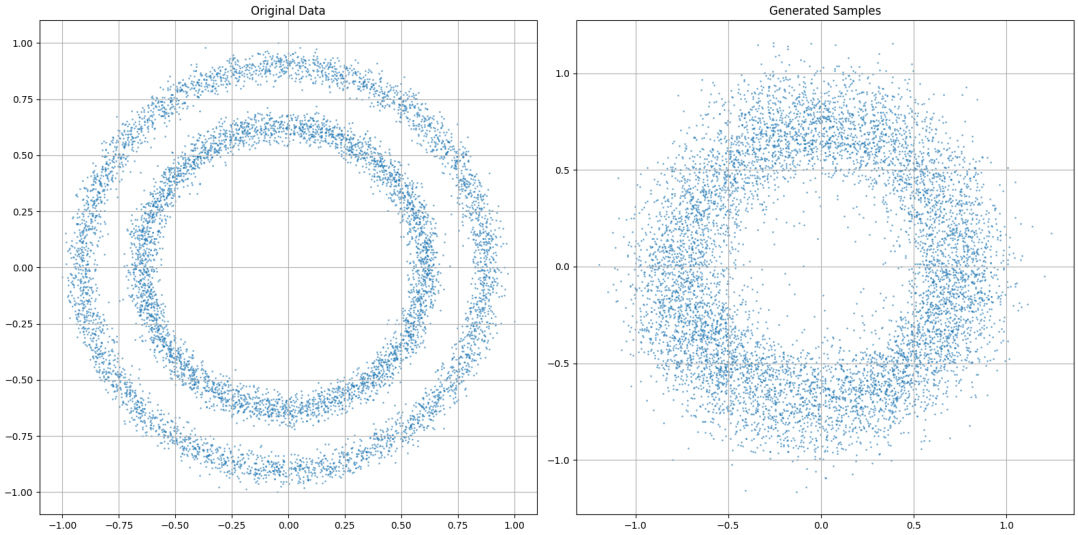
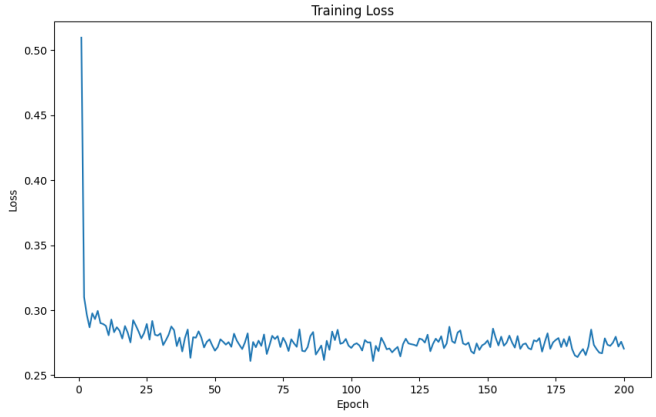


Figure 4: Comparison of Original Data and Generated Samples

3. Manycircles



Evaluation Metrics:

- **EMD**: 61.5133
- **NLL**: 0.5498

Figure 5: Training Loss and Evaluation Metrics

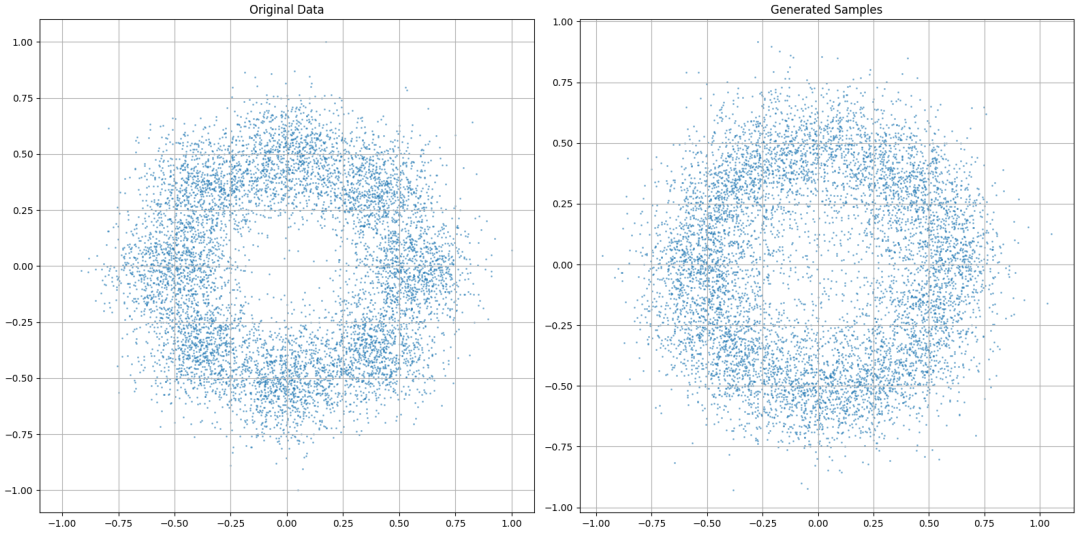
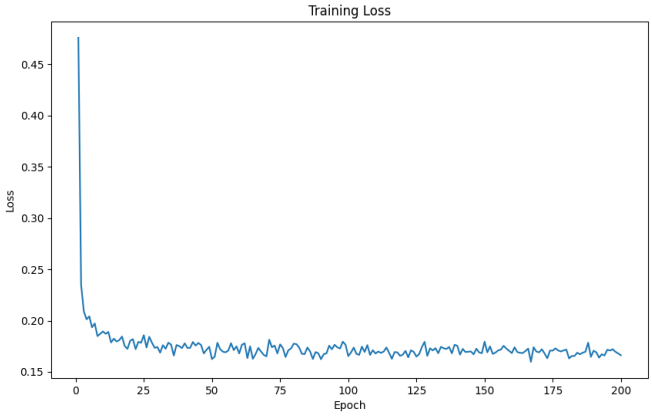


Figure 6: Comparison of Original Data and Generated Samples

4. Blobs



Evaluation Metrics:

- **EMD**: 645.3247
- **NLL**: 0.7221

Figure 7: Training Loss and Evaluation Metrics

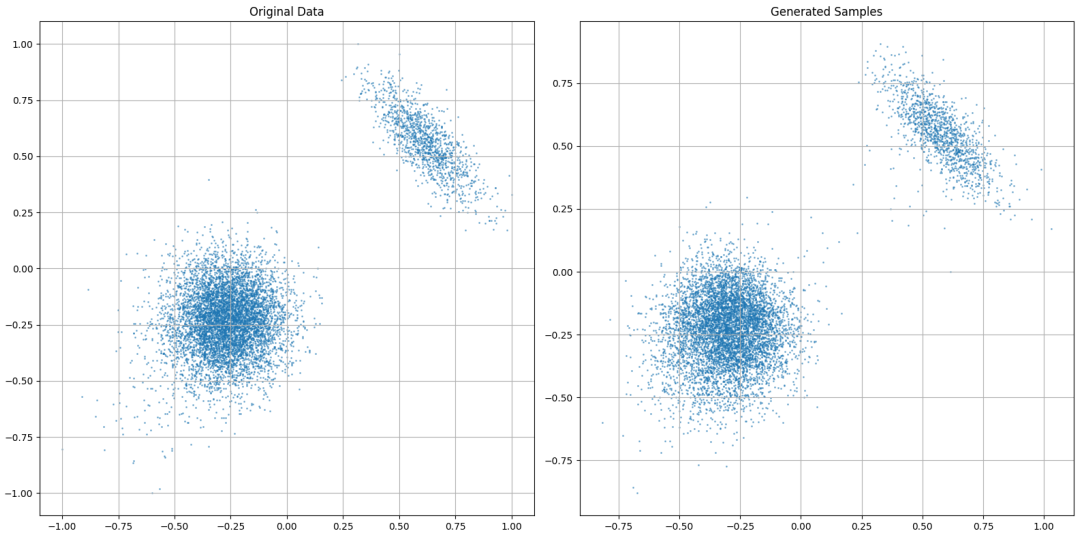
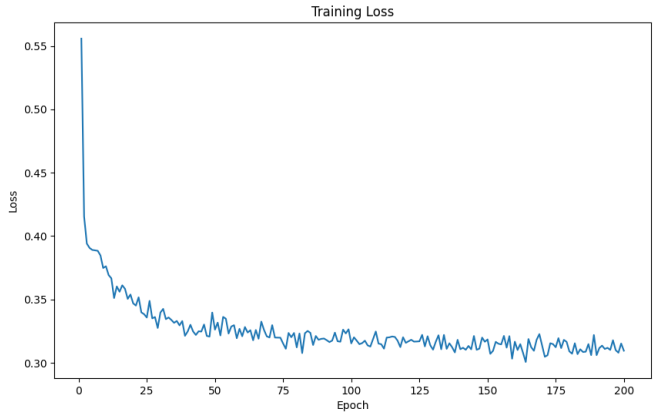


Figure 8: Comparison of Original Data and Generated Samples

5. Helix



Evaluation Metrics:

- **EMD**: 144.0211
- **NLL**: 1.5195

Figure 9: Training Loss and Evaluation Metrics

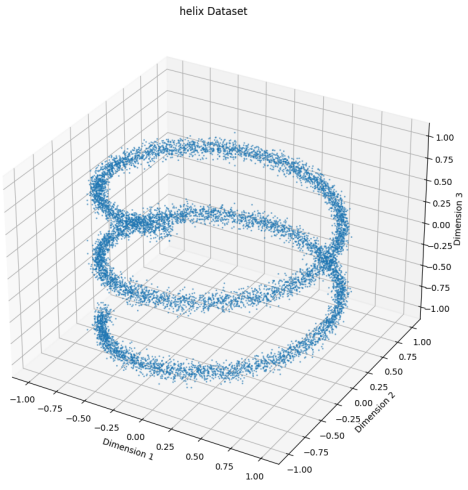


Figure 10: Original Data

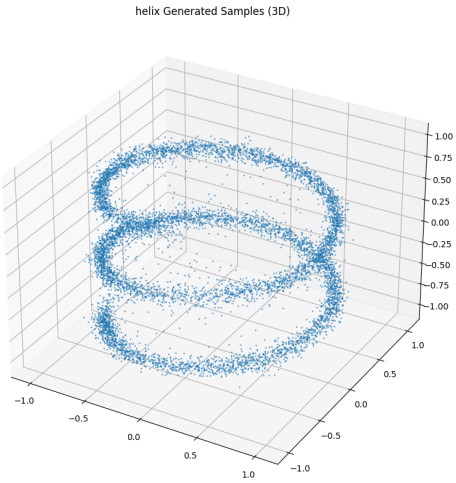
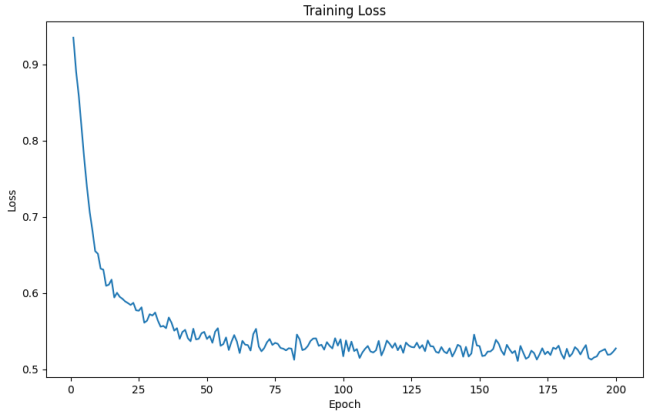


Figure 11: Generated Samples

1.1.2 Study the effects of hyperparameters

a) Number of diffusion steps (T)

We experiment this for T = 10, 50, 100, 150 & 200. We consider Helix Dataset for this task. We kept all other hyperparameters fixed. The following are the results we get for different time steps.



Evaluation Metrics:

1.
   - **EMD**: 119.7350
   - **NLL**: 1.5355

Figure 12: Training Loss and Evaluation Metrics

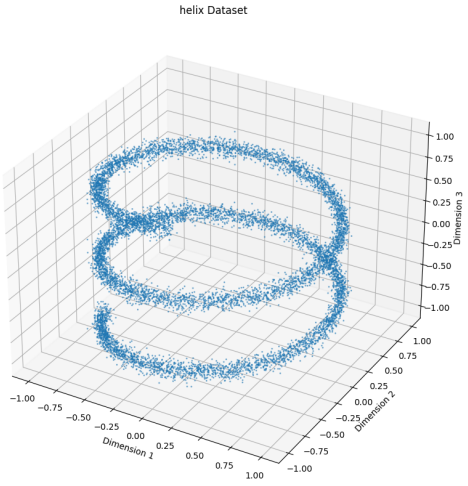


Figure 13: Original Data

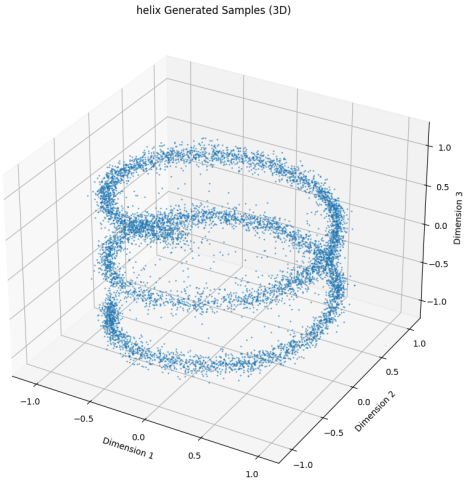
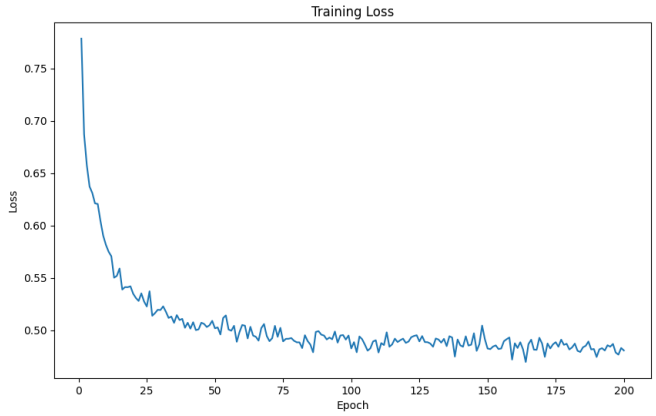


Figure 14: Generated Samples

2. T = 50



Evaluation Metrics:

- **EMD**: 123.1479
- **NLL**: 1.5495

Figure 15: Training Loss and Evaluation Metrics

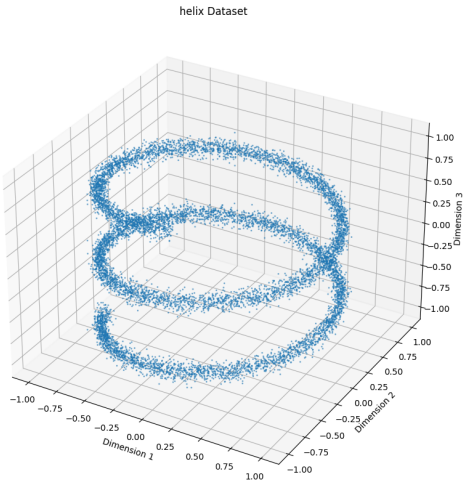


Figure 16: Original Data

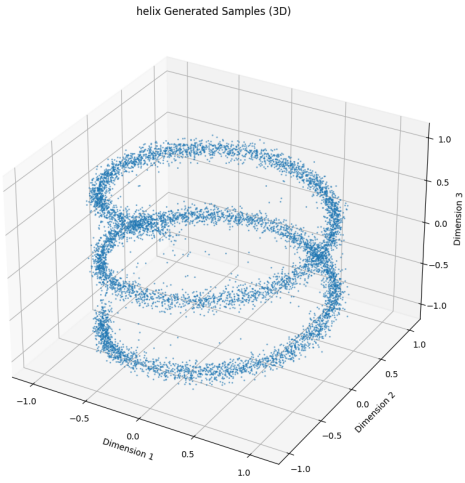
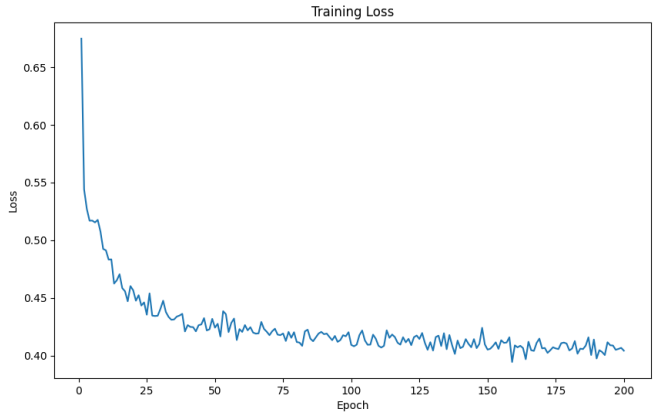


Figure 17: Generated Samples

3. T = 100



Evaluation Metrics:

- **EMD**: 158.0988
- **NLL**: 1.5416

Figure 18: Training Loss and Evaluation Metrics

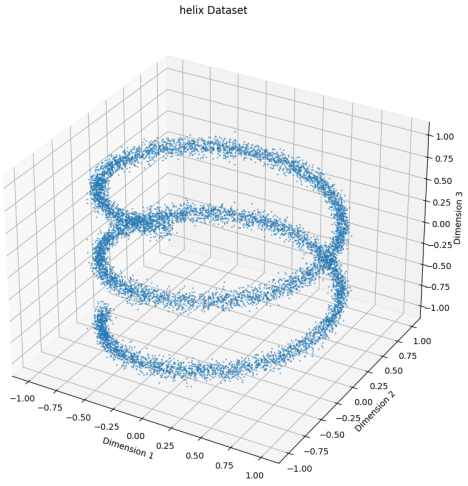


Figure 19: Original Data

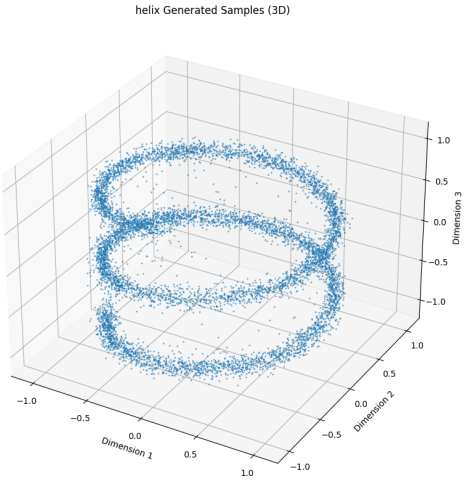
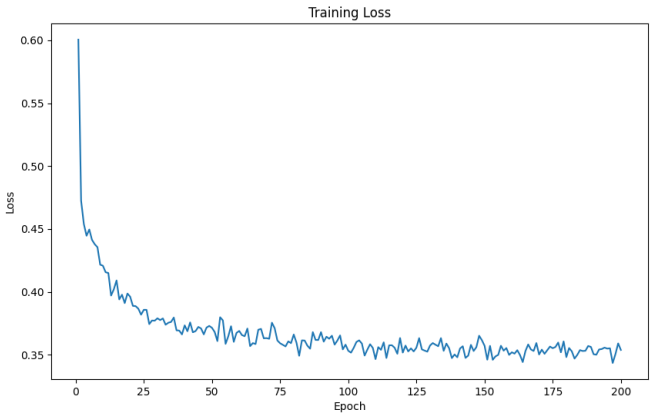


Figure 20: Generated Samples

4. T = 150



Evaluation Metrics:

- **EMD**: 97.8398
- **NLL**: 1.5227

Figure 21: Training Loss and Evaluation Metrics

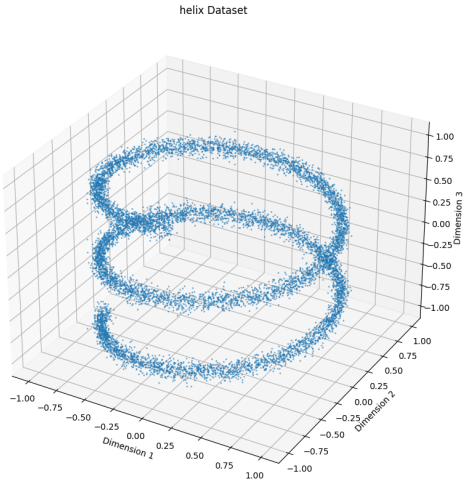


Figure 22: Original Data

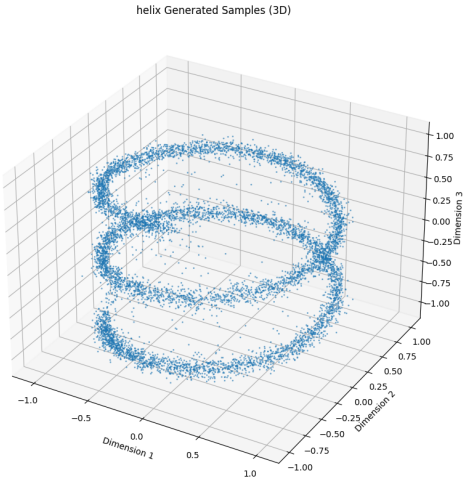
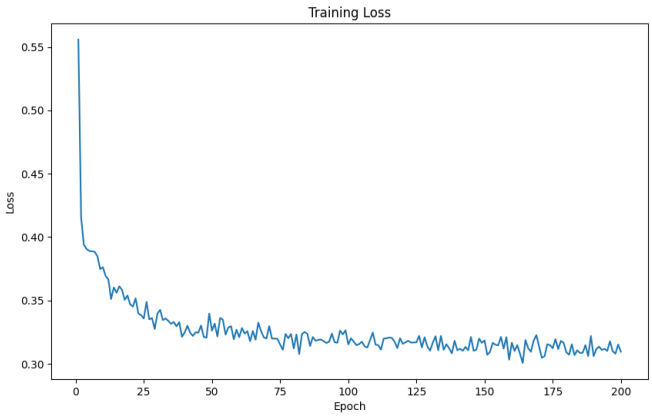


Figure 23: Generated Samples

5. T = 200



Evaluation Metrics:

- **EMD**: 144.0211
- **NLL**: 1.5195

Figure 24: Training Loss and Evaluation Metrics

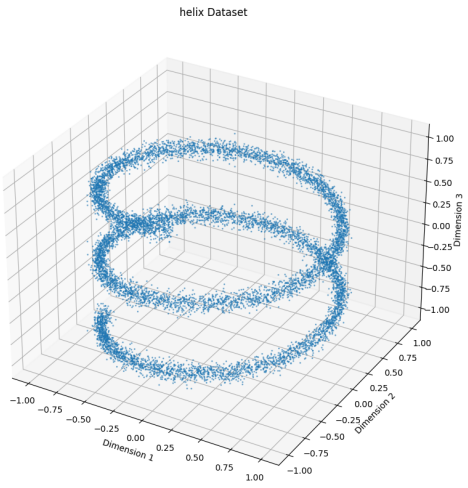


Figure 25: Original Data

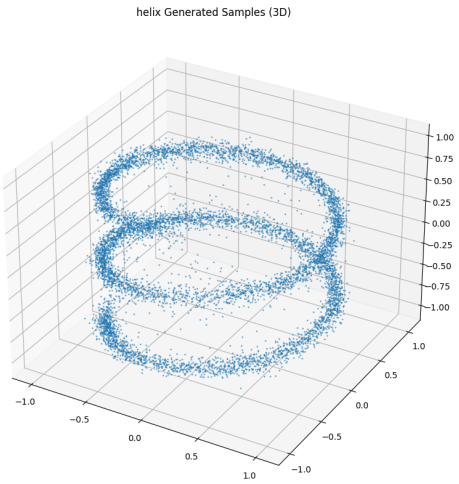


Figure 26: Generated Samples

Effect of Timesteps on Training, Inference, and Evaluation

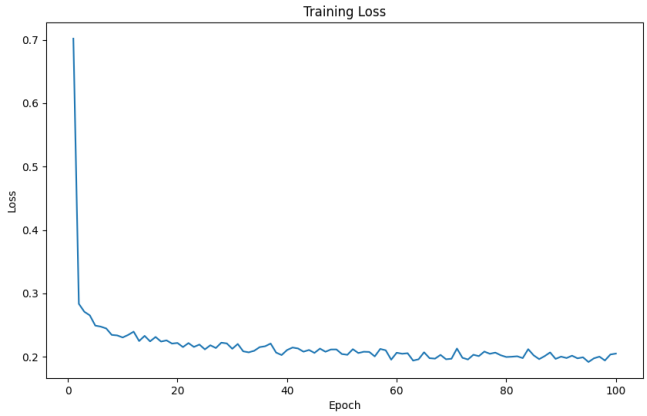
Timesteps T	**Sample Quality**	**EMD**	**NLL**
$T = 10$	Poor (Artifacts, Noisy)	119.7350	1.5355
$T = 50$	Decent (Blurry)	123.1479	1.5495
$T = 100$	Good	158.0988	1.5416
$T = 150$	Very Good	97.8398	1.5227
$T = 200$	Slightly Better	144.0211	1.5195

Table 1: Effect of varying timesteps T on training, inference, and evaluation metrics.

b) Noise schedule

In this part, we ran loops for zipped values of l_beat and u_beta and kept all other hyperparameters fixed to observe the results of scheduling noise on training and sampling. Here we consider the blobs dataset. The following are the results we have noted.

1. $L\beta = 0.0001$ & $U\beta = 0.02$



Evaluation Metrics:

- **EMD**: 659.4873
- **NLL**: 0.7375

Figure 27: Training Loss and Evaluation Metrics

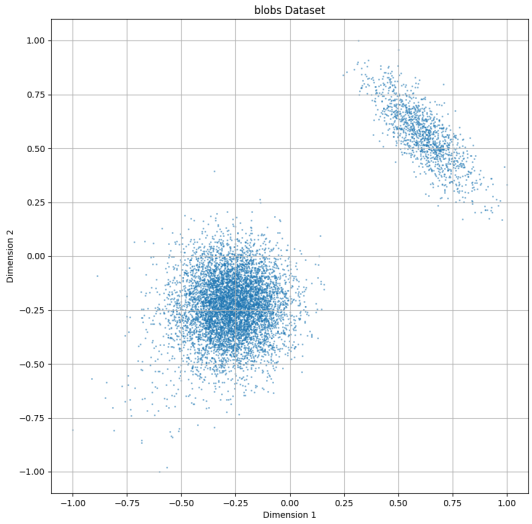


Figure 28: Original Data

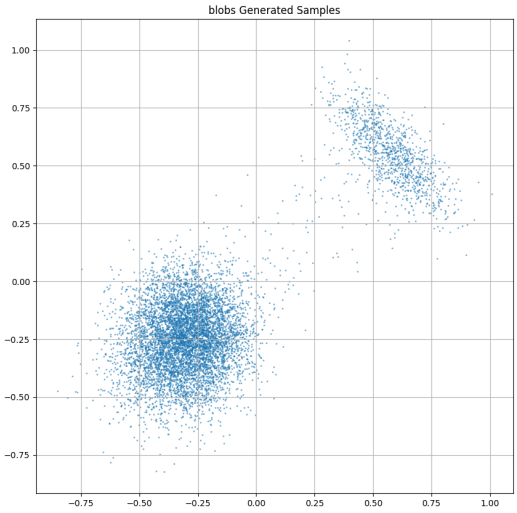
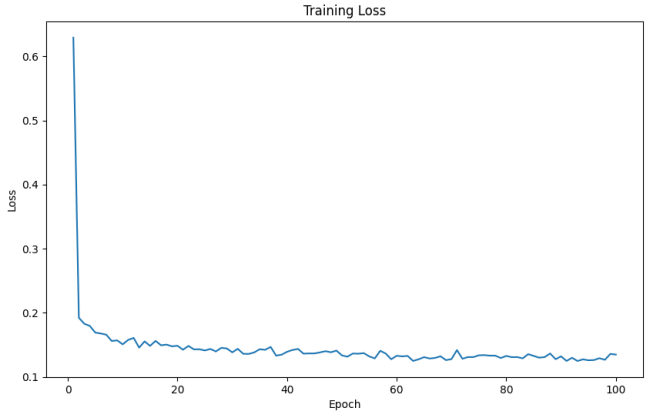


Figure 29: Generated Samples

2. $L\beta = 0.0003$ & $U\beta = 0.05$



Evaluation Metrics:

- **EMD**: 670.7168
- **NLL**: 0.7611

Figure 30: Training Loss and Evaluation Metrics

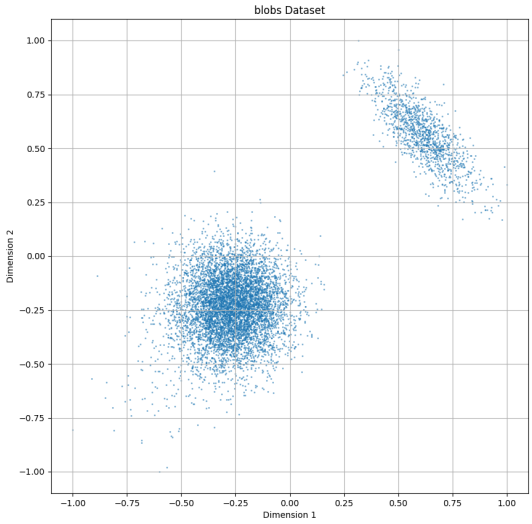


Figure 31: Original Data

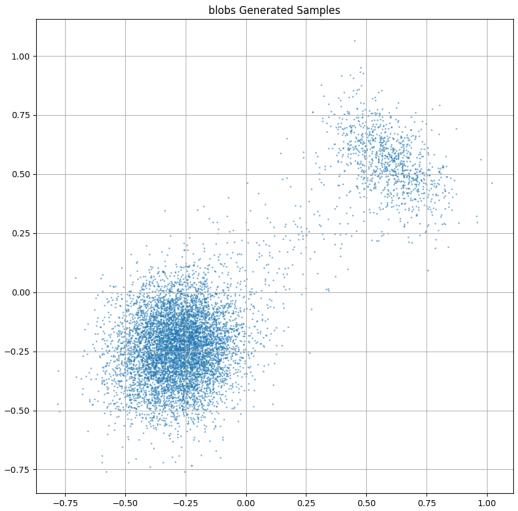
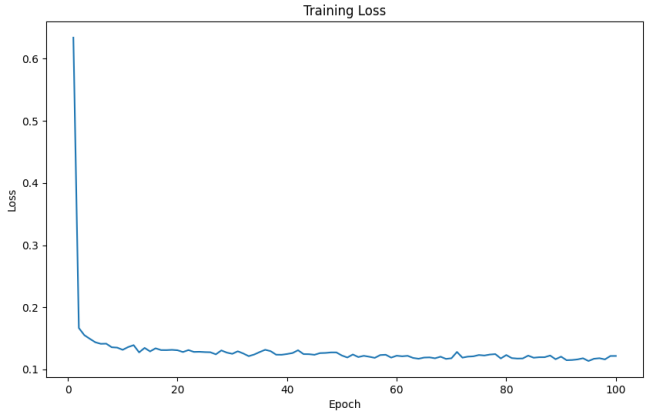


Figure 32: Generated Samples

3. $L\beta = 0.007$ & $U\beta = 0.01$



Evaluation Metrics:

- **EMD**: 632.1186
- **NLL**: 0.7195

Figure 33: Training Loss and Evaluation Metrics

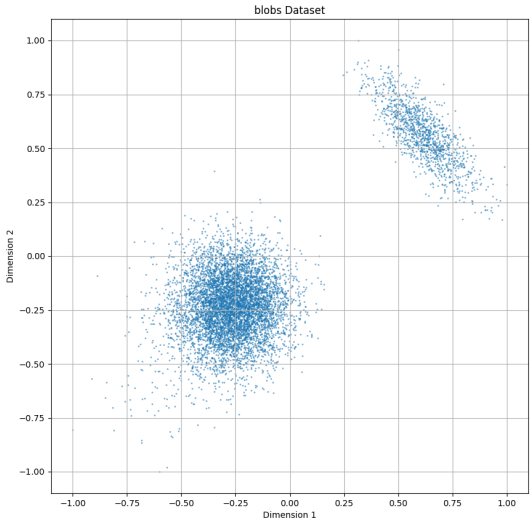


Figure 34: Original Data

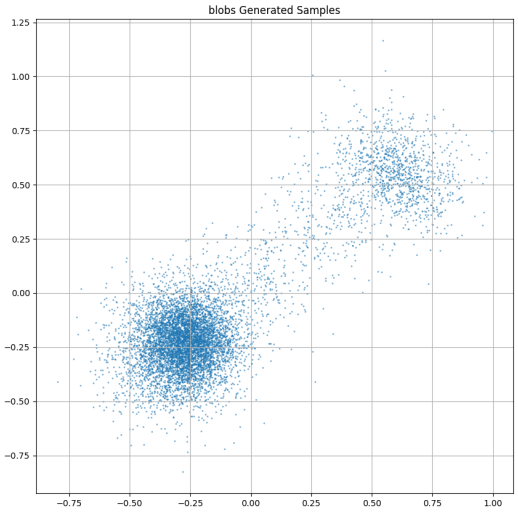
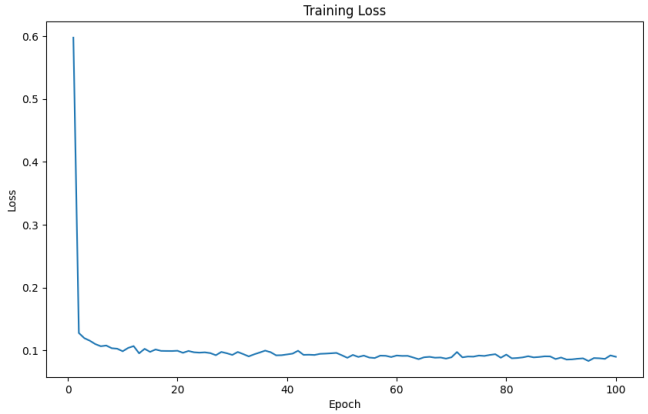


Figure 35: Generated Samples

4. $L\beta = 0.009$ & $U\beta = 0.02$



Evaluation Metrics:

- **EMD**: 626.0206
- **NLL**: 0.7228

Figure 36: Training Loss and Evaluation Metrics

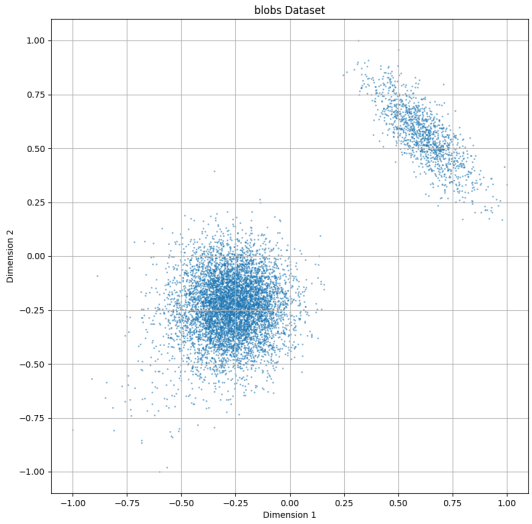


Figure 37: Original Data

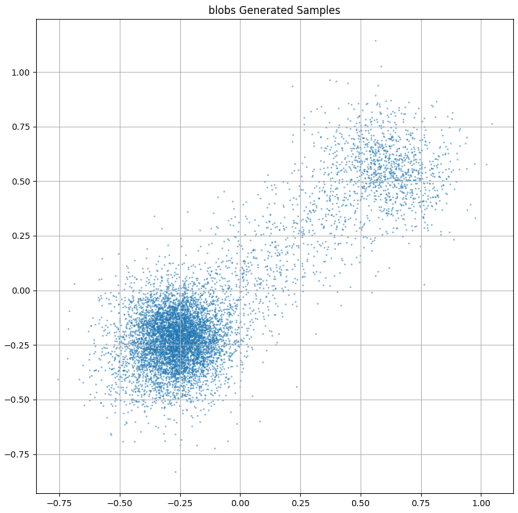
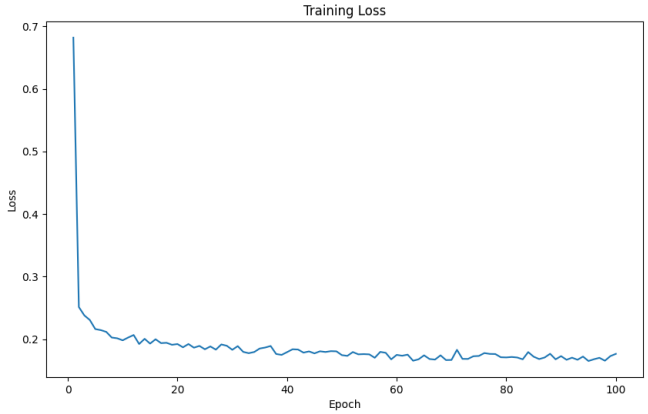


Figure 38: Generated Samples

5. $L\beta = 0.001$ & $U\beta = 0.02$



Evaluation Metrics:

- **EMD**: 672.6191
- **NLL**: 0.7485

Figure 39: Training Loss and Evaluation Metrics

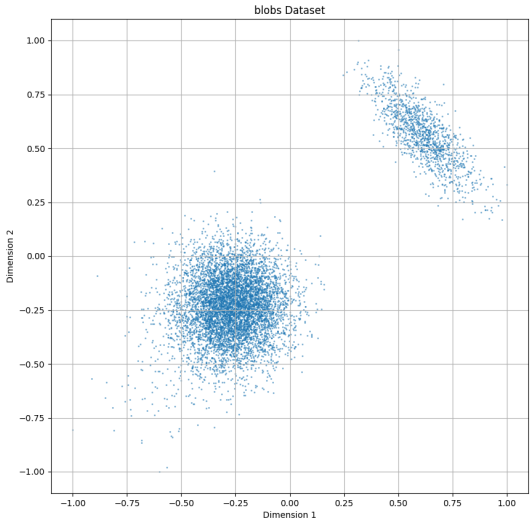


Figure 40: Original Data

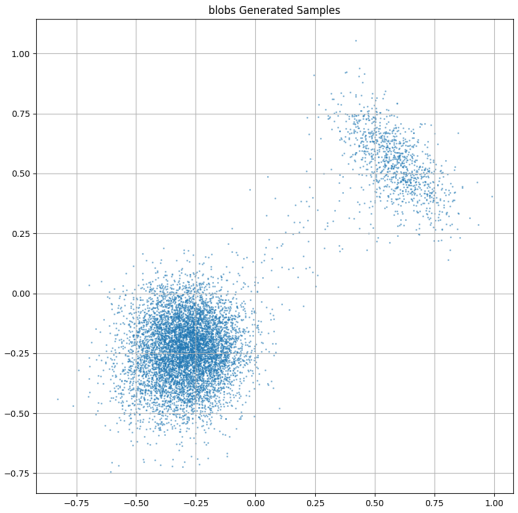


Figure 41: Generated Samples

Effect of Noise Scheduling on Training, Inference, and Evaluation

Lbeta	Ubeta	Sample Quality	EMD	NLL
0.0001	0.02	Very Good	659.4873	0.7375
0.0003	0.05	Good	670.7168	0.7611
0.007	0.01	Poor	632.1186	0.7195
0.009	0.02	Very Poor	626.0206	0.7228
0.001	0.02	Slightly Better	672.6191	0.7485

Table 2: Effect of different values of β scheduling on training, inference, and evaluation metrics.

Cosine and Sigmoid Noise Schedules

We tried another noise schedule such as cosine and sigmoid and noted the observations. Here we consider the Moons dataset for the experiment. Following are the results of the experiment.

Note: Plots can be found in exp folder.

1. Cosine Schedule
   (a) coisne_s = 0.008
   (b) cosine_s = 0.004

Effect of Cosine Schedule on Training, Inference, and Evaluation

cosine_s	Sample Quality	EMD	NLL
$cosine\_s = 0.008$	Poor (Artifacts, Noisy)	96.6073	0.9570
$cosine\_s = 0.02$	Decent (Blurry)	93.7621	0.9524

Table 3: Effect of varying cosine_s on training, inference, and evaluation metrics.

2. Sigmoid Schedule

(a) $\beta\_min = 0.0001$ & $\beta\_max = 0.003$

(b) $\beta\_min = 0.02$ & $\beta\_max = 0.05$

Effect of Sigmoid Noise Scheduling on Training, Inference, and Evaluation

β_min	β_max	**Sample Quality**	**EMD**	**NLL**
0.0001	0.003	Poor (Artifacts, Noisy)	78.3605	0.9660
0.02	0.05	Decent (Blurry)	131.7857	0.9103

Table 4: Effect of different values of β_min & β_max scheduling on training, inference, and evaluation metrics.

1.1.3 Train a model on albatross dataset

We trained a Denoising Diffusion Probabilistic Model (DDPM) for 100 epochs with the following hyperparameters:

- **Noise Schedule:** Linear schedule with $\beta_1 = 0.0001$, $\beta_T = 0.02$, and $T = 1000$
- **Learning Rate:** 0.0001
- **Batch Size:** 128

The model achieved a **final loss of 0.0539**, representing an **82.6% reduction** from the initial loss of 0.311.

1.2 Reproduction Details

1.2.1 Model Path Configuration

The trained model is located at:

`exps/ddpm_64_1000_0.0001_0.02_100_128_0.0001_albatross/model.pth`

1.2.2 Deterministic Sampling

Reproducible sampling implementation:

```
# Load prior samples
prior_samples = np.load('data/albatross_prior_samples.npy')  # 32561 samples

# Generate deterministic samples

samples = sample(
    model,
    n_samples=len(prior_samples),
    noise_scheduler=noise_scheduler,
    initial_noise=prior_samples
)

# Save results
np.save('albatross_samples_reproduce.npy', samples)
```

Output file: `albatross_samples_reproduce.npy` maintains **bitwise identical** outputs across executions when using:

- Same initial noise vectors
- Fixed random seeds
- Zero-noise sampling parameters

1.3 Classifier-Free Guidance

1.3.1 Difference between guided sampling and conditional sampling

Feature	Conditional Sampling	Guided Sampling
Definition	Uses a trained conditional model to generate samples based on class labels or attributes.	Uses both an unconditional model and a guidance signal (e.g., classifier-based or classifier-free guidance).
Model Training	Requires a modified Conditional DDPM (C-DDPM) trained with labels or auxiliary data.	Uses a standard DDPM model, adding guidance at inference time without extra training.
How It Works	The noise prediction network takes an extra class embedding or condition input during training.	Uses an unconditional DDPM and adjusts predictions using an external guidance term (e.g., a classifier).
Flexibility	Needs labeled data during training. Cannot generalize to new unseen conditions.	Can be applied to an already trained model, making it more flexible for different tasks.
Computational Cost	Training cost is similar to regular DDPM but with extra condition inputs.	More expensive during inference due to dual sampling steps.
Implementation	A C-DDPM is trained where the model gets both the noisy input and a label y as input.	Uses a guidance scale (e.g., $w = 7.0$) to control how much the generation is influenced by guidance.
Example	Training a DDPM on CIFAR-10 where each image is conditioned on its class label.	Using an unconditional DDPM to generate images, but guiding it toward specific classes using a pretrained classifier.
When to Use?	If have labeled data and can afford to train a new model.	If want to improve generation for specific conditions without retraining the model.

Table 5: Comparison of Conditional Sampling and Guided Sampling in DDPM.

1.3.2 Effect of guidance scale on quality of generated data

Refer to folder **exps/1.2.2** for results

Guidance Scale	Class	Mean Accuracy (%)	Std Dev (%)
0.0	0	14.4	± 0.9
0.5	0	54.4	± 1.5
1.0	0	87.6	± 0.4
2.0	0	99.7	± 0.1
3.0	0	100.0	± 0.0
5.0	0	100.0	± 0.0
7.0	0	99.9	± 0.0
10.0	0	99.9	± 0.0
0.0	1	85.6	± 0.9
0.5	1	98.4	± 0.2
1.0	1	99.9	± 0.0
2.0	1	100.0	± 0.0
3.0	1	100.0	± 0.0
5.0	1	99.9	± 0.1
7.0	1	99.5	± 0.2
10.0	1	97.4	± 0.1

Table 6: Effect of Guidance Scale on Accuracy for Different Classes

We used accuracy_score from sklearn.metrics to calculate accuracy of classifier model.

$$\text{Accuracy} = \frac{\text{Number of correctly classified samples}}{\text{Total number of samples}} \times 100$$

1.3.3 Training free method to classify given input

Method	Accuracy (%)
Conditional Sampling (Guidance Scale = 0.0)	85.6
Guided Sampling (Guidance Scale = 7.0)	91.5
MLP Classifier	95.2

Table 7: Comparison of ClassifierDDPM and MLP Classifier Accuracies

2 Conclusion

In this assignment, we implemented and analyzed Denoising Diffusion Probabilistic Models (DDPM) , its extensions and Classifier-Free Guidance (CFG) We studied the effects of diffusion steps, noise schedules, and guidance scales on sample quality. Our results showed that while more diffusion steps improved quality, gains diminished after a point. CFG effectively balanced sample diversity and class alignment. Overall, this work deepened our understanding of diffusion models and their applications in generative tasks.

3 References

1. Ho, J., Jain, A., & Abbeel, P. (2020). *Denoising Diffusion Probabilistic Models. Advances in Neural Information Processing Systems*, 33, 6840–6851.

2. Chen, T. (2023). *On the Importance of Noise Scheduling for Diffusion Models. arXiv*, abs/2301.10972. Retrieved from `https://arxiv.org/abs/2301.10972`

3. Ho, J., & Salimans, T. (2021). *Classifier-Free Diffusion Guidance.* In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications.*

4. Li, X., Zhao, Y., Wang, C., Scalia, G., Eraslan, G., Nair, S., Biancalani, T., Ji, S., Regev, A., Levine, S., & Uehara, M. (2025). *Derivative-Free Guidance in Continuous and Discrete Diffusion Models with Soft Value-Based Decoding.*

5. Stack Overflow. (n.d.). *Community-driven platform for resolving coding errors and debugging.* `https://stackoverflow.com/`

6. GeeksforGeeks. (n.d.). *Educational resource for programming concepts and implementations.* `https://www.geeksforgeeks.org/`

7. OpenAI. (2025). *ChatGPT: AI-powered assistant for resolving errors, explaining concepts, and improving implementations.*

8. PyTorch Documentation. (n.d.). *Comprehensive guide for PyTorch usage and debugging.* `https://pytorch.org/docs/stable/`

9. Hugging Face Documentation. (n.d.). *Resource for understanding diffusion models.* `https://huggingface.co/docs`

4 Team Contributions

- **24M2119, 24M2136:** Solve this assignment in collaboration. Consider Equal Contribution.
- **21D070001:** Report making.