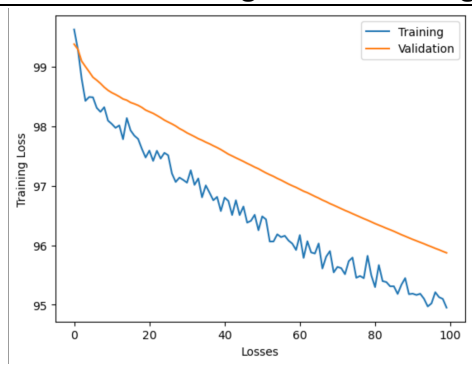
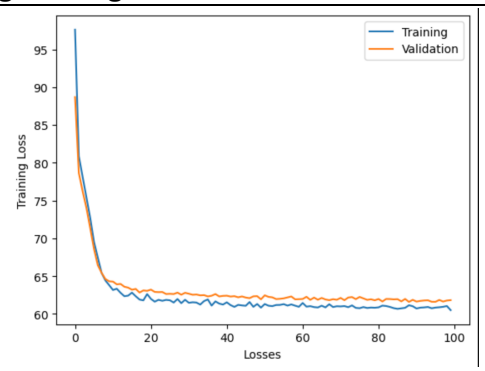
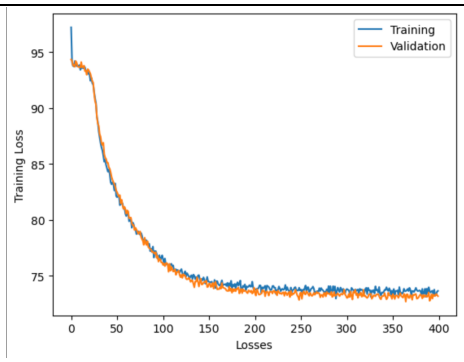
Denoising Score Matching - a technique used in unsupervised learning to train a model (such as a neural network) by matching the gradient of the log probability density of the data. Adam optimizer is used to minimize the denoising score matching loss during training.



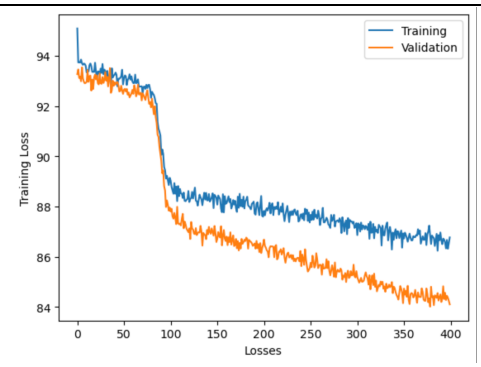
CHECKERBOARD



GAUSSIAN

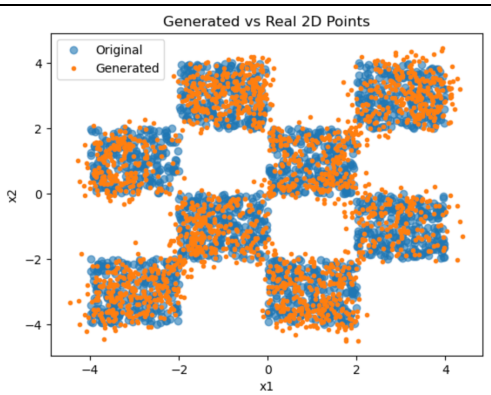


PINWHEEL

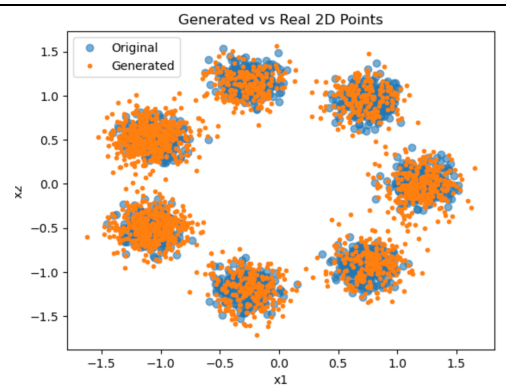


SPIRAL

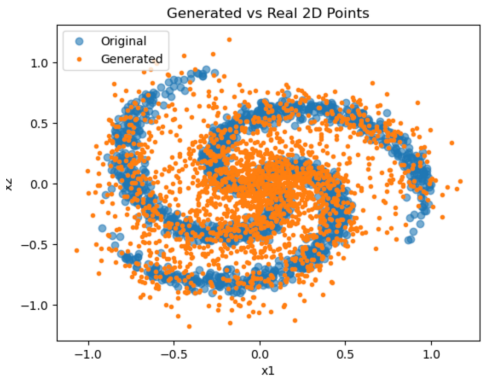
Table 1- DSM Loss Curves



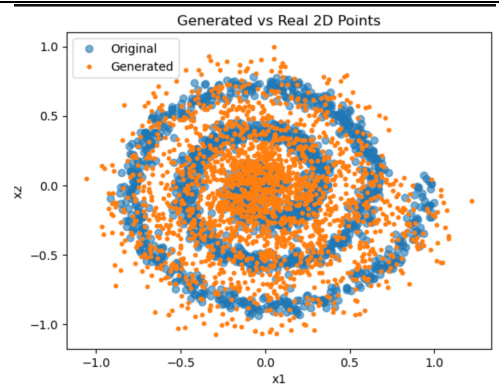
CHECKERBOARD



GAUSSIAN



PINWHEEL



SPIRAL

Table 2 - Data Sampled using DSM

Denoising Diffusion Probabilistic Models – Diffusion technique, noise is slowly added to data until it turns to pure noise and the process is reversed to get back the original (or SIMILAR to original) dataset.

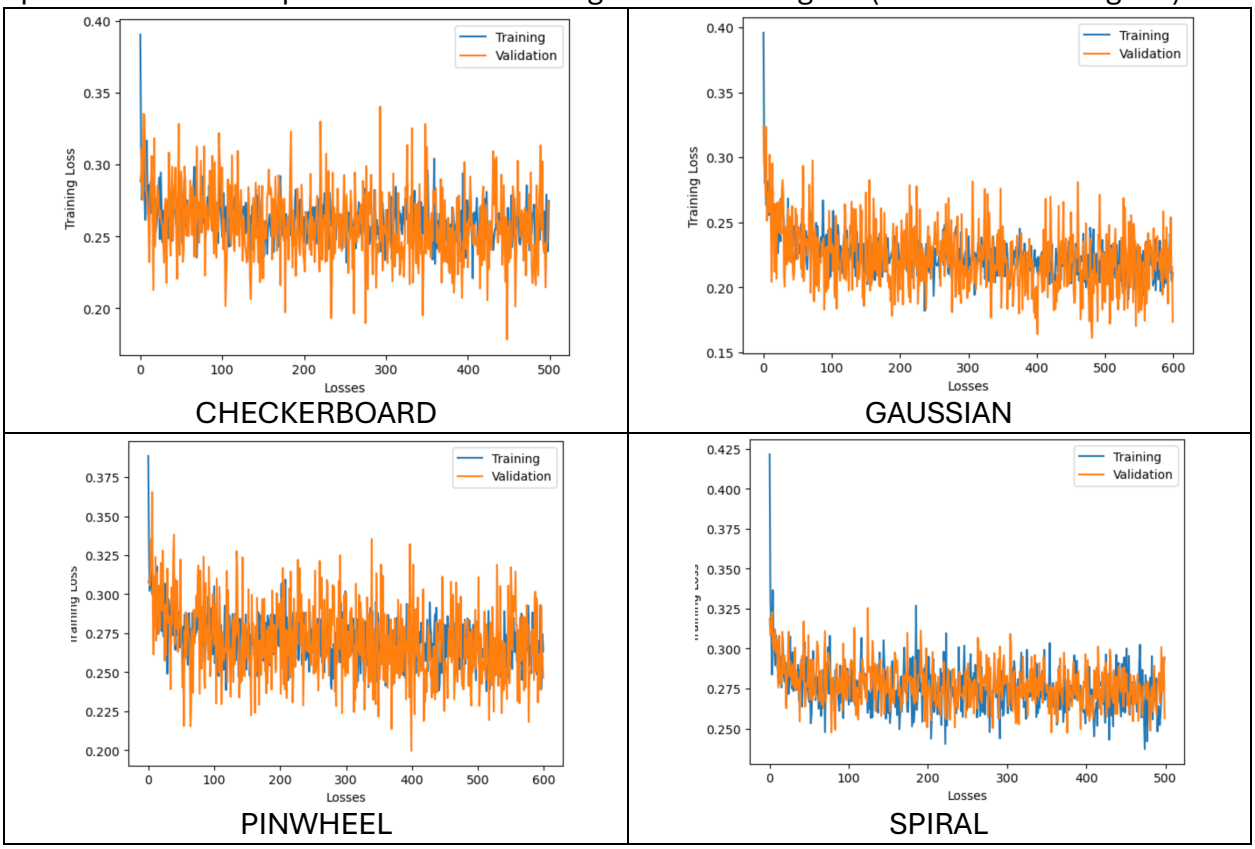


Table 3 - Loss Curves using DDPM

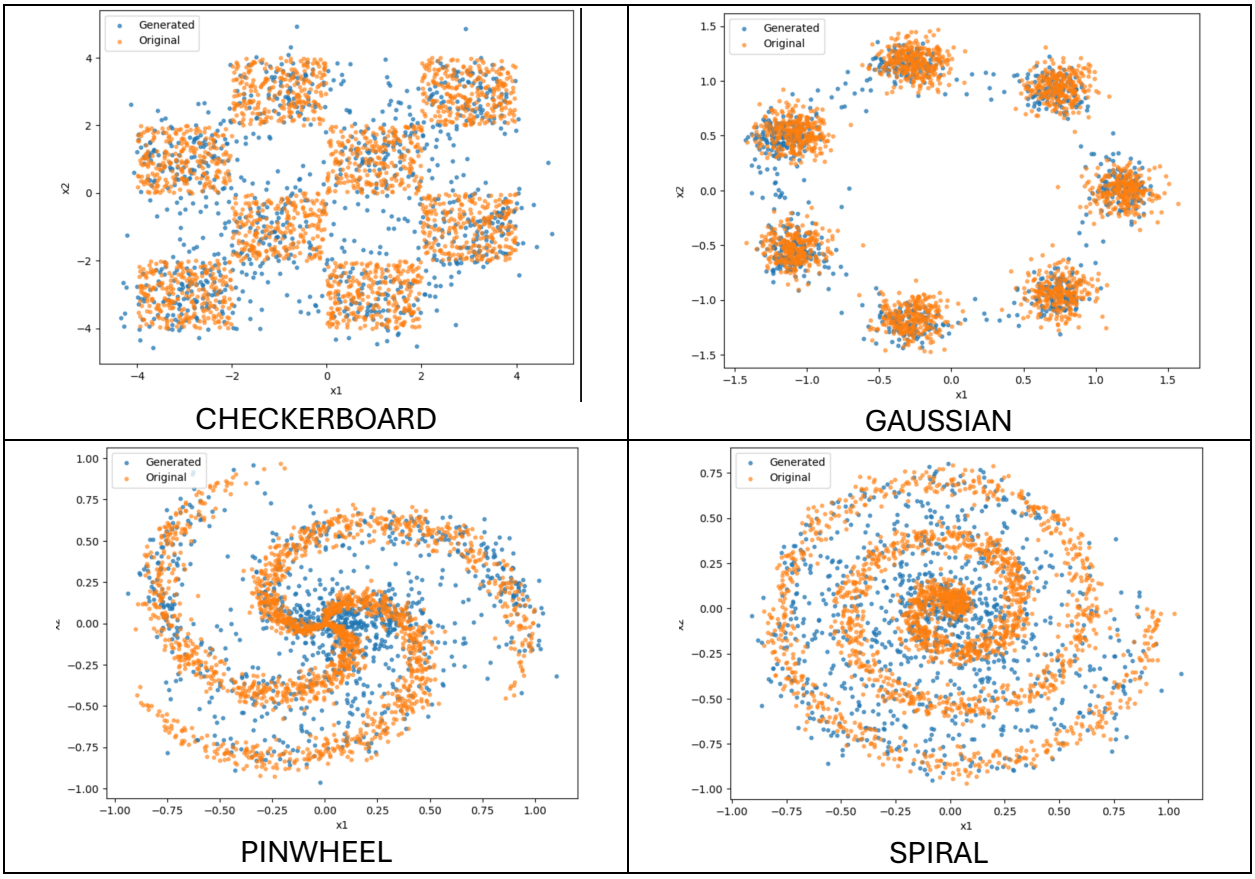


Table 4 - Data Sampled using DDPM

Generative Adversarial Networks (GANs) – This consists of 2 neural networks working against each other. The generator must create new samples like original training dataset, and the discriminator is supposed to identify if data is original or newly generated. Working against each other, both keep getting better until the generator can sufficiently fool a good discriminator and we can then sample new datapoints like the original training dataset.

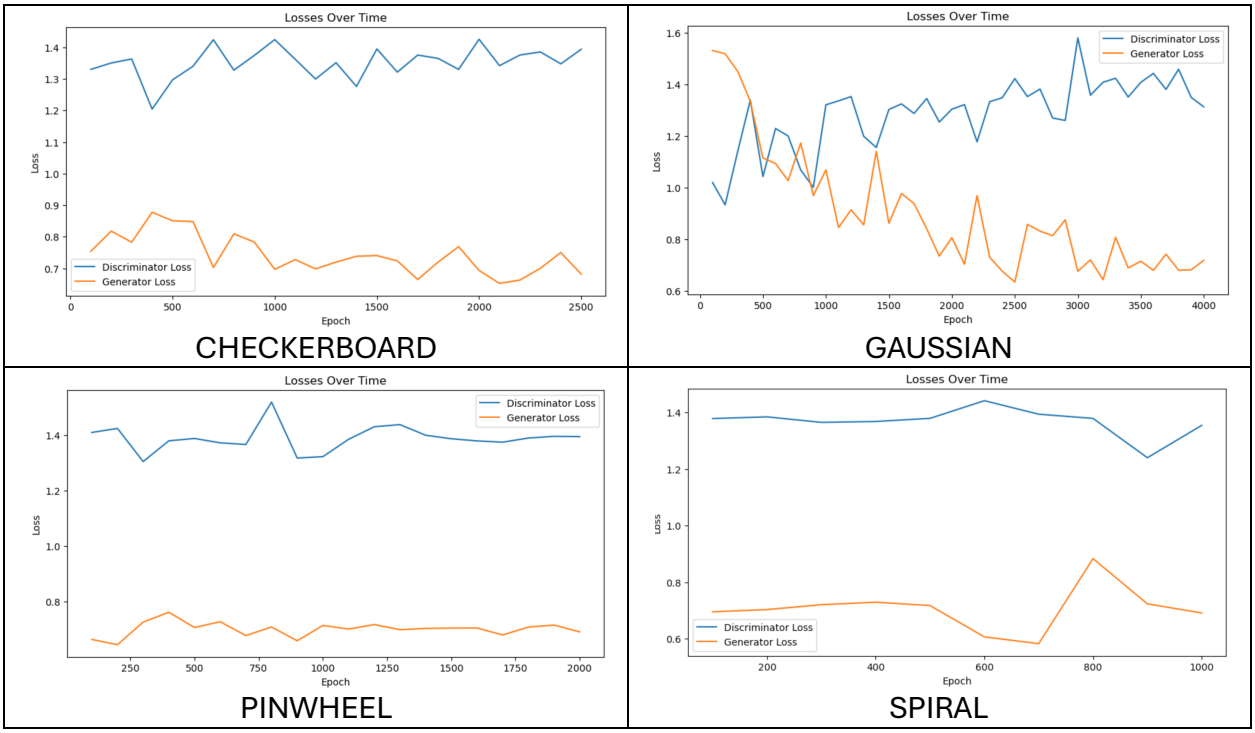


Table 5 - Loss Curves using GANs

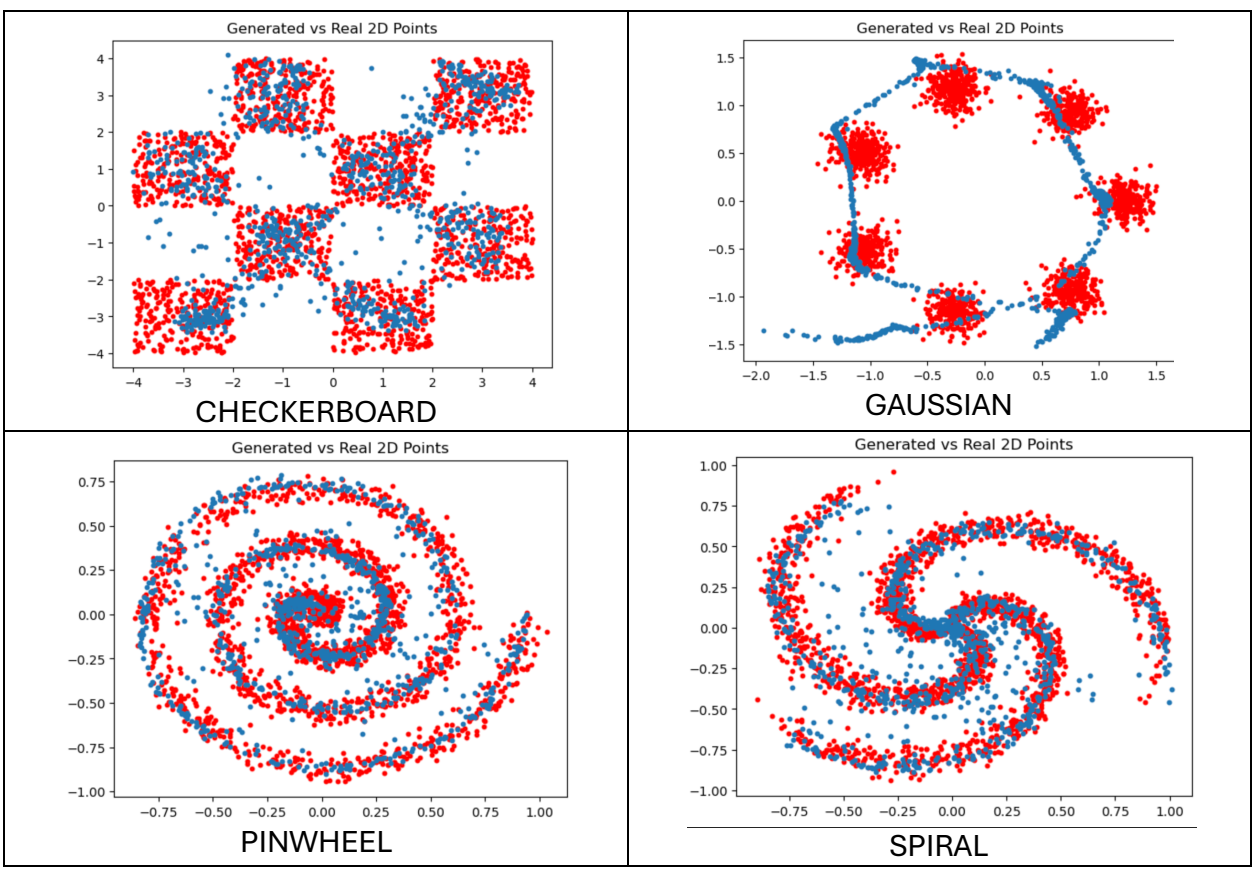


Table 6 - Data Sampled using GANs

Qualitative Comparison –
From the plots of generated samples alongside original training dataset, we can see DSM and GANs perform very well, as can also be seen from the training and validation curves as well. For Gaussian dataset, DSM was the best method. GANs were best for checkerboard and spiral dataset, and DDPM worked best for pinwheel dataset.

Quantitative Comparison –
The method used for quantitative comparison is Kernel Density Estimation and Log Likelihood. KDE is a way to estimate PDF of a random variable, used when we want to infer underlying probability distribution given a set of samples.
Log Likelihood is a measure used to evaluate the probability of observing a given data under a specific statistical model. In our method, we use it to measure the similarity between the generated data and the actual data from a holdout set generated using the data generator function.
Probability values are between 0 and 1, and if we take logarithm, hence we get a negative final value. At the end, since all values are negative, so we want values as close to 0 so they are overall bigger. Hence, this is one of the numerical ways to evaluate performance of different models.

	CHECKERBOARD	GAUSSIAN	PINWHEEL	SPIRAL
DSM	-4.07	-1.52	-0.87	-0.87
DDPM	-4.16	-1.54	-0.83	-0.85
GANs	-4.03	-2.00	-1.05	-0.84

GANS

```python
class Generator(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(Generator, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.BatchNorm1d(128),
            nn.ReLU(True),
            nn.Linear(128, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(True),
            nn.Linear(512, output_dim),
        )

    def forward(self, x):
        return self.net(x)

class Discriminator(nn.Module):
    def __init__(self, input_dim):
        super(Discriminator, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.1),
            nn.Linear(512, 128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.1),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.net(x)
```

DDPM

```python
class DiffusionModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(DiffusionModel, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, 8*hidden_dim)
        self.fc3 = nn.Linear(8*hidden_dim, hidden_dim)
        self.fc4 = nn.Linear(hidden_dim, output_dim)
        self.relu = nn.ReLU()
        self.proj1 = nn.Linear(hidden_dim, 8*hidden_dim)
        self.proj2 = nn.Linear(8*hidden_dim, hidden_dim)

    def forward(self, x, t):
        x = torch.cat((x, t), dim=1)
        x1 = self.relu(self.fc1(x))
        x2 = self.relu(self.fc2(x1)) + self.proj1(x1)  # A
        x3 = self.relu(self.fc3(x2)) + self.proj2(x2)  # A
        return self.fc4(x3)
```

Notes –
All code, files, exported model files, numpy files of generated samples, plots, hyperparameters, model architecture are zipped in the attached file.