Exploring the Combination of Diffusion Models and Generative Adversarial Networks

Group 11
Lauren Rouse (s4742379) · Malaika Vaz (s4699270) · Xuran Wang (s4838862)
Ganesh Channaiah (s4919002) · Mats Martinussen (s4946842)

1 Introduction

In the past few years, generative models have made great progress in the field of image generation. Popular systems such as OpenAI's DALL-E (Betker et al., 2023) and GPT-4o (OpenAI et al., 2024) have demonstrated how powerful generative AI can be in creating high quality images. This advance largely rely on deep learning architectures like Generative Adversarial Networks (GANs) and diffusion models. Our project explores the idea of combining the two approaches into a hybrid model, with the goal of producing the best performing model possible.

This approach has seen some success in recent years, as highlighted by the results of Kim et al. (2024). Their paper suggests that GAN and diffusion models can be combined in a way that synergistically enhances the quality of generated images. We intend to employ a similar idea in our project to fine-tune an existing diffusion model by attaching a discriminator, as in GANs.

Our project uses pre-trained open-source diffusion and GAN models. We also trained diffusion models from scratch—not to optimize their standalone performance, but to enable discriminator feedback at key points in the training loop. This let us experiment with various training strategies and analyze their effect on performance.

In particular, we found that our approach leads to noticeable improvements in the more challenging setting of unconditioned image generation, where the model has no label or category guidance. Considering that we use limited computing resources, this result is particularly encouraging. It indicates that even lightweight adversarial feedback can enhance the realism of diffusion-generated images.

Our approach involves fine-tuning existing diffusion models using feedback from the discriminator from a GAN-architecture. The diffusion model follows a standard U-Net architecture, and the discriminator follows different forms of convolutional neural network architectures. These will be described in more detail later on.

Initially, we tested a variety of methods on an unconditioned diffusion model for the MNIST dataset. We then employed the best techniques on both a conditioned model for the MNIST dataset, as well as an unconditioned model for the CIFAR-10 dataset.

We quickly realized there were many potential methods to explore. The key differences lay in what the discriminator was tasked with predicting—such as the final image, the noise, or a noisy intermediate—and when its feedback was applied to the diffusion model: during a single denoising step, after the full denoising process, or only at sampling time. Not all approaches lead to clear improvements, but they all helped us better understand the limits of our method.

We make our code freely available on GitHub[1].

2 Related Work

This project has been primarily inspired by the work of Kim et al. (2024). Their paper proposed a simple mapping network to link the latent spaces of a pre-trained GAN and a diffusion model with the aim of generating photo-realistic face images that outperformed existing diffusion and GAN-based methods. This was achieved using a learning-based GAN inversion, with a diffusion model serving as the encoder (Kim et al., 2024).

[1] https://github.com/Joov95/stat3007-G11-2025

The models employed to execute this method were as follows: StyleGAN and EG3D were used as pre-trained 2D and 3D GANs, respectively, and ControlNet was selected as the diffusion-based encoder (Kim et al., 2024). The model architecture was trained and tested on the CelebAMask-HQ dataset (Lee et al., 2019), and the evaluation of the method using text prompts with semantic masks or scribble maps as input for the 2D and 3D versions is shown in Figure 1

Input conditions	Method	Model	Domain	FID↓	LPIPS↓	SSIM↑	ID↑	ACC↑	mIoU↑
Text + semantic mask	TediGAN [58]	GAN	2D	54.83	0.31	0.62	0.63	81.68	40.01
	IDE-3D [51]	GAN	3D	**39.05**	0.40	0.41	0.54	47.07	10.98
	UaC [35]	Diffusion	2D	45.87	0.38	0.59	0.32	81.49	42.68
	ControlNet [62]	Diffusion	2D	46.41	0.41	0.53	0.30	82.42	42.77
	Collaborative [19]	Diffusion	2D	48.23	0.39	0.62	0.31	74.06	30.69
	Ours	GAN	2D	46.68	0.30	0.63	0.76	**83.41**	**43.82**
	Ours	GAN	3D	44.91	**0.28**	**0.64**	**0.78**	83.05	43.74
Text + scribble map	ControlNet [62]	Diffusion	2D	93.26	0.52	0.25	0.21	-	-
	Ours	GAN	2D	55.60	**0.32**	**0.56**	**0.72**	-	-
	Ours	GAN	3D	**48.76**	0.34	0.49	0.62	-	-

Figure 1: Quantitative results of multi-modal face image generation on CelebAMask-HQ with annotated text prompts

Kim et al. (2023) take a different approach by introducing discriminator guidance at sampling time, rather than during training. Their method refines pre-trained diffusion models by adjusting the denoising process based on feedback from a separately trained discriminator, leading to improved image realism without requiring joint training.

In general, the work of Kim et al. (2024) and Kim et al. (2023) shows promising results in experimenting with the combination of two distinct deep learning architectures to enhance image generation. The results clearly show that it is possible to integrate GANs and diffusion models in a way that leverages the strengths of both methods to create a hybrid model that outperforms existing GAN and diffusion-based methods. Building on this, our project takes a different approach by integrating discriminator guidance directly into the diffusion process rather than at the latent or sampling level.

3 Methods

This section outlines the two models used to build our hybrid architecture: a diffusion model and the discriminator component of a GAN. While we experimented with variations—such as conditioned vs. unconditioned diffusion models, and time-conditioned vs. unconditioned discriminators with either single or multiple inputs—the overall architectures remained consistent. Minor adjustments were made depending on the dataset (e.g., MNIST vs. CIFAR-10) to account for differences in input size.

3.1 Diffusion Model

Due to their architecture, diffusion models are generally expensive and time consuming to train. Given this, and more importantly, our time and resources constraint for this project, we decided to use an architecture we knew would work, namely the Google's Deep Denoising Diffusion Probabilistic Model (DDPM) Ho, Jain, and Abbeel (2020) as our diffusion model. The DDPM model is a parameterized Markov chain trained using variational inference to generate data samples that match a target distribution.(Ho, Jain, and Abbeel, 2020).

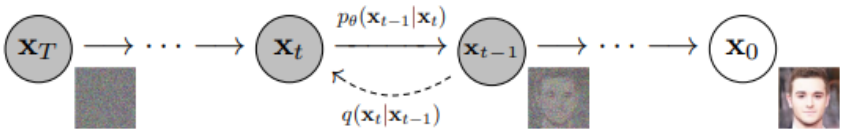

Figure 2: Directed graphical model (from Ho, Jain, and Abbeel (2020))

In the forward diffusion process, Gaussian noise is gradually added to the original data sample over a series of steps, following a fixed variance schedule β_t , until the image becomes nearly indistinguishable from random noise (Ho, Jain, and Abbeel, 2020). The reverse process aims to undo this corruption. A U-Net is trained to denoise the data step by step, approximating the reverse transitions of the diffusion process and reconstructing an image from noise.

During training, the model learns to predict the noise added at each timestep using a simple mean squared error loss. While the objective is derived from a variational framework, it reduces in practice to a straightforward regression task. The training and sampling procedures are outlined in Figure 3.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure 3: Training and Sampling algorithm for DDPM

3.2 Generative Adversarial Network

A typical GAN features an adversarial framework comprised of two distinct models: a generator and a discriminator. The aim of the generator is to create new data samples from the training data distribution, while the discriminator attempts to distinguish between real and generated data from the generator.

The Deep Convolutional Generative Adversarial Network (DCGAN) is an extension of this framework, first proposed by Radford, Met, and Chintala (2016), that uses convolutional net architecture in both the generator and the discriminator, and applies batch normalisation to make the DCGAN stable to train.

For our problem, we incorporated the DCGAN discriminator in our hybrid model, using the feedback from the discriminator to fine-tune our diffusion model. As with our diffusion model, we have chosen to use a discriminator from an open source DCGAN model that has been proven to work on our datasets[2].

Although pre-trained weights are available, we trained the discriminator from scratch in most methods, as it was designed to predict noise rather than images, which the pre-trained version is predicting. The architecture of our chosen discriminator is shown in Figure4, with the input size being adjusted according to the dataset.

[2] `https://github.com/csinva/gan-vae-pretrained-pytorch/blob/master/mnist_dcgan/dcgan.py`

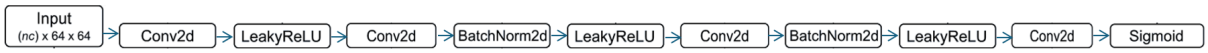

Figure 4: DCGAN Discriminator Architecture

The discriminator is comprised of multiple strided convolution layers, batch norm layers, and LeakyReLU activations. It takes as input a 1x64x64 (can be changed) input image, processes it, and, through a Sigmoid activation function, outputs a scalar probability that the input is from the real data distribution (Inkawhich, 2018).

The code for our discriminator implementation is given in the appendix, Figure 15.

4 Experiments

This section will detail the specifics of our experimental methodology to develop the hybrid model, including the datasets utilised, the training details, the evaluation metrics used, and our findings for each method trialed. The details of the different algorithm training methods are structured according to the order of experimentation.

4.1 Dataset

The datasets used for this project were MNIST and CIFAR-10.

The MNIST dataset is a collection of grayscale handwritten digits, along with associated labels. The images represent digits from 0 to 9, and each image has a resolution of 28x28 pixels. This dataset contains 60,000 training images and 10,000 testing images. The MNIST dataset was chosen for this project with the aim of performing smaller-scale experiments to determine the best performing techniques while developing our algorithm. These findings would then be incorporated to extend our model to the CIFAR-10 dataset. A sample can be found in the appendix, Figure 13.

The CIFAR-10 dataset is a collection of 60,000 colour images along with associated labels. The CIFAR-10 images have a resolution of 32x32 pixels. Each image is from one of ten mutually exclusive classes, with each class comprising of 6000 images. The ten classes are: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is separated into 5 training batches and 1 test batch, each with 10,000 images (that is a roughly 80:20 split for training and testing). A sample can be found in the appendix, Figure 14.

4.2 Algorithms

DDPM model The diffusion model was trained using an A100 GPU with a batch size of 128 and a schedule of beta values from 1e-4 to 0.02 with 300 timesteps. The model was trained on 5 epochs. An Adam optimizer was used with a learning rate of 1e-3 and an MSE loss function.

Two versions of this model were trained: an unconditioned UNet and a class conditioned UNet model (both trained on MNIST). The key difference with the class conditioned UNet model was that a one hot encoded representation of a number was included in input and classifier-free guidance was added in sampling. Additionally, we attempted to fine-tune an unconditioned UNet that was trained on CIFAR-10.

Method 1: Fine-tuning diffusion model based on feedback from full denoising process. In our first attempt at improving the model, only the diffusion model was updated based on feedback on the final output of the diffusion model. The hyperparameter combination used were:

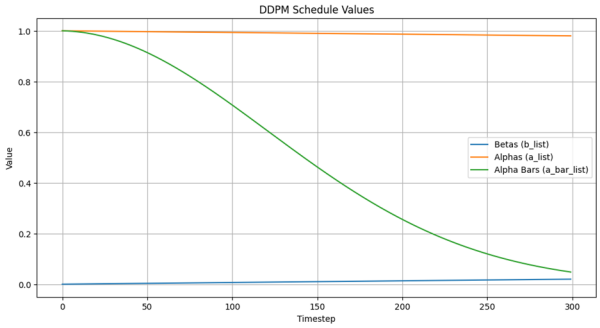

Figure 5: DDPM schedule values at each timestep

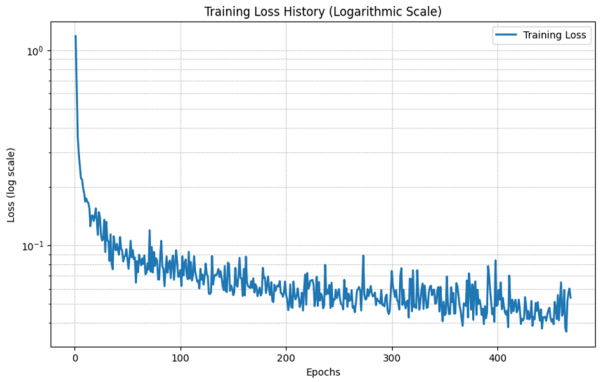

Figure 6: Training loss for the class conditioned UNet

- Loss function: BCE (binary cross entropy) Loss
- num iter: 10
- Learning rate - discriminator: 1e-4
- num timesteps: 300
- seed offset: 1
- Optimizer: Adam

Method 2: Fine-tuning both the diffusion model and discriminator based on feedback from full denoising process. To overcome the challenges of Method 1 (the diffusion model very quickly fooled the discriminator), we decided to experiment with updating both the diffusion model and the discriminator based on the final output of the diffusion mode. This method showed improvement, however, a very small batch size need to be used to avoid GPU memory issues due to the high compute requirements of backtracking the gradient through all 300 denoising iterations. As such, this was insufficient to achieve meaningful results on the performance of this technique.

Method 3: Fine-tune diffusion model, train discriminator from scratch, both on noise. In this method, we attempted to have the discriminator determine whether the given data is real noise or generated noise, rather than attempting to distinguish real vs generated for images. This method provided a better balance between the two models. Initially, the discriminator alone was trained for one epoch, then both the diffusion model and the discriminator were trained for 4 more epochs. The initial hyperparameters used were:

- Loss function: bce (binary cross entropy)

- Weight initialization of discriminator: Pytorch default
- Learning rate - discriminator: 1e-4
- Learning rate - generator: 1e-5
- batch size: 128
- timesteps: 300
- Optimizer: Adam, beta1 = 1e-4 and beta2 = 0.02

Through trial and error, the optimal hyperparameters were determined (discussed further in Results).

Method 3.5: Time-conditioned discriminator. An extension of the previous method described, the time-conditioned discriminator was the most notable variation of the algorithm attempted, as it provided the best result of all trialled methods. In this method, the time-step, i.e. where in the denoising process we are, was also provided to the discriminator.

Method 4: Train on the next generated image in the denoising loop instead of noise. In this method, we simply follow the sampling algorithm to generate the next image in the denoising loop using the estimated noise from the diffusion model. As this only involves some mathematical operations on the diffusion model's output, this generally does not affect backpropagation through the diffusion model. The idea was that by looking at something more realistic, the discriminator could be better at distinguishing real from fake. We attempted to increase model complexity by adding an additional convolutional layer and making the layers wider; however, this did not have any impact on performance.

4.3 Metrics

The metrics used to evaluate our generative models were described by Betzalel et al., 2022. They aim to provide objective measurements of qualitative features of the models such as diversity and realism. They are not perfect metrics; this will be elaborated upon in each score's subsection. However, they should be good enough to at least provide a comparison between our best models.

4.3.1 Inception Score (IS)

The Inception Score (IS) utilizes a pre-trained Inception v3 classifier (Szegedy et al., 2015). It is a common metric used when evaluating generative image models. It is based on two principles:

- The conditional label distribution $p_\theta(y|x)$ should have low entropy (each image should clearly belong to one class).
- The marginal label distribution $p_\theta(y)$ should have high entropy (the set of images should cover many classes).

Unfortunately, in our case, the first assumption does not necessarily hold. This is because Inception v3 was trained on ImageNet, whilst our models are trained on MNIST and CIFAR-10. Hence, the classes for our use case will be different to the classes used by the Inception model; each image in our dataset may not belong cleanly to one ImageNet class.

One way around this would be to use transfer learning to retrain the fully connected step of Inception[3]. Inception would then be able to classify images on the MNIST and CIFAR-10

[3]This idea was actually attempted by S. Colianni on Kaggle, achieving roughly 95% classification accuracy on MNIST (https://www.kaggle.com/code/scolianni/how-good-is-inception-v3-at-mnist).

datasets, and the first assumption could hold. Due to time and resource constraints, this idea was discarded and so we proceed with an imperfect metric.

IS is computed as:

$$IS = \exp\left(\mathbb{E}_{x \sim p_G}\left[KL(p_\theta(y|x) \| p_\theta(y))\right]\right)$$

A higher score implies that the generated images are both distinctive and diverse. The code for our implementation of IS is shown in the appendix, Figure 16.

4.3.2 Fréchet Inception Distance (FID)

The next metric against which our models will be evaluated is the so-called Fréchet Inception Distance (FID). This score also utilizes an Inception v3 model - however, the classifications themselves are not used. Instead, the features that are extracted before the final fully-connected (classification) layer are used to generate multivariate Gaussian distributions of both the real and generated images. These distributions are then compared using the FID score, where generated images have distribution $N(\mu_g, \Sigma_g)$ and real images have distribution $N(\mu_r, \Sigma_r)$.

$$FID = \|\mu_r - \mu_g\|^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

A lower score implies that the two distributions are more similar to one another - this can be interpreted as the generated images being similar to the training images i.e. more "realistic."

Since the FID only uses the features of Inception v3 and not the classes themselves, this should be a more reliable metric for our use case. Unfortunately, FID uses the sample covariance, which is a biased estimator of covariance. After the non-linear transformations used to calculate FID, the bias is not negligible - in particular, with low sample sizes, FID tends to be much higher. The end result of this is that FID may still be helpful for comparing models, though the exact numbers we achieve should not be compared to other FID scores as ours will be substantially higher due to low sample sizes.

The code for our FID implementation is shown in the appendix, Figure 17.

4.3.3 Kernel Inception Distance (KID)

The final metric used for model comparison is the so-called Kernel Inception Distance (KID). This functions quite similarly to the FID score - the main difference, however, is that KID uses the features of Inception to generate distributions with polynomial kernels, which are then compared using Maximum Mean Discrepancy (MMD). Since no covariance matrix is required, the score remains statistically unbiased - this makes it particularly useful in our case, where we are unable to work with large sample sizes. Similarly to FID, a lower KID score means that the distributions are more similar to one another.

The KID (or, more accurately, the MMD that we compute between kernels) is given by:

$$MMD^2(X, Y) = \frac{1}{m(m-1)} \sum_i \sum_{j \neq i} k(\mathbf{r_i}, \mathbf{r_j}) - 2\frac{1}{mn} \sum_i \sum_j k(\mathbf{r_i}, \mathbf{g_j}) + \frac{1}{n(n-1)} \sum_i \sum_{j \neq i} k(\mathbf{g_i}, \mathbf{g_j})$$

where $(g_1, g_2, ..., g_n)$ are the generated samples and $(r_1, r_2, ..., r_m)$ are the real samples.

The code implementation of this is shown in the appendix, Figure 18.

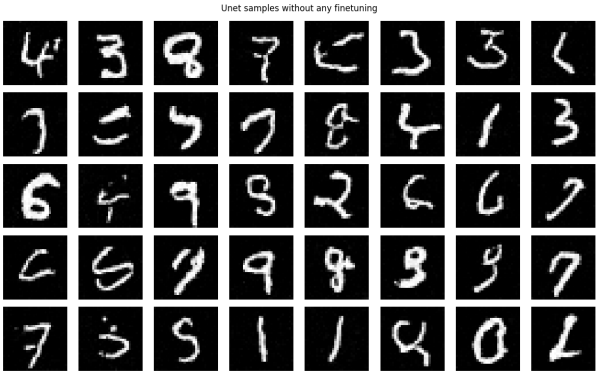

Figure 7: Unconditioned model with no fine-tuning applied.

4.4 Results

4.4.1 MNIST - Unconditioned

Below is a sample of images produced by the diffusion model with no fine-tuning from the discriminator.

The model was then fine-tuned via a variety of methods.

Method 1: only updating the diffusion model based on feedback on the full denoising process. This method did not work well, as the diffusion model very quickly learned how to fool the pre-trained discriminator. The diffusion model either converges to outputting one single image that fools the discriminator every time, or learns an image manifold that does not look anything like real images. This is simply because the pre-trained discriminator is not robust enough. For this to work, the discriminator would have to know all kind of pixel-combinations and if they are realistic, which it simply does not. A sample of the images generated can be found in the appendix, Figure 19.

Method 2: updating both the diffusion model and the discriminator based on feedback on the full denoising process. This method also did not work, although it performed better than method one. Since this method requires backtracking the gradient through all the denoising iterations (300 for our model), we have to use a very small batch size to not run into GPU memory issues. A batch size this small is prone to overfitting, which makes it hard for the model to generalize. Also, due to resource intensity, we do not have resources to achieve meaningful results, even if the method would have worked well. This method was attempted with both a pre-trained discriminator and a discriminator that was trained from scratch, with poor results either way. A sample of the images generated can be found in the appendix, Figure 20.

Method 3: Fine-tune diffusion model, train discriminator from scratch, both on noise. This method works to some degree. Instead of having the discriminator guess on final images, we make it guess if something is real noise or noise generated by the diffusion model. Many hyperparameter combinations were tested, with the best results being found with:

- Loss function: bce (binary cross entropy)
- Weight initialization of discriminator: Default by Pytorch
- Learning rate - discriminator: 1e-4
- Learning rate - generator: 1e-5

- batch size: 128
- Optimizer: Adam, bet1 = 0.5 and beta2 = 0.999
- Train only discriminator for one epoch, then both for 4 more epochs.

The optimal hyperparameters were chosen based on trial and error. A sample of images from this method can be seen in the appendix, Figure 21.

Method 3.5: Providing the current denoising time-step to the discriminator. This appears to have produced the best results out of all of the tested methods, and qualitatively appears more clear and sensible than the model without fine-tuning.

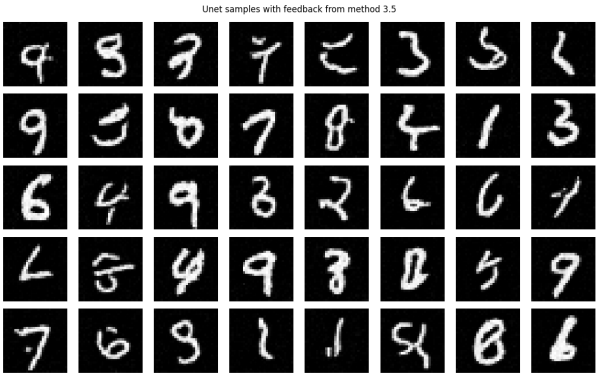

Figure 8: Unconditioned model with feedback from method 3.5.

Method 4: Train on the next generated image in the denoising loop instead of noise This method did not work very effectively. The discriminator did not manage to see any difference between the two and gave them identical scores. This is likely because one single denoising step removes very small amounts of noise (about 1/300 in our case), so the images are very similar. A sample of generated images can be found in the appendix, Figure 22

4.4.2 MNIST - Conditioned

Below is a sample of images produced by the conditioned diffusion model with no fine-tuning from the discriminator.

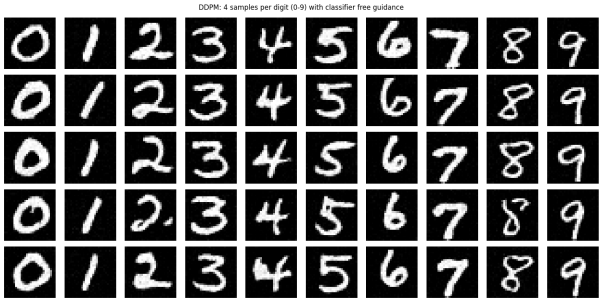

Figure 9: Conditioned model with no feedback from the discriminator.

Since method 3.5 had the best result on the unconditioned model, we apply it to the conditioned model. The results of this are shown below. Unfortunately, though, it appears as though no obvious improvement was made - in fact, the two seem nearly identical. The reason for this is likely that the conditioned diffusion model is already too powerful to be significantly improved by this process.

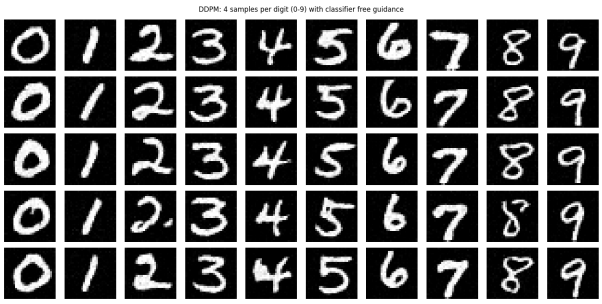

Figure 10: Conditioned model with feedback from method 3.5.

4.4.3 CIFAR-10

Below is a sample of images produced by the diffusion model trained on CIFAR-10 with no fine-tuning from the discriminator.

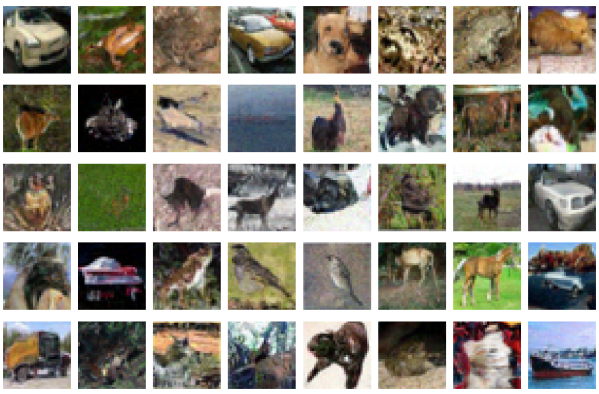

Figure 11: Diffusion model trained on CIFAR-10 with no fine-tuning

Unfortunately, fine-tuning this model appears to have had negative effects on the image quality. Applying method 3.5 to this model has made the images qualitatively more noisy, less distinct and less sensible. It is possible that this was due to a poor choice of hyperparameters - due to time constraints, only the hyperparameters mentioned earlier could be tested. A model trained on a more complex dataset (such as CIFAR-10 with its three colour channels and many class labels) would likely be more sensitive to hyperparameters, so this could warrant further exploration.

We also experimented with the approach proposed by Kim et al. (2023), which applies discriminator guidance during sampling without updating either the diffusion model or the discriminator during training. This method is appealing due to its stability and simplicity. However, it relies heavily on a well-trained discriminator, which we lacked. As a result, the guidance signal was too weak or inconsistent, and the generated images failed to resemble meaningful outputs.

Figure 12: Diffusion model trained on CIFAR-10 with feedback from method 3.5

4.4.4 Best model comparisons

The best models were found to be:

- Unconditioned - Method 3.5: Providing the current denoising time-step to the discriminator.
- Conditioned - Method 3.5
- CIFAR-10 - Method 3.5

Their metrics are provided below. Due to technical constraints with the size of the CIFAR-10 models, some scores could not be computed.

Score \ Method	Unconditioned without fine-tuning	Unconditioned best method	Conditioned without fine-tuning	Conditioned best method	CIFAR-10 without fine-tuning	CIFAR-10 best method
IS	2.14 ± 0.16	2.10 ± 0.12	1.99 ± 0.13	1.99 ± 0.13	X	4.52 ± 0.57
FID	54.91	45.86	84.74	84.74	X	103.32
KID	0.0348	0.0236	0.0719	0.0718	X	X

Table 1: Comparison of best methods using IS, FID and KID scores.

The results more or less align with what was qualitatively observed. The unconditioned model was the most improved by fine-tuning, with a substantially lower FID and KID score indicating a higher realism, which aligns with what was seen in the images. The conditioned model also showed almost no change in metrics between the baseline model and the best method, as expected.

Oddly, however, the conditioned models have much higher FID and KID scores than the unconditioned model, even though the conditioned model appeared to produce much clearer images. This is likely because the ImageNet dataset does not contain classes for handwritten digits. Recall that Inception, the model used in calculating these metrics, was trained on the ImageNet dataset. It is possible that, as the generated images resemble their corresponding MNIST classes more, they contain features less relevant to the ImageNet dataset. As a result, Inception may not be extracting relevant enough features for a true comparison between the real and generated images, resulting in higher "distances" (in the distribution sense) between the two datasets. This could be another symptom of the imperfect metrics we use; perhaps a model similar to inception but trained to extract features from the MNIST dataset would provide results closer to what is expected.

Due to the technical constraints in obtaining the scores for the CIFAR-10 models, a quantitative analysis is difficult. However, we can see that the CIFAR-10 model had a substantially

higher IS than the other models. Recall that one of the assumptions of the IS method was that each image should clearly belong to one (ImageNet) class. Whilst this assumption likely still does not hold true for CIFAR-10, it is probable that the images in CIFAR-10 more closely resemble the images from ImageNet, being that they are both datasets of real world objects. This, along with the fact that the coloured images of CIFAR-10 allow for more diversity in images, would explain the much higher IS compared to the models trained on the MNIST dataset. However, this does not mean that the CIFAR-10 model performed well at all - qualitatively, the images were substantially worse after fine-tuning, as discussed earlier.

5 Conclusion

5.1 Limitations

We were limited by GPU memory and time, which means that we were only able to train our models for a few epochs. With more computing power, we expect that these techniques could be used to produce higher quality models than the ones we managed. This is especially true in the case of our model for the CIFAR-10 dataset, where due to time constraints we were unable to test a large variety of hyperparameters. This means we are unable to conclusively say whether our method could work well in extending a diffusion model for CIFAR-10.

The conditional diffusion model was, in all likelihood, too powerful to be meaningfully improved by our techniques, at least with our computing power. It is possible that, with more computing resources, these methods could still be used to improve slightly upon a conditional diffusion model for the MNIST dataset.

Our evaluation metrics are imperfect and could be improved in a multitude of ways. An Inception-like classifier trained to classify MNIST images would be a much more suitable model to use for the metrics used to evaluate the models. Due to time constraints, we were unable to implement this. A more powerful computing setup that allows for larger sample batches would also improve the quality of FID as a metric, as its bias would become less apparent with a higher sample size.

5.2 Final Thoughts

We present a novel approach for generating images, by combining the architectures of diffusion and generative adversarial networks. We experimented with three main types of models - an unconditioned diffusion model, trained on MNIST, a conditioned diffusion model, also trained on MNIST, and an unconditioned model, trained on CIFAR-10. Multiple methods were tested, with the best method found to be fine-tuning the diffusion model and training a discriminator from scratch alongside it, both on the intermediate noise from each diffusion step. In addition, providing the time-step to the discriminator seemed to have a positive effect on the results. The technique showed promising results in the difficult realm of the unconditioned models. However, the conditioned model was already quite robust and difficult to improve upon. Furthermroe, the diffusion model trained on CIFAR-10 appeared to have been made worse by our methods, though it is unclear if, given enough experimentation with hyperparameters, these techniques could still be used to improve the model.

Contribution

- Mats: Design and implementation of the various methods tested.
- Lauren: Further research into metrics; Introduction, Conclusion, Metrics and Results sections of report.
- Malaika: Reviewed the literature and wrote Related Work, Methods, Dataset and Algorithm sections of report.
- Xuran: Initial implementation of metrics; initial research into metrics. Retrieved results of the various methods.
- Ganesh: Experimentation with various hyperparameters.

References

Betker, J. et al. (2023). *Improving Image Generation With Better Captions*. https://cdn.openai.com/papers/dall-e-3.pdf.

Betzalel, E. et al. (2022). *A Study on the Evaluation of Generative Models*. arXiv: 2206.10935 [cs.LG]. URL: https://arxiv.org/abs/2206.10935.

Ho, J., A. Jain, and P. Abbeel (2020). *Denoising Diffusion Probabilistic Models*. arXiv: 2006.11239 [cs.LG]. URL: https://arxiv.org/abs/2006.11239.

Inkawhich, N. (2018). *DCGAN Tutorial*. URL: https://docs.pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html.

Kim, D. et al. (2023). *Refining Generative Process with Discriminator Guidance in Score-based Diffusion Models*. arXiv: 2211.17091 [cs.CV]. URL: https://arxiv.org/abs/2211.17091.

Kim, T. et al. (2024). "Diffusion-Driven GAN Inversion for Multi-Modal Face Image Generation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lee, C.-H. et al. (2019). *MaskGAN: Towards Diverse and Interactive Facial Image Manipulation*. arXiv: 1907.11922 [cs.CV]. URL: https://arxiv.org/abs/1907.11922.

OpenAI et al. (2024). *GPT-4o System Card*. arXiv: 2410.21276 [cs.CL]. URL: https://arxiv.org/abs/2410.21276.

Radford, A., L. Met, and S. Chintala (2016). *Denoising Diffusion Probabilistic Models*. arXiv: 1511.06434 [cs.LG]. URL: https://arxiv.org/pdf/1511.06434.

Szegedy, C. et al. (2015). *Rethinking the Inception Architecture for Computer Vision*. arXiv: 1512.00567 [cs.CV]. URL: https://arxiv.org/abs/1512.00567.

Appendix

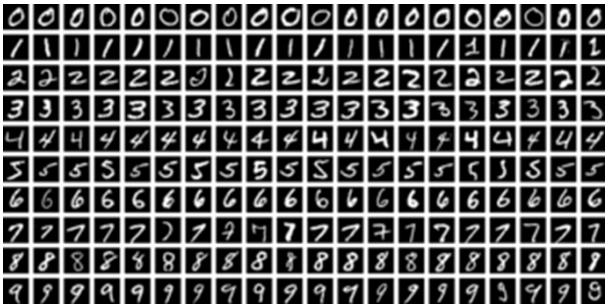

Figure 13: Sample of MNIST images

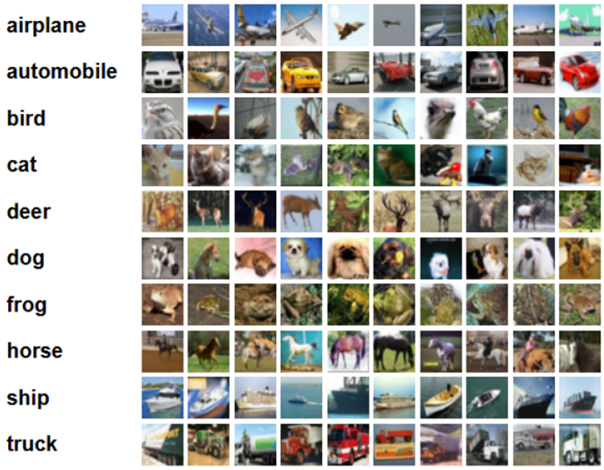

Figure 14: Sample of CIFAR-10 images

```python
class Discriminator(nn.Module):
    def __init__(self, ngpu, nc=1, ndf=64):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 64 x 64
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 32 x 32
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*4) x 8 x 8
            nn.Conv2d(ndf * 4, 1, 4, 2, 1, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        if input.is_cuda and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)
        return output.view(-1, 1).squeeze(1)
```

Figure 15: Python code for DCGAN discriminator implementation

```python
def calculate_inception_score(probs, splits=10):
    scores = []
    n = len(probs)

    #split it up for stability (outliers are less impactful)
    for i in range(splits):
        part = probs[i * n // splits: (i + 1) * n // splits]
        py = np.mean(part, axis=0)
        kl_div = part * (np.log(part + 1e-10) - np.log(py + 1e-10))
        scores.append(np.exp(np.mean(np.sum(kl_div, axis=1))))
    return np.mean(scores), np.std(scores)

# Usage
# probs = get_softmax_preds(gen_images, inception_model)
# mean_is, std_is = calculate_inception_score(probs)
```

Figure 16: Python code for Inception Score implementation.

```python
# FID
def calculate_fid(mu1, sigma1, mu2, sigma2):
    diff = mu1 - mu2
    covmean = sqrtm(sigma1 @ sigma2)
    if np.iscomplexobj(covmean):
        covmean = covmean.real
    return diff.dot(diff) + np.trace(sigma1 + sigma2 - 2 * covmean)

# Usage
# features_real = get_inception_features(real_images, inception_model)
# features_gen = get_inception_features(gen_images, inception_model)

# mu1, sigma1 = features_real.mean(axis=0), np.cov(features_real, rowvar=False)
# mu2, sigma2 = features_gen.mean(axis=0), np.cov(features_gen, rowvar=False)

# fid = calculate_fid(mu1, sigma1, mu2, sigma2)
```

Figure 17: Python code for Fréchet Inception Distance implementation.

```python
# KID
def calculate_kid(real_features, gen_features):
    k_rr = polynomial_kernel(real_features, real_features, degree=3, gamma=None, coef0=1)
    k_gg = polynomial_kernel(gen_features, gen_features, degree=3, gamma=None, coef0=1)
    k_rg = polynomial_kernel(real_features, gen_features, degree=3, gamma=None, coef0=1)

    m, n = len(real_features), len(gen_features)
    return (np.sum(k_rr) - np.trace(k_rr)) / (m * (m - 1)) + \
           (np.sum(k_gg) - np.trace(k_gg)) / (n * (n - 1)) - \
           2 * np.sum(k_rg) / (m * n)

# Usage
# features_real = get_inception_features(real_images, inception_model)
# features_gen = get_inception_features(gen_images, inception_model)

# kid = calculate_kid(features_real, features_gen)
```

Figure 18: Python code for Kernel Inception Distance (KID) implementation.

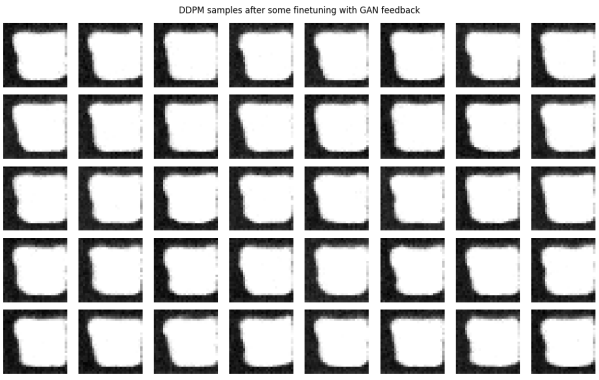

Figure 19: Unconditioned model with feedback from method 1.

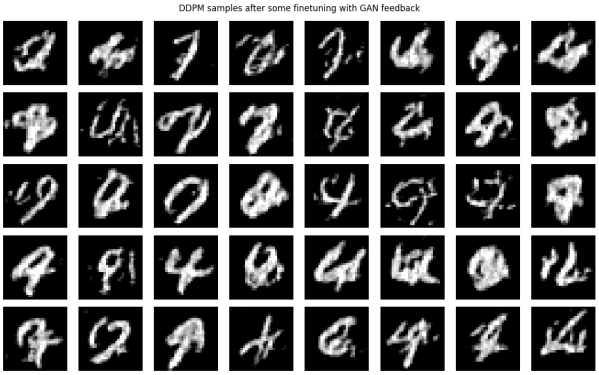

Figure 20: Unconditioned model with feedback from method 2.

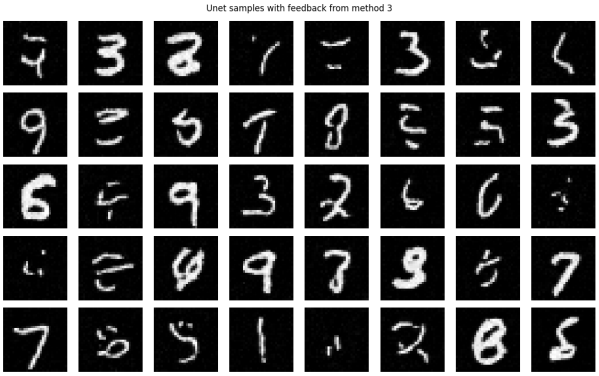

Figure 21: Unconditioned model with feedback from method 3.

Figure 22: Unconditioned model with feedback from method 4.

We give consent for this to be used as a teaching resource.