SAiDL REPORT

Kamal Kumar Manchenella

April 2025

Contents

1 Overview

This report covers the implementation and analysis of two major components:

- **Core Machine Learning (Core ML):** This part includes data preprocessing, selection of model architecture, training procedures, and performance evaluation using standard metrics.
- **Diffusion Models:** This section details the use of diffusion processes for generative modeling. It includes a breakdown of how diffusion works and many methods to calculate how well diffusion is doing.

The report will answer key questions and draw relevant conclusions from graphs and other implemented code outputs.

2 Core Machine Learning

2.1 Objective

The goal of the Core ML component is to explore the robustness of various loss functions on a noisy classification task. Specifically, we train a CNN on a noisy version of CIFAR-10 and compare performance across several custom-designed loss functions.

2.2 Loss Functions

Five loss functions were evaluated:

Evaluated Loss Functions

Five loss functions were evaluated:

- **Cross Entropy (CE)** – The standard baseline loss for classification:

$$CE = -\sum_{k=1}^{K} q(k|x) \log p(k|x)$$

- **Focal Loss (FL)** – Down-weights easy examples to focus learning on hard, misclassified ones:

$$FL = -\sum_{k=1}^{K} q(k|x)(1 - p(k|x))^{\gamma} \log p(k|x), \quad \text{where } \gamma \geq 0$$

- **Normalized Cross Entropy (NCE)** – Normalizes loss using the entropy of the predictions, improving robustness to label noise:

$$\text{NCE} = \frac{-\sum_{k=1}^{K} q(k|x) \log p(k|x)}{-\sum_{j=1}^{K} \sum_{k=1}^{K} q(y = j|x) \log p(k|x)} = \log \frac{\prod_{k=1}^{K} p(k|x)}{p(y|x)}$$

- **Normalized Focal Loss (NFL)** – Combines the benefits of FL and NCE for noise-robust training:

$$\text{NFL} = \frac{-\sum_{k=1}^{K} q(k|x)(1 - p(k|x))^{\gamma} \log p(k|x)}{-\sum_{j=1}^{K} \sum_{k=1}^{K} q(y = j|x)(1 - p(k|x))^{\gamma} \log p(k|x)} = \log \prod_{k=1}^{K} (1 - p(k|x))^{\gamma} p(k|x) \cdot (1 - p(y|x))^{\gamma} p$$

- **Active Passive Loss (APL)** – A linear combination of CE (active) and Reverse Cross Entropy (passive) that balances fitting and noise tolerance:

$$\mathcal{L}_{\text{APL}} = \alpha \cdot \mathcal{L}_{\text{Active}} + \beta \cdot \mathcal{L}_{\text{Passive}}$$

Each loss function was implemented as a custom PyTorch class to enable flexible experimentation.

2.3 Model Architecture

S

Layer	Configuration
Conv2D	3 → 32 channels, 3 × 3 kernel, padding=1
ReLU + MaxPool2D	2 × 2 pool size
Conv2D	32 → 64 channels, 3 × 3 kernel, padding=1
ReLU + MaxPool2D	2 × 2 pool size
Fully Connected	64 × 8 × 8 → 256
ReLU + Dropout	dropout rate = 0.5
Fully Connected	256 → 10

Table 1: CNN Architecture for CIFAR-10 Classification (based on custom model)

2.4 Dataset and Noise Injection

Experiments were conducted using CIFAR-10. To simulate real-world conditions, symmetric label noise was injected at 40%:

- Noise was applied uniformly across all classes.
- A custom data set wrapper randomly replaced labels according to the desired noise rate.

2.5 Training Setup

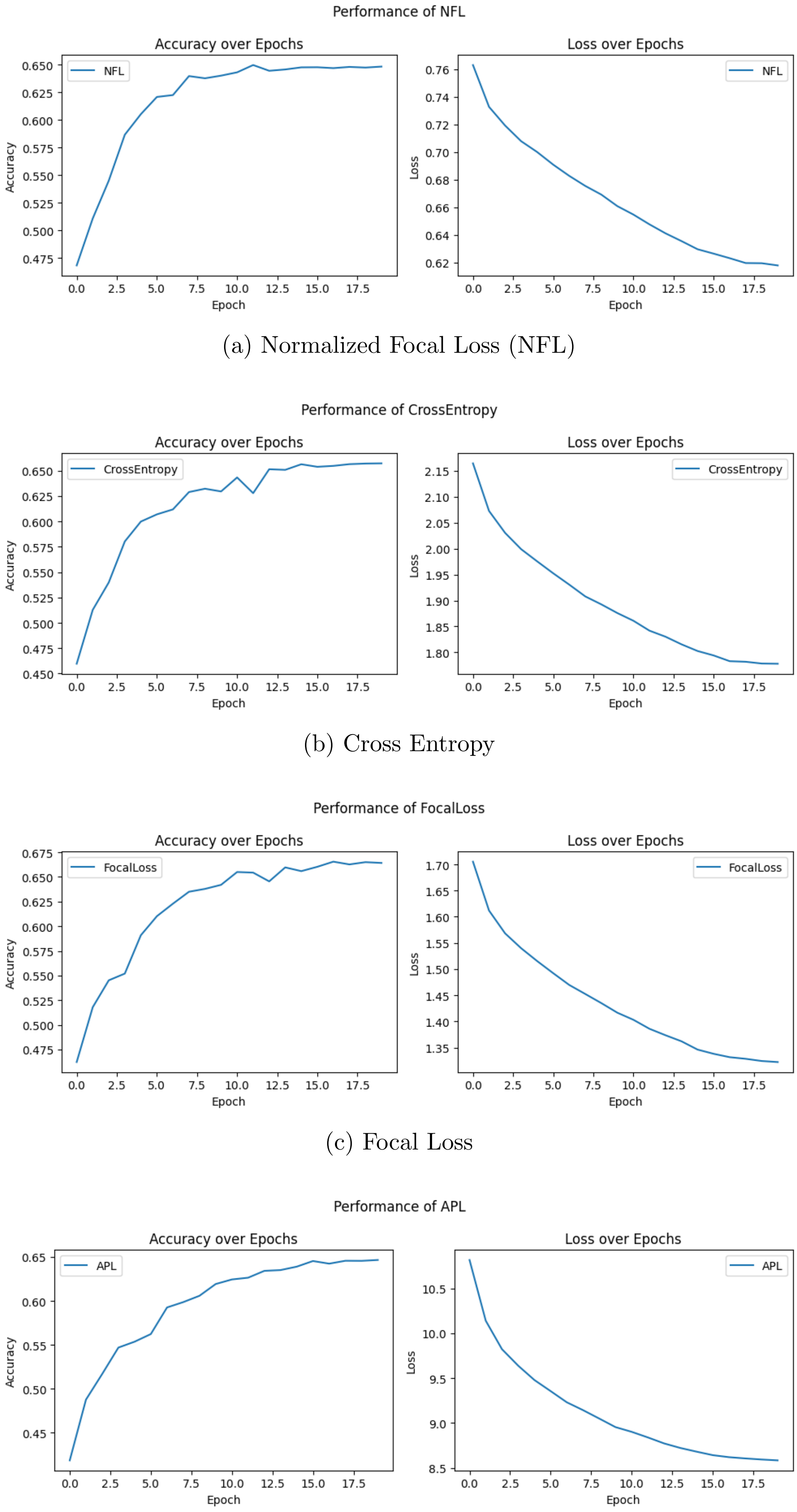
Noise Rate	Alpha (α)	Beta (β)
0.25	0.7	0.3
0.50	0.5	0.5
0.75	0.3	0.7

Table 2: APL Parameters for Different Noise Rates

- Optimizer: Adam, learning rate = 0.0005
- Scheduler: Cosine Annealing
- Epochs: 15
- Batch Size: 128
- Device: CUDA (if available)

2.6 Evaluation Metrics

Models were evaluated using the accuracy of the test set and training loss curves over epochs. The following graphs summarize model performance:



(a) Normalized Focal Loss (NFL)

(b) Cross Entropy

(c) Focal Loss

(d) Active Passive Loss (APL)

Figure 1: Accuracy and Loss over Epochs for Various Loss Functions

2.6.1 Effect of Noise Rates on Loss Function Performance

In this section, we analyze how different loss functions perform under varying label noise conditions. We use noise rates of 0.25, 0.5, and 0.75, and track both training loss and accuracy over epochs.

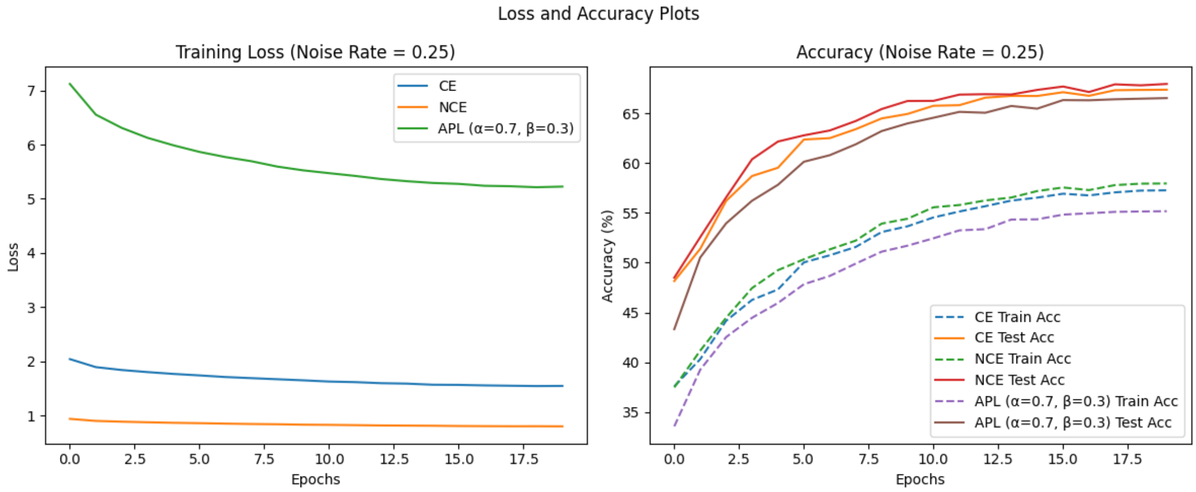


Figure 2: Training and Accuracy Curves at Noise Rate = 0.25

Noise Rate = 0.25 Analysis: At low noise, CE and NCE show strong performance. APL lags in both train and test accuracy likely due to its passive component under emphasizing the clean labels when noise is minimal.

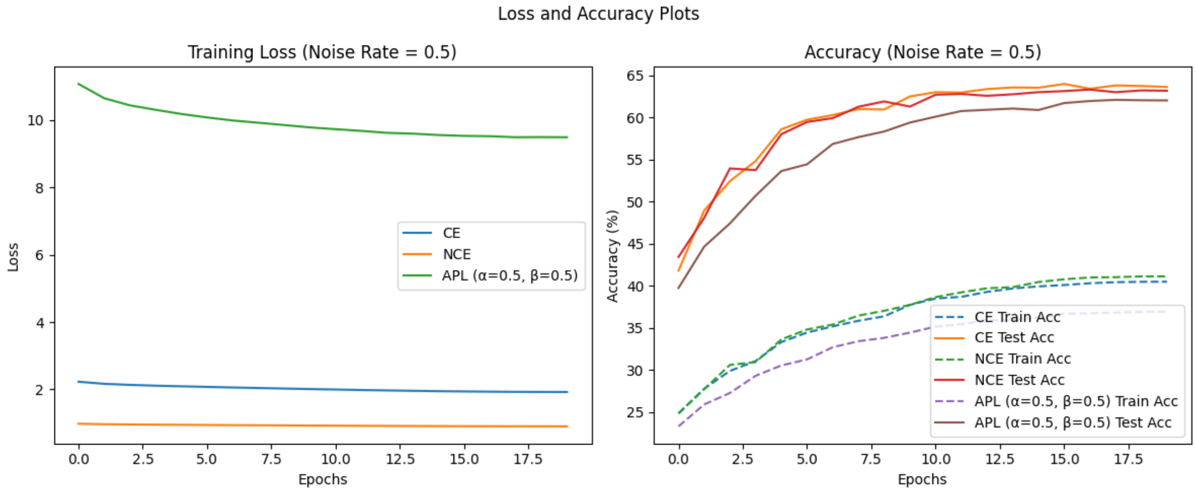


Figure 3: Training and Accuracy Curves at Noise Rate = 0.5

Noise Rate = 0.5 Analysis: NCE maintains robustness, while CE starts to degrade. APL remains competitive in test accuracy, benefiting from the balance between active and passive terms at moderate noise.

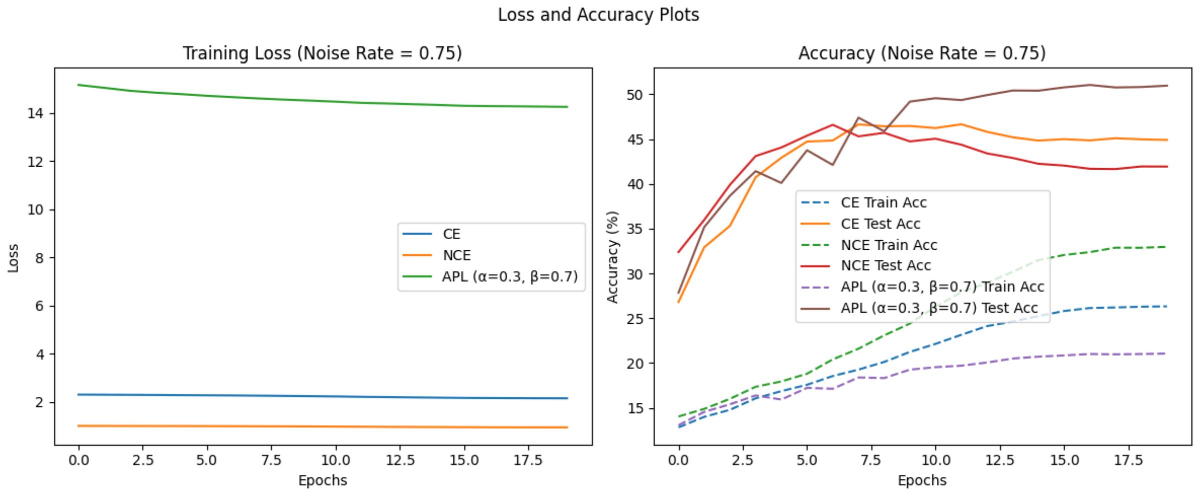


Figure 4: Training and Accuracy Curves at Noise Rate = 0.75

Noise Rate = 0.75 Analysis: At high noise, APL outperforms both CE and NCE in test accuracy. This shows the value of the passive loss in avoiding overfitting to noisy labels.

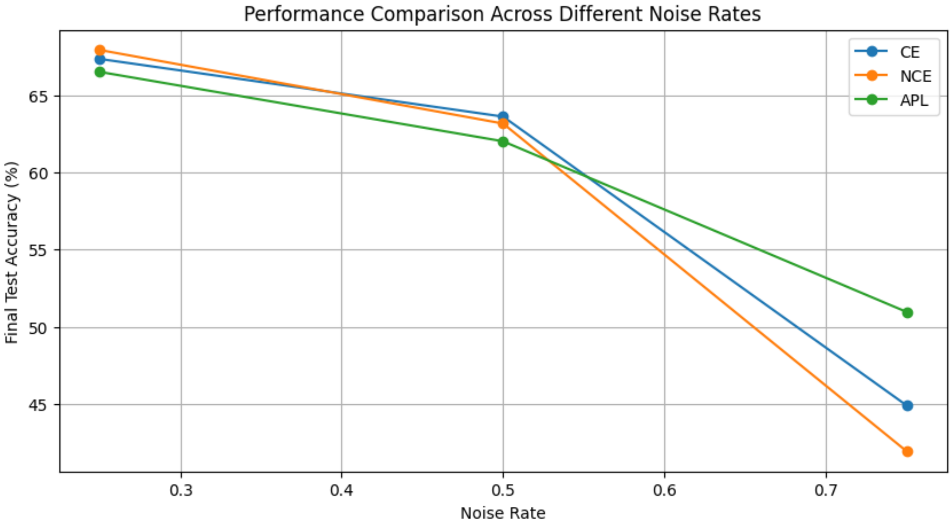


Figure 5: Final Test Accuracy Comparison Across Noise Rates

Final Comparison Across All Noise Levels Analysis: APL performs the worst at low noise but scales better as noise increases. This confirms its design goal: mitigating high label noise impact.

2.7 Conclusion

Custom loss functions significantly influence performance in noisy settings. APL outperformed traditional cross-entropy, confirming its strength in handling label noise. These insights lay the groundwork for using robust loss functions in real-world noisy datasets.

3 Diffusion

3.1 Objective

The goal of this section is to implement a diffusion-based generative model capable of learning the data distribution and synthesizing realistic samples. Diffusion models gradually add noise to the data and then learn to reverse the process, effectively denoising and generating new data.

3.2 Methodology

The implementation follows the Denoising Diffusion Probabilistic Models (DDPM) framework. The key components include:

- A forward diffusion process that adds Gaussian noise to an image across a fixed number of timesteps.
- A learned reverse process using a neural network (typically a U-Net) to predict and remove noise at each timestep.
- Training the model to minimize the mean squared error (MSE) between the true noise and the predicted noise.

3.2.1 1. Noise Addition

- The backbone architecture is based on a simplified U-Net and Gaussian noise is incrementally added over 1000 timesteps.
- The model is trained using the MSE loss between the predicted and actual noise vectors.
- Training was performed on the CIFAR-10 dataset, resized and normalized for diffusion.A cosine schedule was used for β_t noise variance.

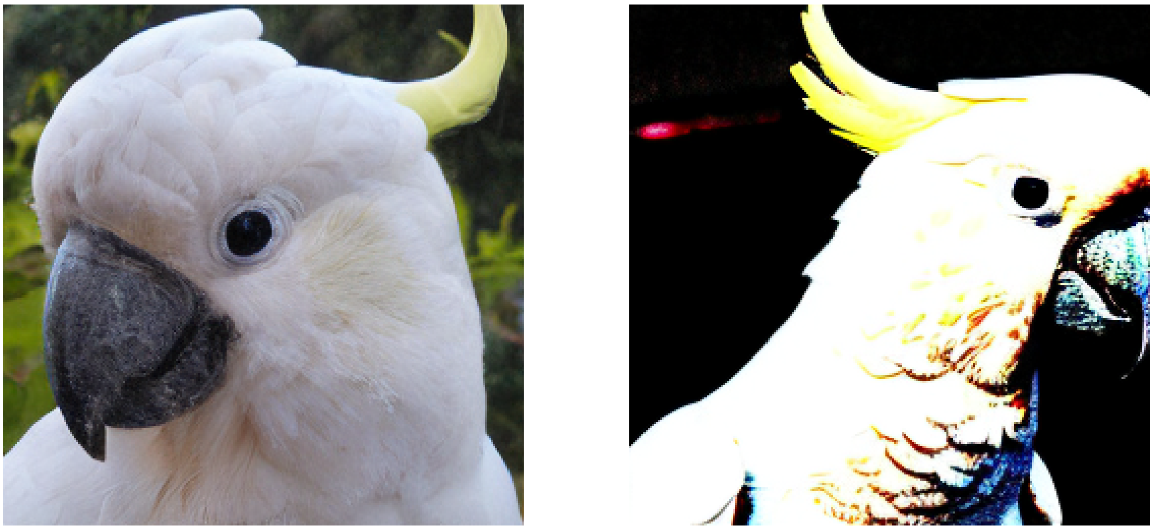
3.2.2 2. Reverse-Diffusion

Once trained, the model performs reverse diffusion:

- Starting from random Gaussian noise, the model iteratively denoises the image one timestep at a time.
- The final output resembles the training data distribution (CIFAR-10 in this case).

3.3 Observations:

3.3.1 Part 1:



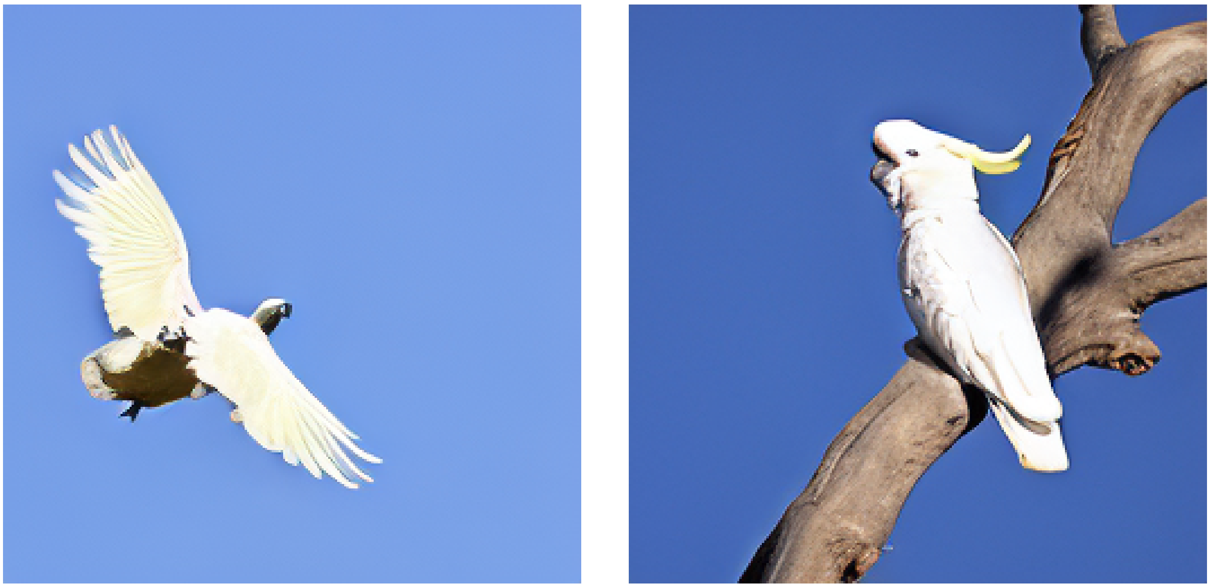
(a) Low CFG – Diverse, underfitting-like outputs

(b) High CFG – Sharp, overfitting-like outputs

Figure 6: Effect of CFG (Classifier-Free Guidance) on sample generation

At low values of CFG: Weaker guidance, meaning the model doesn't strongly enforce the class label. Images may be more diverse, creative, or even a bit off-label, often resembling underfitting-like results.

At high values of CFG: Very strong guidance — the model strictly follows the class conditioning, which can lead to overfitting-like results with less diversity in generations.



(a) Low Sampling Steps – Fast but blurry output

(b) High Sampling Steps – Crisp but slow output

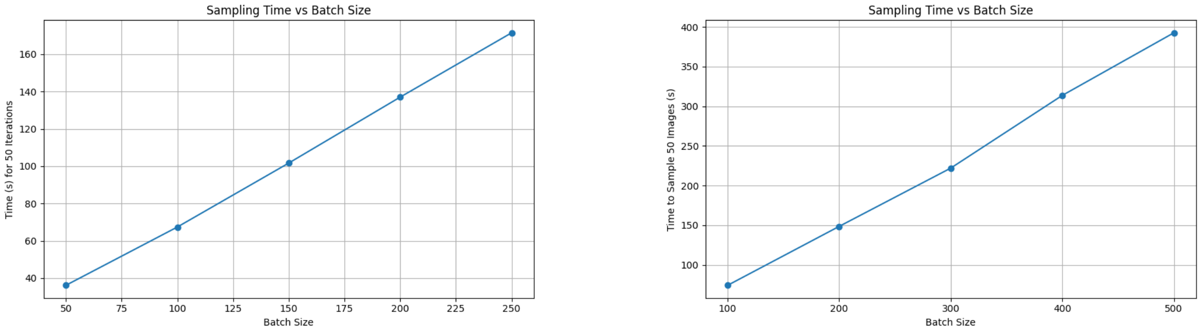
Figure 7: Effect of Sampling Steps on Diffusion Model Output

At Low Sample Steps: Image appears blurry due to noise not being removed entirely because of the low number of denoising steps. Output is fast.

At High Sample Steps: Image appears very detailed and crisp due to noise being completely removed. Output is very slow and there are diminishing returns past a point.

3.3.2 Part 2:

Difference between DiT Attention block and xformers Attention block



(a) Xformers Output

(b) DiT Output

Figure 8: Comparison between outputs of the Xformers-based DiT and standard DiT architecture.

The figure above compares the visual outputs from two transformer blocks. On the left, the image generated using the Xformers-optimized attention mechanism demonstrates faster sampling and similar quality. On the right, the standard DiT output is shown. While both maintain structure and content, the Xformers version often achieves comparable fidelity with lower computational cost, making it more efficient for large-scale diffusion tasks.

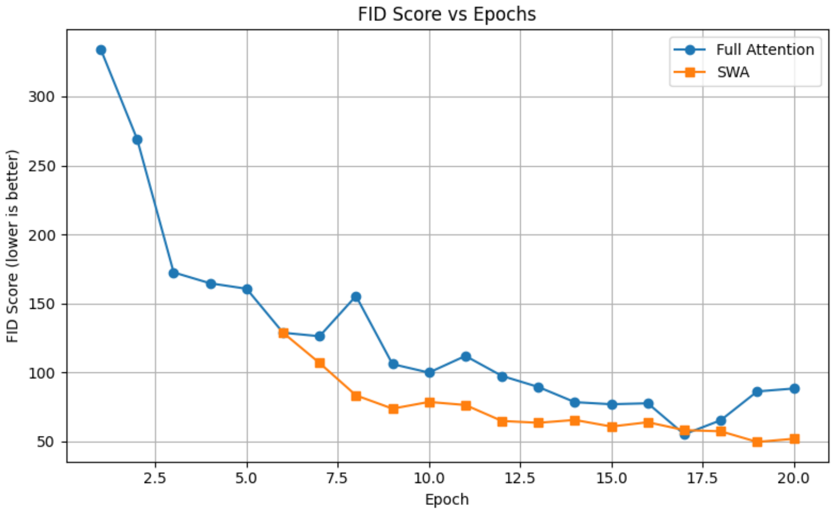


Figure 9: FID score comparison between DiT variants.

For any two probability distributions P and Q over R^d having finite first and second moments, the Fréchet distance is defined by:

$$\text{dist}_F^2(P, Q) := \inf_{\gamma \in \Gamma(P,Q)} E_{(x,y)\sim\gamma} \|x - y\|^2 , \tag{1}$$

Stochastic Weight Averaging (SWA) typically achieves a lower FID score than using the final model from full loss training because it averages weights from multiple points in the training trajectory, leading to a flatter and more generalizable solution. This results in improved sample quality and diversity, which is directly reflected in a better (lower) FID score.

SWA starts after a few epochs (after 6 in above graph) to avoid averaging unstable early weights. Early training involves noisy updates, which can harm convergence. Delaying SWA ensures only well-optimized weights contribute to the average, improving stability and generalization.

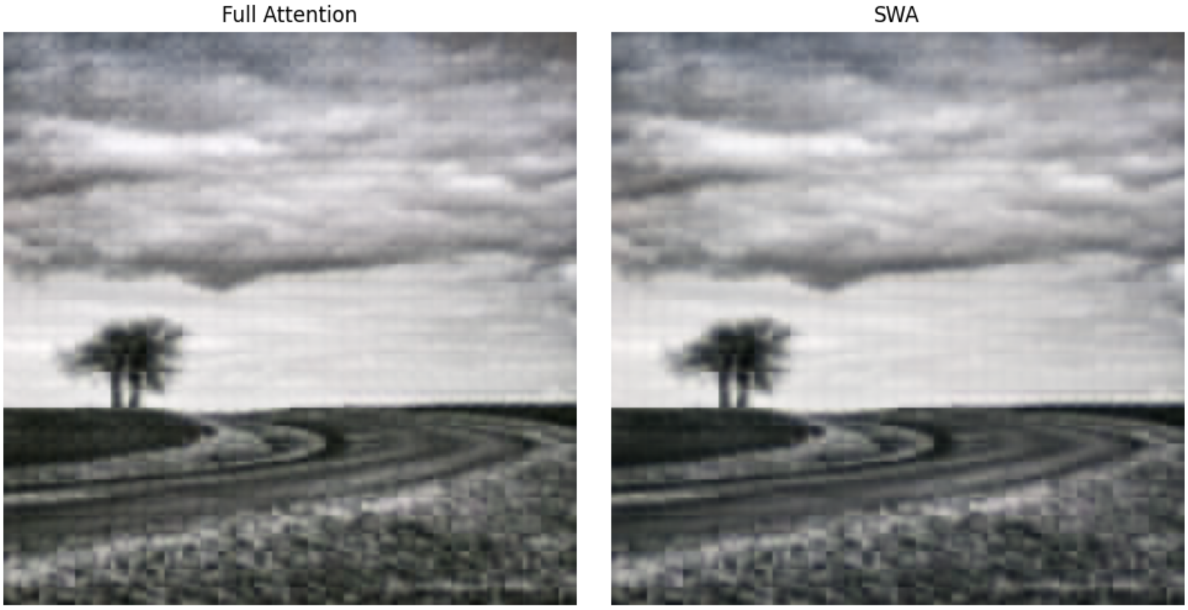


Figure 10: FID (Full) = 156.92, FID (SWA) = 113.05.

SWA having a lesser FID appears less pixilated and grainy.

3.3.3 Part 3 (i):

For two probability distributions P and Q over R^d, the MMD distance with respect to a positive definite kernel k is defined by:

$$\text{dist}_{\text{MMD}}^2(P, Q) := E_{\mathbf{x},\mathbf{x}'}[k(\mathbf{x}, \mathbf{x}')] + E_{\mathbf{y},\mathbf{y}'}[k(\mathbf{y}, \mathbf{y}')] - 2E_{\mathbf{x},\mathbf{y}}[k(\mathbf{x}, \mathbf{y})], \tag{2}$$

1. *Does CMMD correlate with better image generation?*

Yes, CMMD correlated well with better image generation. Lower CMMD scores indicated that generated images better matched the distribution of real images conditioned on labels, supporting both fidelity and conditional consistency.

2. *What assumptions are made for FID calculation?*

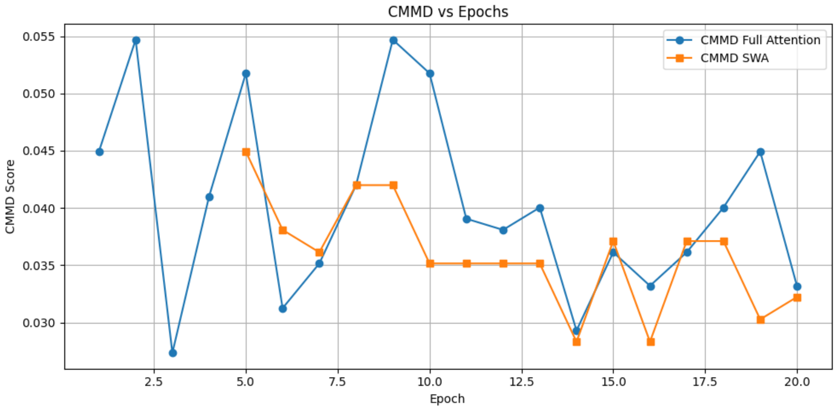


Figure 11: CMMD Score Comparison between DiT Variants

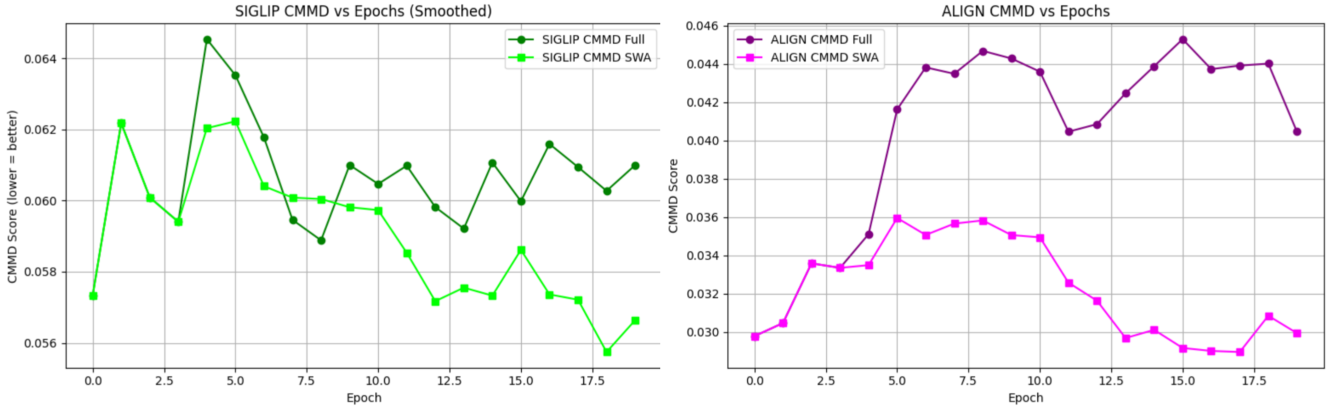
- FID assumes image features (from InceptionV3) follow a **multivariate Gaussian distribution**.
- It compares only the **marginal distributions** of real and generated features — *ignoring the conditioning* (e.g., class labels).

This makes FID less reliable than CMMD

3. How does CMMD improve on those?

CMMD does not assume Gaussianity and measures discrepancy between conditional distributions using kernel methods. It accounts for how well generated samples match real samples given the same condition (e.g., label), making it more appropriate for conditional generative models like DiT.

3.3.4 Part 3 (ii):



(a) SigLIP MMD: SWA vs Full

(b) ALIGN MMD: SWA vs Full

Figure 12: Comparison of MMD scores using SigLIP and ALIGN features for both SWA and full loss DiT models. Lower scores indicate better alignment with the real data distribution.

3.4 Conclusion

Upon compared SigLIP ALIGN CLIP and FID

- **Use** `ALIGN + SWA` if your goal is to maximize ***image realism and quality*** (FID).
- **Choose** `SIGLIP + SWA` if your goal is to improve how well the images match the captions, even if the image quality is a bit lower.
- **SWA consistently improves** both **CMMD** and **FID** across models and should be used when possible.

ALIGN SWA had the lowest CMMD score