INFO8010: Brain Cancer Detection Project Report

Ayoub Assaoud[1]

[1] *ayoub.assaoud@student.uliege.be (s207227)*

I. ABSTRACT

II. INTRODUCTION

For this project we propose to build a generative deep learning system that can generate anime faces. The system will be trained on a dataset of anime faces and will use denoising diffusion probabilistic models (DDPM) to generate new faces. The goal is to create a system that can generate acceptable quality anime faces that are diverse and realistic by drawing samples, and naturally for denoising by encoding-decoding the image. The project will also explore the use of different architectures and the impact of self-attention on standard DDPM to improve the quality of the generated faces.

Through this project, we:

- Collecte and preprocess over than 80,000 anime face images, applying normalization, resizing, and few augmentations.
- Design and train two distinct deep learning models, mainly noise prediction technique and have a quick view on score-based energy model, experimenting with different architectures and training strategies.
- Evaluate performance using some basic metrics.

In the following sections, we'll walk through related work, our data pipeline, model Architecture, and results.

III. RELATED WORK

A. Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models (DDPM), introduced by Ho et al. [1], laid the foundation for a class of generative models that iteratively reverse a diffusion process to synthesize high-quality data samples. The original DDPM framework employs a convolutional U-Net architecture to model the denoising function, demonstrating impressive results in image generation without relying on attention mechanisms.

Subsequent works have extended this paradigm by integrating transformers to better capture long-range dependencies and enable multimodal conditioning, particularly in text-to-image synthesis tasks. Models such as GLIDE [2], Imagen [3], and DALL·E 2 [4] exemplify this shift, incorporating transformer-based components to enhance expressiveness and scalability.

Our work builds on this trajectory by exploring the role of transformers within the DDPM framework, aiming to enhance diffusion modeling with the representational power of attention-based architectures, as well as discussing some limitations and choices.

B. Modified Forward Diffusion Kernel

Recent efforts have explored modifications to the forward diffusion kernel in DDPMs to improve sampling efficiency and image quality. The LearnOpenCV guide [5] discusses a variant where the noise schedule and kernel are adapted to better preserve semantic structure during the forward process. By refining the way noise is injected, this approach aims to maintain more meaningful latent representations, which in turn facilitates more accurate denoising during reverse diffusion.

Such kernel-based modifications offer a promising direction for enhancing DDPMs without fundamentally altering their probabilistic framework. Our work complements these ideas by investigating how transformer-based architectures can further improve the expressiveness and flexibility of the denoising model.

IV. METHODS

A. Data

FIG. 1. Images from the anime faces dataset.

1. *Dataset Description*

we took the **Anime Face Dataset** ($63k$ images), in addition to **soumikrakshit Anime Face** ($21k$ images).

2. Data Preprocessing

Our preprocessing pipeline was implemented in Python using PyTorch library, and included the following steps:

- **Cleaning**: To insure a better quality of dataset, we removed all images for which the size is less than 64x64, resulting in 80k remaining images.
- **Resizing:** All images were resized to a fixed size of 64x64 pixels.
- **Normalization:** We normalize the pixel values to the range [-1, 1], to keep consistent with the noise distribution. an it seems to be much more effective than constraining the model to output values in the [0, 255] range, as it allows for better training stability and convergence.
- **Data Augmentation:** We applied random horizontal flipping, no rotation or cropping was applied because most faces were not centered, or touching the frame which could lead to poor generated samples.

3. Spliting Data:

We split the dataset into training set and validation set, with a ratio of 90:10, althought this splitting is not strictly necessary for DDPMs, and not that significant than it could be for classification tasks, we still did it to monitor the training process and avoid overfitting, to have at least an idea about the model's denoising performance on unseen data.

B. Architecture

We use a denoising diffusion probabilistic model (DDPM) as the backbone of our generative system. The DDPM consists of a forward diffusion process that gradually adds gaussian noise to the data, and a reverse diffusion process that learns to denoise the data. This architecture allows for high-quality image generation by sampling from the a known distribution $q(x_T) \sim \mathcal{N}(0, I)$ then gradually denoising at each step with our model.

the same architecture can be used to approximate the score function of the data distribution $s_\theta(x, t) \approx \nabla \log p_t(x_t)$.

1. Description of the general architecture for the denoiser model

- **U-Net:** A U-Net architecture is used as the denoiser model that predicts the original noise $\epsilon \approx \epsilon_\theta(x_t, t)$ for the DDPM.
- **ResNet Blocks:** We incorporate residual blocks to improve the flow of gradients during training, which helps in learning complex features, thanks to skip connections.
- **Attention Mechanisms:** We incorporate attention mechanisms to improve the model's ability to focus on relevant features during the denoising process.

The Unet architecture consists of Three main components: a chain of $len(down_channels)$ **DownEncoder**, a chain of $len(mid_channels)$ **Bottleneck**, and a chain of $len(down_channels)$ **UpDecoder**, to preserve symmetric feature mapping.

The **DownEncoder** reduces the spatial dimensions (H,W) of the input image while increasing the number of feature channels, which is called downs ampling, allowing the model to learn hierarchical representations.

The **Bottleneck** connects the DownEncoder and UpDecoder, serving as a bridge that captures the most abstract features.

The **UpDecoder** then reconstructs the image by progressively increasing the spatial dimensions while reducing the number of feature channels (up sampling) and concatenates the correspondding levels DownEncoder outputs with those of the UpDecoder.

Each of these subcomponents is composed of a number of $ModelParams.num\_(down\|mid\|up)\_layers$ **block layers**, each of these block layers consists of a stack of ResNet block followed by a self-attention layer (optional in our case see discussion) and finally an optional convolutive layer (downsampling or upsampling), see figure bellow.

The ResNet block consists of two convolutional layers with Group normalization and SiLU activation, allowing the model to learn residual mappings, and the input is added to a **positional embedding learnable vector** after the first convolutional layer, whereas the self-attention layer allows the model to focus on relevant features in the input image, improving the quality of the generated images.

The DownEncoder uses strided convolutions to reduce the spatial dimensions, while the UpDecoder uses transposed convolutions to increase them. The Bottleneck uses standard convolutions without striding.

2. Motivating choices

the choice of activation function: we chose SiLU (Sigmoid Linear Unit) as the activation function for the model, which is known to improve the flow of gradients and enhance the model's ability to learn complex features. SiLU has been shown to outperform ReLU in many cases, especially in image-tasks. in other point of view, image pixels are normalized to the range [-1, 1], and the SiLU activation function is well-suited for this range, as it smoothly transitions between negative and positive values. $SiLU(x) = x \cdot \sigma(x)$

the choice of normalization: we used Group Normalization instead of Batch Normalization, to normalize the features within groups independently from batches, which is more effective in stabilizing training, especially that in our cases

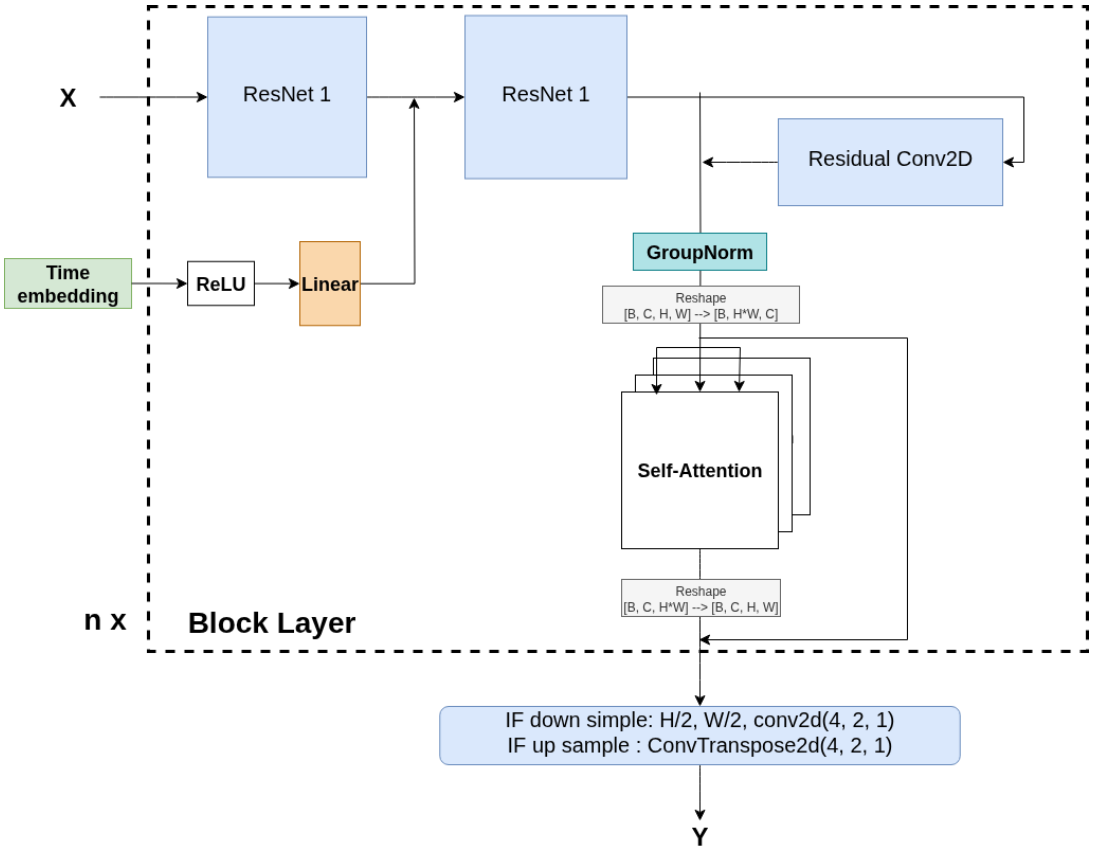

FIG. 2. Block layer Architecture with ResNet blocks and self-attention layers, depending on the subcomponent, the number n is equal to $ModelParams.num\_(down\|mid\|up)\_layers$

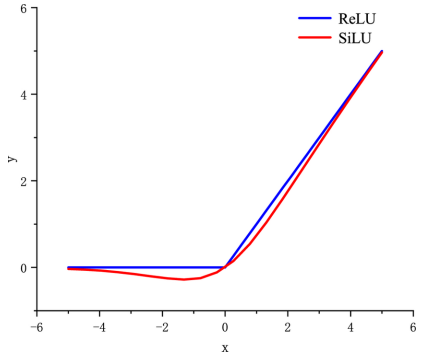

FIG. 3. SiLU and ReLU Activation Functions, from www.researchgate.net

batch size was set to 6, which is small, because we hit the GPU's Limit, it is also good for variable batch size while evaluation.

the choice of attention mechanism: as modern architectures often incorporate attention mechanisms, we used self-attention layers to helps the model to capture long-range dependencies and relationships between pixels, which is crucial for generating high-quality images.

the shape of the input of self-attention is : $[B, H * W, C_i]$ respectively batch, height, weight and out_channels of a given subcomponent. Which makes relationships between pixels more explicit than $[B, C_i, H * W]$, we explored the impact for this decision especially for Mega 74M model in the sections bellow.

this mechanism has an impact, as we will see in the comming sections, that there is a trade-off between an attention block for each block layer and the model's performance. we will get a tour over the architecture we have tried in the order of their complexity.

3. *Recapitulative Table:*

It is important to mentionne that at the beginng, we took $[B, C_i, H * W]$ as the input shape for attention, and the stratigy to enhance performance was to increase the capacity by increasing stages of DownEncoder and UpDecoder, as well as the number of block layers in each subcomponent, but it was not effective, thus we changed the input shape to $[B, H * W, C_i]$ and we tried to increase the capacity by increasing the number of channels in DownEncoder and UpDecoder, as well as the number of block layers in each subcomponent.

NB: $down\_channel = up\_channels$

Model Name	Params (M)	*Down Channels*	Down Layers	*Mid Channels*	Mid Layers	Attention	Time Emb Size
mini	10	[32,64,128,256]	2	[256,256,128]	2	full	128
Simple	49	[32,64,128,256,512]	3	[512,512,256]	2	full	128
Mega	74	[32,64,128,256,512]	5	[512,512,256]	3	full	128
Giga	85	[128,256,256,512]	5	[512,512,256]	5	partial	128

TABLE I. Model architecture parameters for each model.

4. *Mini U-Net DDPM 10M Architecture:*

a. Bad Attention Input Shape: $[B, C_i, H*W]$ we use $[B, C_i, H*W]$ as input shape for attention, it never generate a face, producing some noisy mono-color backgrounds, with a 2 little dark (eyes-like) stains, then it continiousely decreading quality until generating uncomprehensive noise with almost the same grade of colors.

Its best (still poor) performence is reached in first epochs. However, is was remarked that the model is still denoising validation set fairly enough.

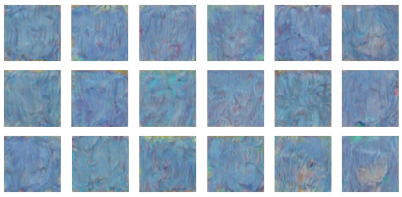

FIG. 4. Samples from Mini U-Net DDPM 10M Architecture at epoch 5 with bad attention.

b. Good Attention Input Shape: $[B, H*W, C_i]$ The idea is to know weither we can reach an acceptable quality of the images, strictely increasing down channels in the Down-Decoder, with attention in each block layer.

the model is not capable of generating coherent sampled images in the first epochs, its best performence is reached at the 15th epoch, but some regions in the latent space x_T are empty or under-represented, such that we obtain some images with the same color, some images that are identical to original data, which indicates that model is overfitting and begins to collapse withour being capable to generalize. but the model is still denoising fairly good as larger models.

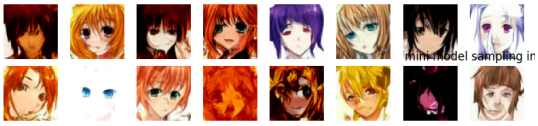

FIG. 5. Samples from Mini U-Net DDPM 10M Architecture at epoch 15.

Having less empty images may indicated that latent variable space is compact, in other words, the model is learning a more structured representation of the data, which can be due to its small capacity, but we aim a better results.

5. *Simple U-Net DDPM 49M Architecture:*

the second try was to augment the capacity of the model by increasing the number of block layers to 3 and an additional layer of DownEncoder (automatically UpDecoder) to reach 512 channels, while preserving the good input shape.

the performence has increased, but the sampling quality is still poor and not diverse, some samples are either white or black backgrounds, and struggles to generalize even after many epochs.

FIG. 6. Samples from Mega U-Net DDPM 74M Architecture at epoch 13.

6. *Mega U-Net DDPM 74M Architecture:*

here we augment the capacity of the model by increasing the number of block layers of DownEncoder and UpDecoder to 5, Bottleneck to 3 layers. Feed attention mechanism with inputs of the shape $[B, C_i, H*W]$, which focuses on the relationships across channels instead of pixel-to-pixel relationships, resulted in bad performance. However, Mega was capable of generating some reasonable samples, but they were sharp, slightly noisy and poorly divergent.

FIG. 7. Samples from Mega U-Net DDPM 74M Architecture, bad input shape, at respectively 6, 8 and 16 epochs.

We notice that epochs between 5 and 9, every epoch has a common samples pattern, such as hair colors, face color, sharpness, the model stick to one pattern, similar faces, thus we beleive that attention mechanism has its role on this issue as well as the limited size of batch (2)[6]. then at later epochs, the model starts to generate pure noise with a fixed mean that corresponds to ebony.

Here it was the first trigger that shifted our approach to think about the architecture choices and specially input shape of attention, as we noticed that the model is not capable of generating diverse samples, and it seems to be stuck in a local minimum, which is a common issue in deep learning models, especially when the model is too complex or the data is not diverse enough.

7. *Giga U-Net DDPM 85M Architecture:*

Here we decided to increase the capacity of the model by upgrading the size of channels to become $[128, 256, 256, 512]$, refer to table IV B 3, you will notice also that channels are increasing non strictly in the DownEncoder and UpDecoder, while keeping the same number of block layers to 5 for both DownEncoder and UpDecoder, and increasing to 5 for Bottleneck.

In addition, we decided that a transformer for each block layer may be not a good idea, as it requires too much data to train and more prone to noise, as well as for computaion limited ressource issues, thus in Giga model, we apply the attention layer as follows:

Layer	**Pos 1**	**Pos 2**	**Pos 3**	**Pos 4**	**Pos 5**
Down	False	True	False	True	True
Mid	False	True	False	-	-
Up	False	True	False	True	False

TABLE II. Attention application across Block layers

Such that the first block layer does not contain an attention layer, the idea is to obtain some meaningful input for the transformer, our best-choice is inspired from GPT, BERT and modern Large languages models that take as input a meaningful semantic vectors from pretrained embedding technologies like word2vec, FastText..., then passes through a transformer-based encoder/decoder block.

Thus we do the same thing for images, but with convolution-oriented layers (such ResNet) that tend to extract meaningful features from images, to feed them to our self-attention layers within the block layer.

Let us take an overview on the samples generated by Giga model in the first epochs.

FIG. 8. Samples from Giga U-Net DDPM 85M Architecture respectively after 1, 3, 7, 10, 17 and 22 epochs.

We can see that the model is capable of generating coherent faces, with diversity, while impresively achieving the same performance for smaller models in even earlier epochs.

It also produces cleaner images even in the earlier epochs in comparison with Mega 74M model, which is not the case with smaller models that contains full transformer.

We believe that this significant improvement is not only due to the capacity of the model given the additional 10M parameters, but also to the change of the architecture, notably by omitting some attention layers in the early stages, as well as allowing for bigger batch sizes (at most 6) which lead to a stability in training, which can be seen in the train and validation loss plots in the next section.

What was also surprising is the fact that the model is capable of generating coherent faces very early, which was not the case with smaller models, and it seems to be more stable during training, as we can see in the train and validation loss plots.

C. Training

Our training technique is based on standard denoising diffusion probabilistic models (DDPM) paper. we trained the model to predict the original noise ϵ given a noisy image x_t and a time step t. The model learns to denoise the image by minimizing the Mean Squared Error (MSE) between the predicted noise and the true noise.

we draw a timestep from a uniform distribution between 0 and T=1000, for β_t linearly increasing from $1e-4$ up to $2e-2$ a noise ϵ from normal distribution and a data point from dataset (in batches).

as mentionned previously in section IV A 3, we split the dataset into training and validation sets, in order to visualize how much the model is able to denoise the validation images, and see how the model 'imagines' the original data.

here is a visualization of denoising validation set in :

FIG. 9. Denoising a sample from validation set for Giga, Mini, Simple and Mega at step 272873

the expression to retreive the original image in one step we used is :

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t} \cdot \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{\alpha_t}}$$

1. Batch Size and Dropout

Model	Batch Size	Dropout	learning rate
Giga	6	0.1	2e-4
Mega	2	0.1	2e-4
Simple	3	0	1e-4
Mini	5	0	1e-4

TABLE III. Model batch sizes and dropout values

2. Noise Scheduling:

we used a **linear scheduler** for the noise β_t, which is a common choice in DDPMs, as it allows for a smooth transition from low to high noise levels during training. This linear schedule helps the model to learn to denoise images effectively at different noise levels, and it has been shown to work well in practice, It was recommanded to use other strategies like cosine annealing specialy for small datasets, but we estimate that ours is sufficentely large.

3. Parameters

for learning rate, we trained on $1e-4$ for Mini and Simple, and $2e-4$ for Giga and Mega models.

4. Training Procedure

5. Optimizer

We used the AdamW optimizer, which is an extension of the Adam optimizer with weight decay. AdamW helps to prevent overfitting by adding a weight decay term to the loss function, which dampens oscillations and large gradients.

the desicion to use AdamW was based on its effectiveness in training deep learning models, especially in the context of image tasks. It combines the benefits of adaptive learning rates and weight decay.

we kept the default parameters for AdamW, which are:

β_1	β_2	ϵ	weight decay	learning rate
0.9	0.999	1e-8	1e-4	1e-4

TABLE IV. Optimizer Parameters

6. Loss Function

the loss function used for training the models is Mean Squared Error. It measures the dissimilarity between the predicted noise and the true noise. The Mean Squared Error loss is defined as:

$$\mathcal{L} = \mathbb{E}_{x\sim p(x), t\sim\mathcal{U}, \epsilon\sim\mathcal{N}} ||(\epsilon - \epsilon_\theta(x_t, t))||^2$$

we choosed to not consider the upsampling factor in the MSE $\frac{1}{2\sigma_t^2}\frac{(1-\sigma_t)^2}{(1-\hat{\sigma}_t)\sigma_t}$, our goal is not the maximum likelyhood estimation but a better samples.

V. RESULTS

VI. DISCUSSION

A. Mode collapse

As observed in the Mega 74M model, the training process exhibited signs of mode collapse, where the model generated

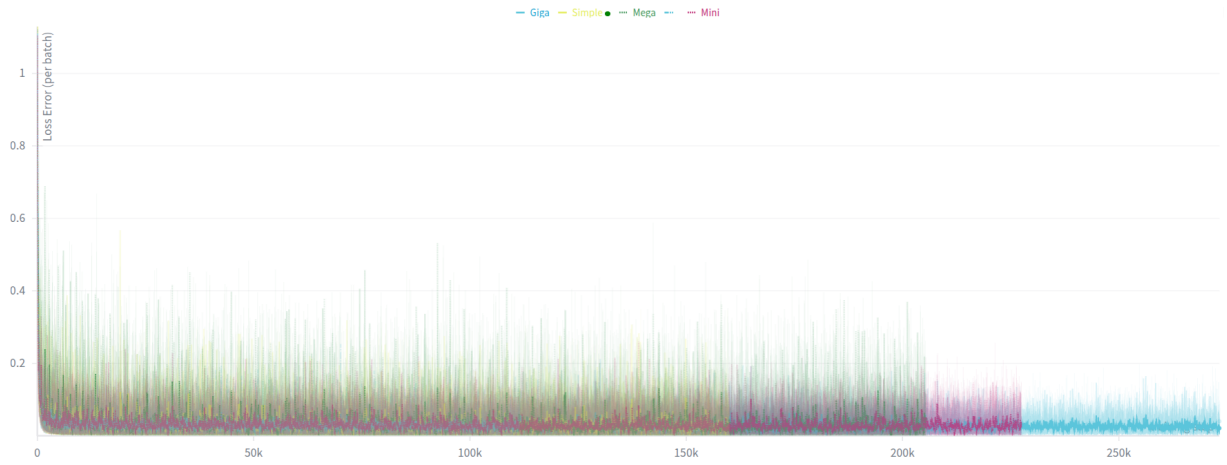

FIG. 11. Evolution of Train Loss per step for each model.

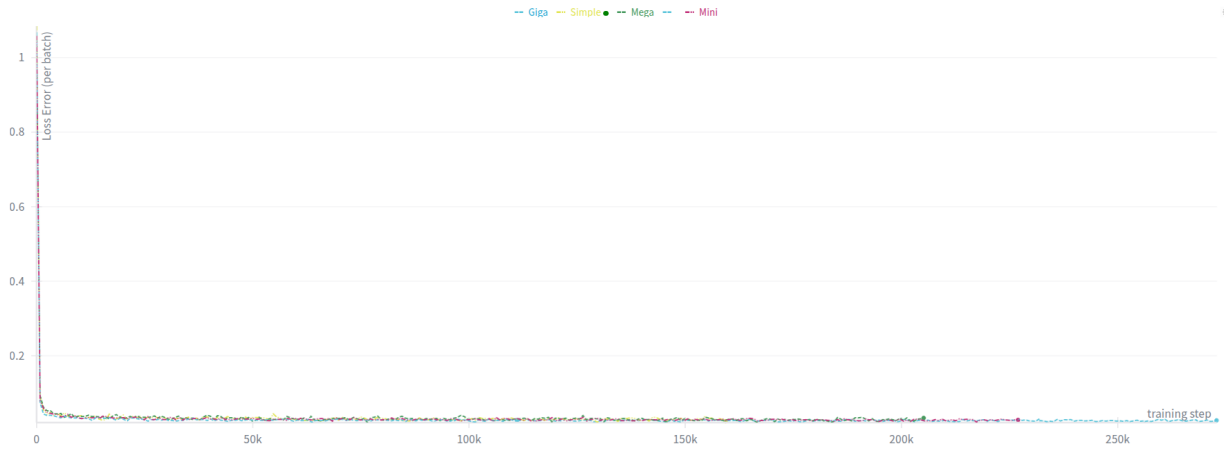

FIG. 12. Evolution of Validation Loss per step for each model.

a limited variety of outputs despite having a diverse training dataset. This phenomenon can be attributed to the model's inability to effectively explore the latent space, leading to a convergence towards a few dominant modes despite the model size. this issue is common for HVAEs, when the assumption on the distribution of latent variable is too constraining or when the chaining is too short, as we can see in the Mega model, as we progress in the training the quality improves, but in the Mega models we reach the best generation performence early between the 6th and 7th epochs, an then the quality is contiinously decreasing.

B. Self-Attention in Unet

Incorporating self-attention mechanisms into the U-Net architecture enhances its ability to capture long-range dependencies in the input data. This ability to attend to relevant features from distant pixels can be crucial for accurate predictions, and demonstrates the potential of self-attention in tasks requiring contextual understanding even for image generation.

C. Positional Embedding for Self-Attention

To further improve the performance of self-attention in our model, we can explore the use of additional positional embeddings suited for transformer as it wasn't directely imbedded. These embeddings provide the model with information about the relative positions of a set of pixels in the input sequence, which could be used for attention input of the shape $[B, C_i, H * W]$, to allow some subset of pixels attending each others, which is essential for maintaining the spatial structure of the data., instead of making relationships explicit using $[B, H * W, C_i]$.

D. the shape of attention-input

E. About possible improvements

F. The deceiving performance of Score-based Energy model

we followed the preconditioning and sampling strategy of the paper [7] to train a score-based energy model, which is a generative model that learns to approximate the score function of the data distribution. The model is trained to predict the score function at different noise levels, and it can generate samples by sampling from the learned distribution.

we obsered very poor of ED, such that we observed some huge fletuations in the train losses that varies from 0.01 to more than 4000 , which indicates unstability

[1] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *arXiv preprint arXiv:2006.11239*, 2020.

[2] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, K. McGrew, and I. Sutskever, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," *arXiv preprint arXiv:2112.10741*, 2021.

[3] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. Ghasemipour, A. Kolesnikov, T. Salimans, J. Ho, D. J. Fleet, and Q. V. Le, "Imagen: Scaling up diffusion models for text-to-image generation," *arXiv preprint arXiv:2205.11487*, 2022.

[4] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, 2022.

[5] V. Singh, "An in-depth guide to denoising diffusion probabilistic models (ddpm)," https://learnopencv.com/denoising-diffusion-probabilistic-models/, 2023, accessed: 2025-08-13.

[6] The same resource 2x(Nvidia a5000) allows training Mega on 2 batches, while allowing 6 batches for Giga.

[7] T. Karras, M. Aittala, T. Aila, and S. Laine, "Elucidating the design space of diffusion-based generative models," *NeurIPS*, 2022. [Online]. Available: https://arxiv.org/abs/2206.00364

Appendix A: Sampled Images for Mini 10M Architecture with bad attention input shape $[B, C_i, H * W]$

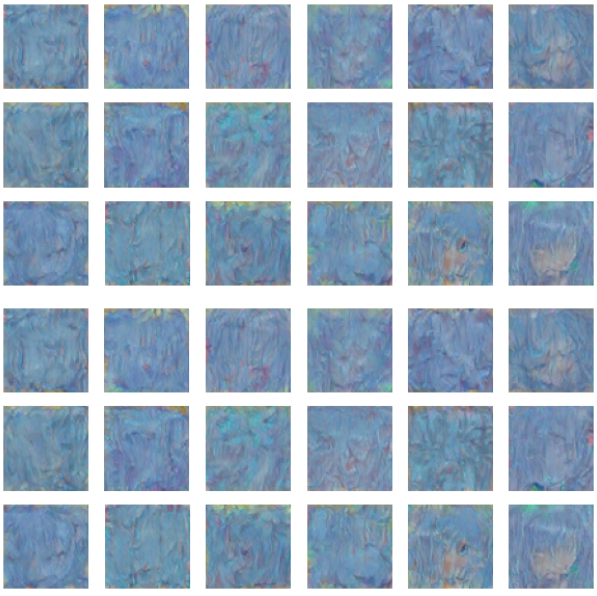

FIG. 13. Mini 10M epochs 2 and 5 (bad attention input shape).

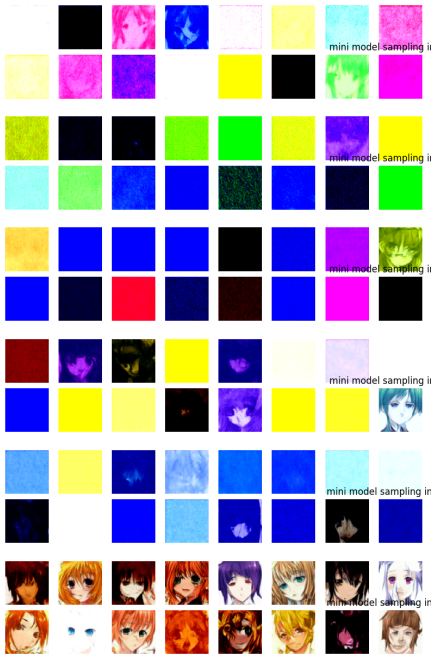

FIG. 14. Mini 10M epochs 1, 3, 6, 9, 12, and 15 (good attention input shape).

Appendix B: Sampled Images for Simple 49M Architecture

FIG. 15. Simple 49M epochs 1, 3, 5, 7, 8, 9, 10, 12, and 13.

Appendix C: Sampled Images for Mega 74M Architecture: input shape $[B, H * W, C_i]$

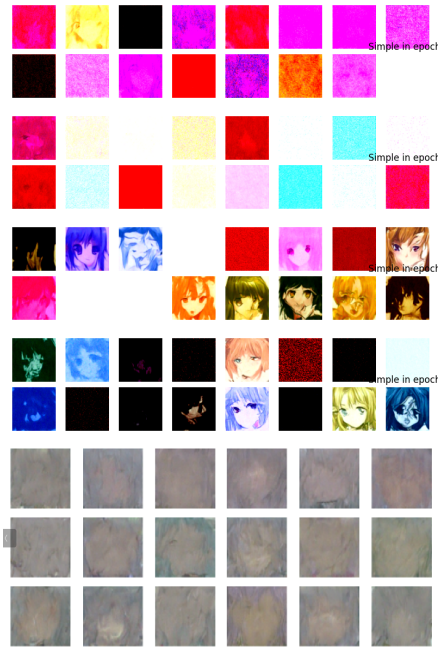

FIG. 16. Mega 74M epochs 1, 2, 4 and 5.

Appendix D: Sampled Images for Mega 74M Architecture: input shape $[B, H * W, C_i]$

Appendix E: Sampled Images for Giga 85M Architecture

FIG. 18. Giga 85M epochs 1, 3, 4, 5, 7, 9, 10, 13, 16, 17, 19, 20, 22.