Investigate the use of Deep Learning for Generative Models

Master Information Technology
Individual Project
Nghia Nguyen Tuan
1273713
ntuan@stud.fra-uas.de

***Abstract*—Generative AI has been making waves lately almost everywhere in the world, from the technology and computer science perspective. The most recent implementation of the Generative AI can be seen in various well-known applications. For example, in the field of Natural Language Processing (NLP), there is the world-wide famous Chat GPT and Siri. On the other hand, in the area of Image Processing, the State-of-the-Art applications includes Dall-E, Midjourney or Stable Diffusion. The focus of this project is to investigate and analyze the use of Deep Learning for a specific type of image generative models, the Diffusion Models.**

Keywords—Generative AI, Generative Models, Diffusion Models

I. Introduction

In an era where countless achievements have been seen rapidly in various areas of expertise due to the enormous growth in technology advancement and science development, Artificial Intelligence (AI) has been more frequently mentioned and widely used than ever. Usually, the so-called AI refers to a more general, powerful type of machine's intelligence, which can even be considered an independent conscious or entity. However, Generative AI is only one specific use of the AI to generate unlimited image variations of outputs based on the input images, hence the "generative" in its name. The more different kinds of data used to train the model, the more diverse the outcome the model can produce. The method of training the AI model is called Machine Learning (ML). According to IBM, ML can be described as a field that focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy [1]. The terms Neural Network (NN) and Deep Learning (DL) are usually associated with ML. In fact, NN is actually a sub-field of ML, and DL is a sub-field of NN. NN is basically an architecture of multiple node layers that is similar the neurons in human brains, and "deep" in DL refers to the number of layers in an NN. Many Generative Diffusion Models are based on this architecture, so they will be the main point of this project. Specifically, it also includes their definition, how they generally work, and their real-world applications.

II. Diffusion Models

Diffusion Models (DMs) are a new class of state-of-the-art generative models that generate diverse high-resolution images. They have already attracted a lot of attention after OpenAI, Nvidia and Google managed to train large-scale models. Example architectures that are based on diffusion models are GLIDE, DALLE-2, Imagen, and Stable Diffusion.

By definition, "a diffusion probabilistic model (or diffusion model for short) is a parameterized Markov chain trained using variational inference to produce samples matching the data after finite time. Transitions of this chain are learned to reverse a diffusion process, which is a Markov chain that gradually adds noise to the data in the opposite direction of sampling until signal is destroyed." [1]. To put it simply, diffusion models functions as a two-step process. First, the input data is added gradually with noise until it is completely covered in noise. This process is also known as "Forward Diffusion". After that, NNs are applied to reverse this process, hence the name "Reverse Diffusion". The input data will be recovered from that noise which is leveraged to generate and obtain new denoised and realistic data samples. Figure 1 [2] demonstrates how DMs work generally.

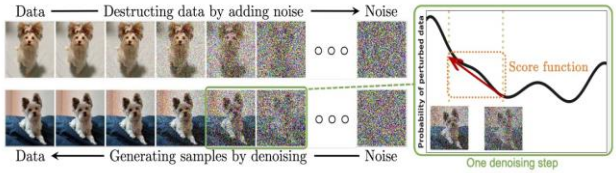


Figure 1: General Process of DMs

Based on this concept, DMs are able to generate brand-new data from the data that it is trained on. For instance, after a diffusion model is trained on images of the existing landscapes (Eiffel Tower, Pyramids, The Great Wall, etc.), it can then generate images of structures and monuments that do not actually exist, by simply mixing up and combining the parameters of the trained images together. Another example is a diffusion model trained on a photos collection of human faces. The results are completely new and realistic images of human faces with various features and expressions, even if those exact faces were not included in the original training dataset.

A. Forward Diffusion

The distribution q in the forward process is defined as Markov Chain given by Figure 2 [3]:

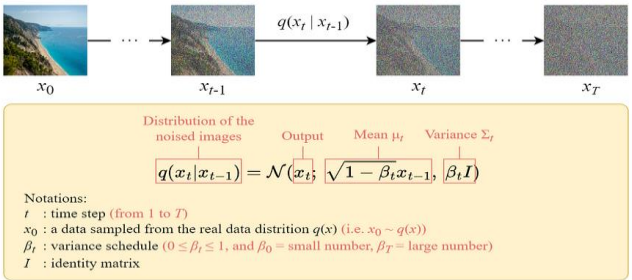


Figure 2: Forward Diffusion

As mentioned before, Foward Diffusion adds noises, specifically Gaussian noise, gradually to the input image x_0. The process is then repeated step by step, which in turn produces a sequence of noisy image samples $x_1, x_2, \ldots, x_t, x_T$, where T is the number of steps in total. As T reaches ∞, the

final sample results in a completely noisy image, similar to being sampled from an isotropic Gaussian distribution.

However, for a large time step, e.g. $t = 500 < T$, we have to apply q 500 times in order to sample x_t. Instead, this process can be improved by transforming it to a closed-form formula. It can be achieved by applying reparameterization:

$$\text{If } z \sim \mathcal{N}(\mu, \sigma^2) \text{ then}$$

$$z = \mu + \sigma\varepsilon \quad \text{where } \varepsilon \sim \mathcal{N}(0,1)$$

With ε being the standard normal distribution that has mean 0 and variance 1, the sampled image x_t can be expressed as below:

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \varepsilon_{t-1}$$

This equation can be expanded recursively, with $\alpha_t = 1 - \beta_t$:

$$\begin{aligned}
x_t &= \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \varepsilon_{t-1} && \varepsilon_0, \ldots, \varepsilon_{t-2}, \varepsilon_{t-1} \sim \mathcal{N}(0, I) \\
&= \sqrt{\alpha_t}\, x_{t-1} + \sqrt{1-\alpha_t}\, \varepsilon_{t-1} && \bar{\varepsilon}_0, \ldots, \bar{\varepsilon}_{t-2}, \bar{\varepsilon}_{t-1} \sim \mathcal{N}(0, I) \\
&= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_{t-1}}\, \varepsilon_{t-2}\right) + \sqrt{1-\alpha_t}\, \varepsilon_{t-1} && \varepsilon \sim \mathcal{N}(0, I) \\
&= \sqrt{\alpha_t \alpha_{t-1}}\, x_{t-2} + \left(\sqrt{\alpha_t(1-\alpha_{t-1})}\, \varepsilon_{t-2} + \sqrt{1-\alpha_t}\, \varepsilon_{t-1}\right) && \alpha_t = 1 - \beta_t \\
&= \sqrt{\alpha_t \alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_t \alpha_{t-1}}\, \bar{\varepsilon}_{t-2} && \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i
\end{aligned}$$

How?

It is necessary to explain how we go from line 4 to line 5. We denote these 2 terms as X and Y, which are regarded as samples from two different normal distributions $X \sim N(0, \alpha_t(1-\alpha_{t-1})I)$ and $Y \sim N(0, (1-\alpha_t)I)$, respectively.

$$= \sqrt{\alpha_t \alpha_{t-1}}\, x_{t-2} + \underbrace{\sqrt{\alpha_t(1-\alpha_{t-1})}\, \varepsilon_{t-2}}_{X} + \underbrace{\sqrt{1-\alpha_t}\, \varepsilon_{t-1}}_{Y} \qquad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$$

$X + Y$

Reparameterization Trick	Underlying Normal Distribution
$0 + \sqrt{\alpha_t(1-\alpha_{t-1})}\, \varepsilon_{t-2}$	$X \sim \mathcal{N}(0,\ \alpha_t(1-\alpha_{t-1})\, I)$
$0 + \sqrt{1-\alpha_t}\, \varepsilon_{t-1}$	$Y \sim \mathcal{N}(0,\ (1-\alpha_t)\, I)$

Additionally, the sum of two normally distributed (independent) random variables is also normally distributed. i.e. if $Z = X + Y$, then $Z \sim N(0, \sigma^2_X + \sigma^2_Y)$. The sum of two variances of X and Y can be calculated as below:

$$\begin{aligned}
Z &\sim \mathcal{N}(0, \sigma_X^2 + \sigma_Y^2) \\
Z &\sim \mathcal{N}(0, (1-\alpha_t\alpha_{t-1})\, I)
\end{aligned}$$

Reparameterization Trick

$$0 + \sqrt{1-\alpha_t\alpha_{t-1}}\, \bar{\varepsilon}_{t-2}$$

$$\begin{aligned}
\sigma_X^2 + \sigma_Y^2 &= \alpha_t(1-\alpha_{t-1}) + (1-\alpha_t) \\
&= \alpha_t - \alpha_t\alpha_{t-1} + 1 - \alpha_t \\
&= 1 - \alpha_t\alpha_{t-1}
\end{aligned}$$

We can repeat this transformation a few more times until the closed-form formula is reached:

$$\begin{aligned}
&= \sqrt{\alpha_t \alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_t \alpha_{t-1}}\, \bar{\varepsilon}_{t-2} \\
&\vdots \\
&= \sqrt{\alpha_t \alpha_{t-1} \cdots \alpha_1}\, x_0 + \sqrt{1-\alpha_t \alpha_{t-1} \cdots \alpha_1}\, \varepsilon \\
&= \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon
\end{aligned}$$

To conclude, the formula for the noisy image x_t is:

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon$$

This equation can be used to directly sample a noisy image x_t at any specific step t. Additionally, it only depends on the input image x_0, which makes the Diffusion Process faster and more convenient.

B. *Reverse Diffusion*

The reverse process can be expressed as in Figure 3 [3]:

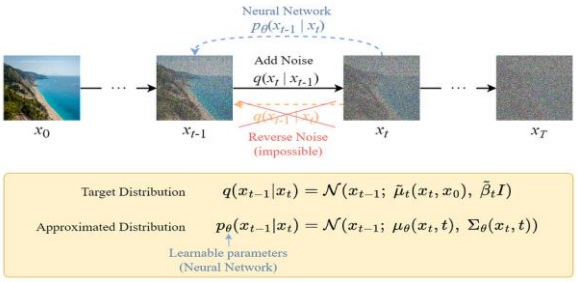


Figure 3: Reverse Diffusion

Unlike Forward Diffusion, $q(x_{t-1}|x_t)$ cannot be used to reverse the noise since it is not possible to be computed. Therefore, NN is required and trained to obtain the approximated distribution $p_\theta(x_{t-1}|x_t)$ instead. In order to do that, we first define the loss for the NN as a Negative Log-Likelihood:

$$\text{Loss} = -\log(p_\theta(x_0))$$

Depends on $x_1, x_2, \ldots, x_T$
Therefore it is intractable!

Then, the Variational Lower Bound is optimized instead of the Loss Function itself. After expanding, it can be expressed in three terms: Constant, Stepwise denoising and Reconstruction. The second term would be the main focus, since the other two can be ignored, due to one being a constant, while ignoring the other increase the sample quality [4].

$$-\log p_\theta(x_0) \le -\log p_\theta(x_0) + D_{\mathrm{KL}}(q(x_{1:T}|x_0) \| p_\theta(x_{1:T}|x_0))$$

$$\vdots$$

$$-\log p_\theta(x_0) \le \mathbb{E}_q\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\right] \quad \text{Variational Lower Bound}$$

$$\vdots$$

$$-\log p_\theta(x_0) \le \mathbb{E}_q\Big[\underbrace{D_{\mathrm{KL}}(q(x_T|x_0) \,\|\, p_\theta(x_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0) \,\|\, p_\theta(x_{t-1}|x_t))}_{L_{t-1}} \underbrace{-\log p_\theta(x_0|x_1)}_{L_0}\Big]$$

① ② ③

- ① No learnable parameters
- ① Just a Gaussian noise
- ② Stepwise denoising term
- ③ Reconstruction term

① → Constant → Ignorable

After a series of derivations of the Stepwise denoising term [4], the mean $\tilde{\mu}_t$ of $q(x_{t-1}|x_t, x_0)$ can be obtained. The approximation μ_θ of this mean $\tilde{\mu}_t$ is what we need.

$$\tilde{\boldsymbol{\mu}}_t(x_t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}_t\right)$$

$$\text{Set} \quad \boldsymbol{\mu}_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}_\theta(x_t, t)\right)$$

The Mean Squared Error (MSE) is then applied to compare these two:

$$\begin{aligned}
L_t &= \mathbb{E}_{x_0,\varepsilon}\left[\frac{1}{2\sigma_t^2}\left\|\tilde{\boldsymbol{\mu}}_t(x_t) - \boldsymbol{\mu}_\theta(x_t, t)\right\|^2\right] \\
&= \mathbb{E}_{x_0,\varepsilon}\left[\frac{1}{2\sigma_t^2}\left\|\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}_t\right) - \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}_\theta(x_t, t)\right)\right\|^2\right] \\
&= \mathbb{E}_{x_0,\varepsilon}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\sigma_t^2}\left\|\boldsymbol{\varepsilon}_t - \boldsymbol{\varepsilon}_\theta(x_t, t)\right\|^2\right]
\end{aligned}$$

ignorable

$$L_t^{\text{simple}} = \mathbb{E}_{t\sim[1,T],x_0,\varepsilon_t}\left[\left\|\boldsymbol{\varepsilon}_t - \boldsymbol{\varepsilon}_\theta(x_t, t)\right\|^2\right]$$

Conclusively, we just need to approximate the noise ε_t using a neural network ε_θ. The training objective is then a simple MSE loss between the true noise added in the forward process and the predicted noise by the model.

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon$$

$$L_{\text{simple}} = \mathbb{E}_{t,x_0,\varepsilon}\left[\|\varepsilon - \varepsilon_\theta(x_t, t)\|^2\right]$$

C. *Architecture*

According to Ho et al. [1], DMs use a UNet-shaped NN for the reverse process (including training) that consists of 5 components: Encoder blocks, Bottleneck blocks, Decoder blocks, Self attention modules, Sinusoidal time embeddings [4]. In the UNet:

1. There are four levels in the encoder and decoder path with bottleneck blocks between them.

2. Each encoder stage comprises two residual blocks with convolutional downsampling except the last level.

3. Each corresponding decoder stage comprises three residual blocks and uses 2x nearest neighbors with convolutions to upsample the input from the previous level.

4. Each stage in the encoder path is connected to the decoder path with skip connections (Concatenate).

5. The model uses "Self-Attention" modules at a single feature map resolution.

6. Every residual block in the model gets the inputs from the previous layer and the Sinusoidal embedding of the current timestep. Embedding is technically a tensor that inform the model of the input's current position in the Markov chain.

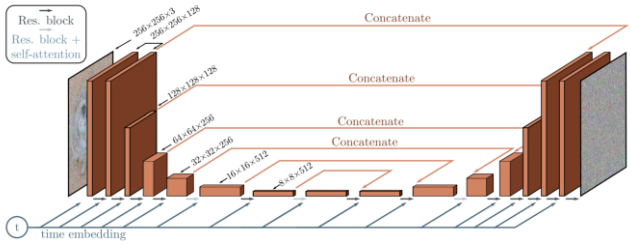


Figure 4: UNet Architecture

D. *Pipeline*

Forward Diffusion:

In each epoch, the image is processed as below:

1. A random time step *t* will be selected for each sample image, e.g. a photo of a beach.

2. The Gaussian noise is applied to the that sample image based on the selected time step *t* accordingly.

3. The time steps are converted to Sinusoidal time embeddings.

The input for the training of the UNet model will consist of both the noisy image and the corresponding embedding. However, the embeddings do not add additional parameters to the model.

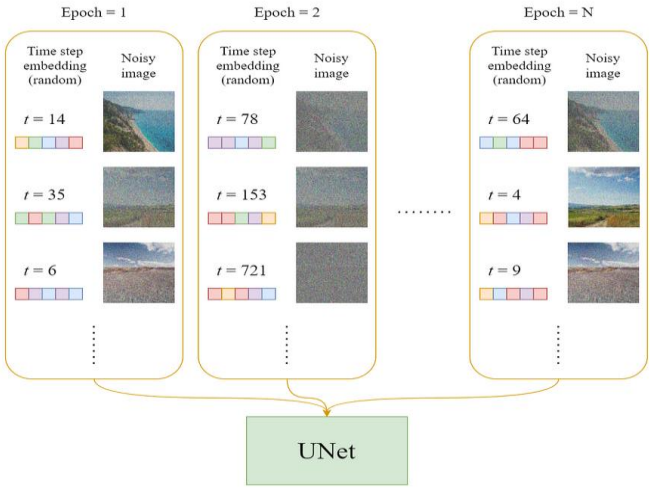


Figure 5: Dataset for training

Training:

In the first epoch, the UNet will predict the noise added in an image, then compare it to the true noise. The difference between the predicted noise and the true noise is represented by a loss function MSE. By using Stochastic Gradient Descent (SGD) with an appropriate learning rate, we can gradually minimize the loss function after each epoch, i.e. the model will predict the noise more and more accurately. After a certain epochs, the maximum accuracy of the model will be reached. After the training is completed, we have a noise predictor capable of estimating the noise added to an image.

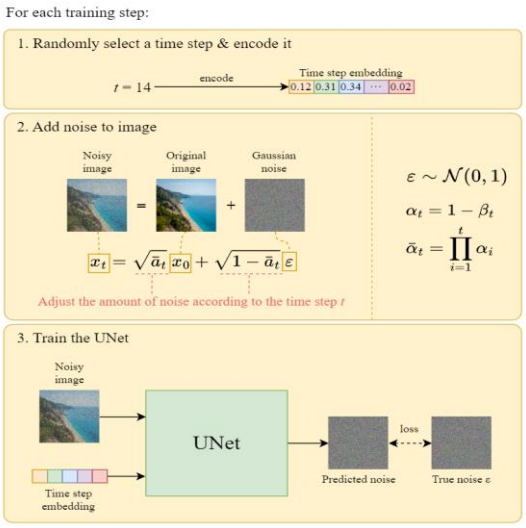


Figure 6: Training Illustration

Reverse Diffusion:

Now we have the noise predictor. To put it to use, we first generate a random noisy image and ask the noise predictor to predict the noise. We then subtract this estimated noise from the original image. Repeating this process a few times will result in a normal image. In summary, the model is now capable of reversing the diffusion process by reducing the predicted noise from the input noisy image, which results in the denoised image.

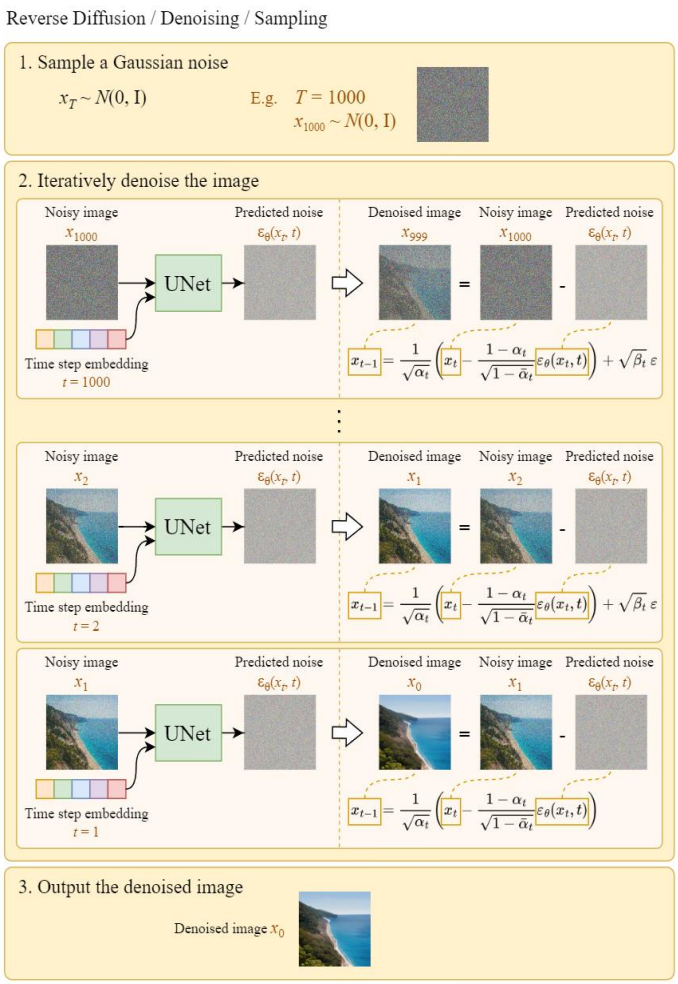


Figure 7: Denoising Illustration

III. Stable Diffusion

Although there are currently many popular applications of Diffusion Models such as GLIDE, DALLE-2, Imagen, etc., Stable Diffusion would be the main approach for this project. The reason is that it is not only based on the previous Diffusion Models, but also published as an open source. The principle behind Stable Diffusion is that instead of applying diffusion directly on a high-dimensional input, it is compressed into a lower-dimensional latent space in order to reduce the computational cost of the training process. This is done using a technique called variational autoencoder (VAE). Firstly, the input is encoded by an encoder network into its latent representation, i.e. $z_t = g(x_t)$, After new data is generated by UNet, it is then upsampled by a decoder network. Figure 7 [5] illustrates the simple representation of this method.

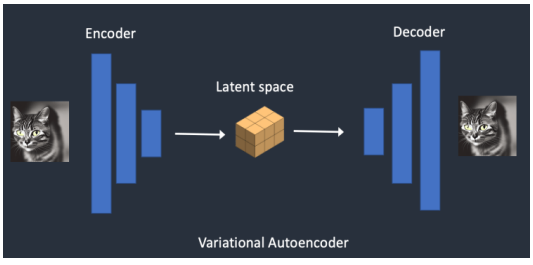


Figure 7: VAE Illustration

To further explain the necessity of VAE, the initial diffusion process occurs in the image (pixel) space, which is enormous. For example, a common 512x512 image with three-color channels (RGB) will have 786432-dimensional space, which is computationally very slow. A latent space of Stable Diffusion is 4x64x64, which is 48 times smaller than the image space. This version of Diffusion Models is also known as Latent Diffusion Models (LDMs), or the so-called Stable Diffusion, which has achieved a near-optimal balance between complexity reduction and detail preservation, leading to a significant boost in both computing speed and visual fidelity.

However, so far, the Diffusion Models can only generate images similar to those that they have been trained on. It wouldn't be really practical if they are not able to generate completely new images based on the user's control, as we have seen nowadays. Therefore, one more step is required. This step is referred to as guided diffusion, which sets the conditions for the sampling process of image generation in order to manipulate the generated outputs.

A. *Text Prompt Process*

The guided diffusion in Stable Diffusion is done via text prompts that help control the generated outputs of the images. With both diffusion process and text prompts, Stable Diffusion is a text-to-image model as a complete package. The purpose of the text prompts is to provide directions for the image generator so that the noise will be subtracted from the image and predicted into what we want. The whole process can be divided into many small steps, as shown in Figure 8 [5].

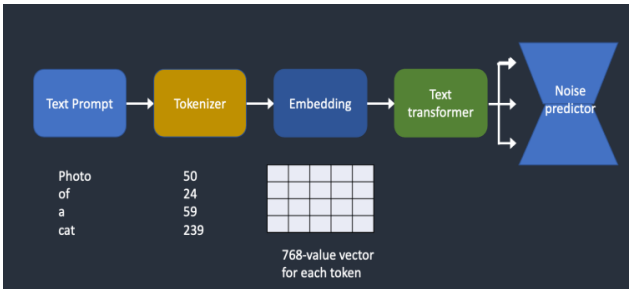


Figure 8: Text Prompt Process

Specifically for each step, we have:

Tokenizer: First, the text prompt is tokenized into numbers for computers to read. This is done by a CLIP tokenizer, which is a DL model developed by OpenAI to produce text descriptions of images. Usually, one word is converted into one token. Stable Diffusion model is limited to 75 tokens in a prompt.

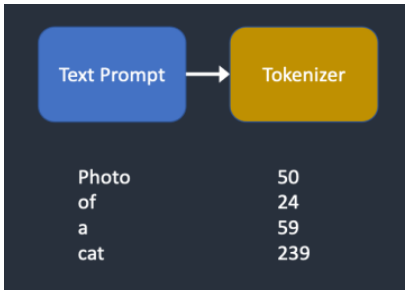


Figure 9: Text Tokenizing

Embedding: Embedding is technically a vector representation of a token, which has 768 values. Each token has its own unique embedding vector. However, words that are closely related to each other, e.g. kid, child, would have similar embeddings.

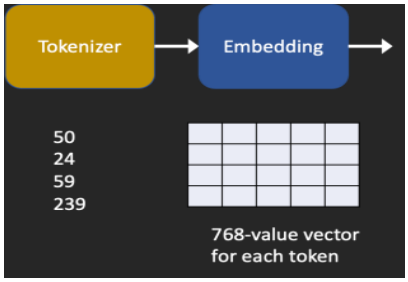


Figure 10: Tokens Embedding

Feeding embeddings to UNet: The embeddings are further processed through a Text transformer, then fed into the UNet (noise predictor).

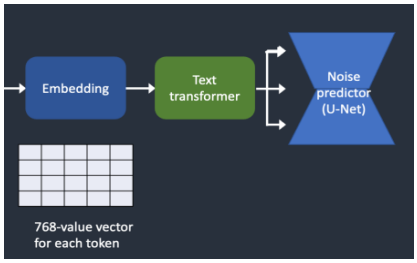


Figure 11: Embedding into Unet

Cross-attention: The UNet uses the output from the Text transformer multiple times through a cross-attention mechanism. To demonstrate this step more clearly, let's try an example. The prompt "A kid with black hair" will enable Stable Diffusion to pair the two words "black" and "hair" together so the goal would be an image of a kid with black hair, not a black kid or a kid with black hat, etc. As a result the reverse diffusion will be lead towards images containing black hair. This step is basically a cross-attention between the prompt and the image.

B. Pipeline

Basically, Stable Diffusion returns an image after a text prompt is given. The complete process will be as follow:

Step 1: Stable Diffusion generates a random tensor in the latent space. This tensor will be denoised into your required image later, but for now, it is completely noised. A parameter called "seed" can be set to specify the tensor. For example, if you keep the same seed = 1337, you will always get that same random tensor.

Step 2: The UNet (noise predictor) predicts the latent noise from the input tensor and text prompt.

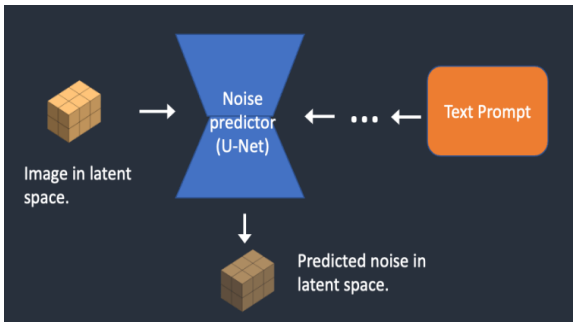


Figure 12: Noise Predictor

Step 3: The new latent image is produced by subtracting the latent predicted noise from the previous latent image.

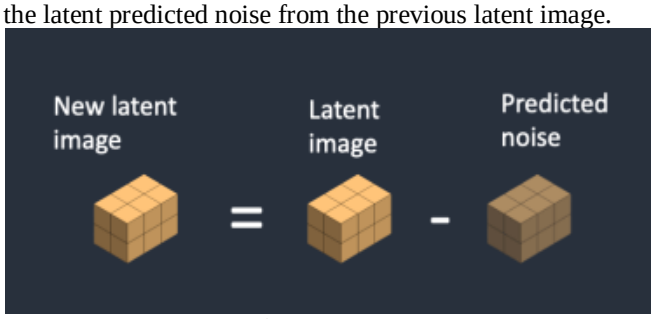


Figure 13: New Latent Image

Step 4: Step 2 and step 3 are repeated for a certain number of sampling steps. For example, if the steps *t* are set to be 500, than they are repeated for 500 times. Usually, the more steps you have, the better the quality of the image, until it reaches the maximum point. The picture below illustrates how the image looks like after each sampling step.

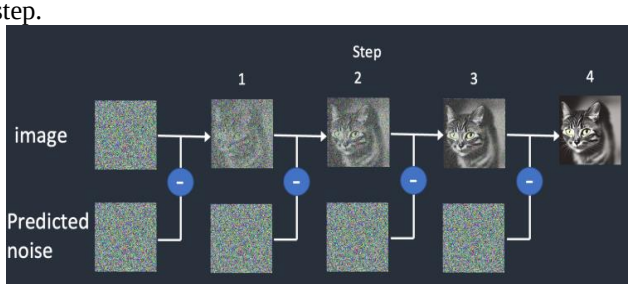


Figure 13: Image Sampling

Step 5: The latent image is converted back to the image (pixel) space by the VAE, which is the output image.

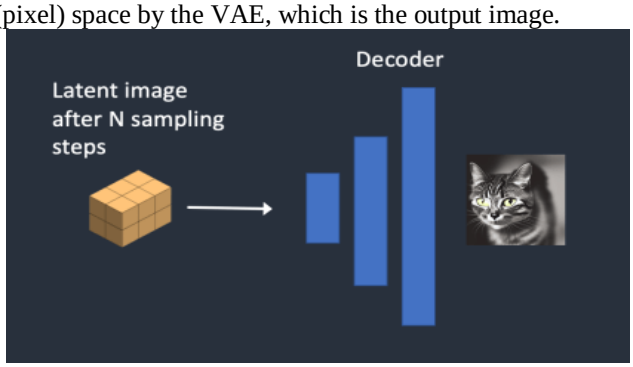
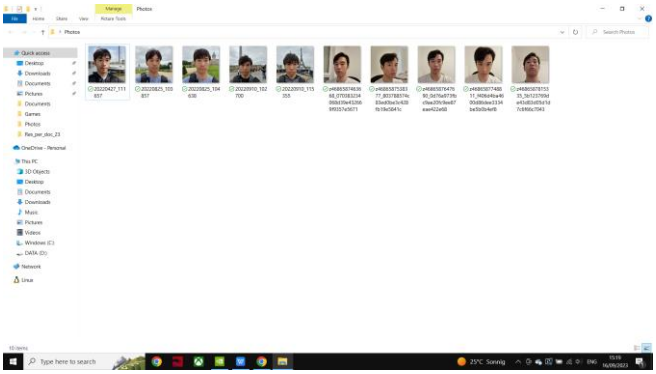


Figure 14: Output

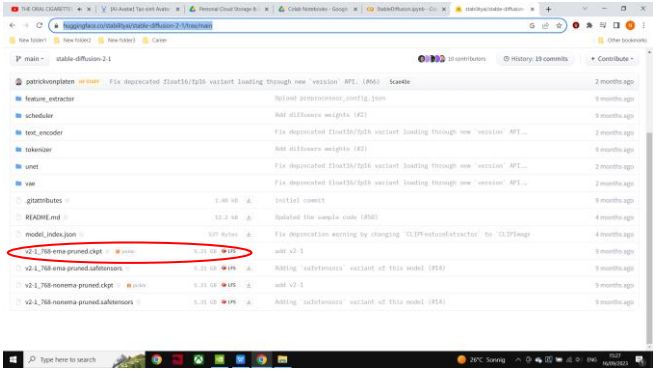
C. Implementation

In this small implementation, Stable Diffusion v2.1 will be used to train with photos of myself. After that, it can generate completely new images of myself, which did not exist in the training dataset. The source code is provided right here: , which can be run on Google Collab.

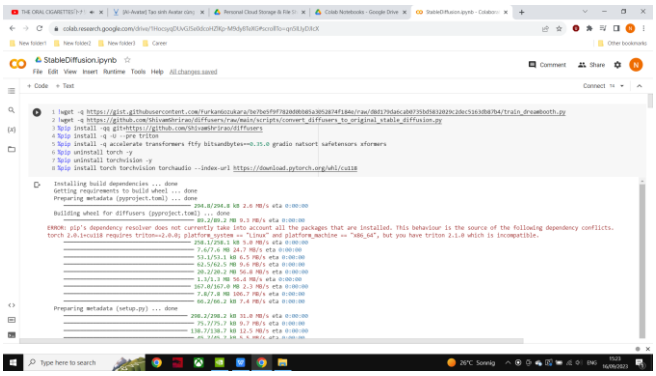
Step 1: The dataset for training is prepared by taking ten photos of myself. These photos are then cropped by an online application BIRME [7] so that their resolutions are 512x512 each. It is the maximum resolution that this model supports.



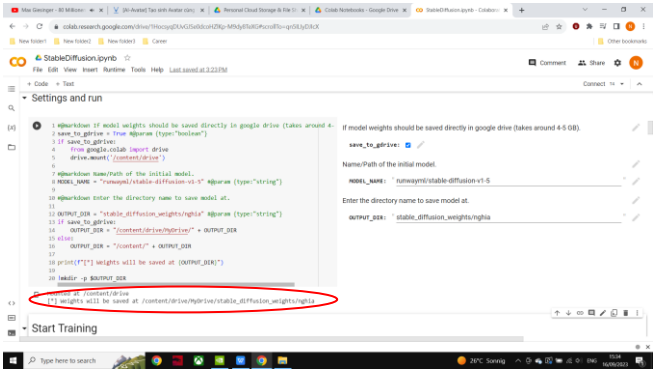
Step 2: Download Stable Diffusion file on HuggingFace [8]. This is the file of an already trained model that has its own weights and biases, which is why it is quite large (5.21 GB).



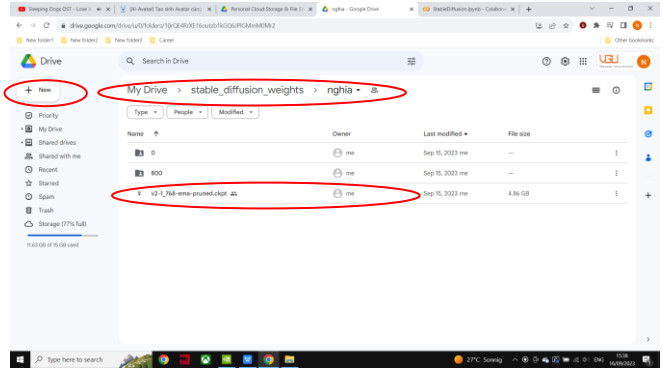
Step 3: Open the code with Google Collab. It can also be run directly on your PC using Jupyter Notebook. After that, start running the code cells. The first cell is for installing the necessary modules and libraries.



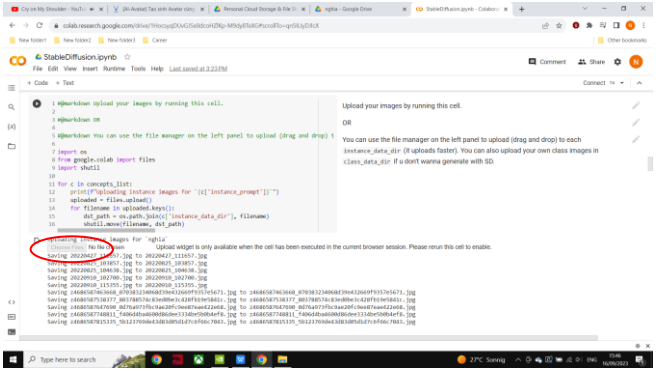
Step 4: The code shown below will create a new directory on my Google Drive so that I can upload the earlier downloaded file here. Remember to check save_to_gdrive box.



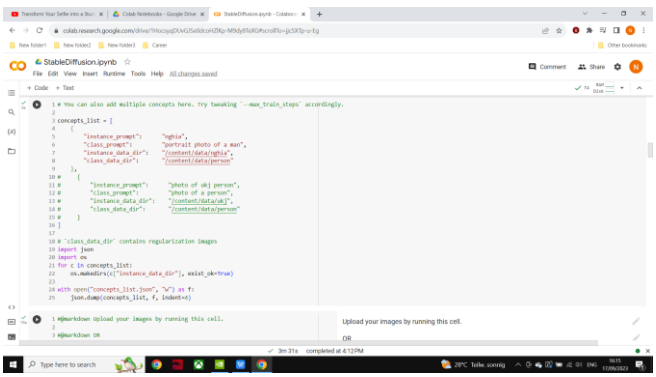
Step 5: Open my Google Drive, then navigate to the specified path above and upload the Stable Diffusion file with “New”.



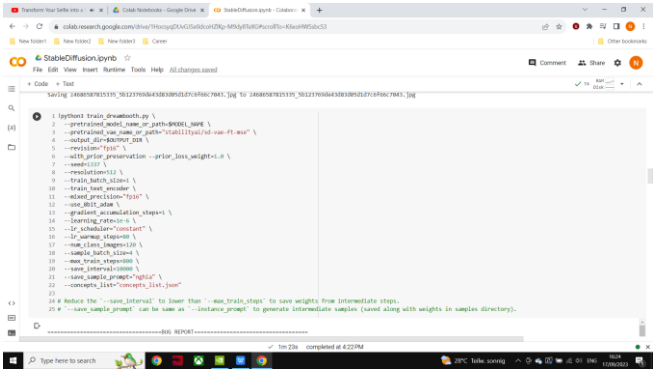
Step 6: Continue running the code cell-by-cell. When reaching the below code cell, click on “Choose Files” and select my training dataset of 10 photos of myself.



Step 7: Define the prompt for the training images. After training, the model will generate the image of myself based on the prompt “nghia”.



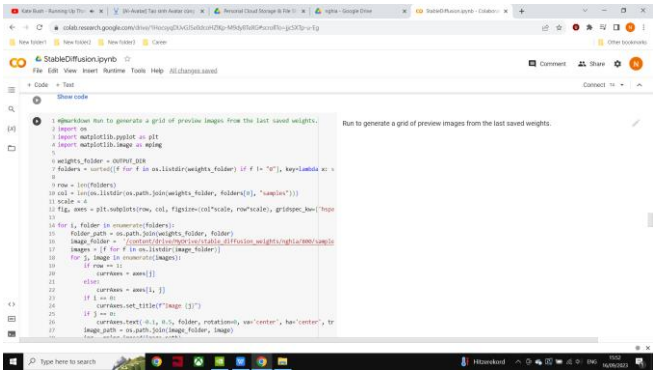
The training process can be started with the code cell below.



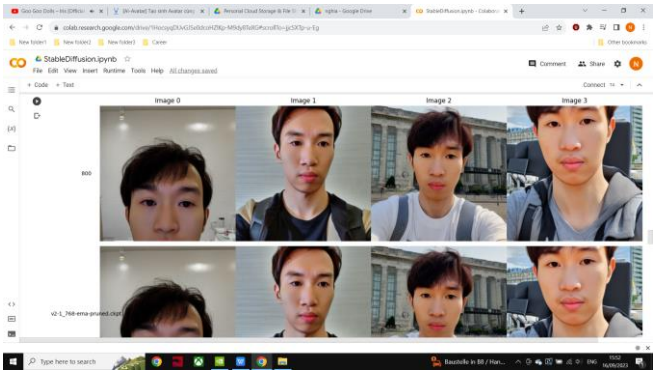
There are many parameters related to this process. Here are the explanation for the most important ones:

- seed=1557: It is a random tensor generated by Stable Diffusion as mentioned in **Step 1** in the *Pipeline* section.
- resolution=512: the maximum resolution supported by this model.
- max_train_steps=800: the model is trained 800 times.
- num_class_images=120: this parameter equals to the number of training images (10 in this case) multiplies with 12.

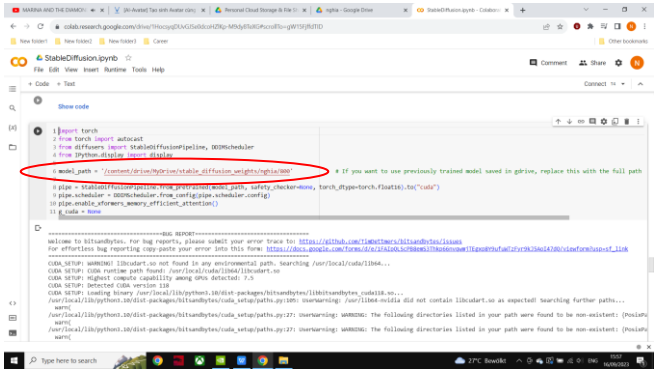
Step 8: After the training is done, run the next code cell to show the sample images that the model generates by itself.



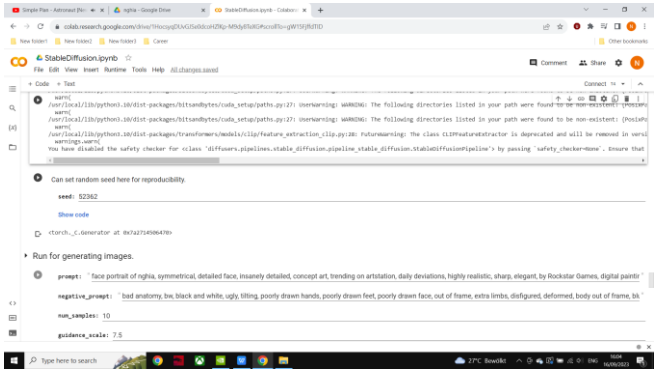
Here are the sample images:



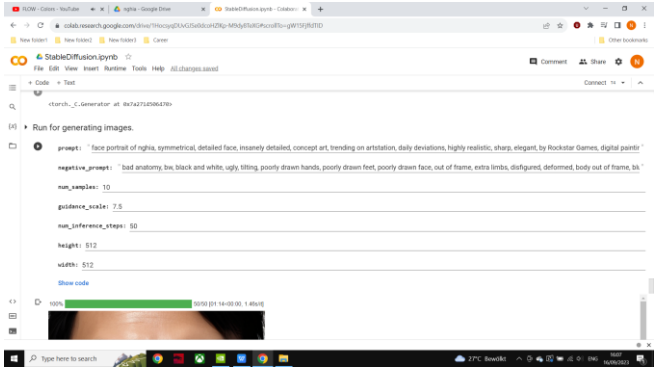
Step 9: The weights file of the recently trained model is saved in a directory in Google Drive so that it can be used directly for the text prompts. In the code, this directory is specified as “model_path”.



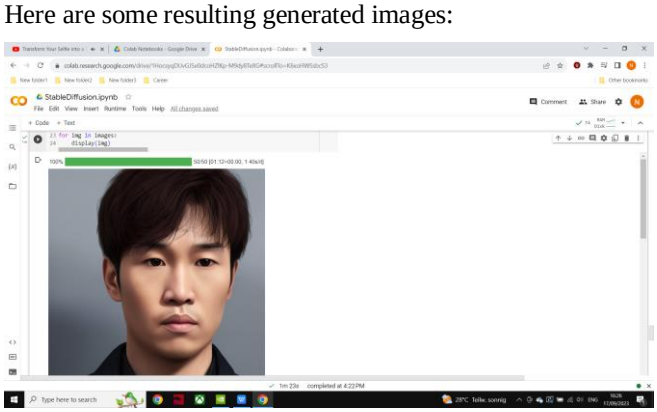
Step 10: A variable called seed is a number that can be set so that same images can be generated again. Specifically, with the same prompt, for the same seed, the same images will be created, regardless of how many times the images are generated.

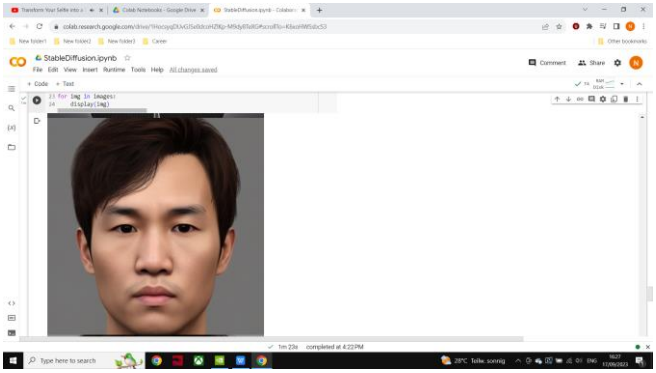


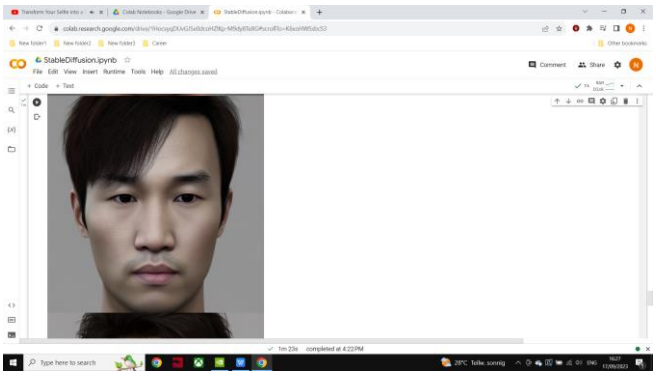
Step 11: Enter the prompt (and negative prompt optionally) to specify how you want the generated images to be. I will generate images of myself with the prompt “nghia”, with a few additional prompts to further specify the conditions for the images. There are many examples of prompts for various results on Lexica.



Here are some resulting generated images:







IV. Conclusion

Diffusion Models, or specifically Stable Diffusion have been making waves recently in the fields of AI Generative Models. The idea behind this model is to train an NN to reverse the noise-adding process, then based on the user's prompt to generate completely new images. Additionally, Stable Diffusion is an open source, so an already trained model can be easily installed for free, with additional training with my own dataset to create a unique new prompt for the model. This new prompt can be combined with various sample prompts on Lexica to generate an endless amount of different variations of outputs, limited only by the users' creativity.

Acknowledgment

I would like to express my thanks and gratitude to Professor Nauth for his invaluable guidance and constructive support during the process of conducting this research paper.

References

[1] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic models," arXiv.org, 16-Dec-2020. [Online]. Available: https://arxiv.org/abs/2006.11239.

[2] An introduction to diffusion models for machine learning. (n.d.). https://encord.com/blog/diffusion-models/

[3] Steins. (2023, July 11). Diffusion model clearly explained! - Steins - medium. Medium. https://medium.com/@steinsfu/diffusion-model-clearly-explained-cd331bd41166

[4] Singh, V., & Singh, V. (2023). An In-Depth Guide to Denoising Diffusion Probabilistic Models – From theory to implementation. LearnOpenCV – Learn OpenCV, PyTorch, Keras, Tensorflow With Examples and Tutorials. https://learnopencv.com/denoising-diffusion-probabilistic-models/

[5] Andrew. (2023, September 27). How does Stable Diffusion work? Stable Diffusion Art. https://stable-diffusion-art.com/how-stable-diffusion-work/

[6] https://colab.research.google.com/github/FurkanGozukara/Stable-Diffusion/blob/main/DreamBooth/ShivamShriraoDreamBooth.ipynb#scrollTo=5vDpCxId1aCm

[7] https://www.birme.net/.

[8] https://huggingface.co/stabilityai/stable-diffusion-2-1/tree/main.