# Diffusion Models for Machine Learning
## CS 584 Final Report

1st Isaias Rivera
*College of Computing*
*Illinois Institute of Technology*
Chicago, Illinois, USA
irivera3@hawk.iit.edu

## I. Introduction

Diffusion models are a type of generative model that utilizes the principles of diffusion to generate realistic data. There are different types of diffusion models that have the same underlying idea including, diffusion probabilistic models, noise-conditioned score network, and denoising diffusion probabilistic models [1].

The basic idea behind diffusion-based generative models is that noise is first added to an initial input, where the resulting noisy image is then processed to subsequently remove that noise, while preserving the underlying structure of the image. This process is done multiple times where, at each step of this process, the noise level is decreased, and the resulting image becomes more and more refined. After this training, the model can then be used to generate data by passing random noise through the learned denoising process.

Diffusion-based generative models can function either unconditionally or guided such as with GLIDE's text-guided diffusion model, trading "diversity for fidelity" [2]. There are other examples of this online such as with Google's Imagen, and OpenAI's DALL-E 2.

There are many variations [3] [4] [5] [6] and different improvements [7] [8] to diffusion models that are being actively researched and have already been implemented, however, as in my proposal I mainly have wanted to explore implementing my own in an easily digestible format.

## II. Focus

The main focus of my project has been with Denoising diffusion probabilistic models, or DDPM for short, as I have found the most literature revolving around this variation on diffusion models. I have not gone further than DDPMs as I wanted to to first get some results before moving on. Regardless, the majority of my research has been with understanding both the math and implementation of every step with a DDPM [9] [10] [11].

## III. Implementation

Thus far, I have a very basic implementation of the forward and backward process of a DDPM. For some components of the process I am still working towards fully understanding and developing an easy to understand notebook, however, I do have a working model, albeit, relatively crude and unoptimized.

My main notebook uses torch to implement everything, however, I have found many implementations that use a variety of APIs.

### A. The Forward Process

The forward process is the step that involves adding noise to data. Given a number of timesteps, gaussian noise is added to data at each step, where subsequent steps are dependent on the output of the previous step. The main equation that describes this, is as follows

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}) := \prod_{t=1}^{T} \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1},\ \beta_t\mathbf{I})$$

Where $T$ is the total number of timesteps, $\beta_t$ is the variance schedule, where $\beta_t \in (0, 1)$ and $\beta_1 < \beta_2...$, it describes the amount of noise we want to add to each timestep. This schedule can simply be set as linear,

### B. The Reverse Process

The reverse process involves sampling the forward process using a random timestep and predicting the noise in that sample. Where a neural network is trained to predict this noise.

The following equation describes this process.

$$p_\theta(x_{0:T}) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

$$:= p(x_T) \prod_{t=1}^{T} \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

Where $X_{t-1}$ is predicted using the current timestep.

### C. Timestep Embedding

Timestep Embedding, or positional embedding, is used to ensure that the model can "know" where

Another thing that I need to touch on is the position embedding, because the model depends on t, in terms of knowing what is the noise level is at any given step, what essentially is done is the timestep is embedded into the input using what is called sinusoidal position embedding. How this is integrated is that a vector is generated for each step and is passed alongside the input data into the U-net model as an additional channel. This embedding helps capture the dynamics of the data over time.

*D. Loss Function*

*E. U-Net*

*F. Training*

*G. Sampling*

*H. Simplified Example*

*I. Improvements*

*J. Comparisons*

*K. Recent Developments*

## REFERENCES


[1] L. Weng, "What are diffusion models?," *lilianweng.github.io*, Jul 2021.

[2] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," 2021.

[3] J. Ho and T. Salimans, "Classifier-free diffusion guidance," 2022.

[4] A. Nain, "Denoising diffusion probabilistic model," 2022.

[5] A. Q. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. Mcgrew, I. Sutskever, and M. Chen, "GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models," in *Proceedings of the 39th International Conference on Machine Learning* (K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, eds.), vol. 162 of *Proceedings of Machine Learning Research*, pp. 16784–16804, PMLR, 17–23 Jul 2022.

[6] N. R. Kashif Rasul, "The annotated diffusion model," 2022.

[7] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," 2021.

[8] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," 2022.

[9] A. Nain, "All you need to know about gaussian distribution."

[10] A. Nain, "A deep dive into ddpms."

[11] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.