Denoising Diffusion Probabilistic Models (DDPM) for Protein Structure Generation Technical Report

# Introduction:

Diffusion Models have demonstrated unprecedented capabilities in a wide range of applications in the context of generation, where the main purpose is to generate meaningful novel signals, from images, sound, to trajectories, or to enhance a given signal (inpainting, image quality enhancement, etc…) More recently, Diffusion Models were adopted to tackle some of the most complex challenges in biology and drug discovery. Among applications of such models that attracted the most attention in the Biomedical AI field, is De Novo Protein sequence and structure generation, where the idea is to find novel functional proteins, for example, to engineer binders to specific peptides in the context of personalized medicine. Furthermore, diffusion models have made it possible to generate proteins with the specific desired properties and characteristics thanks to conditional generation. In short, the problem we attempt to solve through this work, is to train a diffusion model in a way that it becomes capable of generating novel protein backbone structures.

# Proposed Approach:

The elaborated approach consists in training a UNet-based DDPM model on translating pure Gaussian noise into a meaningful protein backbone, using a UNet model for noise estimation ( the classic approach).   
During training, at each iteration, a batch of inputs (protein backbone structures) is loaded, then an amount of Gaussian noise is added to each input in the batch, following the forward process of the DDPM model as described in the following sections. Each noised structure is then passed to the noise estimation network of the DDPM model to predict the added noise, following the reverse operation as discussed in later sections. A loss value is then calculated with respect to the predicted and actual noise.

During inference, Gaussian noise is sampled in the shape that matches that of the protein backbones, then over T timesteps, the model gradually denoises the structure by predicting the amount of noise that could have been added at each time step, until it reaches a shape that visually reflects a stable protein.

## II.1. Model Architecture:

In this section, we describe in detail the model architecture, starting by breaking down the global DDPM framework, then providing details on the UNet module.

### II.1.1. DDPM Framework:

The DDPM framework is mainly made of 3 main building blocks: the forward operation, the reverse operation and the noise estimation network.

**Forward process:** As per the original DDPM paper Ho et al. (2020) the forward operation performs the gradual Gaussian noise addition to an input over time steps to achieve and is defined as follows:

where

This basically means that a noised input at timestep is given by the cumulative product of each couple of successive noising iterations. It is also important to mention that the amount of noise added by the forward process is also decided by a schedule .

It has been proven in Ho et al. (2020) that we can achieve knowing at any time step in one single forward step without having to gradually add noise over timesteps. This is achieved thanks to the following equality:

where and

Below is the code used to set these variables:



Given the properties of Gaussian distributions, after some derivation we see that in application, is obtained by the following:  
  
 where is the randomly sampled Gaussian noise.

Below is the implementation of the calculation:  




**Reverse Process:** Moving on to the reverse or denoising process, during training, the model has to predict the amount of noise added at time step given and . The predicted noise is denoted where represents the parameters of the noise estimation network. The model weights are updated in a way to maximize the similarity between the predicted and actual noise. The loss function is discussed in the coming sections.

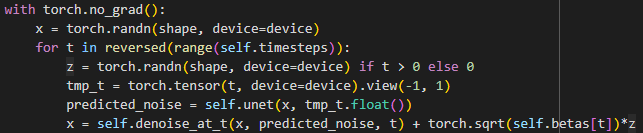


At sampling stage, we start from pure Gaussian noise then over time steps, looping in reverse, we gradually denoise the input until we hopefully achieve a meaningful signal at time step .

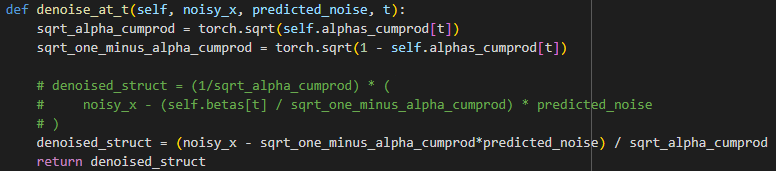
At each time step, we feed the noisy structure to the noise estimation network, then perform the denoising operation that removes the predicted noise.

As shown in Ho et al. (2020), the reverse of the forward operation applied on and that results in is defined as follows:

+ where



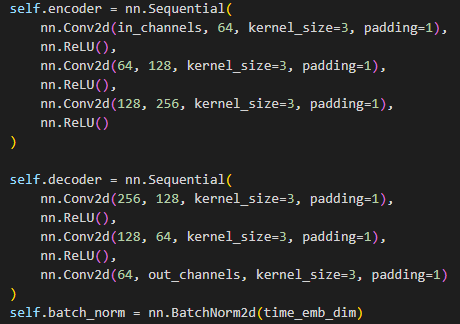
As previously mentioned, the end goal is to be able to generate structures that are both meaningful and novel in the same time, and that’s where the comes into play, its purpose to add an additional noise to the denoised image at each time step to promote diversity. Our understanding here is that, without that additional noise, and in case the model models our data distribution very well, the sampling process will always yield structures that are very similar to previously seen ones, and the process becomes somewhat deterministic. The amount of additional noise added also follows the noising schedule .



In our case, most likely due to inconsistencies in our implementation, the conventional denoising formula shown in the paper didn’t result in any improvement throughout the training process. But we were able to visually observe a slight correlation between the original and denoised structures by making a small adjustment in the denoising method as shown in the snippet above. This observation was made by visualizing samples with added noise and their denoised versions, during training, to get a sense of what the model is learning. The question we are trying to answer through this is: **“Does the resemblance between the noisy and denoised version increase as the model trains or not ?”**

**Noise estimation network:** Since the UNet architecture was proven to work well in the context of diffusion, we decided to also follow Ho et al. (2020) in the choice of the noise estimation network. Also, we sought sensibility in this choice due to the shape of the output we expect to get out of this module. In our case, generating protein backbone structures means operating on and generating 3D data, which can be handled effectively with the encoder (convolution) - decoder (deconvolution) shape of the UNet architecture.

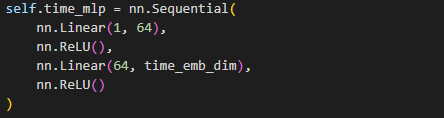
To achieve a minimalistic version of our framework, we decided to keep the UNet module as simple as possible, where the encoder is made of three convolutional layers (Conv2D) each followed by a ReLU activation. The decoder is also made of three convolutional layers, the first two followed by a ReLU activation, with one main difference being the sizes of the input and output channels.



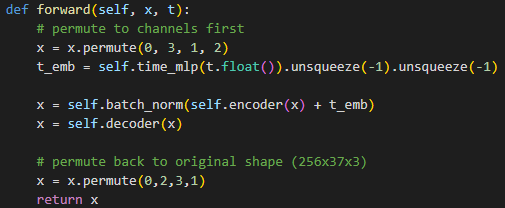
In this minimal implementation, and as opposed to conventional implementations, we do not incorporate ResNet blocks, residual connections, nor attention blocks. The input is passed through the encoder layer, then at the bottleneck, the last encoder state is combined with the time embedding.

The time embedding is added to make the noise estimation conditioned on the time step, in order to feed it to the network, it is linearly projected to the same dimension as the last encoder state.

The way we embed the time signal is also subject to improvement, other implementations have used sinusoidal positional encoding to represent the time step information.



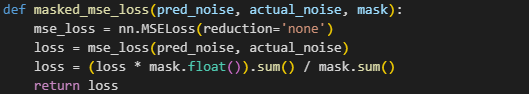
The combination of the last encoder state with the time embedding is then passed to the first decoder layer, the output of which is fed to the Batch Normalization layer, then the last decoder layer outputs the predicted noise with the same shape as the input.



**Loss function:** In Ho et al. (2020), the idea is to learn a distribution that is as similar as possible to the real data distribution. The distance between the learnt distribution and real distribution is given by the KL divergence denoted . It is demonstrated that minimizing means maximizing the log likelihood also referred to as (evidence) which can be done by maximizing the Evidence Lower BOund (ELBO).

In practice, since the model predicts noise, we penalize it using the Mean Squared Error (MSE) between the predicted and true noise, and a minimal MSE should correspond to a maximal ELBO.

One more important detail about the loss is that, our protein sequences come in different lengths, meaning that their backbone structures have different shapes, thus, they have all been padded or cropped to match a fixed maximum length, depending on whether a given input is longer than the maximum length or shorter. What this implies is that the sampled added and predicted noise are also shaped in a padded manner. Thus, an atom mask is calculated (already done in the initial notebook) for each input, to indicate null value and padded positions. As instructed in the notebook, the mask has to be used in the loss calculation to only account for valid positions and penalize the model for errors at such positions.



## II.2. Model training:

In this section, we share details concerning the training process. Details include training logic, hardware used and model parameters.

**Training logic:** The training process consists in iterating the training data over multiple epochs, and at each iteration, a batch of preprocessed protein backbones is loaded. Since the atom mask informs on positions of null values and padding cells, it comes in 2D, and it has to be reshaped to match the shape of the backbones to allow for later multiplication, to that end, each atom mask is converted to a stack of 3 copies of itself.



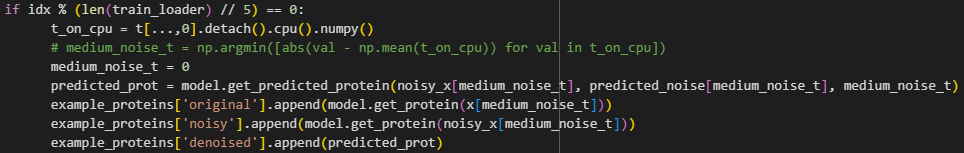
As can be inferred from previous sections, at each iteration, after processing the atom masks, time steps are randomly sampled, where is the batch size, to then apply the forward process on the batch of loaded inputs. The noisy batch is fed to the UNet module to estimate added noise. Next, a loss value is calculated and the gradient is back propagated.







In addition, every time the model sees 20% of the whole dataset, the original, noisy and predicted backbones are saved to later review the model’s learning progress throughout training. That is why shuffling is disabled, to make sure that at each epoch we’re saving data corresponding to the same samples.

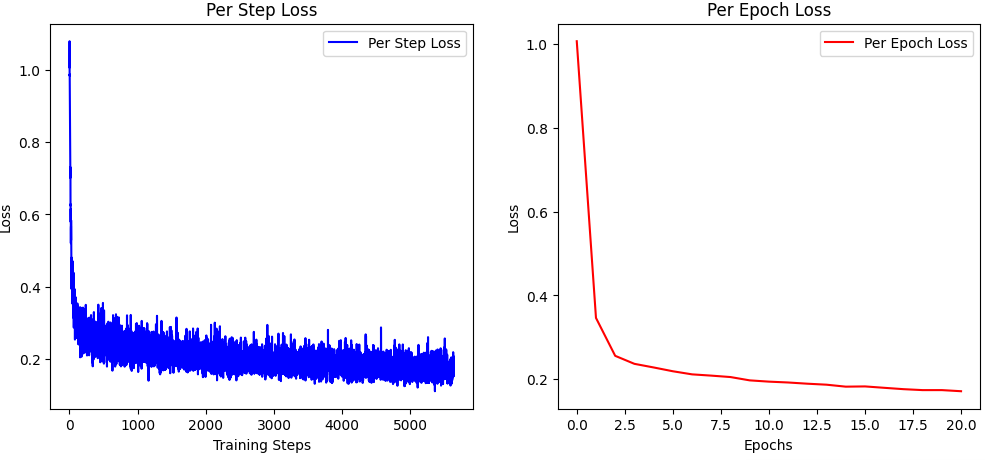


**Model parameters:**

* Batch size: is set to the maximum possible value without exceeding GPU memory limit, respecting the rule. For the final retained model, that maximum batch size was 64.
* Max length: All proteins are processed to match a maximum length of 128 residues. The choice of a rather short length aims to make the task easier, due to the simplicity and size of the trained model.
* Number of epochs: The highest number of epochs we’ve experimented with was 20, as we noticed that with all model configurations and sizes, they converge and stagnate before 20 epochs.
* Learning rate: This depended on the model size and had to be adjusted accordingly, for the last retained model, we achieve a (seemingly) stable training.
* Time embedding dimension: was arbitrarily set to 256, to match the size of the output of the last UNet encoder layer.
* Time steps: We found it common to use 1000 for time steps across different implementations.
* Betas (noise schedule): start value set to , end value set to , as in [Lucidrain’s implementation](https://github.com/lucidrains/denoising-diffusion-pytorch/blob/main/denoising_diffusion_pytorch/denoising_diffusion_pytorch.py)

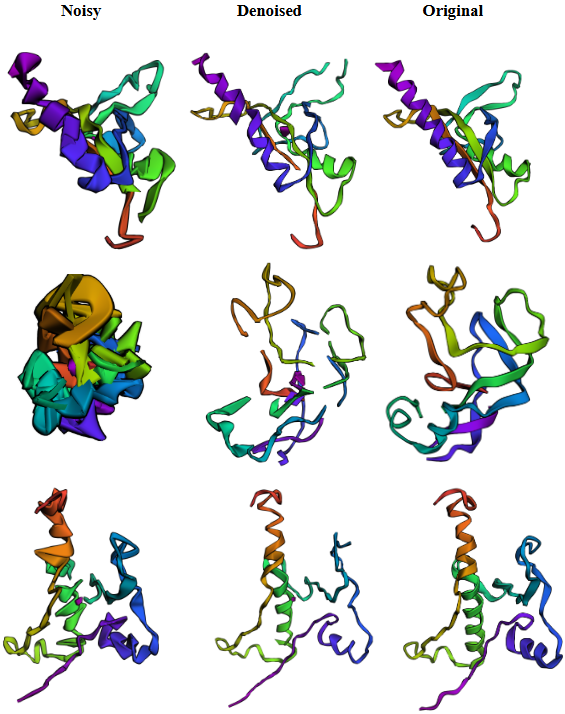
## II.3. Visualization & Evaluation:

**Loss values during training:** For the model’s assessment, the first thing we looked at was the loss values which informs on the model’s stability during training and could indicate if the learning rate should be adjusted. Over 20 epochs, we got a quick decrease to ~0.4 in terms of MSE Loss, then the model slowly kept improving to converge at around ~0.17. During training, we log the loss per iteration (batch loss) and the epoch loss.

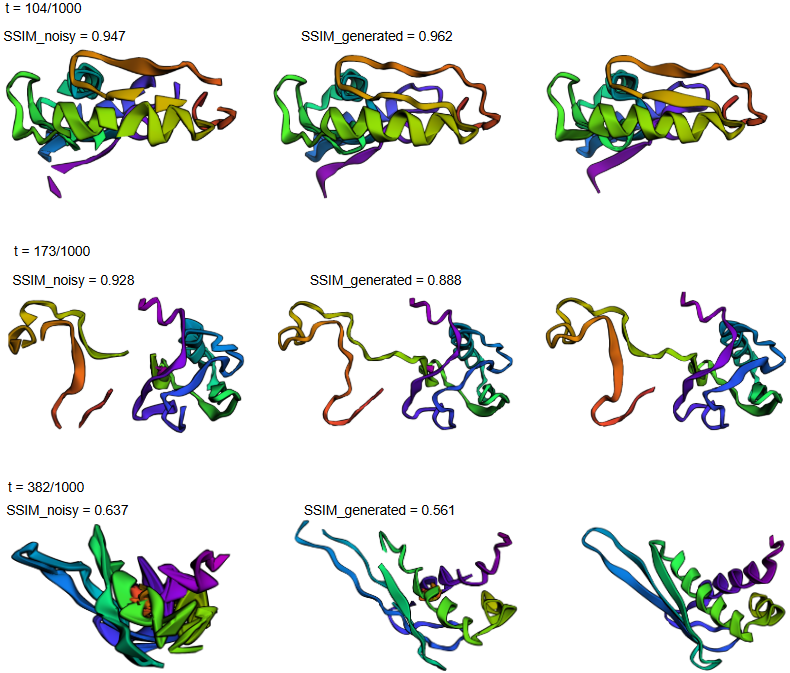


**Visualizing generated backbones:** The next thing is the visual evaluation of the model’s generations. As mentioned in previous sections, during training, we save a few training examples, their noisy versions, the time step and the corresponding model predictions, which are used after training to reconstruct original images based on the noisy image and the predicted noise.

Here we display a few denoising results we saved during training:



**Structural Similarity Index (SSIM):** In order to measure the similarity between a real backbone and its corresponding reconstructed version by removing the predicted noise from the noisy structure), we calculate the SSIM metric, which is typically used to evaluate diffusion models (higher is better). When evaluating the model with the SSIM, we have to take into consideration the time step . When comparing the SSIM obtained on the same sample by the same model, a higher value of this metric doesn’t necessarily mean better performance, higher means better generation only at equal time steps. Here we provide a few generation examples, and for each, we report the time step and the value of SSIM:



**Evaluating sampling (Generation from pure noise):** At this stage, the model tries to gradually build a new structure starting from Gaussian noise. Unfortunately, we weren’t able to get the sampling to work with the current model, when the output of the sampling function is passed to the renderer, it shows complete blank. We think of this to be due to two possible reasons, 1) a mistake in the code, or 2) the model hasn’t been trained well enough to be able to generate a sensible output from pure noise. We previously saw the model’s ability to get noisy inputs closer to the real form, but that is an easier task as it starts from noisy data and not pure noise.

## III. Improvement perspectives and next steps:

As mentioned in the above sections, the aim of this work was to implement a minimal functional framework, thus, it presents much room for improvement on many levels, including model architecture, noising strategy, denoising method and training:

* **Model architecture:** First, it is worth mentioning that this is considered as a baseline in diffusion, and has already been applied in protein structure generation. One way to better model this problem, could be by training a more sophisticated architecture, such as ProteinMPNN in Dauparas et al. (2022), which was used to implement Chroma, a generative model introduced in Ingraham et al. (2023) (Illuminating protein space with a programmable generative model) and trained to generate both the structure and the corresponding sequence simultaneously.  
     
  As for our model, it can benefit from a deeper structure with more parameters and more advanced operations. For example, the noise estimation network can be enhanced by adding residual connections to combine the encoder layers’ outputs with those of the decoder. Also, the integration of attention blocks could help model contextual information and relationships, which has been proven in other works to be very useful. Additionally, the encoder layers can be transformed into ResNet blocks instead of just a succession of convolutional layers and ReLU activations.  
    
  Furthermore, an important asset of the CATH dataset is that it provides the amino acid sequences and corresponding backbones. Knowing that in nature, the structure of a protein and how it folds, strongly depends on the chain of amino acids, it would make sense to investigate the impact of injecting sequence information to the noise estimation network, to move from simple unconditional generation to sequence-informed generation. An easy and straightforward way to achieve this would be to add a pre-trained protein LLM component, that computes sequence embeddings, which can be aggregated into a single representation, projected to the time embedding dimension then injected into the network at the bottleneck level of the UNet.
* **Noising strategy:** As proposed in Ingraham et al. (2023), noising can be done by collapsing all coordinates to positions near the center of mass of the protein, then the model is tasked to figure out where to place back each atom to regain the original structure, one benefit here is that at sampling stage, the model starts from an input that is easier to digest than pure Gaussian noise.
* **Denoising method:** In our work, we’ve used the classical denoising process as proposed in the DDP paper. As a possible way to improve this, we can adopt the denoising process as introduced in Song et al. (2021) Denoising Diffusion Implicit Models (DDIM).
* **Training**: A better training loop can definitely enhance the end model performance. We’ve noticed that the loss drops drastically to the vicinity of 0.4 then slowly stagnates at ~0.17. We haven’t noticed any worrying fluctuations but one way to further smoothen the model’s behavior is by using learning rate scheduling (eg. OneCycleLR). Also, a quick and easy win would be to use a bigger batch size if enough hardware is provided, else gradient accumulation can be applied.

## IV. Conclusion:

In this work, we adopted the DDPM framework to solve the protein backbone structure generation task. To that end, we implemented a minimal version that presents much room for improvement on many aspects as discussed in section 3 “Improvement perspectives and next steps”. We concluded through the evaluation step that our final model is far from being completely reliable, but in many examples, it was able to generate meaningful structures from noisy inputs, meaning that the model had captured patterns and has shown potential to achieve acceptable performance.

## V. Acknowledgments:

Finally, we would like to extend our thanks to the wonderful people who took the time to record and upload class sessions, write blog posts, make explanatory videos to breakdown the complex parts of diffusion models and how they are applied in this field and make available complete implementations of the different models.

* [Phil Wang (aka Lucidrains)](https://github.com/lucidrains)
* [Umar Jamil](https://www.youtube.com/@umarjamilai)
* [Rosetta Commons](https://www.youtube.com/@RosettaCommons)