UNIVERSITY OF COPENHAGEN

BSc in Computer Science

Exploring Discrete Denoising Diffusion Models for Protein Design

A Case Study on Limited Data

Mads Christian Frandsen

Supervised by Wouter Boomsma

External examiner: Jes Frellsen

June 10, 2024

Mads Christian Frandsen
Exploring Discrete Denoising Diffusion Models for Protein Design
BSc in Computer Science, June 10, 2024
Supervisor: Wouter Boomsma
External examiner: Jes Frellsen

University of Copenhagen
Faculty of SCIENCE
Department of Computer Science
Universitetsparken 1
2100 Copenhagen Ø

Preface

This bachelor thesis has been completed as part of a 15 ECTS project at the Department of Computer Science, University of Copenhagen (DIKU). Selecting this particular topic has been motivated by a personal desire to work with deep learning and generative machine learning models.

The intended reader of this paper is a student with basic knowledge of computer science equivalent to a Bachelor's degree. The student should be comfortable with probability theory and statistics, and posses knowledge of machine learning techniques similar to those taught in the course Probabilistic Machine Learning at DIKU. No prior specific knowledge of diffusion models or proteins is assumed.

I want to give my appreciations to professor Wouter Boomsma for supervising me with this project. Without his help this thesis could never have existed as he has been critical for feedback, contributing to the idea generation phase and making sure I stick to the time plan.

Also, big thanks to my mother, Rikke Vang, who assisted me in proofreading this paper even though she does not have the slightest interest in computer science, machine learning or proteins.

I hope you will enjoy reading this thesis and gain valuable information.

Mads Christian Frandsen
June 2024

Abstract

Proteins, composed of sequences of amino acids, are fundamental to numerous biological functions and applications, ranging from therapeutic drugs to industrial enzymes and novel bio-materials. The field of protein engineering aims to create novel protein sequences optimized for specific purposes. Recently, deep generative models, particularly diffusion models, have shown remarkable potential in this area due to their ability to learn complex data patterns and generate novel samples that are functionally diverse.

This thesis investigates the implementation and application of Discrete Denoising Diffusion Models (D3PMs) for protein design. Here we evaluate the performance of D3PMs trained on a single protein alignment, diverging from prior models trained on extensive datasets.

The results indicate that D3PMs can generate high-quality, functionally diverse proteins even with limited data. Notably, the ByteNet model exhibited strong alignment with the original protein contact maps and significant Spearman correlation with experimental deep mutational scanning data, outperforming other tested models. These findings underscore the potential of diffusion models in protein engineering, providing a promising approach for applications where large datasets are not available.

The code and data that has been used as a basis for the results and plots can be accessed at `https://github.com/MadsFrandsen/Exploring-Discrete-Denoising-Diffusion-Models-for-Protein-Design`

Contents

1 Introduction 1

2 Theory 2

2.1 Proteins 2

2.2 Diffusion Models 4

2.2.1 Denoising Diffusion Probabilistic Models (DDPMs) 5

2.2.2 Discrete Denoising Diffusion Probabilistic Models (D3PMs) . . . 9

3 Experiments and Results 19

3.1 Image Generation 20

3.1.1 Experimental Setup 20

3.1.2 Results 21

3.2 Protein Generation 23

3.2.1 Experimental Setup 23

3.2.2 Results 25

4 Discussion 36

5 Bibliography 38

A Sequence Logos 41

1 Introduction

In the field of protein engineering, the main objective is to design and construct novel proteins or modify existing ones to optimize them for specific purposes and applications. These applications cover a wider range all the way from therapeutic drug targets to industrial enzymes or novel bio-materials. Motivated by this, deep generative modelling have recently emerged as powerful tools in this endeavour. Particularly, diffusion models have demonstrated significant potential in protein design.

Inspired by the work presented in the EvoDiff paper [2], this thesis explores the application of discrete diffusion models to protein design. Diffusion models, a class of generative models, have demonstrated exceptional capabilities in creating biologically plausible and functionally diverse proteins by simulating a process of gradual noise addition and removal.

The specific focus of this thesis is to evaluate the performance of discrete diffusion models when trained on a single protein alignment. Previous models, such as those discussed in the EvoDiff paper, were trained on extensive datasets comprising millions of sequences. However, diffusion models are known to require vast amounts of data to achieve optimal performance. This research aims to investigate whether these models can still generate high-quality, functionally diverse proteins when trained on significantly smaller datasets. By doing so, the study seeks to uncover the potential limitations and challenges associated with limited data conditions and to assess the viability of discrete diffusion models in data-constrained environments.

Theory 2

2.1 Proteins

Proteins are complex molecules, present in all living organisms and are directly involved in the chemical processes essential for life [4]. Protein molecules are composed of multiple amino acids linked together by peptide bonds, thereby forming long chains, much as beads are arranged on a string. The linear sequence of amino acids within a protein is considered the *primary structure* of the protein, and can be written as a simple string (e.g. AFGHAAV...), where each letter corresponds to a specific amino acid. Naturally occurring proteins are built from a set of only twenty standard amino acids, each with unique chemical properties that contribute to the protein's overall structure and function.

The primary structure of a protein, i.e. its amino acid sequence, is fundamental to its identity and function [5]. It drives the folding and intramolecular bonding of the linear amino acid chain, which ultimately determines the protein's higher-level structures and its unique three-dimensional shape.

The second level of structure, called *secondary structure*, is made up by local patterns of folding in the chain. Here we distinguish between different patterns where α-helices and β-sheets are by far the most common. Most proteins contain multiple helices and sheets, in addition to other less common patterns, and these patterns are stabilized by hydrogen bonds.

The protein chain folds onto itself to achieve the third level of structure, known as the *tertiary structure*. The tertiary structure of a protein is its overall three-dimensional shape, and it is stabilized by various interactions between the different individual amino acids, such as hydrophobic and hydrophilic interactions. The tertiary structure is essential for the protein's functionality, as it determines the spatial arrangement of amino acids that form the different functional regions of the protein.

Finally, the fourth and last level of structure is the *Quaternary structure*. This structure arises when multiple polypeptide chains, each with their own primary, secondary and tertiary structure, come together to form a larger functional unit.

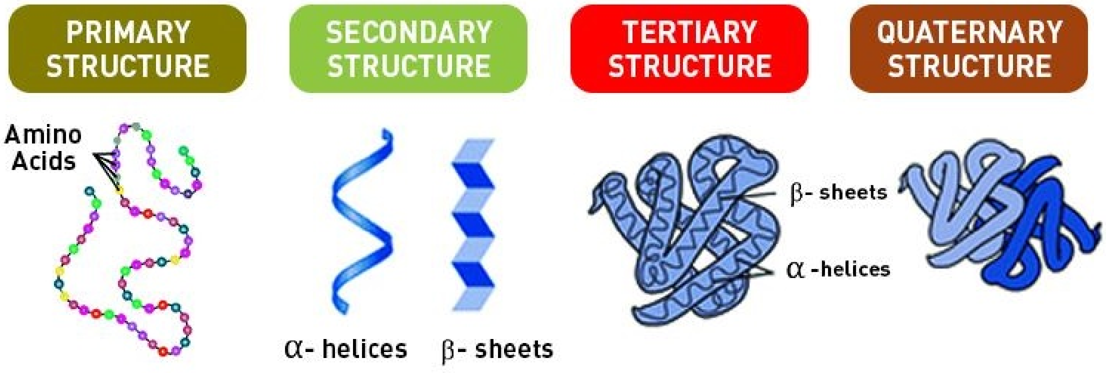


Figure 2.1.: The four levels of protein structure. Image courtesy of [21]

Protein Families and Multiple Sequence Alignments (MSAs)

Proteins are often grouped into families based on their sequence and structural similarities, as well as their functional properties [9]. A protein family is a group of proteins that share common evolutionary origin, reflected by their related functions and similarities in sequence structure. Due to these similarities, we typically see that members of a protein family have similar three-dimensional structures and perform similar biological roles, although they may differ in specific details. Protein families play a crucial role within fields such as protein engineering, where the goal is to develop proteins with specific properties or functions. It is often more efficient to modify an already existing protein with desired properties than to create *de novo* proteins, i.e. new proteins from scratch. By leveraging the natural characteristics of protein families, researchers can more easily adapt and enhance proteins to meet specific needs. This is often done by performing single or multiple mutations—exchanging one or more amino acids in a protein for others—and then testing how these mutations affect the protein's properties.

A collection of three or more similar sequences, such as protein families, are often represented as a *multiple sequence alignment* (MSA). In an MSA, all sequences are adjusted to match the length of the longest sequence, ensuring optimal overlap of amino acids and introducing gaps as needed. This approach provides a clear and organized way to compare protein sequences and serves as valuable input for computational analyses. MSAs are especially of interest when training generative machine learning models, as they provide a structured dataset that captures the evolutionary relationships and conserved regions of protein sequences. This information is crucial for training models to predict protein structures, functions, and to design new proteins with specific properties.

Generative models, such as those based on deep learning, can leverage the patterns and conserved regions identified in MSAs to generate novel protein sequences that maintain the structural and functional characteristics of the protein family. By learning from the variations and conserved motifs within an MSA, these models can predict how changes in the sequence might affect the protein's behavior and stability. This has significant implications for fields such as drug design, where generating proteins with specific binding affinities and activities can lead to the development of new therapeutics.

2.2 Diffusion Models

Diffusion models are originally inspired by non-equilibrium statistical physics, specifically a modelling approach of molecular systems called Langevin dynamics. They were first introduced in [25], and has since emerged as a compelling alternative to traditional generative models such as GANs or VAEs, achieving outstanding results in text-to-image generation [16]. Furthermore, their ability to generate different types of data has been demonstrated across various domains: image [6], text [12], speech [8], music [14], video [27], and time series [23]. This has amongst other things, resulted in popular tools such as Stability AI's Stable Diffusion [1], OpenAI's Sora [19] and DALL-E 3 [18], or Midjourney [13]. More recently, diffusion models have surfaced as a powerful tool for modelling protein sequences [2].

There are multiple variants of diffusion models depending on the type of data one is working with. In this introduction to diffusion models, we consider Denoising Diffusion Probabilistic Models (DDPMs) for image generation [6], as these models have gained a lot of popularity in recent work, and I personally also find them more intuitive to understand. However, the concepts and ideas presented in this section are similar for other modalities. Especially for the discrete case that we will consider afterwards.

2.2.1 Denoising Diffusion Probabilistic Models (DDPMs)

Let's imagine that we take an image and gradually add random noise to each pixel of the image. Eventually after a sufficient number of noising steps, the image itself becomes pure noise and the signal is obliterated. Diffusion models are based on the idea, that we can learn how to reverse this process, i.e. to iteratively recover the initial image from the noisy image. The generative model thus learns how to remove noise, which subsequently allows for new samples to be generated by drawing an image from pure noise and thereafter denoise it.

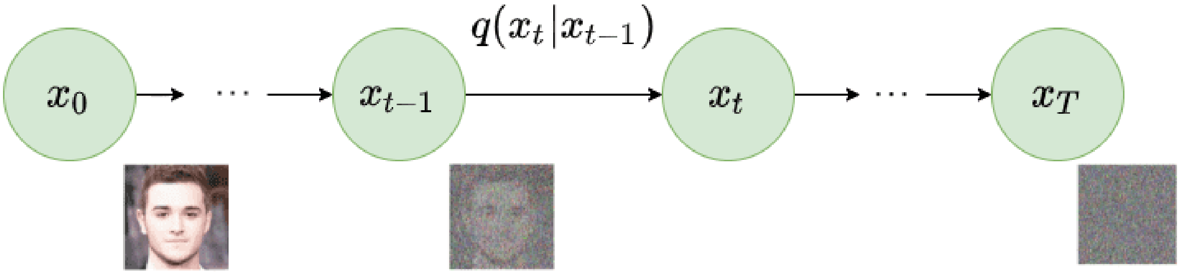


Figure 2.2.: The forward (diffusion) process progressively adds noise to the image. Image courtesy of [6]

In mathematical terms, we call the initial image x_0, and we say that it is obtained from the data distribution $q(x_0)$. The noisy image is then represented by x_T, where T denotes the number of noising steps, also called time-steps, sequentially applied to x_0. All the noising steps $x_1, \ldots, x_T$ are latent variables of the same dimensionality as x_0. We define this noising process as being the *forward process* or the *diffusion process*, with the following distribution:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{2.1}$$

Here we also make the assumption that the forward process is fixed to a Markov chain. A non-Markovian forward process could also have been considered, leading to models such as DDIMs [26], which has the benefit of accelerated sampling.

For DDPMs, the added noise at each step is set to Gaussian, which gives us the following equation for the conditional distribution of the Markov chain:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}) \tag{2.2}$$

where $\beta_1, \ldots, \beta_T \in (0, 1)$ is known as the *variance schedule* or *noise schedule*, and it controls the level of noise added at each step. A key observation here is that if $\beta_1 < \beta_2 < \ldots < \beta_T$, then for $T \to \infty$ the latent variable x_t converges towards an isometric Gaussian random variable:

$$\lim_{t \to \infty} q(x_t | x_0) = \mathcal{N}(0, \mathbf{I}) \tag{2.3}$$

This is a desired property as it enables one to generate samples simply by drawing from a standard Gaussian, since if we pick a value for T that is large enough, then we will have $x_T \sim \mathcal{N}(0, \mathbf{I})$. This also makes sampling similar to how it is done in other generative models such as generative adversarial networks (GANs) or variational autoencoders (VAEs). Various noise schedules have been proposed, among these are the linear [6] and cosine [15] schedules, which are commonly used, and both imply this property. The specific details of these schedules will be covered later.

Importantly, while this specific setup converges to a standard Gaussian, the convergence property can potentially be extended to other distributions as well. By modifying the type of noise and/or the variance schedule, other stationary distributions could be targeted, allowing for flexibility in diffusion models design. This is important for when we consider discrete data instead of continuous data, as we in this scenario no longer will be adding Gaussian noise.

By defining $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} a_s$, the following reparametrization emerges from the forward process:

$$q(x_t | x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t)\mathbf{I}) \tag{2.4}$$

The forward process thus admits sampling x_t at an arbitrary step t in closed form conditioned on x_0. This is quite handy for training the model as we shall later see.

We are now interested in learning the reverse distribution $q(x_{t-1}|x_t)$, because after sampling from $q(x_T) \sim \mathcal{N}(0, \mathbf{I})$, we could just run the process in reverse to obtain a sample from the data distribution $q(x_0)$—or, in other words, a synthetic image. However, we cannot easily compute $q(x_{t-1}|x_t)$ as it requires access to the full data distribution, thus making it intractable. Instead, we aim to approximate it by learning a distribution $p_\theta(x_{0:T})$ called the reverse process:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{2.5}$$

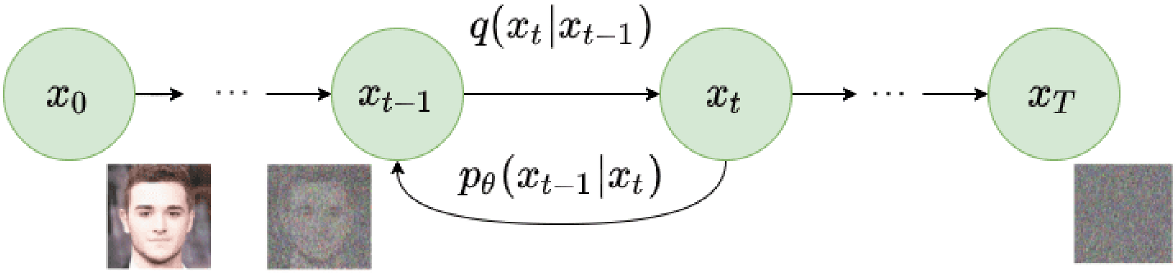


Figure 2.3.: The reverse (inference) process removes the noise to retrieve the initial image. Image courtesy of [6]

Note that similarly to the forward process, the reverse process is also defined as a Markov chain. And specifically for DDPMS, the conditional distribution of the reverse process is defined as:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{2.6}$$

where $\mu_\theta(x_t, t)$ is a neural network and the variance $\Sigma_\theta(x_t, t)$ can either be fixed to a constant or learned. It should however be noted, that [15] showed that learning the variance could be beneficial in reducing the total number of time-steps required for the model to perform well.

The diffusion model's forward process, q, combined with its reverse process, p, conceptually resembles a Variational Autoencoder (VAE) [11] quite well. This leads to the learning objective of optimizing a variational upper bound on the negative log-likelihood, also known as the evidence lower bound (ELBO), for every step of the diffusion process, treating the corrupted data as latent variables:

$$\begin{aligned}
\mathcal{L}_{\text{vb}} &= \sum_{t=0}^{T} \mathcal{L}_{\text{ELBO}} \\
&= \mathbb{E}_{q(x_0)}\left[-\log \frac{p_\theta(x_{0:T})}{q(x_t|x_{t-1})}\right] \\
&= \mathbb{E}_{q(x_0)}\Bigg[\underbrace{D_{\text{KL}}\left[q(x_T|x_0)||p(x_T)\right]}_{L_T} + \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(x_{t-1}|x_0)}\left[D_{\text{KL}}\left[q(x_{t-1}|x_t, x_0)||p(x_{t-1}|x_t)\right]\right]}_{L_{t-1}} \\
&\qquad - \underbrace{\mathbb{E}_{q(x_1|x_0)}\left[\log p_\theta(x_0|x_1)\right]}_{L_0}\Bigg]
\end{aligned} \tag{2.7}$$

where D_{KL} denotes the Kullback-Leibler (**KL**) divergence. Inspecting this loss function, we see that the term L_T can be disregarded since it has no learned parameters thus making it constant during training. Furthermore, when the number of steps T is large enough, we can guarantee that this term will approach zero regardless of the data distribution $q(x_0)$. This is due to the property following from Eq. (2.3), and intuitively it also makes sense, since the accumulation of Gaussian noise will inevitably lead to the convergence toward the standard Gaussian distribution. The term L_{t-1} represents the **KL** divergence between the true posterior distribution of the data at step $t-1$, given both the subsequent noisy observation at step t and the original data point x_0, and the learned approximation of the reverse process at this step. Finally, L_0 is the expected reconstruction error for the first diffusion step. Here it is important to note that even though the reverse distribution $q(x_{t-1}|x_t)$ is intractable, it becomes tractable when conditioned on x_0 due to the Markov property. For the DDPM framework this gives us the following equations:

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t \mathbf{I}) \tag{2.8}$$

$$\text{where} \quad \tilde{\mu}(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} \tag{2.9}$$

$$\text{and} \quad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{2.10}$$

In consequence, all terms (except L_0) of L_{vb} are some **KL** divergence between two Gaussian distributions, thus enabling us to evaluate them with closed form expressions.

During training, we typically apply stochastic gradient descent by considering only one random term of L_{vb}. This is possible due to Eq. (2.4), which allows us to sample x_t at an arbitrary step t. The reader should however note, that in more recent work other loss functions has also been used to optimize diffusion models. We however present L_{vb} here, as it is relevant for diffusion models in discrete state space.

To briefly summarize this introduction on diffusion models, the key takeaways are that diffusion models consists of a forward process $q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$ that corrupts the data $x_0 \sim q(x_0)$ into a sequence of increasingly noisy latent variables $x_{1:T} = x_1, x_2, \ldots, x_T$. The learned reverse Markov process $p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$ gradually denoises the latent variables towards the data distribution. In general, for diffusion models, the distribution $q(x_t|x_{t-1})$ can in theory be arbitrary, however efficient training of p_θ is possible when $q(x_t|x_{t-1})$:

1. Permits efficient sampling of x_t from $q(x_t|x_0)$ for an arbitrary time t, allowing us to randomly sample time-steps and optimize each L_{t-1} term individually with stochastic gradient decent.

2. Has a tractable expression for the forward process posterior $q(x_{t-1}|x_t, x_0)$, which allows us to compute the KL divergences present in the L_{t-1} term.

In essence, the tractability and efficient sampling capabilities of the forward process in diffusion models are critical for the practical training and optimization of these models.

2.2.2 Discrete Denoising Diffusion Probabilistic Models (D3PMs)

We now move from continuous state space to discrete state space, i.e. from continuous data to discrete data. As stated in the previous section, the general idea behind diffusion models are still the same, however with some modifications to the framework in order for it to fit discrete data.

Diffusion models for discrete state spaces were initially first introduced in [25] who considered a diffusion process over binary random variables. This model class was then extended by [7] to categorical random variables with transition matrices characterized by uniform transition probabilities. In this section we will cover discrete denoising diffusion probabilistic models (D3PMs), which will be the model framework that we consider in the rest of this thesis. D3PMs were introduced by [3], and they serve as a more general framework for diffusion with categorical random variables by further extending upon the model class.

The forward process

The idea behind these models is to consider our data as discrete random variables. For example, if we are working with images, then each pixel of the image can be treated as a discrete random variable. Similarly, if we are working with sequences, then each token can be considered a discrete random variable.

We then have that for scalar discrete random variables with K categories $x_t, x_{t-1} \in 1, \ldots, K$ the forward transition probabilities can be represented by transition matrices: $[\boldsymbol{Q}_t]_{ij} = q(x_t = j | x_{t-1} = i)$. This essentially means that the $K \times K$ transition matrix $\boldsymbol{Q}_t$ contains the probabilities of transitioning from any state i to any state j at time-step t.

If we then denote the one-hot version of $x^{(i)}$, i.e., the i-th pixel or token in the data we are considering, with the row vector $\boldsymbol{x}^{(i)}$, we can then write:

$$q(\boldsymbol{x}_t^{(i)} | \boldsymbol{x}_{t-1}^{(i)}) = \text{Cat}(\boldsymbol{x}_t^{(i)}; \boldsymbol{p} = \boldsymbol{x}_{t-1}^{(i)} \boldsymbol{Q}_t) \tag{2.11}$$

where $\text{Cat}(\boldsymbol{x}; \boldsymbol{p})$ is a categorical distribution over the one-hot row vector $\boldsymbol{x}$ with probabilities given by the row vector $\boldsymbol{p}$, and $\boldsymbol{x}_{t-1}\boldsymbol{Q}_t$ is to be understood as a row vector-matrix product. Eq. (2.11) is thus the distribution for a single element of our complete data sample.

We then proceed to make two assumptions; $\boldsymbol{Q}_t$ is applied to each pixel of an image or each token in a sequence independently, and that q factorizes over these higher dimensions as well. Due to these assumptions, we write $q(\boldsymbol{x}_t | \boldsymbol{x}_{t-1})$, i.e. the joint distribution of all pixels or tokens, in terms of a single element, as this distribution is simply just the product of all the individual distributions:

$$q(\boldsymbol{x}_t | \boldsymbol{x}_{t-1}) = \prod_{i=1}^{n} q(\boldsymbol{x}_t^{(i)} | \boldsymbol{x}_{t-1}^{(i)})$$

To simplify notation slightly, we will in the subsequent sections proceed by writing Eq. (2.11) simply as $q(\boldsymbol{x}_t | \boldsymbol{x}_{t-1}) = \text{Cat}(\boldsymbol{x}_t; \boldsymbol{p} = \boldsymbol{x}_{t-1}\boldsymbol{Q}_t)$, but do beware that this is the distribution for a single element that we are referencing.

From the Markov property the t-step marginal can then be defined as the following:

$$q(\boldsymbol{x}_t | \boldsymbol{x}_0) = \text{Cat}(\boldsymbol{x}_t; \boldsymbol{p} = \boldsymbol{x}_0 \overline{\boldsymbol{Q}}_t) \tag{2.12}$$

with $\overline{\boldsymbol{Q}}_t$ defined as the cumulative product; $\overline{\boldsymbol{Q}}_t = \boldsymbol{Q}_1 \boldsymbol{Q}_2 \ldots \boldsymbol{Q}_t$. It should be noted that this allows for easy sampling of x_t at an arbitrary step t, which is one of the desired properties of the forward process.

The reverse process

Now that the forward process has been defined we are then interested in the reverse process, more specifically the forward process posterior $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$, as we need this to be tractable for our loss computation.

In the original paper on D3PMs this posterior distribution is defined as the following:

$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) = \frac{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{x}_0)q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}{q(\boldsymbol{x}_t|\boldsymbol{x}_0)} = \text{Cat}\left(\boldsymbol{x}_{t-1}; \boldsymbol{p} = \frac{\boldsymbol{x}_t\boldsymbol{Q}_t^T \odot \boldsymbol{x}_0\overline{\boldsymbol{Q}}_{t-1}}{\boldsymbol{x}_0\overline{\boldsymbol{Q}}_t\boldsymbol{x}_t^T}\right) \quad (2.13)$$

where $\odot$ represents an element-wise product. However, we will provide a full derivation here, as the result to this equation can otherwise be rather confusing. First we start off with some preliminaries to make things more clear:

- $\boldsymbol{x}_t \in \{0,1\}^k$ (for any t) is a discrete label represented as a one-hot vector. For instance if we are working with image data, $\boldsymbol{x}_t$ would be a single pixel value represented as a one-hot vector.

- $\boldsymbol{Q}_t \in [0,1]^{k\times k}$ is a stochastic matrix whose rows sum to 1, i.e. a transition matrix.

We have to be very careful here with our derivations because some of the expression computed evaluate to vectors (probability distributions over the k classes), and some evaluate to scalars, since they are individual probabilities computed by indexing into a probability distribution. It is also crucial to make it clear when we are referring to the random variables themselves compared to their realisations, i.e. clarifying when they have been observed or conditioned on. Otherwise things will get really confusing. We will thus start by rewriting the first part of Eq. (2.13) such that $\boldsymbol{X}_t$ now denotes the random variable and $\boldsymbol{X}_t = \boldsymbol{x}_t$ its realisation:

$$\begin{aligned} q(\boldsymbol{X}_{t-1}|\boldsymbol{X}_t = \boldsymbol{x}_t, \boldsymbol{X}_0 = \boldsymbol{x}_0) &= \frac{q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{X}_{t-1}, \boldsymbol{X}_0 = \boldsymbol{x}_0)q(\boldsymbol{X}_{t-1}|\boldsymbol{X}_0 = \boldsymbol{x}_0)}{q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{X}_0 = \boldsymbol{x}_0)} && (2.14) \\ &= \frac{q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{X}_{t-1})q(\boldsymbol{X}_{t-1}|\boldsymbol{X}_0 = \boldsymbol{x}_0)}{q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{X}_0 = \boldsymbol{x}_0)} && (2.14.b) \end{aligned}$$

where we make use of the Markov property to get Eq. (2.14.b), since:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{x}_0) = q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$$

We then proceed by defining each individual term in Eq. (2.14.b), starting from left to right in the numerator. From Eq. (2.11) we get the following equation:

$$\underbrace{q(\boldsymbol{X}_t = \boldsymbol{x}_t | \boldsymbol{X}_{t-1} = \boldsymbol{x}_{t-1})}_{1\times 1} = \Big[\underbrace{\boldsymbol{x}_{t-1}}_{1\times k}\underbrace{\boldsymbol{Q}_t}_{k\times k}\Big]\underbrace{\boldsymbol{x}_t^T}_{k\times 1} \tag{2.15}$$

Note that this assumes that we have observed $\boldsymbol{X}_{t-1}$, which we clearly have not since we are trying to compute the conditional distribution over $\boldsymbol{x}_{t-1}$ to begin with. Therefore, we ought to write Eq. (2.15) by enumerating all the possible values $\boldsymbol{x}_{t-1}$ could take, which is simply just the identity matrix $\mathbf{I}_k$. This means that $\boldsymbol{x}_{t-1}$ disappears from Eq. (2.15) and is replaced with the identity matrix:

$$\underbrace{q(\boldsymbol{X}_t = \boldsymbol{x}_t | \boldsymbol{X}_{t-1})}_{1\times k} = \Big[\underbrace{\mathbf{I}_k}_{k\times k}\underbrace{\boldsymbol{Q}_t}_{k\times k}\Big]\underbrace{\boldsymbol{x}_t^T}_{k\times 1} = \boldsymbol{Q}_t\boldsymbol{x}_t^T \tag{2.16}$$

The idea of having a non-observed variable on the conditioning side of the expression may seem a bit weird, however we will elaborate more on this later. Eq (2.16) is now a column vector, and we want to keep things consistent by representing probability distributions as row vectors. We thus abuse notation by redefining Eq. (2.16) so that it is a row vector. This just means transposing the right-hand side of the equation:

$$\underbrace{q(\boldsymbol{X}_t = \boldsymbol{x}_t | \boldsymbol{X}_{t-1})}_{k\times 1} = \Big[\boldsymbol{Q}_t\boldsymbol{x}_t^T\Big]^T = \boldsymbol{x}_t\boldsymbol{Q}_t^T \tag{2.17}$$

For the second term in the numerator of Eq. (2.14.b) we apply Eq. (2.12) and get:

$$\underbrace{q(\boldsymbol{X}_{t-1} | \boldsymbol{X}_0 = \boldsymbol{x}_0)}_{1\times k} = \underbrace{\boldsymbol{x}_0}_{1\times k}\underbrace{\overline{\boldsymbol{Q}}_{t-1}}_{k\times k} \tag{2.18}$$

where $\overline{\boldsymbol{Q}}_{t-1} = \boldsymbol{Q}_1\boldsymbol{Q}_2\ldots\boldsymbol{Q}_{t-1}$. And lastly for the term in the denominator we have:

$$\underbrace{q(\boldsymbol{X}_t = \boldsymbol{x}_t | \boldsymbol{X}_0 = \boldsymbol{x}_0)}_{1\times 1} = \Big[\underbrace{\boldsymbol{x}_0}_{1\times 1}\underbrace{\overline{\boldsymbol{Q}}_t}_{k\times k}\Big]\underbrace{\boldsymbol{x}_t^T}_{k\times 1} \tag{2.19}$$

Putting this all together, we then get:

$$q(\boldsymbol{X}_{t-1} | \boldsymbol{X}_t = \boldsymbol{x}_t, \boldsymbol{X}_0 = \boldsymbol{x}_0) = \mathrm{Cat}\left(\boldsymbol{x}_{t-1}; \frac{\underbrace{\boldsymbol{x}_t\boldsymbol{Q}_t^T}_{\text{vector}} \odot \underbrace{\boldsymbol{x}_0\overline{\boldsymbol{Q}}_{t-1}}_{\text{vector}}}{\underbrace{\boldsymbol{x}_0\overline{\boldsymbol{Q}}_t\boldsymbol{x}_t^T}_{\text{scalar}}}\right) \tag{2.20}$$

which is the equation from the original paper, and we are thus done with the derivation.

Now, in Eq. (2.15) we saw an interesting kind of expression, one where the probability of a particular $\boldsymbol{X}_t$ was being conditioned on a non-observed $\boldsymbol{X}_{t-1}$. Before we elaborate on this, perhaps it is useful to consider all the different possible realisations of $q(\boldsymbol{X}_t|\boldsymbol{X}_{t-1})$:

- $q(\boldsymbol{X}_t|\boldsymbol{X}_{t-1}) \in [0,1]^{k\times k}$: What is the probability distribution over the different values $\boldsymbol{X}_t$ can take, given some unspecified $\boldsymbol{X}_{t-1}$?

- $q(\boldsymbol{X}_t|\boldsymbol{X}_{t-1} = \boldsymbol{x}_{t-1}) \in [0,1]^{1\times k}$: What is the probability distribution over the different values $\boldsymbol{X}_t$ can take, given that we have observed $\boldsymbol{X}_{t-1}$ to be $\boldsymbol{x}_{t-1}$?

- $q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{X}_{t-1} = \boldsymbol{x}_{t-1}) \in [0,1]$: What is the probability of observing $\boldsymbol{X}_t = \boldsymbol{x}_t$, given that we have observed $\boldsymbol{X}_{t-1} = \boldsymbol{x}_{t-1}$?

- $q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{X}_{t-1})$: What is the probability of observing $\boldsymbol{X}_t = \boldsymbol{x}_t$, given that... well, nothing has been observed, so what does this mean?

We know via Eq. (2.15) that $q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{X}_{t-1})$ is a column vector (i.e. a $k \times 1$ matrix) and that its entries encode the following:

$$q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{X}_{t-1}) = \begin{bmatrix} q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{X}_{t-1} = [1,0,...,0]) \\ q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{X}_{t-1} = [0,1,...,0]) \\ \cdots \\ q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{X}_{t-1} = [0,0,...,1]) \end{bmatrix} \tag{2.21}$$

Hence its interpretation is also simple: as a distribution over all possible observed $\boldsymbol{X}_{t-1}$'s, what is the probability of observing $\boldsymbol{X}_t = \boldsymbol{x}_t$? Or, simply consider the j'th element of the vector instead: what is the probability of observing $\boldsymbol{X}_t = \boldsymbol{x}_t$, if we did condition on the observation $\boldsymbol{X}_{t-1}$ was j?

Going back to the forward process posterior, If we make the assumption that the reverse process $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ is also factorized as conditionally independent over the image or sequence elements, similarly to what we did for q, the **KL** divergence between

q and p_θ can be computed by simply summing over all possible values of each random variable:

$$D_{\mathrm{KL}}\left[q(x_{t-1}|x_t,x_0)||p(x_{t-1}|x_t)\right] = \mathbb{E}_{q(x_t|x_{t-1},x_0)}\left[\log\frac{q(x_t|x_{t-1},x_0)}{p_\theta(x_{t-1}|x_t)}\right]$$

$$= \mathbb{E}_{q(x_t|x_{t-1},x_0)}\left[\sum_{i=1}^{n}\left(\log q(\boldsymbol{x}^{(i)}|\boldsymbol{x}_{t-1}^{(i)},\boldsymbol{x}_0^{(i)}) - \log p_\theta(\boldsymbol{x}_{t-1}^{(i)}|\boldsymbol{x}_t^{(i)})\right)\right]$$

$$= \sum_{i=1}^{n}\mathbb{E}_{q(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)},\boldsymbol{x}_0^{(i)})}\left[\log\frac{q(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)},\boldsymbol{x}_0^{(i)})}{p_\theta(\boldsymbol{x}_{t-1}^{(i)}|\boldsymbol{x}_t^{(i)})}\right]$$

This significantly simplifies the computation of the L_{t-1} term in the loss function, as the problem is reduced from high-dimensional joint distributions to low-dimensional individual distributions. We thus satisfy both criteria 1 and 2 discussed at the end of Section 2.2.1.

Markov transition matrices

An advantage of the D3PM framework is the ability to control the data corruption and the denoising process by choosing $\boldsymbol{Q}_t$. This is in notable contrast with continuous diffusion models such as DDPM, for which only additive Gaussian noise has received significant attention. When choosing $\boldsymbol{Q}_t$, there are only two constraints that we have to consider: Rows of $\boldsymbol{Q}_t$ must sum to one to conserve probability mass, and rows of $\overline{\boldsymbol{Q}}_t = \boldsymbol{Q}_1\boldsymbol{Q}_2...\boldsymbol{Q}_t$ must converge to a known stationary distribution when t becomes large.

Many different types of transition matrices can be used within the D3PM framework, however here we will only consider **Uniform** and **Discretized Gaussian** transition matrices:

Uniform: The uniform transition matrix first introduced by [25] for the binary case and, later extended by [7] to the categorical case, can be represented using the following $K \times K$ transition matrix:

$$[\boldsymbol{Q}_t]_{ij} = \begin{cases} 1-\frac{K-1}{K}\beta_t & \text{if} \quad i=j \\ \frac{1}{K}\beta_t & \text{if} \quad i\neq j \end{cases}$$

with $\beta_t \in [0,1]$. It can also be written as $(1-\beta_t)I + \beta_t\mathbb{1}\mathbb{1}^T/K$, where $\mathbb{1}$ is a column vector of all ones. This transition matrix is doubly stochastic with strictly positive entries, hence the stationary distribution is uniform. In this thesis we will refer to models applying this transition matrix as D3PM-uniform due to the transition probability to any other state being uniform.

Discretized Gaussian: For ordinal data we can imitate a continuous space diffusion model by using a discretized, trunctated Gaussian distribution. In order to do so the following $K \times K$ matrix can be used:

$$[\boldsymbol{Q}_t]_{ij} = \begin{cases} \dfrac{\exp\left(-\frac{4|i-j|^2}{(K-1)^2\beta_t}\right)}{\sum_{n=(K-1)}^{K-1} \exp\left(-\frac{4n^2}{(K-1)^2\beta_t}\right)} & \text{if} \quad i \neq j \\ 1 - \sum_{l=0, l\neq i}^{K-1} [\boldsymbol{Q}_t]_{il} & \text{if} \quad i = j \end{cases}$$

An important factor to note about this matrix, is that the diagonal values are assigned to one minus the sum of each row (not including the diagonal entry) thus ensuring that probability mass in conserved. On top of that, due to the normalization of the off-diagonal values over the range $\{-K+1, ..., K-1\}$ the sum of each row, excluding the diagonal entry, is always smaller than 1. This leads to a matrix that is irreducible doubly stochastic, hence the forward process will converge to a uniform stationary distribution. Similar to the continuous Gaussian diffusion model, the parameter β_t influence the variance of the forward process distributions. This transition matrix will transition between more similar states with higher probability, consequently making it well suited for quantized ordinal data such as images.

Noise schedules

Homogeneous to DDPMs, there exists several different strategies for the noise schedule of the forward process. The optimal choice here may depend on factors such as the data or what type of transition matrix is being utilized. For diffusion using the discretized Gaussian transition matrices, the linear schedule has been proven successful [3]. (Bear in mind that a linear schedule for $\boldsymbol{Q}_t$ leads to a non-liner amount of cumulative noise in $\overline{\boldsymbol{Q}}_t$.) On the other hand, for diffusion using uniform transition matrices, we will utilize two different schedules depending on the type of data we are working with. For image data, we follow the steps of the original D3PM paper, and deploy the cosine schedule given by:

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right), \quad s = 0.008$$

Then finally β_t is defined as:

$$\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}$$

Conversely for sequence data, such as proteins, we use a schedule given by $(T-t+1)^{-1}$, which ensures that information is linearly corrupted between x_0 and x_t for all $t < T$. This schedule originates from [25], and it is the same schedule used in the EvoDiff framework [2].

Parameterization of the reverse process

For the parameterization of the reverse process there are two different approaches we can apply. The first approach is to directly predict the logits of $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ using a neural network $\mathrm{nn}_\theta(\boldsymbol{x}_t)$. The second approach, which is slightly more complicated, focuses on using a neural network $\mathrm{nn}_\theta(\boldsymbol{x}_t)$ to predict the logits of a distribution $\tilde{p}_\theta(\tilde{\boldsymbol{x}}_0|\boldsymbol{x}_t)$, which is then combined with $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$ and a summation over one-hot representations of $\boldsymbol{x}_0$ to obtain the following parameterization:

$$p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) \propto \sum_{\tilde{\boldsymbol{x}}_0} q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t|\tilde{\boldsymbol{x}}_0)\tilde{p}_\theta(\tilde{\boldsymbol{x}}_0|\boldsymbol{x}_t) \tag{2.22}$$

where $\tilde{\boldsymbol{x}}_0$ denotes the estimated version of $\boldsymbol{x}_0$. The $\boldsymbol{x}_0$-parameterization, i.e. the second approach, is generally the preferred choice as in practice this parameterization tend to perform the best. Now, Eq. (2.22) can seem somewhat unintuitive due to it including the term $q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t|\tilde{\boldsymbol{x}}_0)$ instead of $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$, hence we will provide a derivation to hopefully make things more clear.

From the Markov property we know that $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) = q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$, however when we derive the reverse of the forward process we need to actually keep it in. In fact, rather than just doing away with $\boldsymbol{x}_0$ completely we instead marginalise it out:

$$\begin{aligned}
q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) &= \frac{q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t)}{q(\boldsymbol{x}_t)} = \frac{\sum_{\boldsymbol{x}_0} q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{x}_0)}{q(\boldsymbol{x}_t)} && (2.23)\\
&= \frac{\sum_{\boldsymbol{x}_0} q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)q(\boldsymbol{x}_0|\boldsymbol{x}_t)q(\boldsymbol{x}_t)}{q(\boldsymbol{x}_t)} && (2.23.b)\\
&= \sum_{\boldsymbol{x}_0} q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)q(\boldsymbol{x}_0|\boldsymbol{x}_t) && (2.23.c)\\
&= \sum_{\boldsymbol{x}_0} \frac{q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t|\boldsymbol{x}_0)q(\boldsymbol{x}_0|\boldsymbol{x}_t)}{q(\boldsymbol{x}_t|\boldsymbol{x}_0)} && (2.23.d)\\
&\propto \sum_{\boldsymbol{x}_0} q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t|\boldsymbol{x}_0)q(\boldsymbol{x}_0|\boldsymbol{x}_t) && (2.23.e)
\end{aligned}$$

Now, this expression is over $q(\boldsymbol{x}_0|\boldsymbol{x}_t)$, which we do not have. What we do however have is our learned reverse process, hence we can just approximate this term with $\tilde{p}_\theta(\tilde{\boldsymbol{x}}_0|\boldsymbol{x}_t)$. Plugging this approximation into Eq. (2.23.e) we thus end up with Eq. (2.22).

With this parameterization, we essentially predict x_0 from our given x_t and then run the forward process q forward to predict the noisier intermediate value. It should however be noted that we drop the denominator because this is hard to compute, and it only acts as a normalization factor, hence it will not affect the end result.

Another thing to note about the x_0-parmeterization is that the **KL** divergence $D_{\text{KL}}\left[q(x_{t-1}|x_t, x_0)||p(x_{t-1}|x_t)\right]$ will be zero if $\tilde{p}_\theta(\tilde{\boldsymbol{x}}_0|\boldsymbol{x}_t)$ places all of its probability mass exactly on the original value x_0, hence the more accurate our prediction of x_0 is, the more the loss will be minimized. The decomposition of $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$ in Eq. (2.13) also provides us with another motivation for using this parameterization. According to this equation, in a given state x_t, the optimal reverse process only takes into account transitions to states for which $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ is non-zero. This intuitively also makes sense, as if we can not make a transition from $\boldsymbol{x}_{t-1}$ to $\boldsymbol{x}_t$, then we should not be able to make this transition the other way, i.e. from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t-1}$. Therefore, the sparsity pattern of $\boldsymbol{Q}_t$ determines the sparsity pattern of the ideal reverse transition probabilities in $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$. The $\boldsymbol{x}_0$-parameterization automatically ensures that the learned reverse probability distribution $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ has the correct sparsity pattern dictated by the choice of the Markov transition matrix $\boldsymbol{Q}_t$.

Loss function, training and sampling

For the learning objective the D3PM framework uses a slightly modified version of L_{vb} from Eq. (2.7). Here an auxiliary denoising objective for the $\boldsymbol{x}_0$-parameterization of the reverse process is introduced, which encourages good predictions of the data $\boldsymbol{x}_0$ at each time-step. This is then combined with the negative variational lower bound, yielding the following loss function:

$$L_\lambda = L_{\text{vb}} + \lambda\, \mathbb{E}_{q(\boldsymbol{x}_0)} \mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)} \left[-\log \tilde{p}_\theta(\boldsymbol{x}_0|\boldsymbol{x}_t)\right] \tag{2.24}$$

where λ is a weighting factor for the auxiliary loss. We see here that the auxiliary loss resembles the cross entropy term L_0 in Eq. (2.7) at time-step $t = 1$. Furthermore, due to the $\boldsymbol{x}_0$-parameterization of $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$, both the auxilary loss term and

$D_{\text{KL}}\left[q(x_T|x_0)||p(x_T)\right]$ in L_{vb} are minimzed exactly when $\tilde{p}_\theta(\tilde{\boldsymbol{x}}_0|\boldsymbol{x}_t)$ has all its mass on the datapoint $\boldsymbol{x}_0$.

To effectively train D3PMs, we employ a systematic approach encapsulated in Algorithm 1. This algorithm iteratively samples data from $q(x_0)$, adds noise through the forward process, and updates the model parameters using gradient descent to minimize the loss function L_λ.

Algorithm 1 Training

```
repeat
    $x_0 \sim q(x_0)$
    $t \sim \text{Uniform}(1, ..., T)$
    $x_t \sim q(x_t|x_0)$                         ▷ We use transition matrices to sample here
    Take gradient decent step on
        $\nabla_\theta \left(L_{\text{vb},t} + \lambda \cdot [-\log \tilde{p}_\theta(\boldsymbol{x}_0|\boldsymbol{x}_t)]\right)$        ▷ Optimize $L_\lambda$.
until converged
```

Once trained, D3PMs utilize Algorithm 2 to generate new samples. This involves sampling an initial noisy state from the prior stationary distribution $p(x_T)$ and iteratively denoising it through the learned reverse process.

Algorithm 2 Sampling

```
$x_T \sim p(x_T)$                    ▷ Sample $x_T$ from the prior stationary distribution
for $t = T, ..., 1$ do
    $x_{t-1} \sim p_\theta(x_{t-1}|x_t)$                 ▷ Sample $x_{t-1}$ using Eq. (2.22)
end for
return $x_0$
```

3 Experiments and Results

In this section, we present the experiments conducted to evaluate the performance of the D3PM models applied to a single MSA. The primary objective of these experiments is to assess the ability of D3PMs to generate high-quality, functionally diverse protein sequences, particularly when trained on limited amounts of data. This is crucial for understanding the potential as well as the limitations of D3PMs in the context of protein engineering, where large datasets may not always be available.

To achieve this, we first implemented a D3PM model on image data using the well-known MNIST dataset. This preliminary step was taken to become familiar with the D3PM framework. We specifically chose image data because it allows for easy evaluation of the model's performance through visual inspection of the generated samples. Additionally, we have previous experience with implementing continuous diffusion models on image data, making this a familiar setting for us.

Building on the insights gained from the image data model, we then implemented a D3PM model on a single MSA. This step aimed to evaluate the model's capability in generating biologically plausible and functionally diverse protein sequences when the data is limited.

In the following sections, we will detail our approach and findings for each of these experiments. We will begin with the implementation and results of the image data model, followed by the implementation and results of the protein sequence model. All models were implemented in Python with the PyTorch machine learning library and trained using Google Colabs A100 GPUs.

3.1 Image Generation

3.1.1 Experimental Setup

For image data, we trained two D3PM models with different transition matrices: one using doubly stochastic uniform transition matrices (D3PM uniform) and another using doubly stochastic discretized Gaussian transition matrices (D3PM Gauss). Similar to the models implemented in the original D3PM paper [3], we for both models use $T = 1000$ time-steps. This ensures that the forward process converges to the stationary distribution within T steps, yielding at most $L_T \approx 10^{-5}$ bits per dimension. Regarding the noise schedules, we use the cosine schedule for the D3PM uniform model and a linear schedule for D3PM Gauss. In the linear schedule we linearly increase β_t from 1×10^{-4} to 0.02.

For the neural network, we followed [3] and opted for a U-Net architecture for both models. This architecture is composed of two distinct parts: an encoder that decreases the resolution and increases the number of channels, and a decoder that does the opposite to retrieve the original shape. The U-Net is based on a Wide ResNet, with weight normalization layers replaced by group normalization layers. The model has a total of four feature map resolutions and two convolutional residual blocks for each resolution level in both the encoder and decoder. At the 8×8 and 4×4 resolution levels, a self-attention block is placed after each residual block, except for the middle block at the final 4×4 level. We incorporate the time-step t into the neural network through a Transformer sinusoidal position embedding in each residual block. A diagram of the network can be seen in Figure 3.1.

As hyperparameters we used a dropout rate of 0.1 in the residual blocks, a learning rate of 2×10^{-5} with the Adam optimizer with standard settings, a batch size of 64 and finally we also applied random horizontal flips as an augmentation during training. When evaluating the model we use an exponential moving average (EMA) for the model parameters with a weight decay factor of 0.9999.

Since we are using a discrete diffusion model, we need to discretize the images. There are several methods to achieve this, but we chose the straightforward approach of treating the image pixel values as integers in the interval [0, 255]. This method is natural because MNIST image pixels by default are represented as grey-scale values.

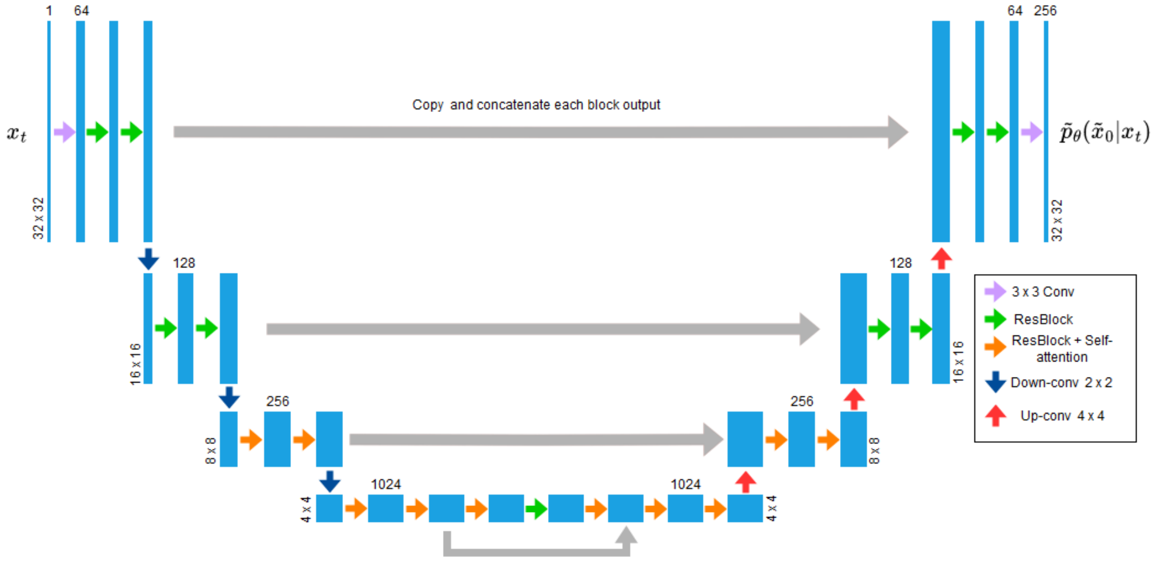


Figure 3.1.: The U-Net architecture of the diffusion models implemented on image data. It processes a grayscale 32×32 image x_t along with a timestep t and outputs the logits for the distribution $\tilde{p}_\theta(\tilde{\boldsymbol{x}}_0|\boldsymbol{x}_t)$.

By discretizing this way, we have a total of $K = 256$ categories, resulting in transition matrices of size 256×256. Additionally, before feeding the images into the neural network, we linearly scale the pixel values to the interval [-1, 1]. This preprocessing step improves training stability. Regarding the training procedure itself we follow Algorithm 1.

3.1.2 Results

From the experiments we found that the model utilizing Gaussian transition matrices performed slightly better according to the negative log-likelihood (NLL) measured in bits per dimension. Table 3.1 shows the NLL comparison of different models. The NLL is estimated on the MNIST test dataset. As we can see from this table, the D3PM models appear to perform slightly worse than the other models. However, we did not spend much time on fine-tuning and optimizing these models, hence there should be a potential to further improve the NLL.

Model	NLL (↓)
Locally Masked PixelCNN	0.65
RNODE	0.97
MintNet	0.98
i-ResNet	1.06
Sliced Iterative Generator	1.34
D3PM uniform $L_{\lambda=0.001}$	≤ 1.96
D3PM Gauss $L_{\lambda=0.001}$	$\leq$1.57

Table 3.1.: Negative Log-Likelihood (NLL) comparison of different models trained on the MNIST dataset. The NLL is estimated using the MNIST test data.

Inspecting the generated images, we can observe that the model's generative capabilities are quite good, although there is still room for improvement. Figure 3.2 shows progressive sampling from the D3PM uniform and D3PM Gauss models. This figure highlights the differences in the denoising process between the two models, dictated by the choice of transition matrices and noise schedules. We can see that the D3PM Uniform model generally produces smoother and more coherent samples earlier in the denoising process compared to the D3PM Gauss model.

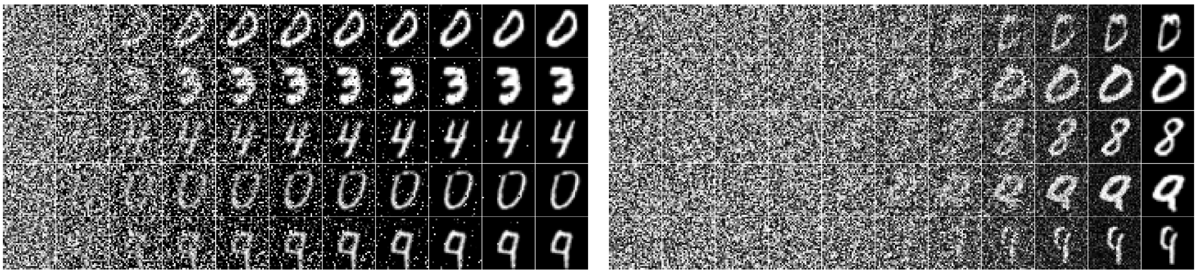


Figure 3.2.: Progressive sampling at $t = 1000, 900, 800, \ldots, 0$ for (left) D3PM uniform and (right) D3PM Gauss trained with $L_{\lambda=0.001}$ loss on MNIST. These samples were not cherry picked.

Figure 3.3 presents non cherry picked samples generated by the two models, further demonstrating their generative capabilities. The samples from both models display a range of quality, with most images somewhat closely resembling the original MNIST digits while others show noticeable deviations. This variability indicates that while both models are capable of generating recognizable digit images, neither consistently outperforms the other based on visual inspection alone.

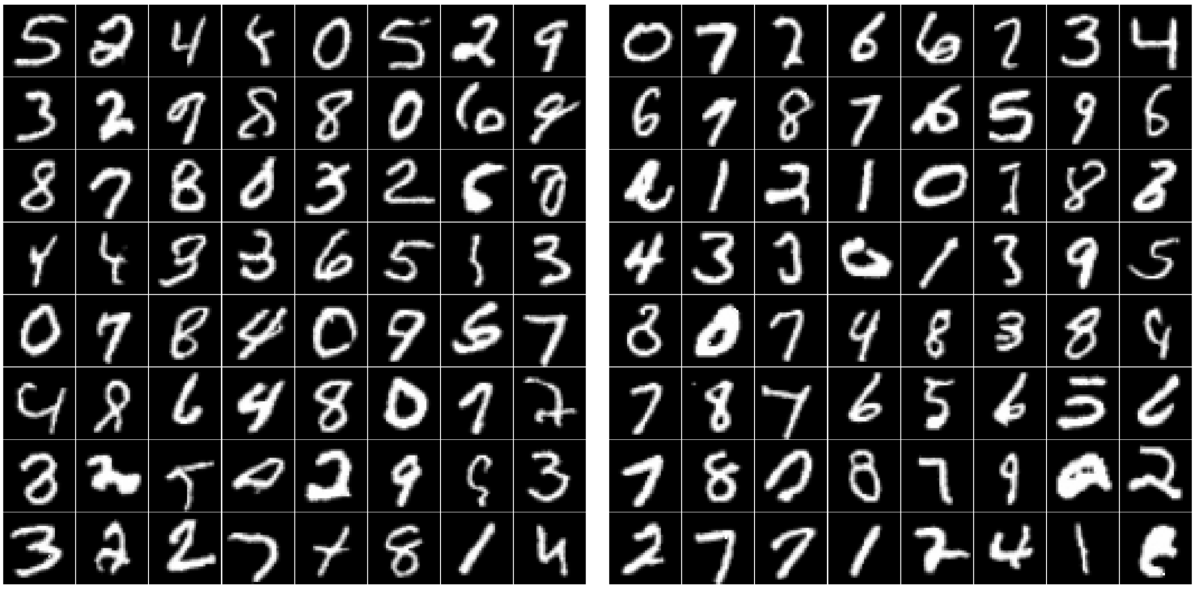


Figure 3.3.: Non cherry picked samples from (left) D3PM uniform and (right) D3PM Gauss

3.2 Protein Generation

3.2.1 Experimental Setup

For protein data, we experimented with training on a single MSA. The alignment used is a variant of BLAT_ECOLX, meaning that the reference sequence is BLAT_ECOLX and all other sequences are aligned to this reference. Generative machine learning models on protein data typically tend to perform well when trained on alignments based on BLAT_ECOLX, hence these alignments are frequently employed as benchmarks for evaluating these models. The MSA contains a total of 14,783 sequences, including the reference sequence, each of length 263.

We train three different D3PM uniform models where we vary the neural network architecture but keep the other model parameters consistent. This iterative process was necessary because the first two models did not perform very well. Uniform transition matrices were chosen following the EvoDiffusion framework, where this is one of the models that they proved successful with. Here we experimented with different values for T and ended up settling for $T = 500$, as this ensured that $L_T \approx 0$ and further increasing the number of steps did not appear to improve performance. For the noise schedule we used a schedule given by $(T - t + 1)^{-1}$, which makes sure that the information is linearly corrupted between x_0 and x_t for all $t < T$.

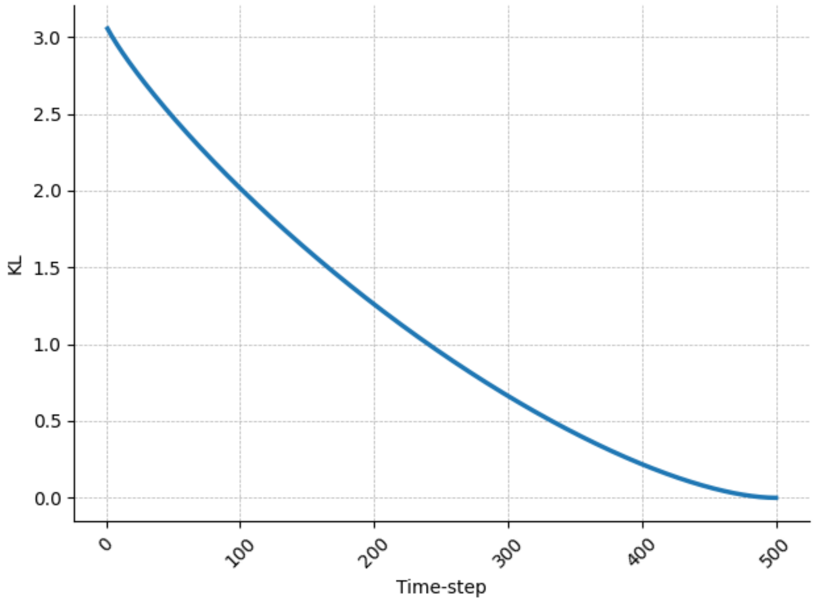


Figure 3.4.: Evolution of $D_{\mathrm{KL}}\left[q(x_T|x_0)||p(x_T)\right]$ as a function of the diffusion time-step t, indicating convergence to a uniform stationary distribution at $t = 500$ as D_{KL} approaches zero.

First, we attempted to follow [2] and apply an MSA-transformer architecture [22]. An MSA-transformer leverages the self-attention mechanism to process MSAs simultaneously, capturing both intra-sequence and inter-sequence relationships by applying attention along the sequences themselves and across the batch of aligned sequences. In total this model had around 114M parameters and included an additional sinusoidal time-step embedding.

Second, we implemented a fully connected neural network, i.e. a Dense model. In this architecture, each neuron in a layer is connected to every neuron in the subsequent layer, allowing for the full flow of information between layers. We applied the architecture typically implemented in Variational Autoencoders (VAEs), consisting of an encoder and a decoder. The layers here consist of linear layers, where in the encoder we downscale the dimensionality of the data and in the decoder we upscale it again. We attempted this architecture specifically because VAEs using these types of neural networks have previously shown great results. Specifically, the model had a total of three layers in both the encoder and decoder with sizes of 2000, 1000, and 300, respectively (in reverse order in the decoder). Additionally, there was a middle latent layer of size 50. An initial embedding layer of dimension 128 was also used to process the input data before it passed through the encoder. Overall, the model comprised roughly 93M parameters.

Lastly, we utilized a ByteNet architecture [10]. A ByteNet is a one-dimensional convolutional neural network designed for sequence-to-sequence tasks. It is composed of two parts, an encoder which encodes the source sequence and a decoder to decode the target sequence. The two network parts are connected by stacking the decoder on top of the encoder. A key feature of a ByteNet is that it utilizes dilated convolutions, which increases its receptive field, allowing it to capture long-term dependencies in sequences. Here we used a dilation rate of 256 and a total of 27 layers. The model had approximately 64M parameters.

Naturally the hyperparameters for the different models vary, however consistent for all of them we used a learning rate of 1×10^{-4} with the Adam optimizer with standard settings, a batch size of 128 for a total of 125.000 steps. The sequences in a batch was randomly sampled from the alignment with uniform probability. Experiments with different ways of preprocessing the data was also conducted. These experiments included removing the columns of the sequences where the reference sequence had lowercase tokens. However, we found that this did not improve the performance of the models, so we opted for minimal preprocessing instead.

After training the models, we generated a total of 14,783 samples for each model, matching the number of samples in our training data. We chose this specific quantity to ensure a one-to-one comparison between the original and generated sequences, facilitating a direct evaluation of the models' performance.

3.2.2 Results

The performance of the D3PM models implemented on protein data was evaluated using three metrics: sequence logos, protein contact maps, and the Spearman correlation coefficient of mutant sequence log-likelihood ratios and deep mutational scanning (DMS) substitutions. These metrics provide a comprehensive view of the models capabilities in generating biologically plausible and functionally diverse protein sequences.

Sequence Logos

Sequence logos are graphical representations that show the conservation and variability of each amino acid at every position in a sequence alignment. A logo consists of stacks of symbols, one stack for each position in the sequence. The overall height of the stack indicates the sequence conservation at that position, while the height of the symbols within the stack indicates the relative frequency of each amino acid at that position. Here, we compare the logos of the original MSA with the generated sequences. Ideally, the generated sequences should show similar conservation and variability of the amino acids, as this would indicate that they preserve key functional and structural regions of the original sequences.

Figure 3.5 displays sequence logos from position 50 to 99 for the original MSA and sequences generated by the three different models. By focusing on this subsection, we aim to ensure that the plots remain clear and readable.

The original sequence logo highlights several highly conserved regions, notably at positions 55, 65, 80, and 95. These tall stacks indicate residues that are crucial for maintaining the functional or structural integrity of the protein. When examining the generated sequences, we observe that all three models effectively replicate these conserved regions, suggesting that the models successfully capture the essential features of the original sequences.

Variability in the original sequence is evident in regions such as positions 55 to 65, where shorter stacks with multiple different amino acids indicate high sequence diversity. Both the ByteNet and Dense models capture this variability effectively, while the MSA-transformer model shows slightly less diversity, potentially indicating a tendency to overfit to conserved regions.

Comparing the performance of the models, the ByteNet and dense models exhibit a closer resemblance to the original sequence logo, maintaining a balance between conservation and variability. The MSA-transformer model, although successful in preserving key features, tends to generate sequences with less variability in regions that are variable in the original.

While we are only examining a subsection of the complete sequence in this analysis, it is important to note that the observed patterns and trends continue consistently across the entire length of the sequences. For a comprehensive view, we refer readers to the appendix of this thesis, where the remaining sections of the logo plots are included for those who are interested in exploring the full sequence logos.

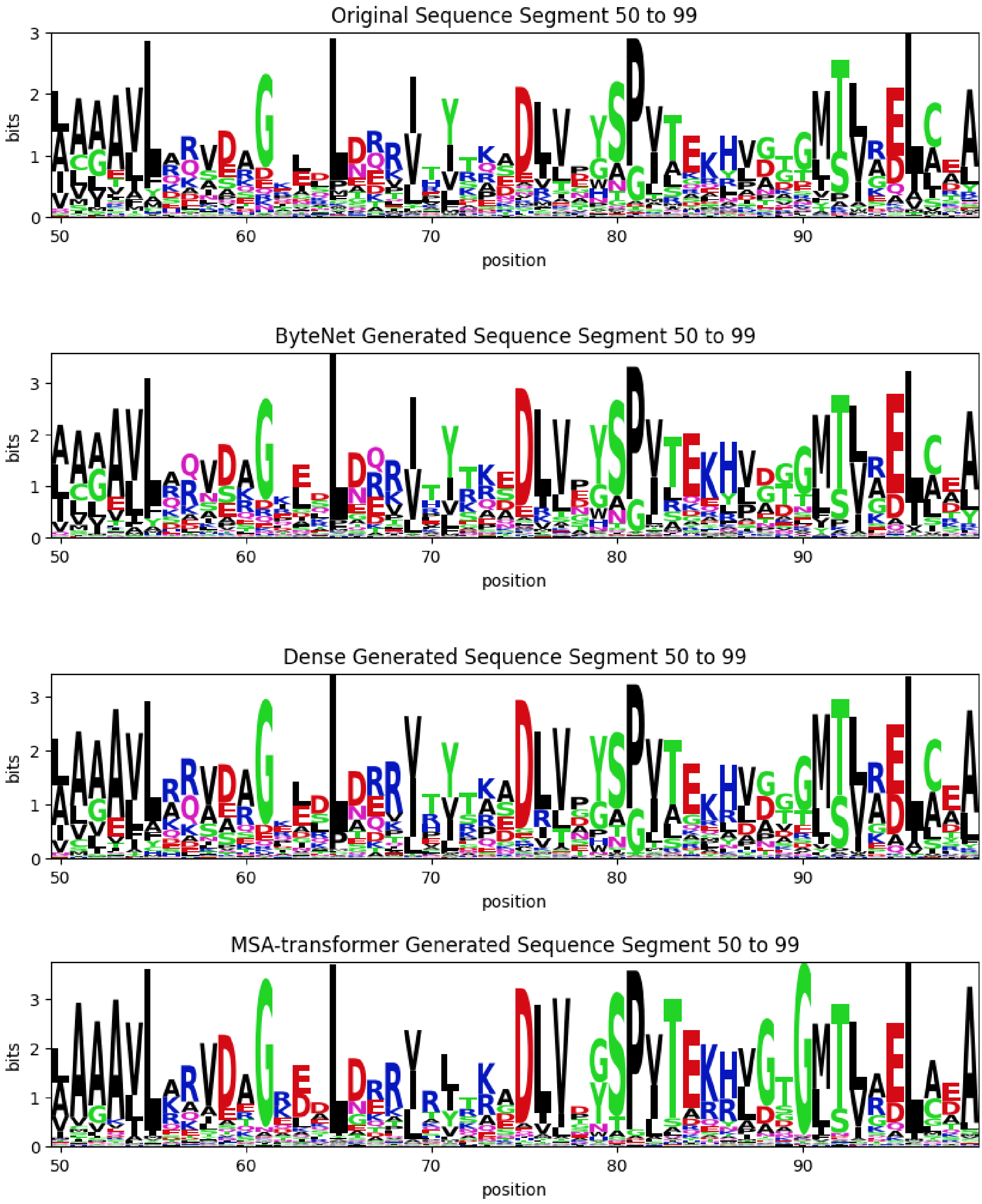


Figure 3.5.: Sequence logos from position 50 to 99 of the original and generated samples from each model. A subsection of the samples is selected in order to make the logos readable.

Contact Maps

Sequence logos are a great tool for visualizing and analysing the primary structure of sequences and identifying conserved residues and motifs. However, they do not capture the three-dimensional dependencies and interactions within proteins, which are crucial for a protein's function and structure. To address this limitation, we supplement our analysis with contact maps.

A contact map is a two-dimensional matrix that represents the proximity of residues in a protein's three-dimensional structure. Each element (i, j) in the matrix indicates whether the amino acids at positions i and j are either in close contact based on some threshold or if there is an evolutionary coupling between them. Evolutionary couplings are pairs of amino acids that show correlated mutations across different species, and they suggest functional and structural interdependencies between residues.

Figure 3.6 displays a contact map of the original MSA we trained on, and it will serve as the reference map in our analysis. In this map, the blue dots represent residues in contact within the molecule (intra-molecular contacts). Orange dots represent residues in contact between the subunits (inter-molecular or multimer contacts). For both blue and orange dots, the contact is determined based on a distance threshold of 5 Angstrom. Finally, the black dots are the evolutionary couplings between residues. The graphics on the bottom and right sides of the contact map show the secondary structure. Arrows correspond to beta strands, and squiggly lines correspond to alpha helices.

The blue diagonal line is expected because naturally there is going to be contact between residues that are close to each other in the primary sequence, e.g., residue i is close to residue $i + 1$. Clusters of blue dots off the diagonal line indicate long-range interactions between residues that are distant in the primary sequence but close in the 3D structure. These interactions are important for a protein's overall fold and stability. The map highlights the 247 most significant and likely accurate couplings. The size of each black dot reflects the corresponding coupling's significance, with larger dots indicating a higher significance.

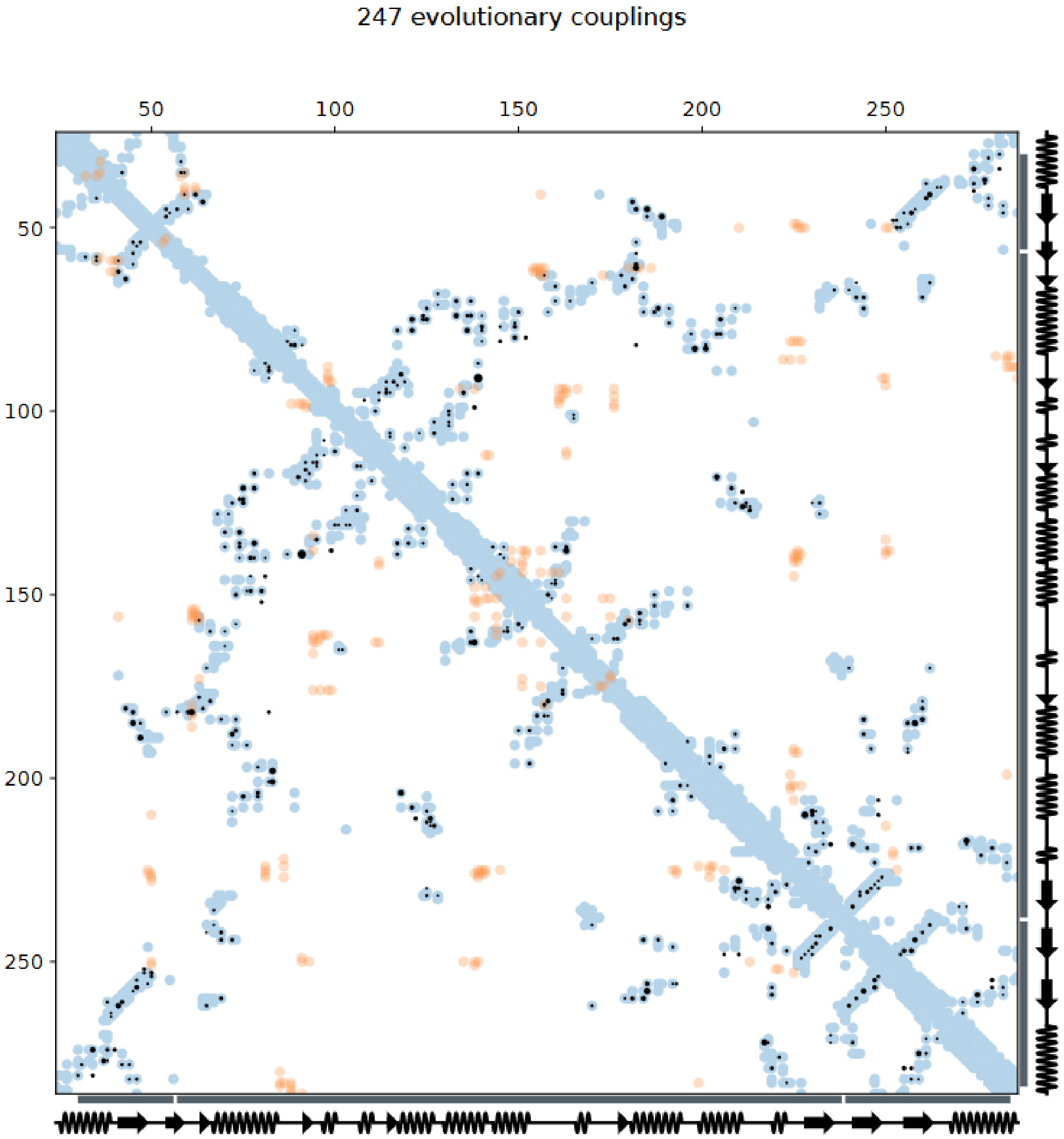


Figure 3.6.: Contact map of the original MSA used for training. This map shows the 247 most significant and likely accurate predicted couplings.

Figures 3.7, 3.8, and 3.9 show contact maps of generated sequences from the ByteNet model, the Dense model, and the MSA-transformer model respectively. Similar to 3.6, each plot contains the 247 most significant couplings.

Comparing these maps to 3.6, we find that the ByteNet model's contact map demonstrates a fairly strong overall correspondence with the original map. It captures many of the couplings along the diagonal as well as several off-diagonal regions, indicating that the model effectively preserves sequential and some long-range interactions. However, there are still regions where many couplings are missing. We also observe two faint lines of couplings running parallel to the diagonal. In earlier versions of the model with lower dilation rates and fewer layers, these parallel lines were way more prominent. It can thus be assumed that the parallel lines indicate potential artifacts due to the model architecture. By further increasing the complexity of the ByteNet model, we could potentially eliminate these artifacts entirely.

From Figure 3.8, we note that this contact map shows a lack of correspondence with the original map. The couplings appear to be scattered all over the place without a discernible pattern. This indicates that the Dense model struggles to capture the true structural and evolutionary relationships present in the original MSA, even though the sequence logo produced by this model looked quite nice.

Figure 3.9 is rather fascinating. Here we see the couplings take an interesting pattern that resembles nothing like the original contact map. This pattern is most likely caused by some sort of artifact related to the model. Transformer architectures are known for requiring relatively large amounts of training data in order to be able to learn. Since our training dataset only contains 14,783 sequences, this could potentially be the cause of this issue. The MSA-transformer model was also the worst performing model according to the sequence logos.

From these contact maps, it thus seems that it is only the ByteNet model that effectively captures the three-dimensional structural interactions and evolutionary couplings present in the original sequences. Despite the presence of some architectural artifacts, the ByteNet model manages to maintain a robust co-evolutionary signal and preserve the majority of both sequential and long-range contacts to a satisfactory extent.

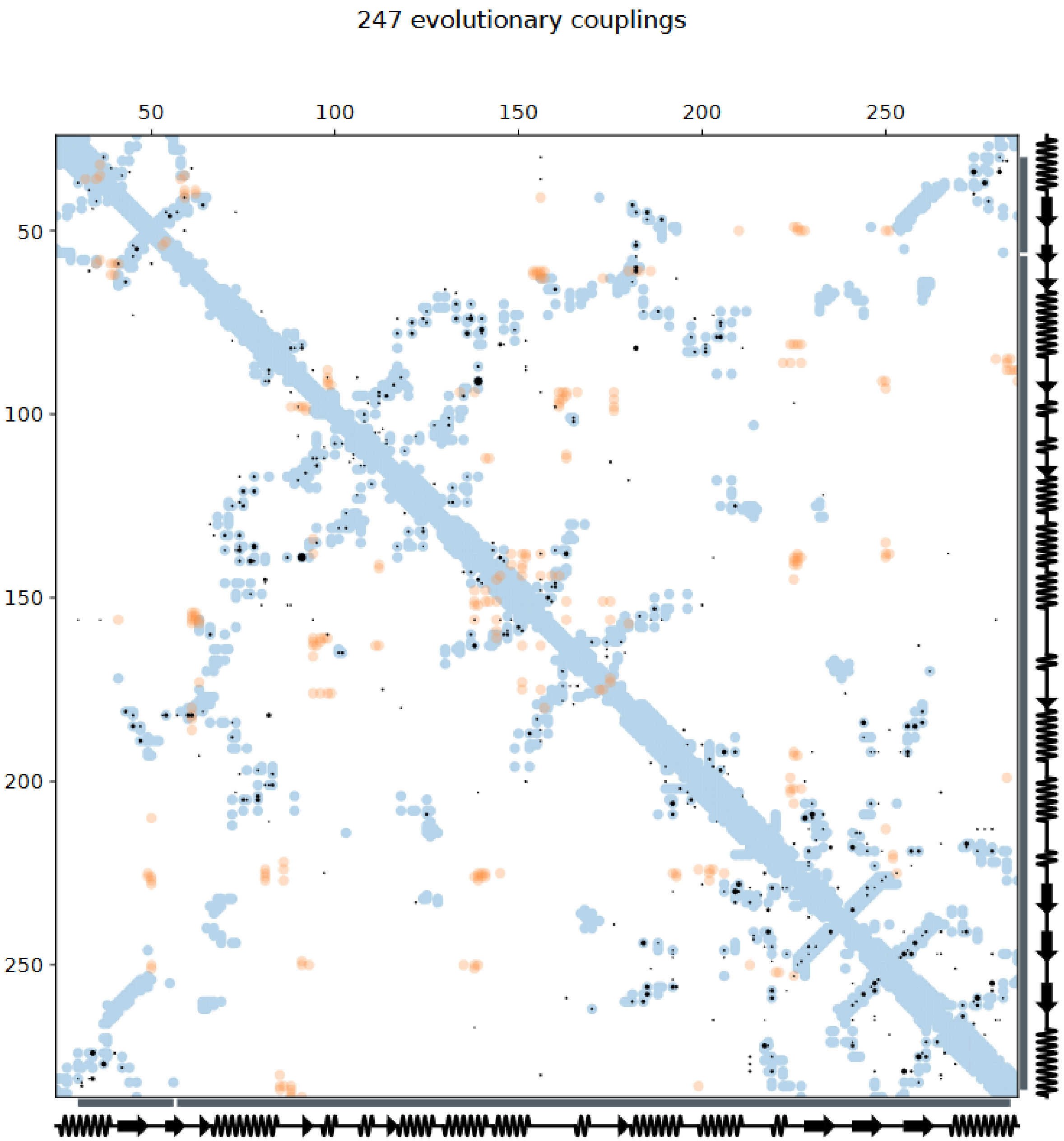


Figure 3.7.: Contact map of ByteNet generated sequences. This map shows the 247 most significant and likely accurate predicted couplings.

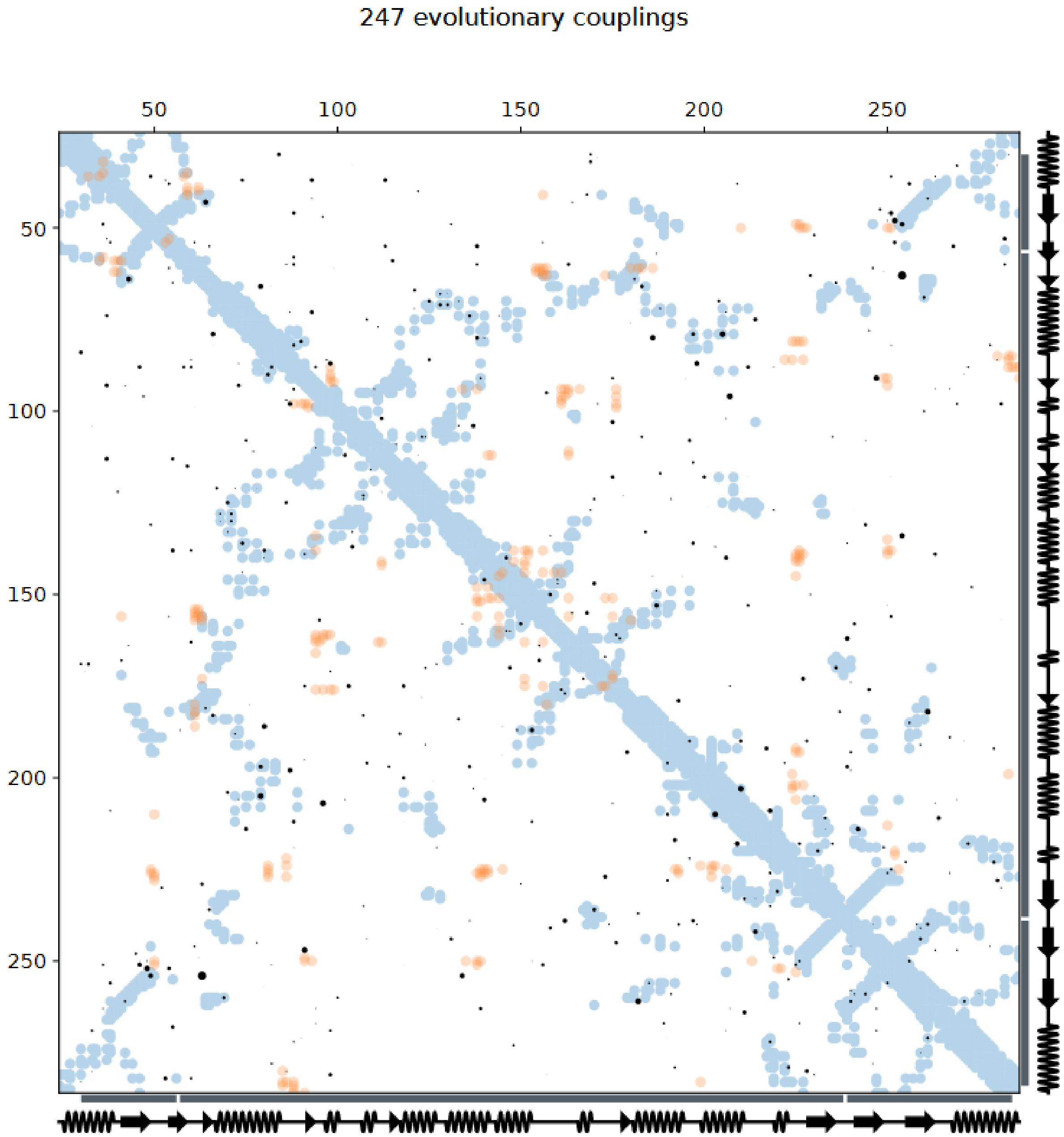


Figure 3.8.: Contact map of Dense generated sequences. This map shows the 247 most significant and likely accurate predicted couplings.

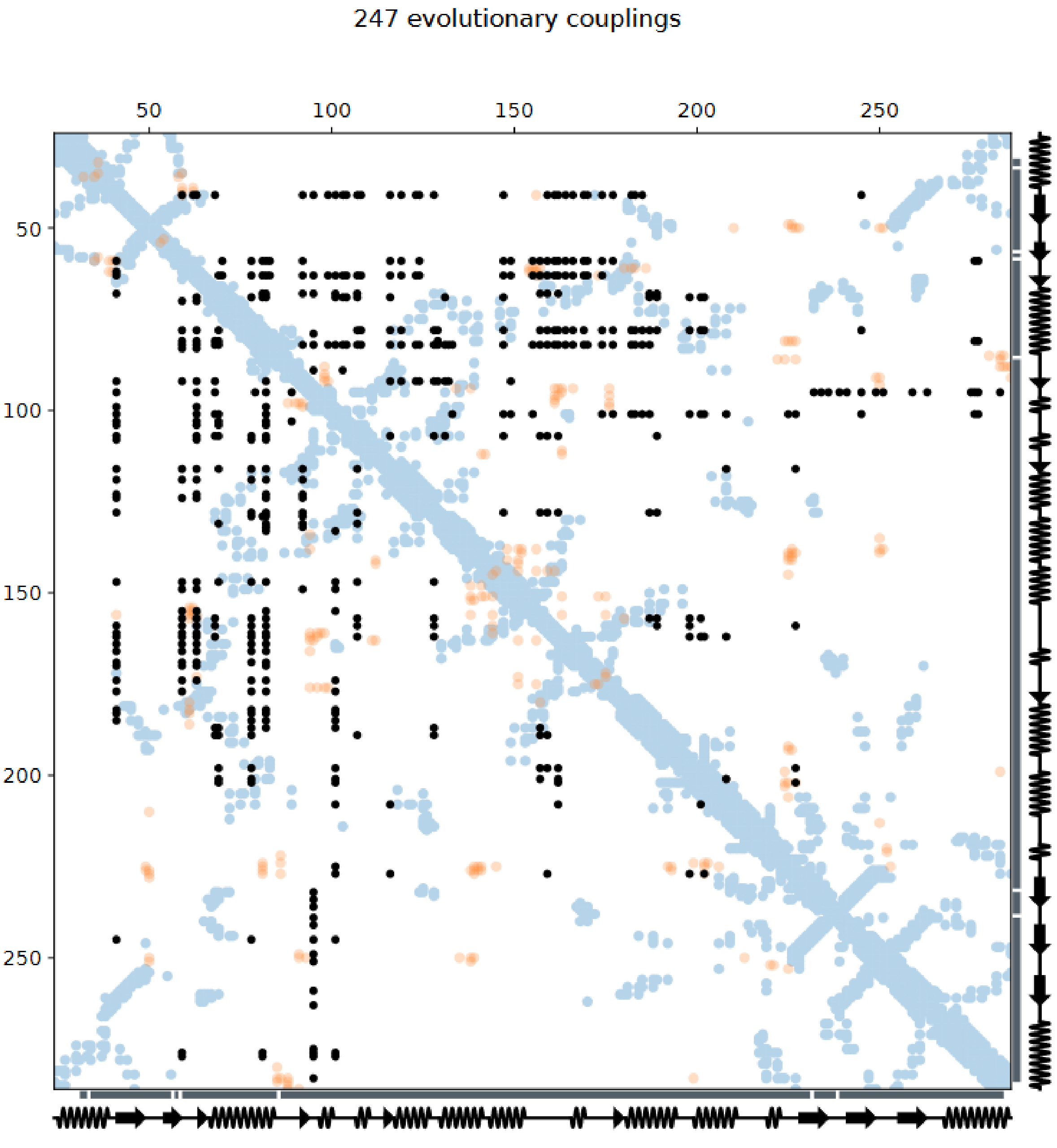


Figure 3.9.: Contact map of MSA-transformer generated sequences. This map shows the 247 most significant and likely accurate predicted couplings.

Spearman correlation coefficient

As a final metric in our evaluation, we use the Spearman correlation coefficient to assess the agreement between the predicted effects of mutations by our models and experimental deep mutational scanning (DMS) data. This method follows the approach by [24] who used the log-ratio, $\log \frac{p(\mathbf{x}^{(\text{Mutant}}|\boldsymbol{\theta})}{p(\mathbf{x}^{(\text{Wild-Type})}|\boldsymbol{\theta})}$ as a heuristic metric for the relative favorability of mutated sequence, $\mathbf{x}^{(\text{Mutant})}$, versus a wild-type $\mathbf{x}^{(\text{Wild-Type})}$.

The log-likelihood ratio serves as a measure of how likely a mutation is to preserve the protein's functions. A positive ratio indicates that the mutant sequence is predicted to be more favorable than the wild-type, whereas a negative ratio suggests the opposite.

Given that direct computation of the log probabilities is intractable for complex models such as diffusion models, we instead choose to approximate these probabilities by replacing each log probability with a lower bound, i.e., the ELBO. To get effective approximations, we use importance sampling, where for each mutation and the wild-type, we take the average of 500 ELBO samples. This should ensure that our approximation is somewhere in the right ballpark.

To validate our models' predictions, we compare the log-likelihood ratios with experimental DMS substitution scores. These scores quantify the observed effects of mutations on protein function, thus providing a benchmark for our predictions. We follow [17] and use the Spearman correlation coefficient, ρ, to measure the rank correlation between the predicted and experimental mutation effects. All of the DMS substitution scores used in this thesis can be found here [20].

Table 3.2 shows the estimated Spearman's rank correlation coefficient between our model scores and experimental measurements. We observe here that the ByteNet model significantly outperforms the other two models, which also aligns with what we expected from the contact maps.

Model	Spearman (↑)
ByteNet	0.263
Dense	0.133
MSA-Transformer	0.074

Table 3.2.: Zero-shot substituion DMS benchmark. Spearman's rank correlation coefficient between our model scores and experimental measurements from BLAT_ECOLX_Stiffler_2015.

Model	Spearman (↑)
SaProt (35M)	0.593
EVmutation	0.707
DeepSequence (single)	0.732
EVE (single)	0.711
Wavenet	0.655
ProtGPT2	0.158
Unirep	0.115
ESM2 (35M)	0.557
ByteNet	0.263

Table 3.3.: Zero-shot substituion DMS benchmark. Spearman's rank correlation coefficient between model scores and experimental measurements from BLAT_ECOLX_Stiffler_2015.

Table 3.3 displays a comparison between our best model, i.e., the ByteNet model, and other protein models. We note from this figure that the discrete diffusion model tends to be on the lower performing end according to these benchmarks. The model is able to beat some of the other generative protein models; however, it does lack quite a bit when compared to the more competitive models. Tables 3.4a and 3.4b show the Spearman correlation for two other DMS assays. Here we see a similar story; however, interestingly enough, the model appears to be slightly more competitive on the BLAT_ECOLX_Deng_2012 assay.

Model	Spearman (↑)
SaProt (35M)	0.406
EVmutation	0.504
DeepSequence (single)	0.508
EVE (single)	0.507
Wavenet	0.445
ProtGPT2	0.107
Unirep	0.075
ESM2 (35M)	0.354
ByteNet	0.231

(a) BLAT_ECOLX_Deng_2012

Model	Spearman (↑)
SaProt (35M)	0.605
EVmutation	0.708
DeepSequence (single)	0.732
EVE (single)	0.709
Wavenet	0.662
ProtGPT2	0.186
Unirep	0.134
ESM2 (35M)	0.555
ByteNet	0.275

(b) BLAT_ECOLX_Firnberg_2014

Table 3.4.: Zero-shot substitution DMS benchmark. Spearman's rank correlation coefficient between model scores and experimental measurements.

4 Discussion

We explored the potential of Discrete Denoising Diffusion Probabilistic Models (D3PMs) in generating high-quality, functionally diverse protein sequences, even when trained on limited data. Our approach involved an iterative process of training three different D3PM models—ByteNet, Dense, and MSA-transformer—on a single MSA to uncover their capabilities and limitations in the context of protein engineering. The results of the experiments conducted have briefly been summarized in Table 4.1.

Experiment	Metric	ByteNet	Dense	MSA-transformer
Protein Generation	Spearman Correlation (ρ)	0.263	0.133	0.074
Sequence Logo Conservation	-	High	High	Moderate
Contact Map Accuracy	-	High	Low	Artifacted

Table 4.1.: Summary of the results from our experiments. These experiments included Sequence Logos, Contact Maps and Spearman Correlation between log-likelihood ratios and experimental results

From the sequence logos, we observed that both the ByteNet and Dense models effectively conserved key positions and captured the conservation patterns of the original sequences. This indicates that these models can maintain essential functional and structural features, which is crucial for generating biologically relevant proteins. The MSA-transformer model, while successful in preserving key positions, showed less variability, potentially due to overfitting to conserved regions. This suggests that the MSA-transformer model may require larger datasets to generalize effectively.

The contact maps further revealed the models' ability to capture three-dimensional interactions within the protein structures. The ByteNet model demonstrated a strong correspondence with the original contact map, preserving many sequential and long-range interactions. This performance highlights the model's robustness in maintaining structural integrity, which is vital for functional protein design. In contrast, the Dense model's contact map showed scattered couplings, indicating difficulties in capturing true structural relationships. The MSA-transformer model exhibited artifacts likely due to the model architecture or insufficient training data, reinforcing the potential need for larger datasets for transformer-based models.

This trend of the ByteNet model outperforming the other two models was also seen in the Spearman correlation coefficient between the predicted effects of mutations by our models and experimental DMS data. Here we saw that the ByteNet model achieved the highest correlation, which aligns well with the findings from the sequence logos and contact maps.

While the results are promising, they also highlight areas for improvement. The ByteNet model, despite its strong performance, exhibited a slight hint of architectural artifacts that could potentially be mitigated by further increasing model complexity. Furthermore, even though the model managed to beat some of the other protein models according to the Spearman correlation coefficient, it still lagged behind compared to the more competitive models in the field. This indicates that, while discrete diffusion models are viable, there is considerable room for enhancement to match or exceed the performance of leading models.

In conclusion, this study demonstrates that D3PMs can generate high-quality protein sequences from limited data, with the ByteNet model standing out in terms of overall performance. These findings provide a foundation for further research into the use of diffusion models in protein engineering, particularly in scenarios where data availability is a constraint. Future work could explore optimizing model architectures and training strategies to enhance performance further and generalize to a broader range of protein sequences. It would be interesting to further explore the models' generative capabilities by incorporating other advanced computational tools and experimental validation techniques. For instance, leveraging algorithms like AlphaFold for predicting the three-dimensional structures of the generated proteins could provide deeper insights into their structural integrity and potential functional domains.

Bibliography

5

[1] Stability AI. *Stable Diffusion*. URL: https://stablediffusionweb.com/.

[2] Sarah Alamdari, Nitya Thakkar, Rianne van den Berg, Alex X. Lu, Nicolo Fusi, Ava P. Amini, and Kevin K. Yang. „Protein generation with evolutionary diffusion: sequence is all you need“. In: *bioRxiv* (2023). eprint: https://www.biorxiv.org/content/early/2023/09/12/2023.09.11.556673.full.pdf.

[3] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. *Structured Denoising Diffusion Models in Discrete State-Spaces*. 2023. arXiv: 2107.03006 [cs.LG].

[4] Encyclopaedia Britannica. *Protein*. URL: https://www.britannica.com/science/protein.

[5] Nature Education. *Protein Structure*. URL: https://www.nature.com/scitable/topicpage/protein-structure-14122136/.

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG].

[7] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. *Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions*. 2021. arXiv: 2102.05379 [stat.ML].

[8] Rongjie Huang, Max W. Y. Lam, Jun Wang, Dan Su, Dong Yu, Yi Ren, and Zhou Zhao. *FastDiff: A Fast Conditional Diffusion Model for High-Quality Speech Synthesis*. 2022. arXiv: 2204.09934 [eess.AS].

[9] EMBL’s European Bioinformatics Institute. *What are protein families?* URL: https://www.ebi.ac.uk/training/online/courses/protein-classification-intro-ebi-resources/protein-classification/what-are-protein-families/.

[10] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. *Neural Machine Translation in Linear Time*. 2017. arXiv: 1610.10099 [cs.CL].

[11] Diederik P. Kingma and Max Welling. „An Introduction to Variational Autoencoders“. In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[12] Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. *Diffusion-LM Improves Controllable Text Generation*. 2022. arXiv: 2205.14217 [cs.CL].

[13] Midjourney. *Midjourney*. URL: https://www.midjourney.com/.

[14] Gautam Mittal, Jesse Engel, Curtis Hawthorne, and Ian Simon. *Symbolic Music Generation with Diffusion Models*. 2021. arXiv: 2103.16091 [cs.SD].

[15] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. 2021. arXiv: 2102.09672 [cs.LG].

[16] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. *GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models*. 2022. arXiv: 2112.10741 [cs.CV].

[17] Pascal Notin, Aaron Kollasch, Daniel Ritter, *et al.* „ProteinGym: Large-Scale Benchmarks for Protein Fitness Prediction and Design“. In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. Vol. 36. Curran Associates, Inc., 2023, pp. 64331–64379.

[18] OpenAI. *DALL-E 3*. URL: https://openai.com/dall-e-3.

[19] OpenAI. *Sora*. URL: https://openai.com/sora.

[20] ProteinGym. *ProteinGym*. URL: https://proteingym.org/.

[21] *Proteins: Structure And Functions*. https://byjus.com/biology/proteins-structure-and-functions/. Accessed: 2024-05-12.

[22] Roshan Rao, Jason Liu, Robert Verkuil, Joshua Meier, John F. Canny, Pieter Abbeel, Tom Sercu, and Alexander Rives. „MSA Transformer“. In: *bioRxiv* (2021). eprint: https://www.biorxiv.org/content/early/2021/02/13/2021.02.12.430858.full.pdf.

[23] Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. *Autoregressive Denoising Diffusion Models for Multivariate Probabilistic Time Series Forecasting*. 2021. arXiv: 2101.12072 [cs.LG].

[24] Adam J. Riesselman, John B. Ingraham, and Debora S. Marks. „Deep generative models of genetic variation capture mutation effects“. In: *bioRxiv* (2017). eprint: `https://www.biorxiv.org/content/early/2017/12/18/235655.1.full.pdf`.

[25] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015. arXiv: `1503.03585 [cs.LG]`.

[26] Jiaming Song, Chenlin Meng, and Stefano Ermon. *Denoising Diffusion Implicit Models*. 2022. arXiv: `2010.02502 [cs.LG]`.

[27] Ruihan Yang, Prakhar Srivastava, and Stephan Mandt. *Diffusion Probabilistic Modeling for Video Generation*. 2022. arXiv: `2203.09481 [cs.CV]`.

A

Sequence Logos

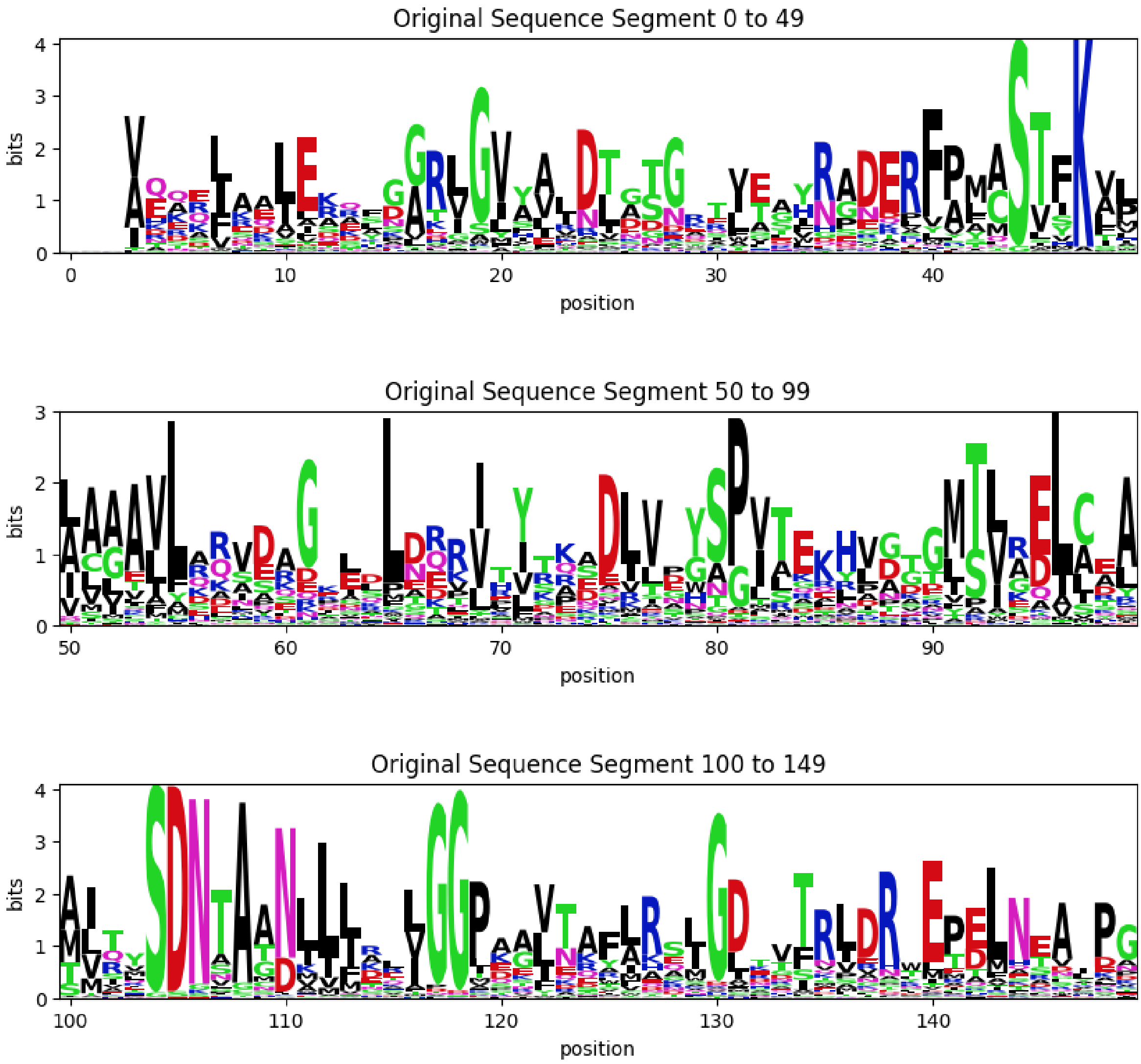


Figure A.1.: Sequence logos from position 0 to 149 of the original sequences

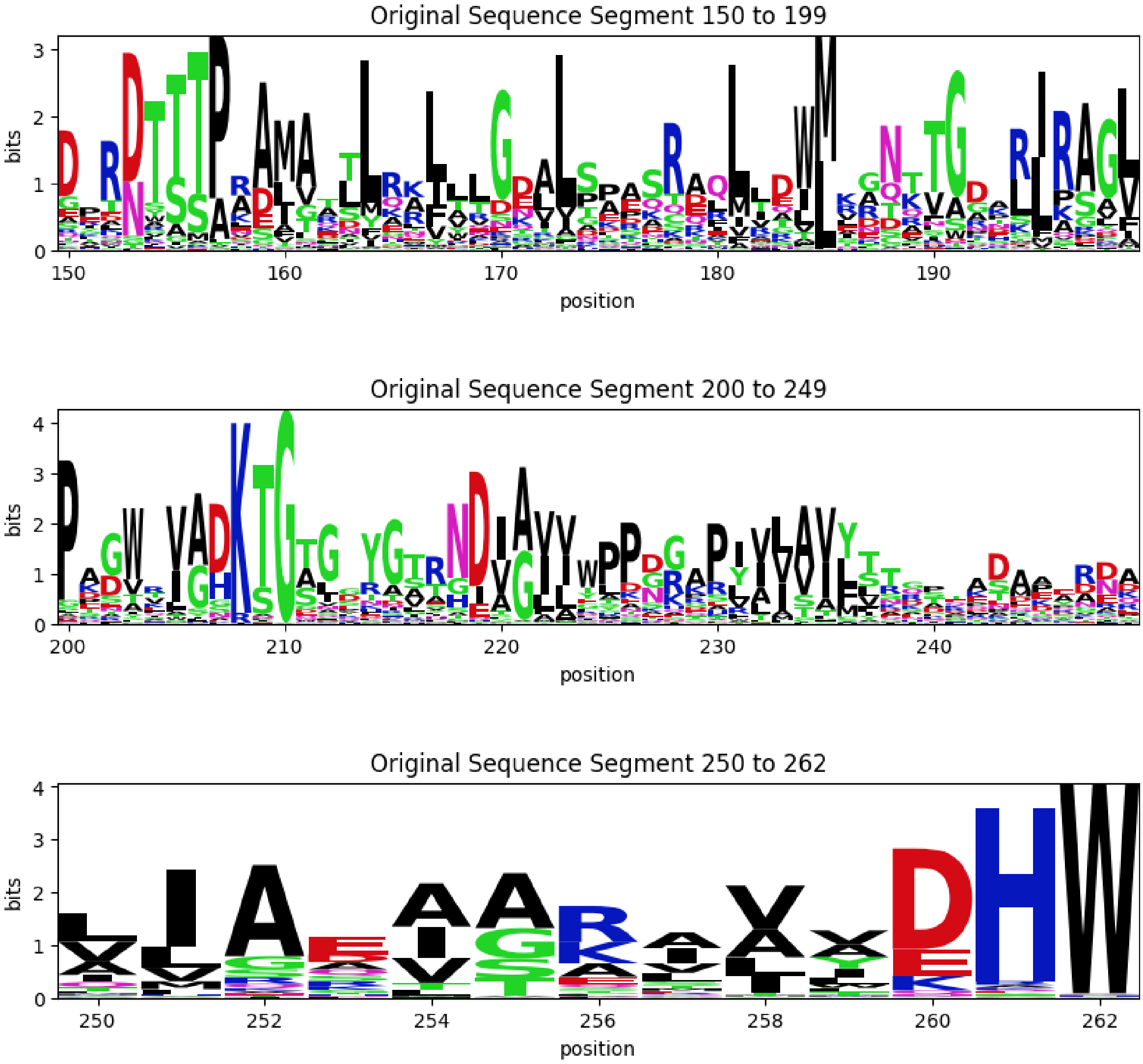


Figure A.2.: Sequence logos from position 150 to 262 of the original sequences

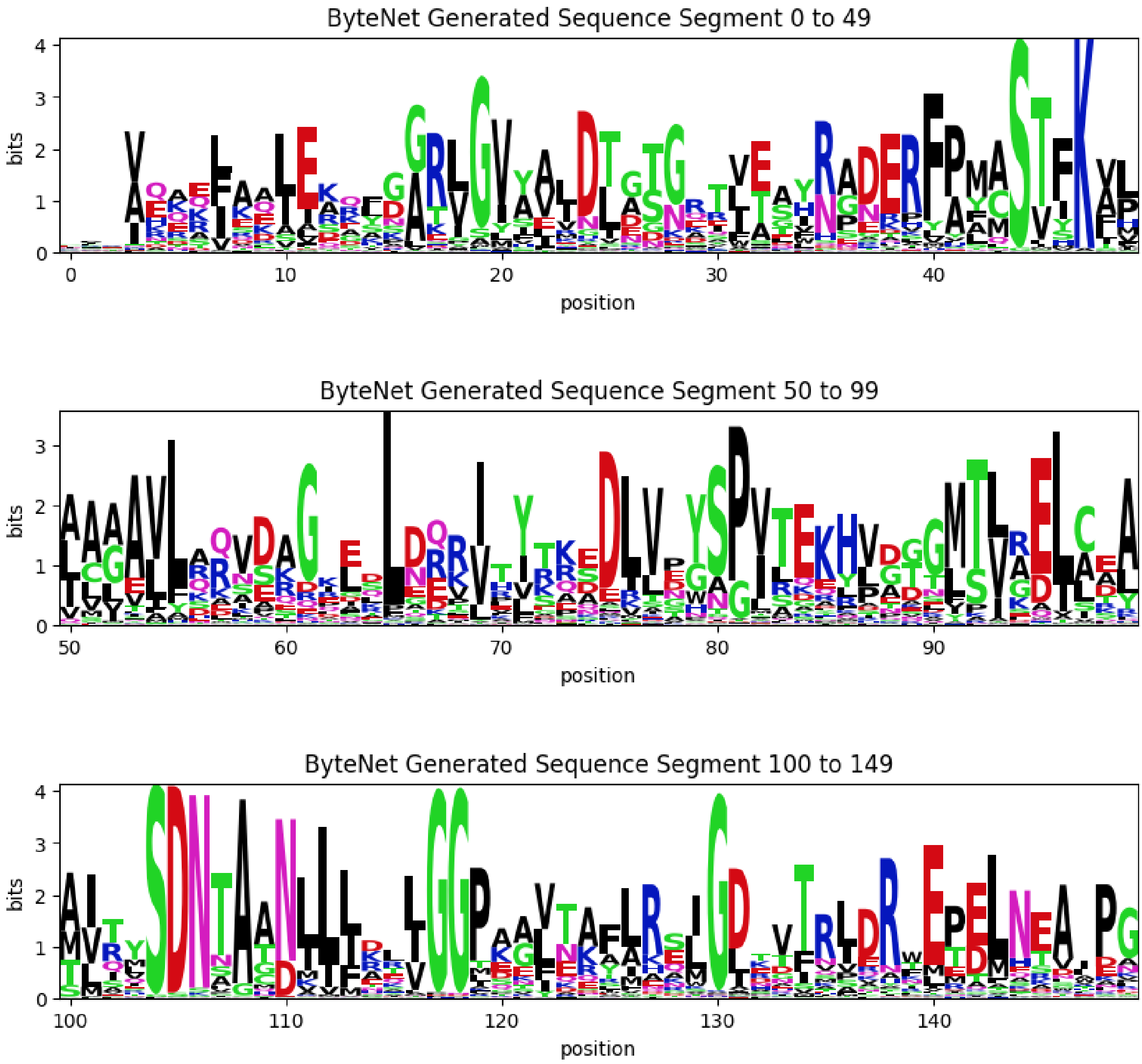


Figure A.3.: Sequence logos from position 0 to 149 of the ByteNet generated sequences

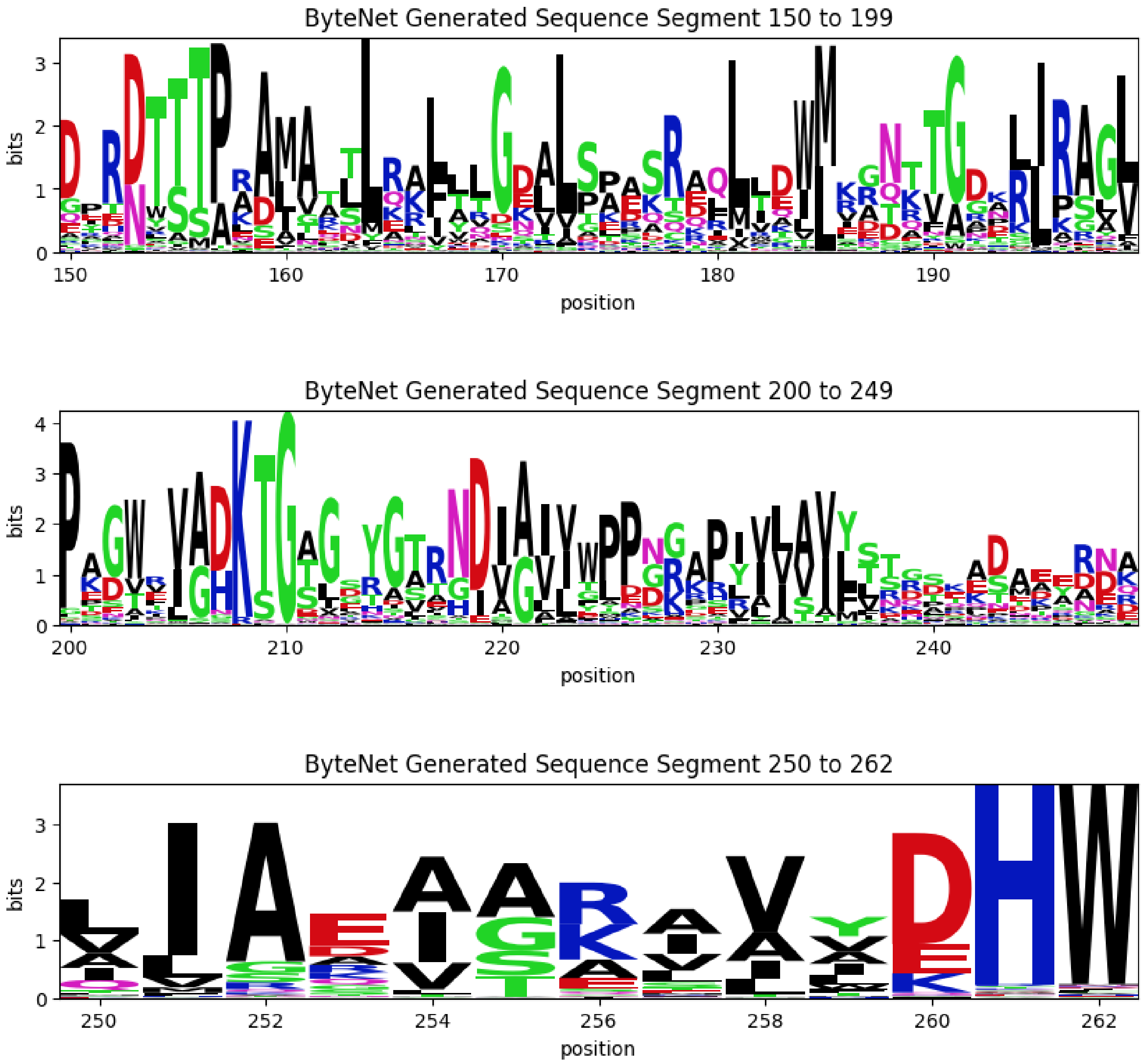


Figure A.4.: Sequence logos from position 150 to 262 of the ByteNet generated sequences

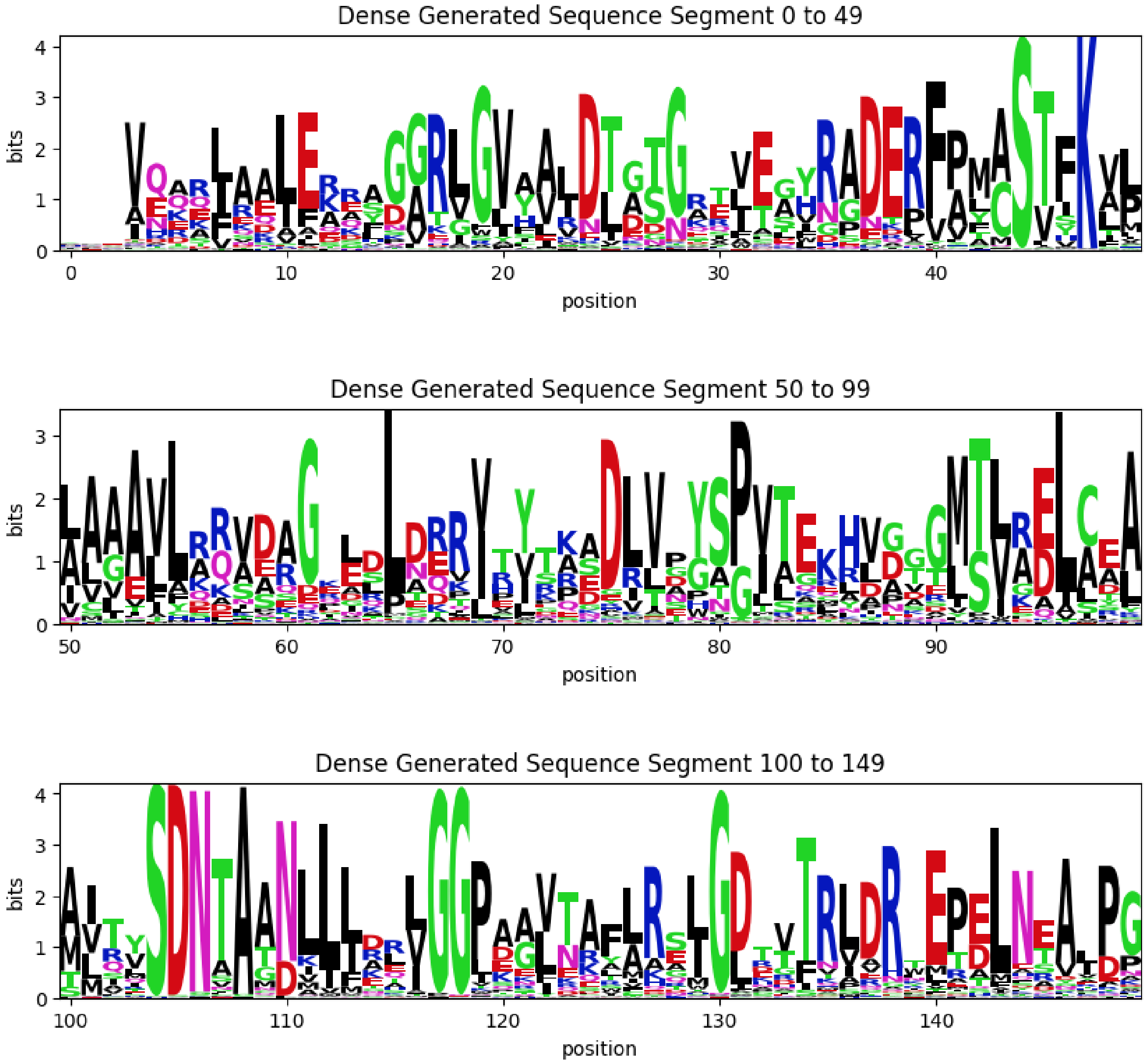
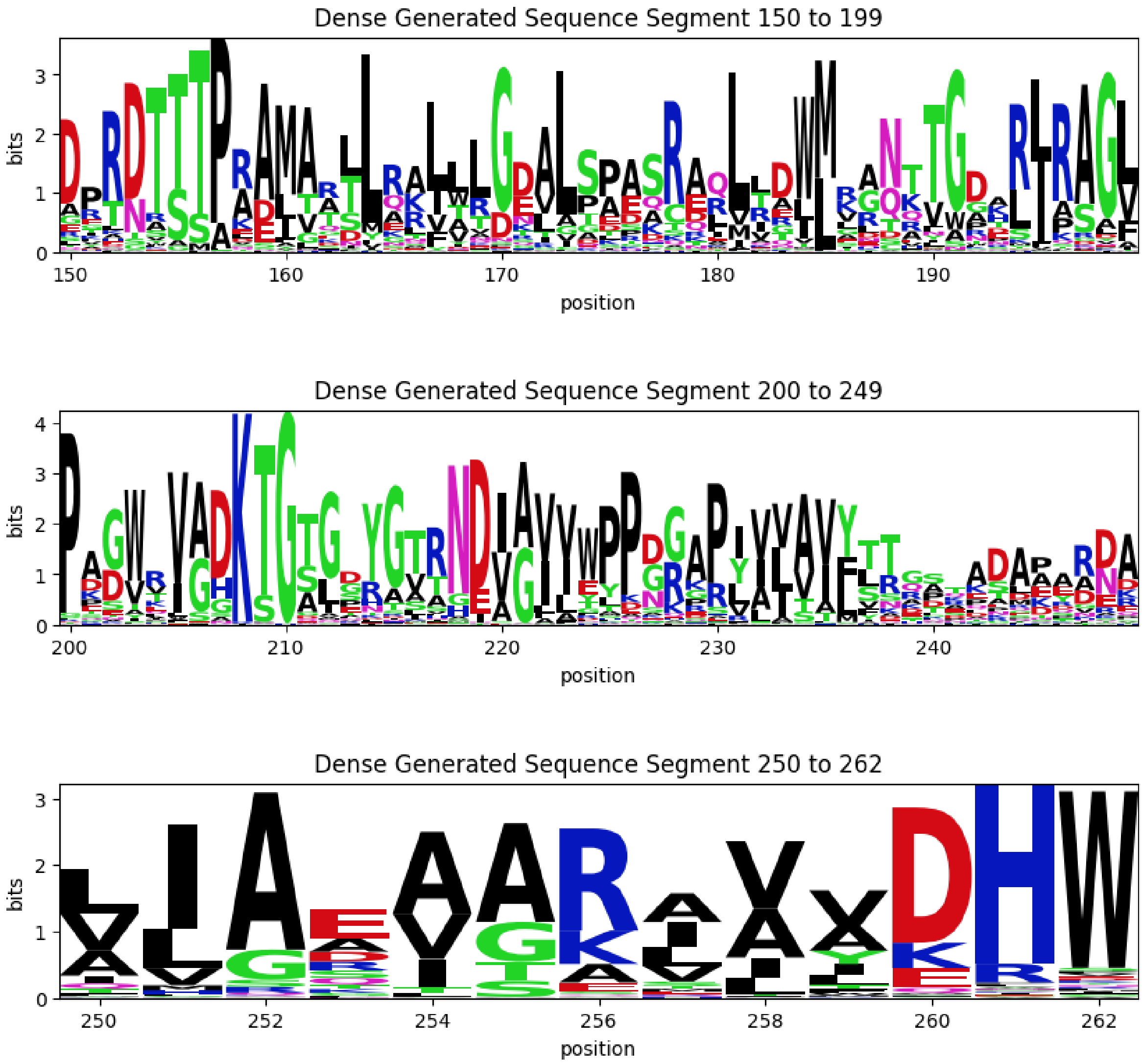


Figure A.5.: Sequence logos from position 0 to 149 of the Dense generated sequences


Dense Generated Sequence Segment 150 to 199

Dense Generated Sequence Segment 200 to 249

Dense Generated Sequence Segment 250 to 262

Figure A.6.: Sequence logos from position 150 to 262 of the Dense generated sequences

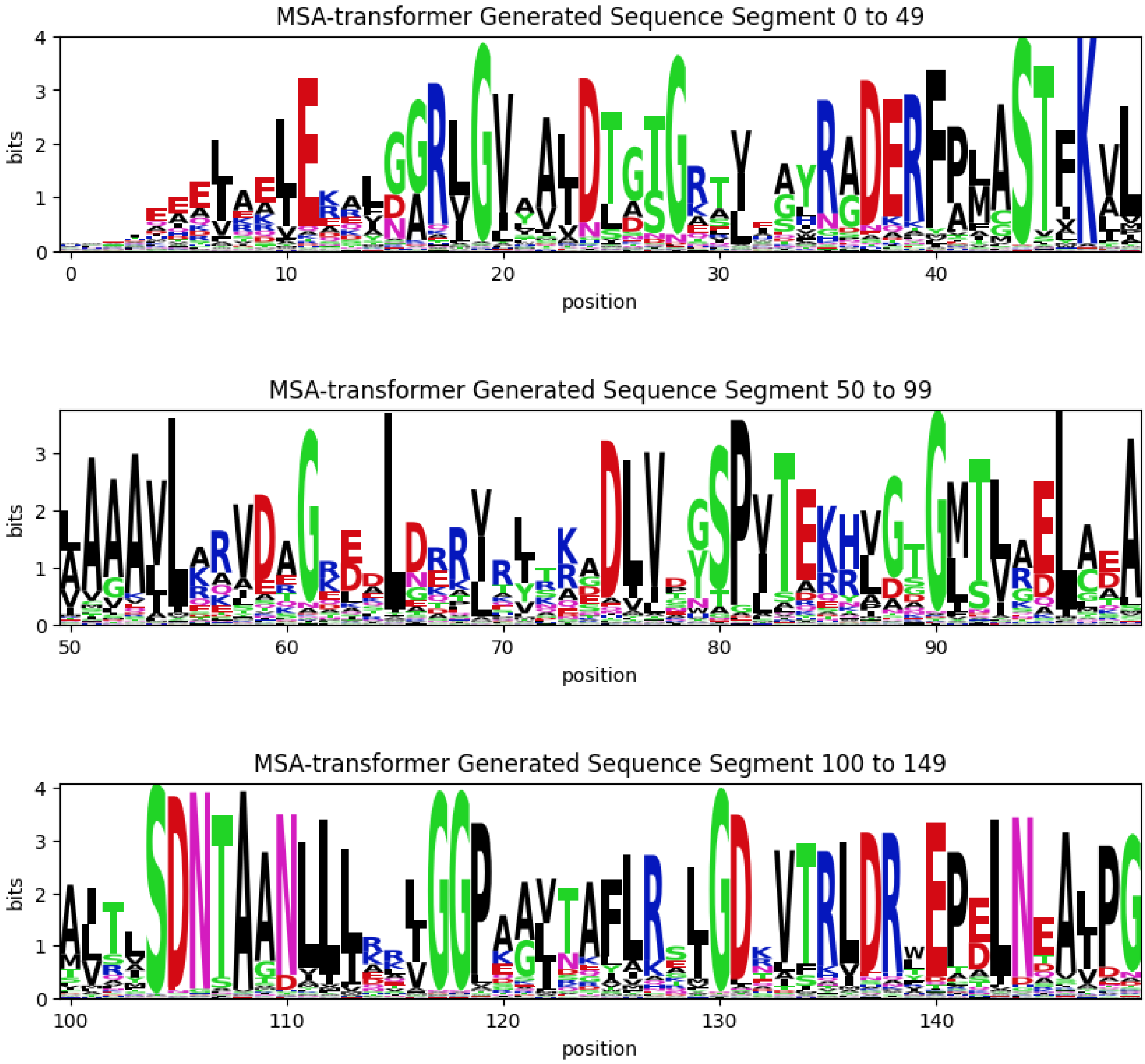


Figure A.7.: Sequence logos from position 0 to 149 of the MSA-transformer generated sequences

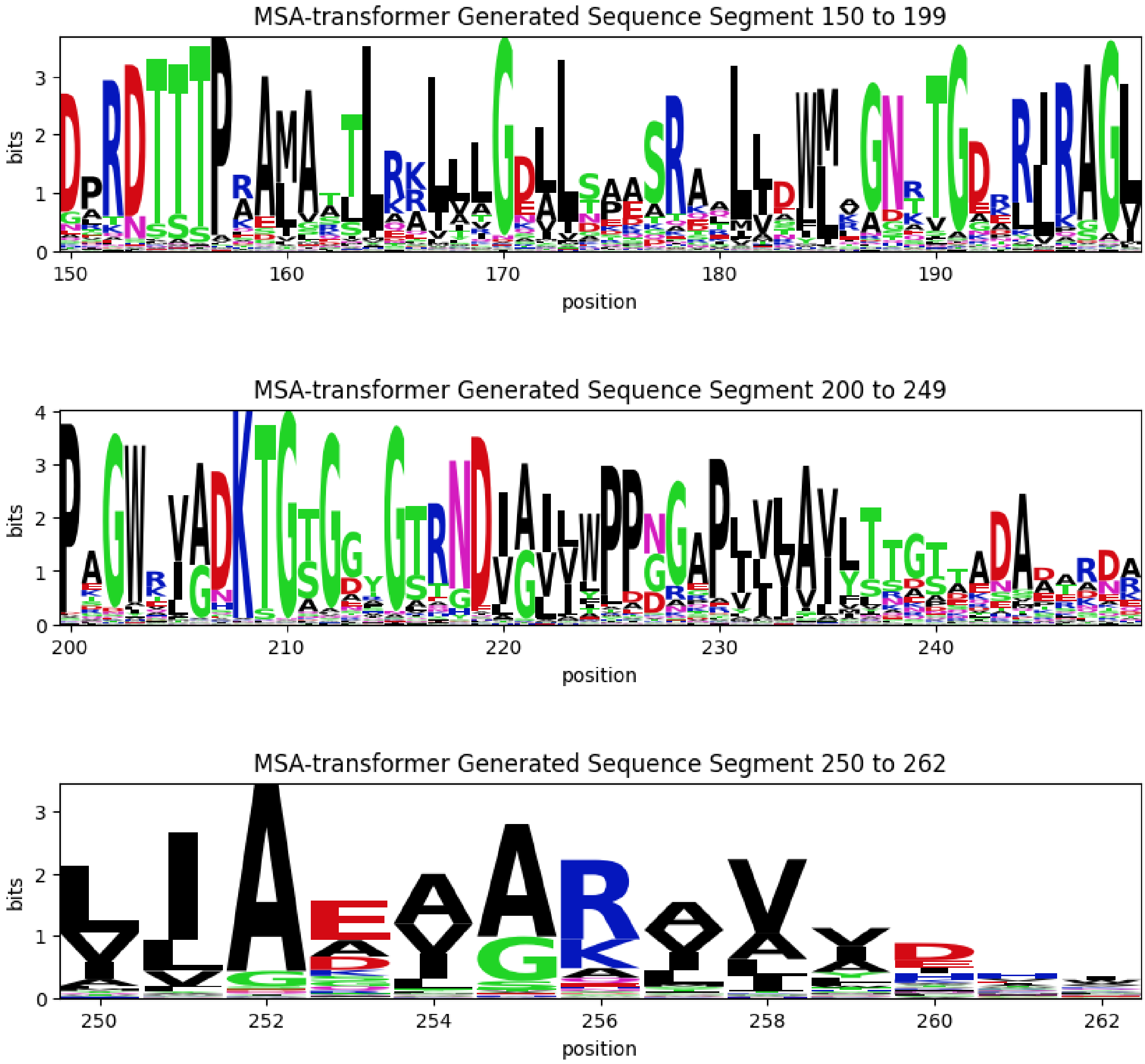


Figure A.8.: Sequence logos from position 150 to 262 of the MSA-transformer generated sequences