FACULTY OF ENGINEERING
AND ARCHITECTURE

Optimizing diffusion models for generative antibody design

Simon Vermeir
Student number: 01512101

Supervisors: Prof. dr. ir. Thomas Demeester, Prof. dr. Kathleen Marchal

Master's dissertation submitted in order to obtain the degree of Master of Science in Computer Science Engineering

Academic year 2024-2025

GHENT
UNIVERSITY

Abstract

Selecting new drug candidates such as antibodies is a challenging task. In today's drug design tool chain generative diffusion models are employed. However, these models require large datasets to train on. In antibody design this is a problem as the amount of publicly available data is limited. To counter this issue, we propose the use of a negative guidance technique to improve sample quality without increasing dataset size.

Recently a new technique for image diffusion has been proposed that aims to get samples closer to the original ground truth data distribution called SIMS. An auxiliary model is trained on synthetic data from a base model trained on the original dataset. At inference time guidance is done between both. The reasoning behind this is that the bias between base and auxiliary has traits of the bias between base and the ground truth. This is used to shift base closer to the ground truth. They show state of the art FID scores on imagenet. Our hypothesis is that by employing this technique on antibody-antigen diffusion similar improvements can be seen.

The proposed integration of SIMS has been applied to Diffab, an accessible and trainable antibody-antigen diffusion model. When a CDR-region is masked out on an antigen-antibody complex, Diffab is able to generate a new CDR-region that fits the rest of the antibody and the antigen. To verify the technique a base model was trained from SabDab using the Diffab framework. Next an auxiliary model initialized with weights from base was trained with synthetic data from base. To achieve reasonable training times a two step training approach was employed. Finally at inference time guidance is done on the predicted noise ϵ by using both models: $\epsilon_{guided} = (1 + \omega)\epsilon_{base} - \omega\epsilon_{aux}$ where ω controls the guidance strength.

On a held out test set of 350 antibody-antigen complexes, the use of SIMS on the (x, y, z) positions has been shown to improve the RMSD of the generated CDRH3 regions by $\sim 8\%$ compared to the base model.

Acknowledgements

When something is important enough you start ignoring the noise. That's what this thesis taught me.

Importantly I want to express my gratitude to the people who have helped me along the way.

To all the professors and faculty who always did their best to take my situation into account and who often helped me in reaching my goals.

To my promotors prof. Thomas Demeester and prof. Kathleen Marchal. For the countless meetings and discussions surrounding this topic. Finally to Johannes Deleu who helped me navigate the intricacies of generative antibody design.

And to my parents and sister for their unconditional support.

Contents

Chapter 1

Introduction

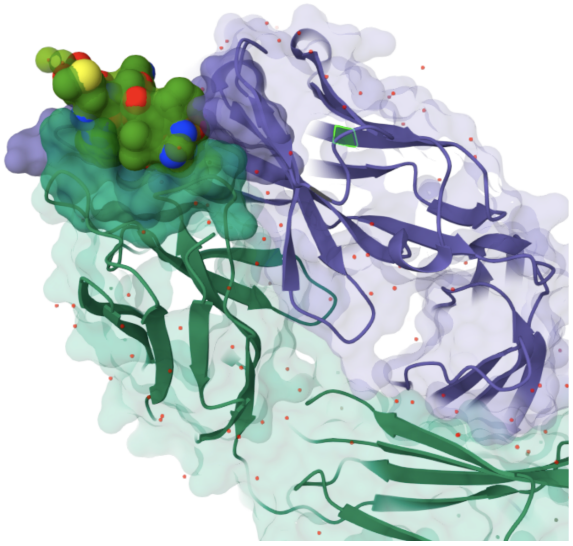


Figure 1.1: An antibody antigen complex. With the heavy and light chain of the antibody translucent and the antigen filled.

1.1 Introduction

In traditional pre-clinical drug design, a significant portion of the time is used to discover and test suitable drug candidates. One subset of drug candidates are antibodies. Antibodies are natural occurring proteins that will identify foreign actors within the body, such as viruses. In immunology, the target that antibodies recognize is called an antigen. An antigen can be the whole foreign element, or just a part of it, such as the spike protein in COVID-19. The antibody will bind to the antigen, forming an antibody-antigen complex. Subsequently, the antibody will signal the immune system to deal with this abnormality. Antibodies are produced by B-cells and can be present freely or locked to the surface of the B-cell. There are specialized classes of antibodies that can signal the immune system in different ways.

A new promising technique in the drug design pipeline is in silico generative antibody

design. Whereby, we use generative (AI) techniques to create suitable antibodies for a particular target. Traditionally many antibodies, by first generating candidates in silico we can reduce the amount of time required to develop a drug.

1.2 Antibodies

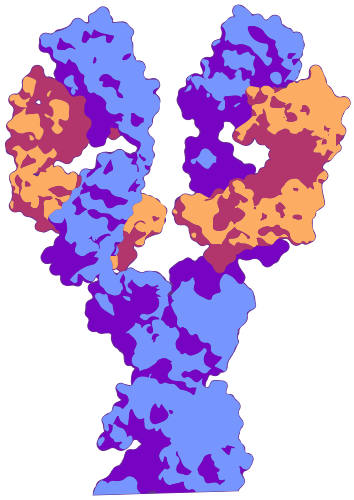


Figure 1.2: An antibody. Source: NIH BioArt [25]

If we want to generate antibodies, a key aspect is understanding the basic structure of such an antibody. The following section will delve into the intricate structure and function of antibodies. It is largely based on the book Molecular Biology of the Cell by **Alberts et al.** [2]

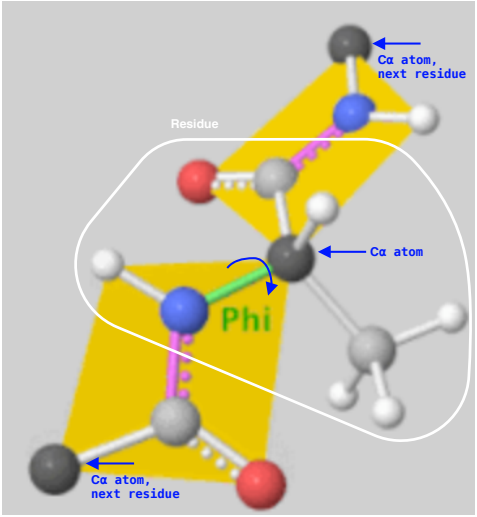
1.2.1 Proteins

An antibody is a type of protein. Proteins perform a vast array of functions within living organisms, including catalyzing metabolic reactions, replicating DNA, responding to stimuli, and transporting molecules from one location to another.

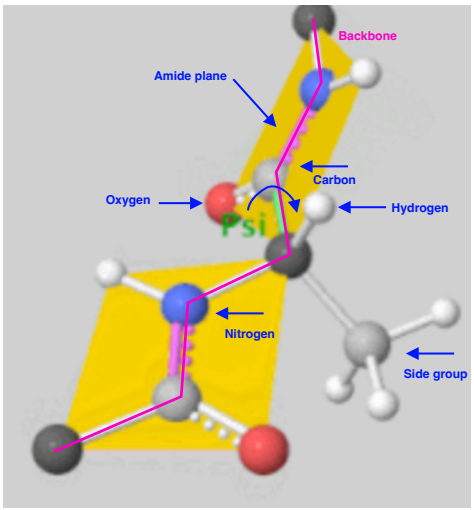
A protein is a large molecule composed of one or more long chains of amino acids.

An amino acid is a molecule. In the most basic form it consists of a central carbon atom (the carbon alpha) bonded to four different groups: an amino group (-NH2), a carboxyl group (-COOH), a hydrogen atom, and a distinctive side chain (R group) that varies among different amino acids. The side chain determines a proteins function and the way it interacts with other molecules. Amino acids also differ in charge, and can be hydrophobic or hydrophilic.

There are 20 standard amino acids. When they connect to each other they form a chain. This is called a protein or a polypeptide chain. The carboxyl group of one amino acid will connect to the amino group of another, and so on. When talking about one amino acid after it is incorporated into a chain it is called a residue (water is lost during linkage). Residue and amino-acid will be used interchangeably in this work.



(a)
Highlighting one residue with its Cα, and the phi angle between Cα and Nitrogen



(b)
Highlighting the different atoms, the backbone, and the psi angle between Cα and the carboxyl carbon.

Figure 1.3: A part of a polypeptide chain. Illustrating the different components and the different angles between neighboring residues. Source: edited GIF from Protopedia [14]

Crucially to remember when representing these amino acids are the shared atoms across them. Namely the carbon alpha, the carboxyl carbon, and the nitrogen of the amino group. These are called the backbone atoms. When you connect amino acids and forget about the side chains for a moment, you can see that these backbone atoms line up to form a chain. Together they form the backbone of the protein. When you connect the side chain groups back to the backbone you get the full polypeptide again. In protein modeling it is common to move between these representations. You can represent a protein in a simplified way by just showing the backbone atoms and ignoring the side chains. When you have the backbone there are algorithms such as PyRosetta [7] that can compute the side chains again and assign them.

The sequence and arrangement of these amino acids in a protein dictate its three-dimensional structure and, consequently, its function. This is because the sequence determines how they will interact with each other some might for example show strong attraction to its neighbors causing the chain to fold in 3D. Since they have different charges etc. When talking about the shape (and the function it carries with this shape in turn) of the protein often it is referred to as the conformation. Moreover not only the amino acids themselves will affect each other but also its environment. Factors such as pH, temperature... can all influence the conformation and stability of the protein. In this work the main way proteins will be modeled is using their captured 3D structures. These structures are most often captured using X-ray crystallography or cryo-electron tomography and stored in databases such as the Protein Data Bank (PDB) [[4] , [6]]. Interestingly these are captured at a very cold temperature and often in a different environment than their source environment. Thus although this work tries its best to computationally represent proteins

as good as possible, modelling the true protein is still an open problem.

Due to the many combinatorial combinations of amino acids, the potential structures and functions of proteins are vast. Theoretically say with a length of 100 amino acids, there are 20^{100} possible combinations. Many of them will be infeasible yet it shows the immense diversity of protein structures and thus their complexity.

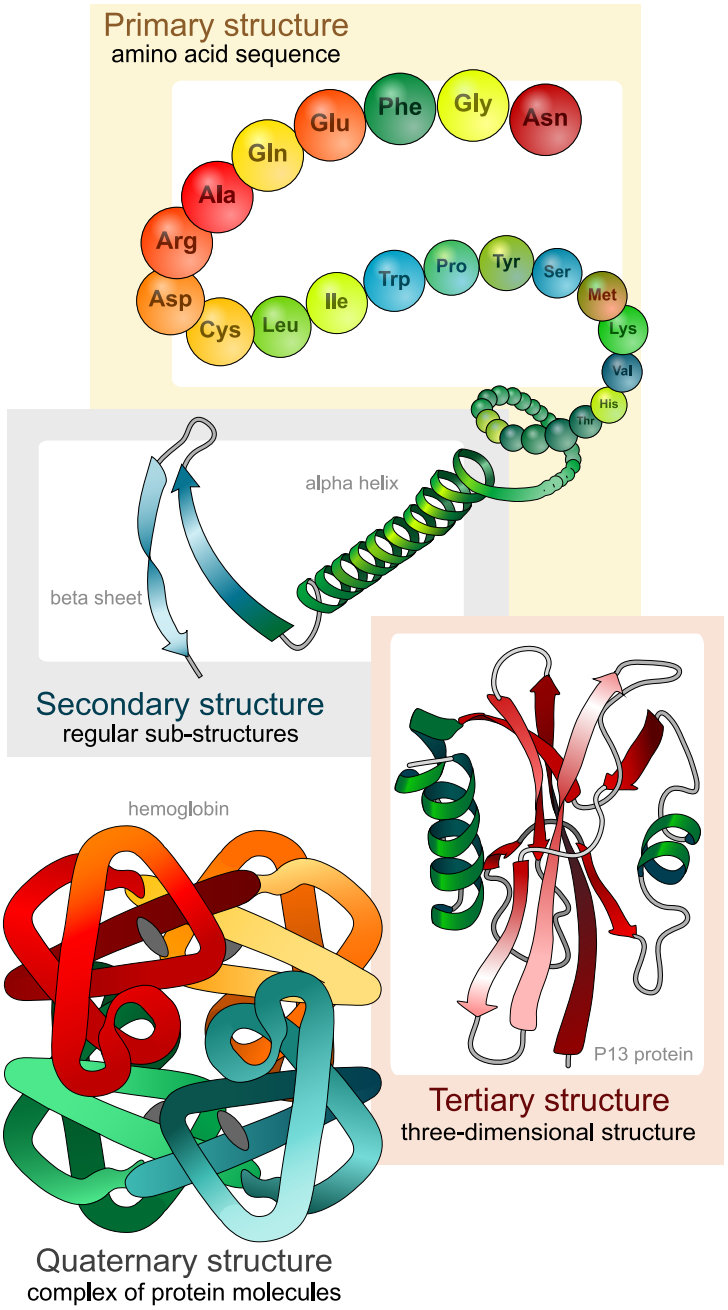


Figure 1.4: The different levels of structure of a protein. Source: Wikimedia Commons [29]

A protein's structure can be described at four levels:

- Primary structure: the linear sequence of amino acids in the polypeptide chain.
- Secondary structure: local folding patterns within the protein, such as alpha-helices and beta-sheets, stabilized by hydrogen bonds.
- Tertiary structure: the overall three-dimensional shape of a single polypeptide chain, determined by interactions between side chains.
- Quaternary structure: the arrangement of multiple polypeptide chains into a functional protein complex.
  - In fact an antibody can be build up of multiple of these chains and thus forms a quaternary structure.

When modelling a polypeptide structure in 3D many different representations can be used to model their structure some common ways:

- All Atomic coordinates: specifying the 3D coordinates of each atom in the protein.
- Only heavy atoms: leaving out the hydrogen atoms. But keeping the sidechain atoms too.
- The backbone representation: focusing on the main chain of the polypeptide, often simplified to show only the C-alpha atoms.
- Model each amino acid as single point (often the C alpha atom). Then assign a rotation to each point. Since now it is an object made up of multiple atoms not just single points in space.
  - Model each backbone atom as a 3D point.
  - Only model the 3D coordinates of the C-alpha atoms. Then assign a rotation to each C-alpha atom to represent the orientation of the residue.
- Using the C alpha position and dihedral angles: representing the protein's conformation through the positions of C alpha atoms and the angles between them. This is illustrated in figure 1.3. Two dihedral angles are visible the phi and psi angles. The phi is between the C alpha and the Nitrogen, while the psi is between the C alpha and the carboxyl Carbon. You can see between both images different conformations.

There is also some distinction between global and local representations of protein structures. You can:

- View the protein as a whole and describe the position and pose in 3D space.
- Focus on individual residues and either:
  - Model their coordinates absolute in 3D space.
  - Model their coordinates relative to the global position of the whole residue.

- You can also model a residue within its global reference frame so in relation to all other residues. Or to its local reference frame, so in relation to its own atoms.

This is important since when you turn and place the protein as whole somewhere else it is still the same protein. But when you change the position and orientation of the individual residues, the function is changed.

In this work the proteins will be represented in the different ways described above. Depending on the task. Therefore it is important be familiar with the terms introduced above.

1.2.2 Structure of Antibodies

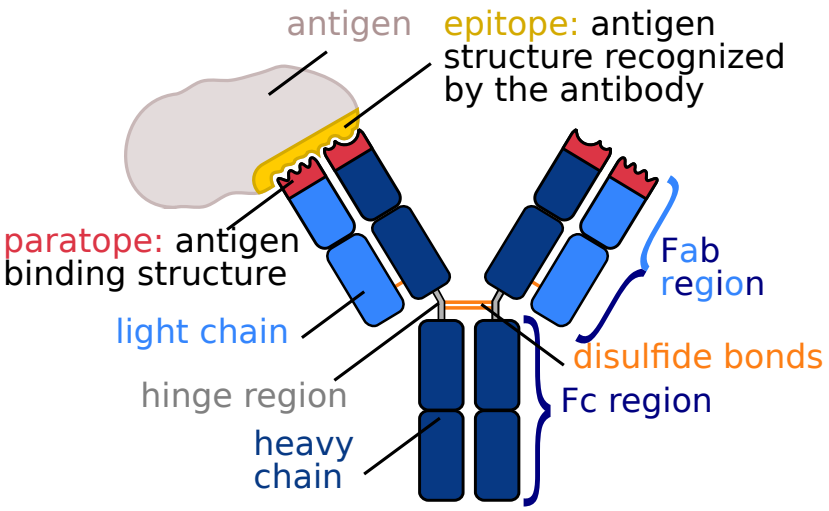


Figure 1.5: The antibody structure. Source: Wikimedia Commons [16]

The general structure of an antibody is a Y shape. It is made up of two polypeptide chains. A heavy chain and a light chain. The heavy chain is typically longer than the light chain. There can be multiple heavy/light chains in an antibody, leading to different classes of antibodies. There are many of them. One common one is the IgG antibody (immunoglobulin G). Figure 1.5 represents this one, and it leads to the typical Y shape.

There are two heavy chains and two light chains. The chains are connected by disulfide bonds. The top part of the Y is called the Fab region, this is the part that binds to the antigen. The bottom part is called the Fc region, this part will signal the immune system. The Fc region is typically fairly constant among antibodies while the Fab region is more variable. Specifically the part that interacts with the antigen is highly variable. That makes sense since it needs to be adapted to the antigen. This part is called the paratope. The part of the antigen that interacts with the paratope of the antibody is the epitope. When the antigen and the antibody are bound together they form an antibody-antigen complex, as shown in figure 1.1. It can also be said that that the antibody is docked onto the antigen.

Figure 1.6: The CDR regions of an antibody (for the heavy chain).

The paratope is made up of three key regions. These are called the complementarity-determining regions, as can bee seen in figure 1.6. Often referred to as the CDRs. Specifically CDR1, CDR2 and CDR3. They are around 7-20 residues in length. Both the heavy chain and the light chain have such regions thus when referring to the regions of the heavy chain they are called HCDR1, HCDR2 and HCDR3. And for the light chain LCDR1, LCDR2 and LCDR3. They can also be referred to as **CDRH1**, **CDRH2** and **CDRH3** for heavy chain and **CDRL1**, **CDRL2** and **CDRL3** for light chain, in this work the latter notation will be used. The CDRs are responsible for the specificity of the antibody-antigen interaction. They vary a lot, and specifically the CDRH3 is the most variable region. They typically don't form a alpha helix or beta sheets but more random shapes. Therefore they are also called loops, and are adapted to the antigen shape.

1.2.3 Antibody refinement

The body has a huge repertoire of stored antibodies. In fact even if the immune system has not seen an antigen before it should have an antibody that binds to it although with low binding affinity. This huge repertoire is possible thanks to V(D)J recombination. When B cells develop in the bone marrow, the DNA of the B-cells undergo this rearrangement. This recombination divides the DNA sequence that encodes the antibody in 3 regions. This process randomly combines variable (V), diversity (D), and joining (J) gene segments to create a unique variable region for each antibody. After this recombination the B cell will express the antibody on its surface. What this recombination of different parts of the genetic code of the antibody achieves is in fact a permutation of the CDR regions and especially the CDR3 region. Using a selection process, the immune system makes sure that newly generated antibodies are stable. If they are not stable, they will die. Or if they show self-reactivity, they will also be destroyed. After passing this test the antibodies are made available for use.

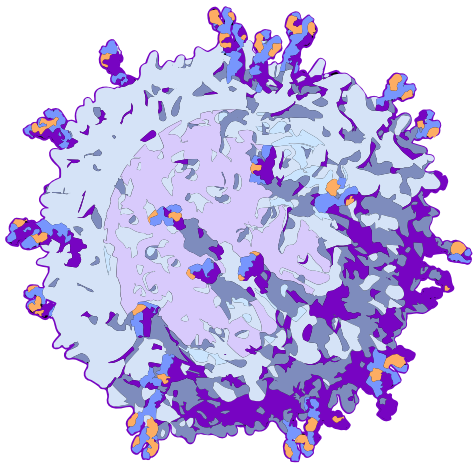


Figure 1.7: Antibodies expressed on the surface of a B cell.
Source: NIH BioArt [26]

The immune system uses affinity maturation to gradually improve antibodies over time so that they fit the antigens better. Using somatic hypermutation, ie when the B cells divide random mutations will be applied to the CDR regions. In this way new versions of the antibody are created. Additionally, a selection process is used, antibodies that bind better to antigens will survive longer, and ones that don't bind well will survive less long. Thus, the antibodies with the best mutation will survive and replicate. This process is repeated. The best B cells (the ones with the best antibodies on their surface) will stop mutating and become memory B cells. In this way, the body will be ready the next time the antigen appears and have a better response.

1.3 Diffusion Models

1.3.1 A brief intro into diffusion models

Diffusion models are a class of generative models that have gained popularity for their ability to generate high-quality samples from complex distributions. They work by modeling the process of diffusion, where data points are gradually transformed into noise and then reconstructed back into data.

The most well know form of diffusion is image diffusion. Here during training the model will learn to denoise an image that has been corrupted with gaussian noise. During inference the model will start from pure noise and gradually denoise it to get a final image. Figure illustrates this process.

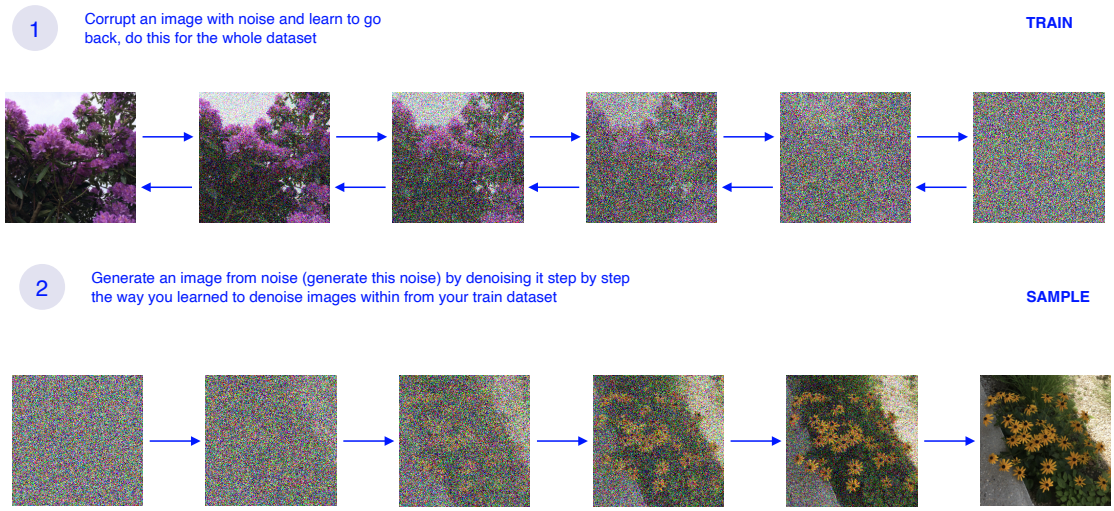


Figure 1.8: The diffusion process.

1.3.2 For antibody design

Recently it has been discovered that just like for creating new images from noise, diffusion models can be employed to create novel antibodies. Just like in the case of image diffusion they will during their training learn to reconstruct a corrupted random representation of an antibody. Then during inference novel antibodies can be generated from this noisy state. While this process is comparable to image diffusion the way these antibodies are represented is different from image diffusion which models pixel values.

In the case of antibodies, typically the backbone representation introduced earlier will be used. In some cases the antibody representation will be in complex with its antigen and the antigen structure will also be visible to the model. When training a part of the backbone will be scrambled and the model will learn to reconstruct this scrambled region. It will keep in mind the antigen and the surrounding antibody context. Then when the model is used a template antibody can be used that is docked to an antigen of interest. The region that we want to generate will be scrambled and the model will generate this region de novo.

The objective for this thesis is to optimize models for generative antibody design. Therefore the focus will be on optimizing models to produce better CDRH3 regions. Since they have the biggest impact, and it makes the problem well defined for the scope of this work. With the goal that these findings can be used to further improve the other regions in the future.

Chapter 2

Literature Review

The following chapter covers a brief review of the literature that was used during the creation of this work. This review intends to highlight the papers that made a considerable impact in shaping this work. It does not intend to explain each technique in detail. However it does cover some fundamental concepts that are crucial for understanding the context of this work.

2.1 Diffusion models

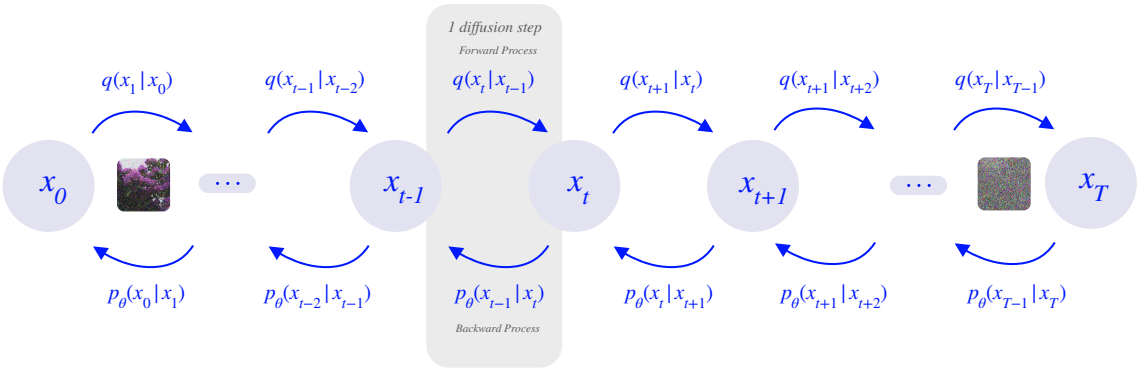


Figure 2.1: The diffusion process modeled as a markov chain. The forward noising process indicated by the q and the learned denoiser indicated by p_θ.

This section intends to highlight to most fundamental aspects and formulations of diffusion models. For a comprehensive mathematical foundation the work *Understanding Diffusion Models: A Unified Perspective* by Calvin Luo [21] is an excellent resource. The following explanation is based on insights from this and from the course *Deep Generative Models (UGent, 2025)* by Dhoedt et al.

2.1.1 General

A diffusion model can be modeled as a *restricted markovian hierarchical VAE (M-HVAE)* as shown in Figure 2.1. The forward process is a fixed noising process that adds noise

to the data. The backward process is a learned denoiser that tries to reverse this noising process. The forward process is defined by $q(x_t \mid x_{t-1})$ and the backward process is defined by $p_\theta(x_{t-1} \mid x_t)$. The forward process is fixed and not learned, the backward process is learned. The goal of training is to learn this backward denoiser. The end of the forward process results in a multi variate gaussian. This is the prior distribution. When using the model we sample a point from this gaussian and apply the learned denoiser. Each results from the denoiser is passed again into it, until we reach the original data space. This is a stochastic process. Thanks to this de novo data can be generated that resembles our original data.

2.1.2 Loss function

The loss function of diffusion models is similar to the ELBO loss used in the VAE [18]. The ELBO stands for Evidence Lower Bound and it defines a lower bound on the log likelihood of the data. The goal is to maximize the log likelihood. By using the ELBO a model can iteratively try to improve this ELBO. The ELBO loss is the negative ELBO since we will try to minimize it.

In the VAE the ELBO can be divided in a reconstruction term where the model tries to reconstruct a sample successfully and a regularization term that tries to make sure the latent dimension has the correct prior. Since the diffusion model a special form is of a M-HVAE the ELBO loss can be defined for this chain:

$$\begin{aligned}\mathcal{L}_{ELBO} = & -\mathbb{E}_{q(x_1|x_0)}\left[\log p_\theta(x_0|x_1)\right] + D_{KL}\left[q(x_T|x_0) \,\|\, p(x_T)\right] \\ & + \sum_{t=2}^{T} \mathbb{E}_{q(x_t|x_0)}\left[D_{KL}\left[q(x_{t-1}|x_t, x_0) \,\|\, p_\theta(x_{t-1}|x_t)\right]\right] \\ = & \, \mathcal{L}_0 + \sum_{t=2}^{T} \mathcal{L}_{t-1} + \mathcal{L}_T\end{aligned} \tag{2.1}$$

The loss now still tries to maximize the log likelihood of the data. The first term $\mathcal{L}_0$ is a reconstruction term that tries to reconstruct the original data from the first noised version. The second term $\mathcal{L}_{t-1}$ is a sum over all intermediate steps where the model tries to learn to denoise from one step to the previous one. This is done by minimizing the KL divergence between the ground truth denoiser and the learned denoiser. The last term $\mathcal{L}_T$ is a regularization term that tries to make sure the last step of the forward process matches the prior distribution.

2.1.3 Denoising

The backward process is defined by $q(x_{t-1} \mid x_t, x_0)$ which is the ground truth denoiser.

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\left(x_{t-1};\, \tilde{\mu}(x_t, x_0),\, \tilde{\beta}_t(x_t, x_0)I\right) \tag{2.2}$$

where $\tilde{\beta}_t$ is the variance of the noise added at step t in the reverse process.

During training the model learns this function. The learned denoiser looks like:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(x_{t-1};\, \mu_\theta(x_t, t),\, \Sigma_\theta(x_t, t)\right) \tag{2.3}$$

where $\Sigma_\theta(x_t, t)$ is the variance predicted by the model at step t. For the covariance a time dependent profile is used so that this does not need to be learned.

$\mu_\theta(x_t, t)$ can be learned directly or we can learn the amount of noise that was added to x_t. This is done trough $\epsilon_\theta(x_t, t)$

Given the noisier sample x_t and the predicted noise $\epsilon_\theta(x_t, t)$ the next step can be computed:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z} \tag{2.4}$$

with $\alpha_t = 1 - \beta_t$ A higher α_t means more signal is preserved (less noise added). This is because it has an inverse relationship to β which is the variance of the noise added at that time step. When the variance is low you retain more of the previous step in the forward process (when set to zero for example you copy over the step), when it is high you add more noise. z is a point sampled from the prior distribution (typically a gaussian) σ controls the amount of noise to add to the backward process, this is needed because each diffusion step is a stochastic process. In each forward diffusion step you sample from a conditional gaussian distribution, not only at T! If σ_t would be zero then each step would be deterministic and we would lose sample diversity.

In this work the epsilon formulation will be used. The function to predict the added noise will be called **EpsilonNet** throughout this work. It will play a crucial role in helping to optimize these models. Figure 2.2 outlines how this epsilon net works. Be aware that this epsilon net is where the deep learning happens. Our epsilon net is a learned function. We want to learn a function that predicts noise. A neural network can be considered a universal function approximate and thus serves as the ideal candidate.

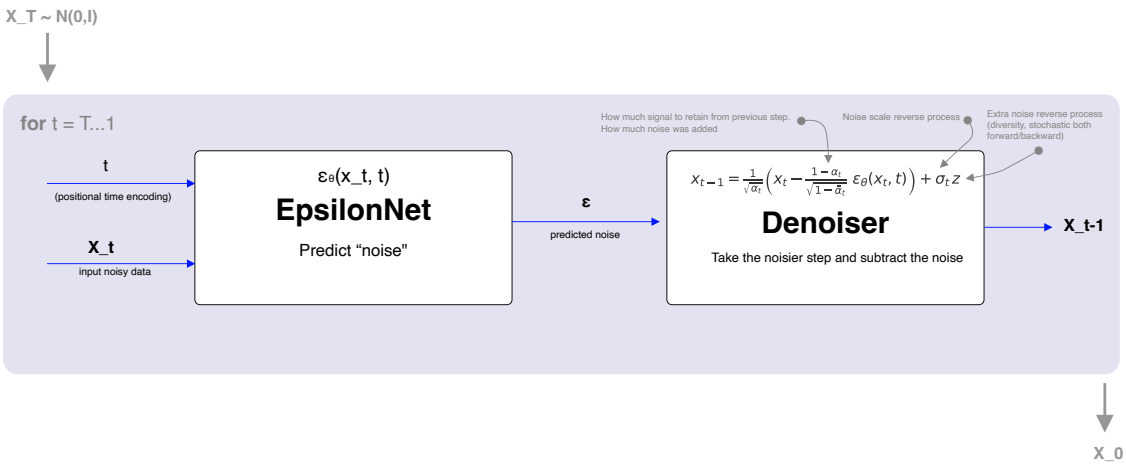


Figure 2.2: EpsilonNet

When sampling from a diffusion model the full diffusion chain is used. Recall that going forward from 0 to T means adding noise. Therefore when denoising we start at time step T and sample from the prior distribution. In the standard implementation this is typically a multivariate gaussian. For example $128 * 128$ in the case we want the generate images of $128 * 128$. In figure 2.2 you can see that this net can be used for multiple time steps, mimicking the classic diffusion chain.

Algorithm 1 outlines the sampling procedure.

Algorithm 1 Sampling

$\mathbf{x}_T \sim \mathcal{N}(0, I)$
for $t = T, \ldots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(0, I)$ if $t > 1$, else $\mathbf{z} = 0$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

2.1.4 Training

Training will always happen for one diffusion step. When implementing a diffusion model a simplified loss is used since, we optimize one diffusion step at a time. Then this loss becomes:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,\epsilon} \left\| \epsilon - \epsilon_\theta(x_t, t) \right\|^2. \tag{2.5}$$

Algorithm 2 outlines the training regime. This regime is possible because a noised version can be computed trough a closed form solution. Consequently the ground truth added noise can be predicted, and the EpsilonNet can learn to predict the added noise. By sampling over all time steps randomly we start to learn a denoiser that can denoise from any time step to the one before it.

Algorithm 2 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \text{Uniform}(\{1, \ldots, T\})$
  $\epsilon \sim \mathcal{N}(0, I)$
  Take gradient descent step on

$$\nabla_\theta \left\| \epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon,\, t \right) \right\|^2$$

until converged

2.2 Diffab

My starting point where two papers by Luo et al. [22] and Zhou et al. [36].

- Antigen-Specific Antibody Design and Optimization with Diffusion-Based Generative Models [22]
- Antigen-Specific Antibody Design via Direct Energy-based Preference Optimization [36]

They both covered antibody-antigen design. The paper by Luo et al [22]. introduced a diffusion model that learned to denoise masked CDR regions on an antibody-antigen complex. Just like in image diffusion this masked region was noised and the model was

trained to denoise it to a suitable CDR region. In this way the model can generate de novo CDR regions that may show better binding affinity and show promise as better drug candidates for example. The second paper builds upon the work of luo et al and introduces energy based optimization to make the structures more biologically relevant. Since in polypeptide chains low energy states are a good indicator that these are suitable ones.

Specifically the **Diffab** was studied in depth and forms just like the paper by the basis for this work.

Diffab introduces novel approach for antibody design by leveraging diffusion models. It focuses on the joint optimization of sequence and structure. Typically the sequence is not jointly generated but first only the antibody backbone structures is generated, afterward the sequence is found using an inverse folding model and subsequently the side chains are placed on the backbone.

The diffab model can be trained to either jointly infer multiple cdr regions or one specific one. Since the limited training data that is available single region optimization produces better results.

In an ideal world a model would invent an antibody from scratch given an antigen. This model works by starting from an antibody-antigen complex. It allows one to specify a cdr region that needs to be optimized. A benefit of this approach is that the problem becomes easier and more feasible to train with the limited data and compute.

To be able to generate a novel antibody this flow would be followed:

1. Find an antibody-antigen complex
2. Mask out the region you want to generate denovo for example CDRH3
3. Use the model
4. A new antibody-antigen complex that has the same framework as the starting one but with a new CDR region.

Internally the model represents antibodies using:

- position
  - Each residue's CA atom is represented by its 3D coordinates (x, y, z).
- orientation (or rotation (often used interchangeably in this work))
  - Each residue's orientation is represented by a rotation matrix (3x3) that describes its spatial orientation. Or by an axis-angle representation.
  - Important to note orientations are described in an equivariant way. Meaning they remain unchanged under global transformations of the complex, such as rotation and translation, ensuring that the model's predictions are consistent regardless of the antibody's position or orientation in space.
- sequence
  - Each residue's amino acid is kept track of. Either represented by letter, number or one-hot encoding.

Figure 2.3, from the diffab paper. Illustrates this diffusion process.

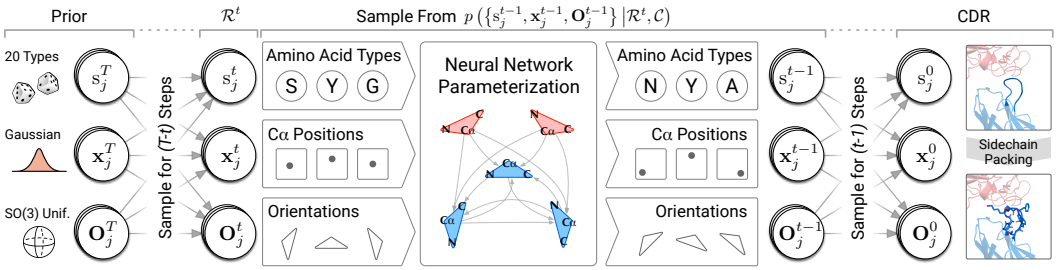


Figure 2.3: Diffab Diffusion Process. Source: Diffab paper [22]

- For position the process is similar to that of the classic diffusion that was discussed above.
- For orientation the forward process operates on SO(3) representations. It adds isotropic gaussian noise at each step. This effectively shakes the rotation each time a bit. Each direction it shakes in is equally likely. All of these random shakes are applied in a sequence. The backward tries to predict these to reconstruct the original rotation.
  - SO(3) is the set of all rotations in 3D space. It can be presented by rotation matrices or by an axis-angle representation.
    - The rotation matrix can be used as follows:

$$\underbrace{v' = Rv}_{\text{rotates the point}} \quad ; \quad \underbrace{[v]_{\text{rot frame}} = R^\top v}_{\text{rotates the frame}}, \qquad v \in \mathbb{R}^3.$$

    - The axis–angle can be encoded as a 3-vector ω whose magnitude $\|\omega\|$ is the rotation angle and whose direction $\omega/\|\omega\|$ is the rotation axis.
- Finally the sequence diffusion can be seen as discretized way of doing diffusion internally this happens by operation on the categorical distribution rather than the gaussian one. The prior is a uniform categorical over the masked sequence.

2.3 SIMS

Diffusion models are very data hungry. They require large amounts of high-quality data to train effectively, which can be a limiting factor in their application. A naive way to augment our training data would be to generate synthetic data. This could be in a simple way by using the existing data to create variations or by using generative models to create entirely new samples. In fact one could use its own trained model to generate new data, then use this data to further train the model. However this strategy typically is not very effective since a model will never model the ground truth perfectly, it will have slight errors. When we keep feeding the model generated data these errors will become worse and worse leading to a model that doesn't perform well anymore.

A new existing approach to address this issue has been introduced in the paper: *Self-Improving Diffusion Models with Synthetic Data* by **Alemohammad et al.** [3].

The authors claim that when feeding a model synthetic data generated by itself, it can lead to a positive feedback loop where the model goes MAD. MAD standing for: model autophagy disorder. Just the type of getting worse and worse that was described earlier.

Their idea uses guidance to counteract this. Guidance is a technique that has been used in diffusion models to steer the generation process towards desired attributes or characteristics. Guidance is an inference time technique were typically the predicted noise is modified to steer the generation process.

Their technique uses guidance alongside a base model that is trained on the original dataset, and an auxiliary model that is trained on synthetic data generated by the base model.

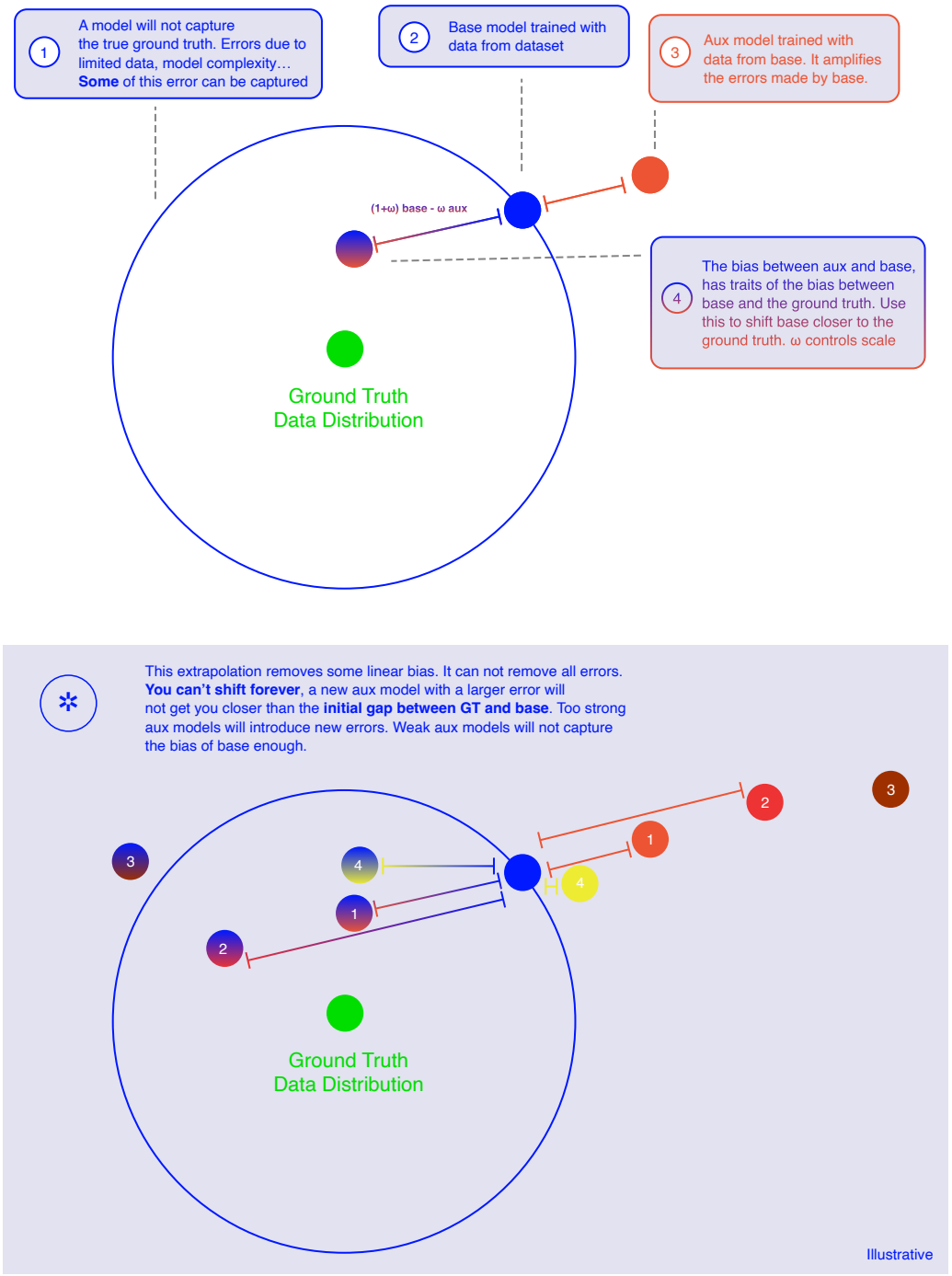


Figure 2.4: The SIMS idea visualized. Inspired by the SIMS paper [3]

Intuitively their technique works as follows:

A trained model will never be as good as the ground truth and will move away from it in some direction. When training a new model based on synthetic data from the first model it will move away even further in a similar way. This error we can measure and use to get closer to the ground truth. This is visualized in figure 2.4.

The algorithm they use to achieve this is outlined in figure 2.5.

Algorithm 1 SIMS Procedure

Input: Training dataset $\mathcal{D}$

Hyperparameters: Synthetic dataset size n_{s}, guidance strength ω, training budget $\mathcal{B}$

1: **Train base diffusion model**: Use dataset $\mathcal{D}$ to train the diffusion model using standard training, resulting in the score function $\boldsymbol{s}_{\theta_{\mathrm{r}}}(\boldsymbol{x}_t, t)$.

2: **Generate auxiliary synthetic data**: Create an internal synthetic dataset $\mathcal{S}$ by generating $n_{\mathrm{s}} = |\mathcal{S}|$ samples from the base diffusion model.

3: **Train auxiliary diffusion model**: Fine-tune the base model using only $\mathcal{S}$ within the training budget $\mathcal{B}$ to obtain $\boldsymbol{s}_{\theta_{\mathrm{s}}}(\boldsymbol{x}_t, t)$. Discard $\mathcal{S}$.

4: **Extrapolate the score function**: Use $\boldsymbol{s}_{\theta_{\mathrm{s}}}(\boldsymbol{x}_t, t)$ to extrapolate backwards from $\boldsymbol{s}_{\theta_{\mathrm{r}}}(\boldsymbol{x}_t, t)$ to the SIMS score function

$$\boldsymbol{s}_\theta(\boldsymbol{x}_t, t) = \boldsymbol{s}_{\theta_{\mathrm{r}}}(\boldsymbol{x}_t, t) - \omega(\boldsymbol{s}_{\theta_{\mathrm{s}}}(\boldsymbol{x}_t, t) - \boldsymbol{s}_{\theta_{\mathrm{r}}}(\boldsymbol{x}_t, t)) = (1+\omega)\boldsymbol{s}_{\theta_{\mathrm{r}}}(\boldsymbol{x}_t, t) - \omega \boldsymbol{s}_{\theta_{\mathrm{s}}}(\boldsymbol{x}_t, t).$$

Synthesize: Generate synthetic data from the model using the SIMS score function $\boldsymbol{s}_\theta(\boldsymbol{x}_t, t)$.

Figure 2.5: The SIMS algorithm. Source: SIMS paper [3]

2.4 Overview

Beyond the topics that were covered above, a brief discovery in the literature surrounding antibody design was done. This was mainly for extra background in understanding and to provide inspiration for implementation techniques.

Table 2.1 gives a concise summary of these with a one line description. This table does not aim to be a review, but rather a collection of key papers that informed this research. In the next section the areas of inquiry are discussed they have been informed by these papers and the discussions with advisors.

It is not an exhaustive list of all the papers that were read, but it does cover the most important ones. The papers are categorized into general, specific, deep learning and miscellaneous. The general category covers papers that are relevant to the overall topic of generative antibody design. The specific category covers papers that are directly related to the topic of this thesis. The deep learning category covers papers that are relevant to the deep learning techniques used in this thesis. The miscellaneous category covers papers that are relevant more for technical background.

Category	Title	Description
General	Accurate structure prediction of biomolecular interactions with AlphaFold 3 [1]	Generate 3D structure from sequence
	De novo design of protein structure and function with RFdiffusion [35]	De novo protein structures
	Boltz-1 [5]	Generate 3D structure from sequence
	Simulating 500 million years of evolution with a language model [13]	Fill in the gap in the sequence, latent space contains 3D representation
Specific	Antigen-Specific Antibody Design and Optimization with Diffusion-Based Generative Models for Protein Structures [22]	Joint Sequence and structure optimization of antibodies
	Antigen-Specific Antibody Design via Direct Energy-based Preference Optimization [36]	Same but added RL with energy optimization
DL	Understanding Diffusion Models: A Unified Perspective [21]	Understanding math behind DDPM
	Understanding Reinforcement Learning-Based Fine-Tuning of Diffusion Models: A Tutorial and Review [33]	RL for DDPM
	Self-Improving Diffusion Models with Synthetic Data [3]	Augmenting the dataset with synthetic data for DDPM
Misc	Optimization of therapeutic antibodies by predicting antigen specificity from antibody sequence via deep learning [23]	A model that predicts if it will bind (for in vivo data)
	SE(3)-transformers: 3D roto-translation equivariant attention networks [12]	simply put: be insensitive to the rotation of the 3D structure
	Lie Groups for Computer Vision [11]	Lie algebra useful for SO(3) transformations

Table 2.1: Overview of relevant literature.

2.5 Areas of inquiry

After exploring the state of the art, and numerous advisory meetings, four key areas were identified that are worth exploring with the thesis. See Figure 2.6 for details.

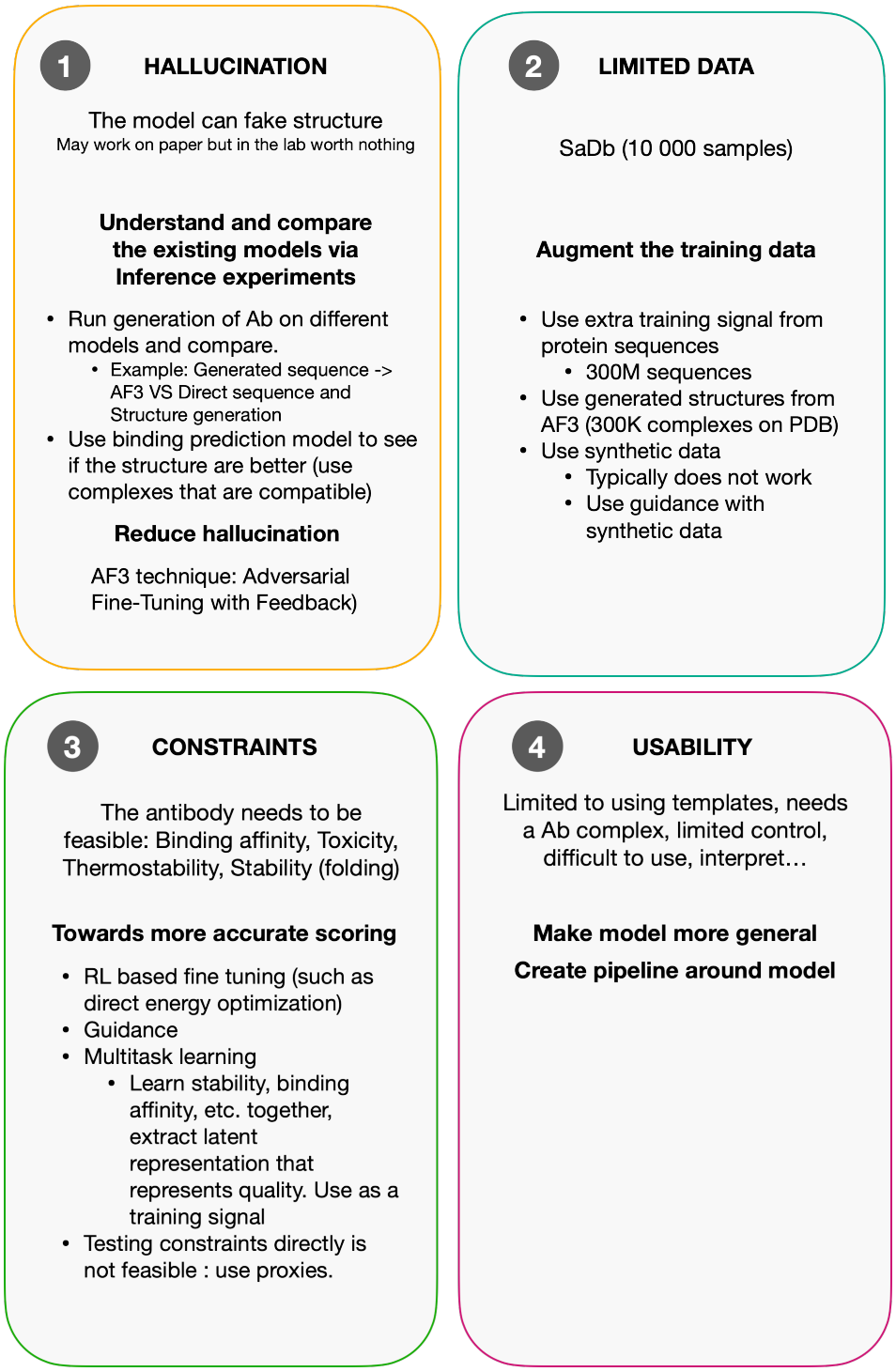


Figure 2.6: Areas of inquiry

Chapter 3

Foundations

The chapter foundations covers the underpinnings of the proposed framework for improving generative antibody design (DGAD). Including: the hypothesis we put forward, the chosen model, model architecture, and terminology used throughout this work.

3.1 Hypothesis

In the literature review the SIMS technique for image diffusion was discussed as a way to improve sample quality without increasing the dataset size. This by employing an auxiliary model trained on synthetic data trained by our base model. At inference time the auxiliary model is used to guide the base model towards samples that are closer to the ground truth data distribution. The idea is that the error produced by the auxiliary model can be used to adjust the predictions of the base model, effectively "nudging" it towards the ground truth.

This technique can be seen as a form of augmenting our dataset, without increasing the ground truth dataset. This is precisely one of the key challenges that was covered in the areas of inquiry.

The available structures of antigen-antibody complexes is limited. One well known database of such structures is SabDab [10]. This database contains only a few thousand structures. While this is a good start, it is still a limited amount of data for training a deep learning model.

> **Hypothesis**
>
> *Can negative guidance improve diffusion models for generative antibody design. By first training an auxiliary model using synthetic data from the base model. Then using the auxiliary model to guide the predictions of the base model closer to the ground truth data distribution.*

Thus the hypothesis of this thesis is to test if the SIMS [3] technique can also be applied on diffusion models for generative antibody design.

3.2 Choosing a model

In this work the focus lies on verifying if this hypothesis holds. Therefore it would not be productive to create a new model from scratch. Instead it is better to build on top of an existing model. This way the focus can be on the proposed method and not on building a new model. Naturally the explored Diffab [22] serves as a suitable candidate.

In antibody generation, it is common to build the design in separate steps. First, a diffusion model is used to generate the backbone structure. Then, the sequence is designed using an inverse folding model such as ProteinMPNN [9]. Finally side-chain packing algorithms are applied to place the side-chain conformations on top of the backbone. This approach can work well, but it depends on multiple components and make training and inference more complex.

DiffAb offers something more integrated. It models sequence and structure together, predicting the amino acid type, backbone position, and orientation of each residue in the same diffusion process.

Practical considerations also played a role. The *DiffAb* codebase was well structured and allowed for direct extension. Just as important, it was trainable on the available infrastructure at IDLab [27]. In this way, it combined both scientific interest and technical feasibility.

Additionally the dataset it used was *SabDab*[10] a trusted dataset of antibody-antigen complexes. This dataset provided a rich source of examples for training and evaluation.

For these reasons *DiffAb* was selected as the foundation of this work. It serves as the baseline against which the proposed DGAD framework is developed and evaluated.

3.3 Diffab

In the literature review, the DiffAb model was introduced as a novel approach for antibody design by leveraging diffusion models. It focuses on the joint optimization of sequence and structure.

3.3.1 Data pipeline

The discussed guidance technique is an inference time technique where the sampling process of diffab will be modified. To enable this a good understanding of the internal data representation of Diffab is crucial.

At a high level the data flows trough the model as shown in Figure 3.1.

The PDB file (that contains the 3D structure, for every atom it's position along with other information) is used as the input. This file is then processed to extract relevant features and create a suitable representation for the model. This involves several steps, including parsing the PDB file, extracting atomic coordinates, and creating masks for different regions of the protein. The final output is a batch of data that can be fed into the diffusion model, as shown in Figure 3.1. The diffusion itself will operate only on the backbone atoms. However, it does use the context of the full structure. Then after diffusion the new backbone is available. Diffab will use the original template complex and put only the generated backbone atoms in it's place (for single region optimization). Next it is saved as a new PDB file. Finally, an external tool like PyRosetta [7] can be used to perform side-chain packing.

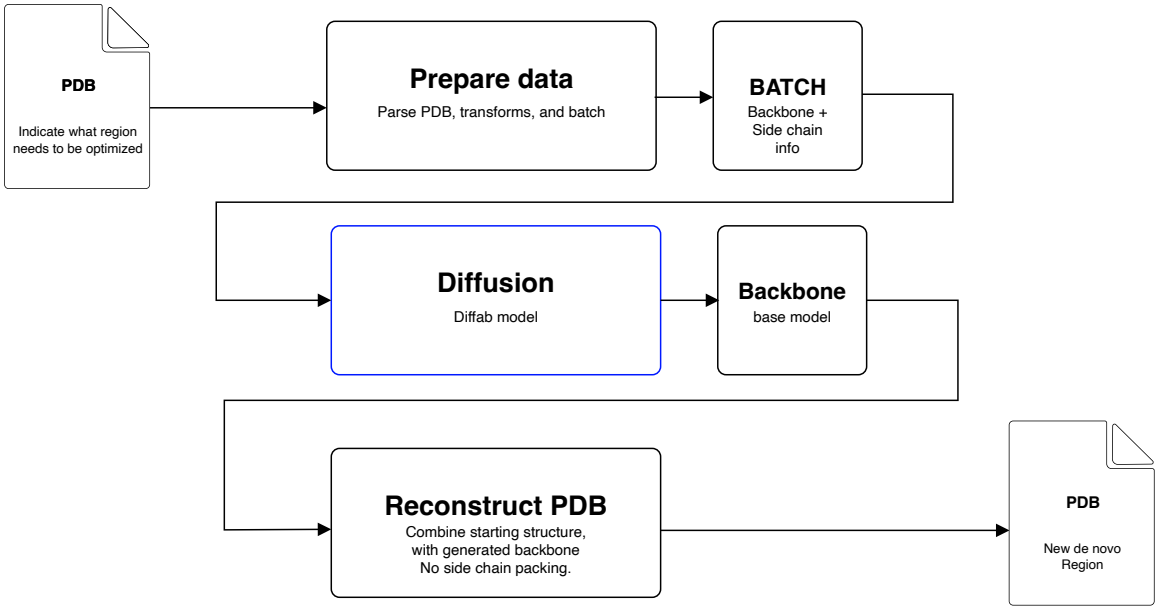


Figure 3.1: Sampling with Diffab

Zooming in on the data preparation a lot of transforms happen before a batch is ready to be fed trough the model. Figure outlines the whole flow from PDB to batch, with a short explanation for each step. [1]

[1] Tip: in the pdf you can zoom in on this is text. Most diagrams in this work are zoom ready.

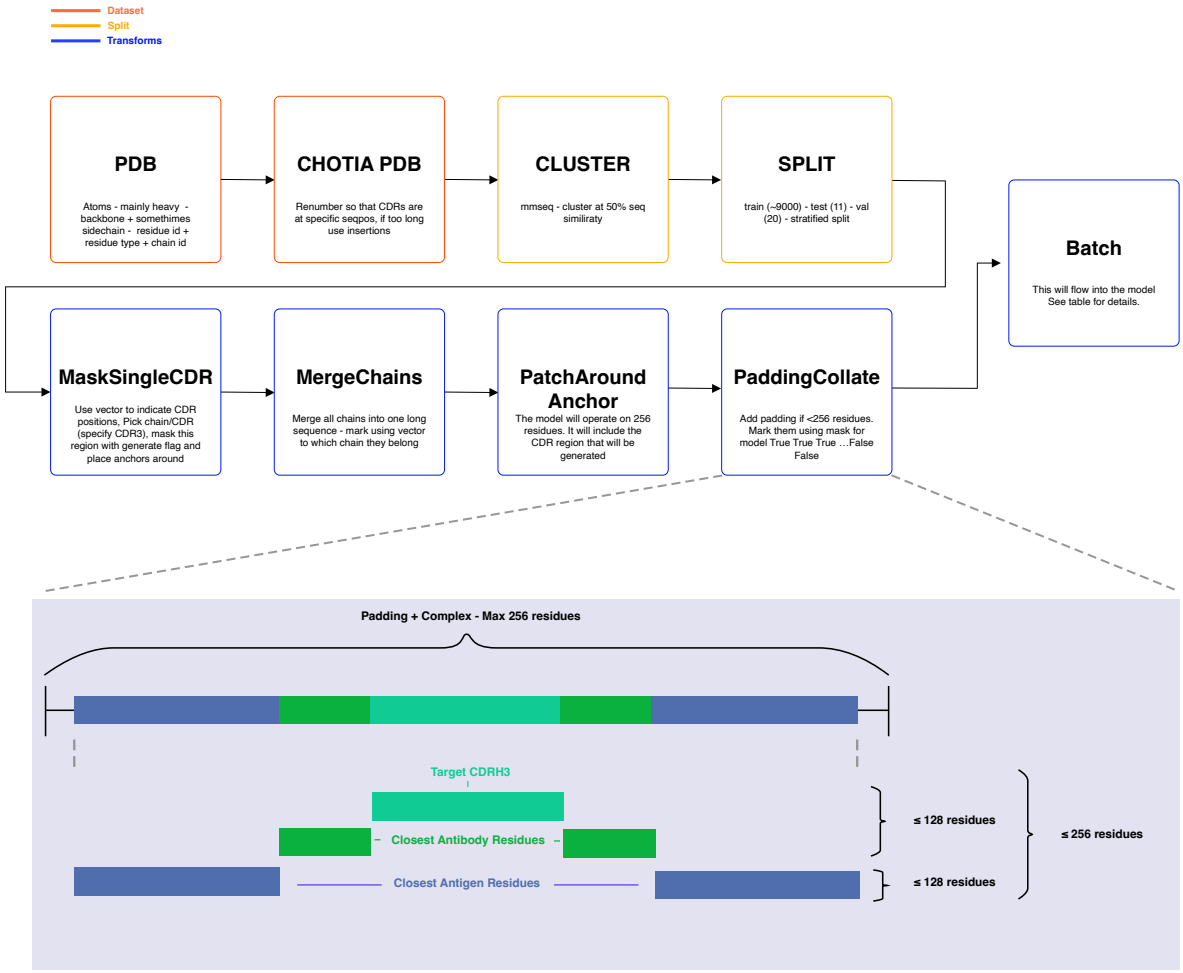


Figure 3.2: Data preparation pipeline

Table 3.1 outlines the final structure of a batch. (This table was also used as a reference during implementation.). This batch is now ready for diffusion.

Key	Purpose	Shape	Example Value
chain_id	Chain identifier	(256, batch_size)	[H, H, L, A]
icode	Insertion code	(256, batch_size)	[, A, ,]
chain_nb	Chain index	(batch_size, 256)	[0, 0, 1, 2]
resseq	Residue sequence number (PDB)	(batch_size, 256)	[1, 2, 1, 1]
res_nb	Internal residue index	(batch_size, 256)	[0, 1, 2, 3]
aa	Amino acid type (as integer code)	(batch_size, 256)	[0, 4, 7, 1]
pos_heavyatom	Heavy atom coordinates	(batch_size, 256, max_atoms, 3)	[25.76, 6.87, 3.76]
mask_heavyatom	Mask for used heavy atoms	(batch_size, 256, max_atoms)	[true, true, false]
generate_flag	Mask for residues to generate	(256, batch_size)	[false, true, true, false]
cdr_flag	Mask for CDR residues	(256, batch_size)	[false, true, true, false]
anchor_flag	Mask for anchor residues	(256, batch_size)	[true, false, false, true]
fragment_type	Residue type (heavy/-light/antigen)	(256, batch_size)	[0, 0, 1, 2]
origin	Patch origin (for centering)	(batch_size)	[12.3, 8.7, 5.1]
patch_idx	Indices mapping patch to full structure	(256, batch_size)	[0, 1, 2, 5]
mask	Mask for real vs. padded positions	(batch_size, 256)	[true, true, true, false]

Table 3.1: Batch dictionary keys, their purpose, shapes, and example values.

3.3.2 Diffusion process implementation

The sampling process of Diffab is at high level very similar to the sampling procedure discussed in the literature review on diffusion models. Figure 3.3 outlines the flow trough this epsilon net.

Like in traditional diffusion models the denoising process happens in a series of steps. In each step the added noise is predicted using the epsilon net and removed from the current noisy sample using a denoiser. This is done iteratively until the final denoised sample is obtained. Remember that denoising is the reverse process. It happens from times step T to 0. Thus when sample t enters the epsilon net and is then passed to the denoiser the sample t_{-1} is predicted.

The epsilon net takes as input the current noisy sample. For diffab this is represented as the backbone position, the orientation and the amino acid associated for each residue. Additionally a context embedding is passed to the epsilon net. Finally also the current time step is included as input. In this way we don't need to use a separate epsilon net for each time step. This is an important factor in making diffusion models work, since weights can be shared across time steps.

The context embedding contains:

- **Residue features**: Information specific to each residue. This includes the amino acid type, dihedral angles, masking information (for example when not used, when generated)
- **Pairwise features**: Information about the interactions between pairs of residues. This for example includes angles between backbone atoms.

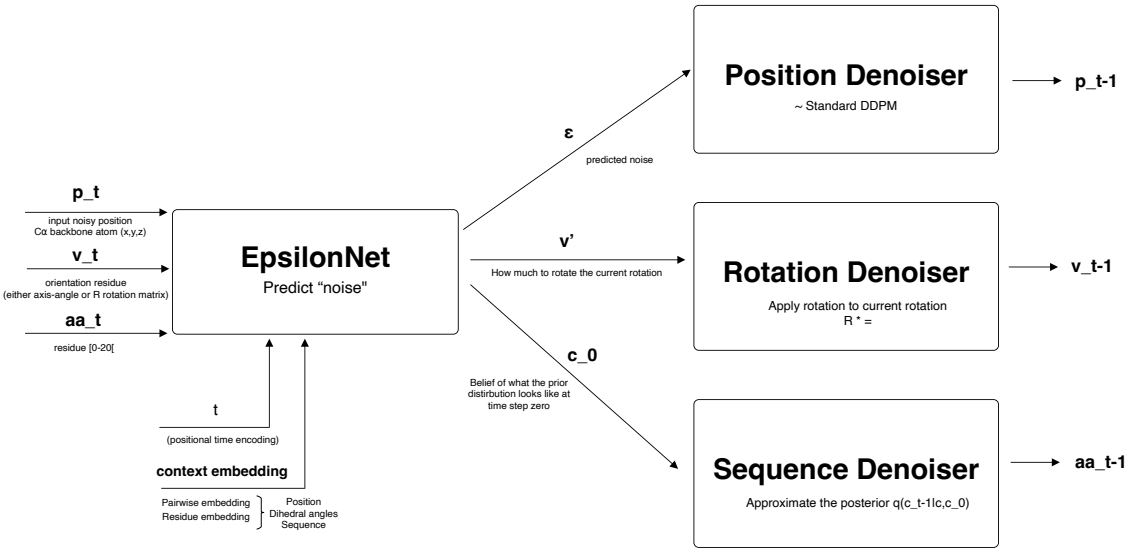


Figure 3.3: Denoising in Diffab

A key in understanding this epsilon net is that unlike images where the denoising happens on all pixels, in diffab denoising happens for each residue in the sequence separately. Thus the denoiser only works in the case of position only on the x,y,z coordinate.

Whereas for images it would not operate on a flattened or 2D representation of all pixels. So for $10x10$ grayscale this would amount to 100 values. In diffab only one pixel is considered. At first glance this might seem odd, and it might not seem to work since it does not keep in mind the rest of the structure. However, remember that a context embedding is also passed to the denoiser. This is like a summary of the rest of the structure. The denoiser uses an attention mechanism internally and can thus attend to all residues in the sequence. For each residue the context of the whole structure is passed along and thus denoising does not happen in a vacuum. The big benefit is that this makes the model much more efficient. The number of parameters can be much lower. This is crucial since compared to image datasets that often have million of samples our training dataset has only around $9k$ samples. This again stems back to the idea of weight sharing, weights are shared across residues. Additionally parallelism is increased since denoising happens not only across the batch dimension but also across residues.

The epsilonnet of diffab has three different heads (the last part of the neural network):

- **Position head**: This head is responsible for predicting the noise added to each position. It thus predicts ϵ like standard diffusion.
- **Orientation head**: This head predicts the next orientation that has to be applied.
- **Amino acid head**: This head predicts a categorical distribution over the amino acid types for each residue. The distribution represents the prior belief of the original distribution.

Next unlike traditional diffusion now a separate denoiser is used for each head. For the position this denoiser operates similarly to original diffusion.

For the orientation a special denoiser is used. Conceptually it works by applying rotation after rotation. Imagine a cube in step T we start from an initial rotation, the next prediction says how much to rotate this cube, and so on. If we sequence all these rotations we get the final rotation.

For the amino acid denoising a categorical denoising is done. This denoiser uses both the current amino acid sequence, and the predicted prior c_0. First each amino acid is turned into a categorical distribution by using a one-hot encoding which creates c_t. Then the conditional distribution is calculated using these two which mimics $q(c_{t-1}|c_t, c_0)$. When calculating both are mixed with a flat categorical distribution since this mimics the forward diffusion process.

After the conditional distribution has been calculated, the next amino acid is sampled from this distribution. This sampled amino acid is then used in the next denoising step.

Finally after all denoising steps are done, the final denoised sample is obtained. This sample contains the predicted backbone positions, orientations, and amino acid types for each residue in the sequence.

3.4 Terminology

Throughout this work, several terms are used frequently. To ensure clarity, the following definitions are provided:

- **Paper model/results**: The original Diffab model, using the weights provided by the authors.

- **Original model**: The original Diffab model, but retrained from scratch using the SabDab dataset. No changes made to the codebase.
- **Base model**: The DiffAb model trained on the CDR3 region with the new data split and sampling strategy.
- **Aux model**: The auxiliary model trained on synthetic data generated by the base model.
- **DGAD (framework)**: The proposed framework that integrates negative guidance into the diffusion process for improved antibody design. DGAD stands for **D**eep **G**enerative **A**ntibody **D**esign.

Chapter 4

Methodology

The methodology chapter cover the key steps that were taken to improve the starting Diffab model [22]. This includes refining the data split, enhancing the sampling strategy, and implementing negative guidance as a proxy to augmenting the dataset thanks to an auxiliary model trained on synthetic data.

4.1 Improving the datasplit

The original Diffab model uses a datasplit with a test set of 11 samples, and a validation set of 20 samples. To the best of our knowledge. All the other samples are used for training.

The split happens on cluster level. This means that all samples from a single cluster are either in the training, validation, or test set, but not in multiple sets at once. This is to make sure the split is as much i.i.d as possible. The clustering in diffab happens with *MMseqs2* [32] at 50% CDR sequence similarity.

Although keeping the training set as large as possible makes sense to train the diffusion model since the dataset is already quite small this inhibits us to have a test set that has statistical significance when comparing different techniques.

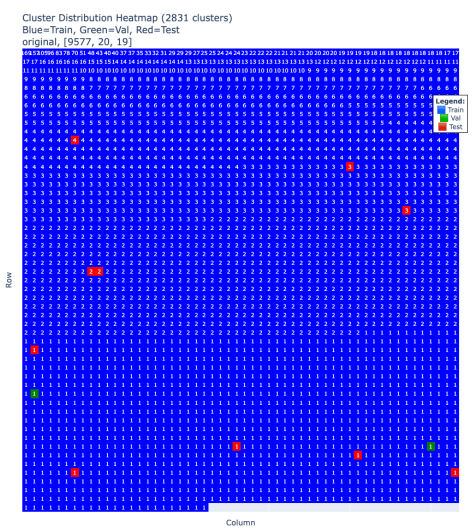
Therefore a new datasplit was created. A split is defined as : $[ratio_{train}, ratio_{val}, ratio_{test}]$

Creation of the new split To create the datasplit a few approaches were considered:

- **Manual**: You could in theory pick your split manually like in the original paper. However, this doesn't allow for seed experimentation, and is labour intensive.
- **Basic**: The basic split happens on cluster level if we want a split like $[0.80, 0.10, 0.10]$. Pick for each cluster randomly to what set it belongs, with these probabilities. Without replacement. Although this will not exactly be equal to the intended ratios across samples it will be close enough.
- **Heuristic**: Using the previous approach the clusters are split according to our intended distribution however some sets might by accident receive a large cluster while other might get small ones. To have more balance we could first sort the clusters by size, then group clusters of equal size together, and finally within each group pick the clusters randomly according to the ratios. When it is uneven we assign them to train. This approach was tested but needed a lot of further tweaking to get a good split for the bigger clusters (since there are not a lot of them)

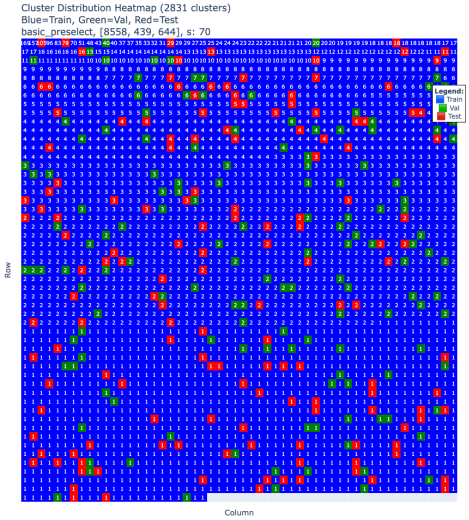
- **Hybrid**: The basic split showed similar results to the heuristic ones. However, picking a good split for the bigger clusters was difficult and depended a lot on the seed. With the hybrid approach only the big clusters were split manually, and the rest was done randomly with a seed. Moreover, the test samples of the original Diffab paper are always included in test, this can be useful when comparing results. In the end this method made sure that the test and val set had some big clusters without taking away too many samples from train. While retaining the benefit of using different seeds. **This strategy was eventually chosen.**

Figure 4.1 shows the difference between the original split and the base split. The original split is used in the original model (for comparison), and the base split is used in the base model of this thesis.



(a)

Before Original split: the base split has only a few samples in the test and val set. It ignores large clusters, and the test samples are hard coded.



(b)

After Base split: this is the final base split. It uses a hybrid approach; the big clusters are split manually (first two rows). The rest are split randomly using a seed. The test and val set are more representative of the overall dataset, while retaining enough samples to train.

Figure 4.1: Comparing the original split and the base split. Each cell in the matrix represents a cluster. The number in the cluster is the amount of samples. The largest clusters are at the top left and it decreases from left to right and top to bottom. The color indicates if the cluster is in the train, val or test set.

4.2 Improving sampling

Although, the diffab paper uses a clustered split, it does not use this information during training. Each sample is picked uniformly at random from the training set. Moreover, epochs are not employed and it just picks randomly this means some samples will be repeated more often than others and in theory some might never be seen by the model. This means that if a cluster is large it will be overrepresented during training, while small clusters will be underrepresented. This could lead to overfitting on the large clusters, and poor generalization on the small clusters.

The other extreme sampling strategy is: First randomly shuffle the clusters, then pick one sample from each cluster in a round robin fashion. This way each cluster is represented equally during training. However, this means that some samples of the smaller clusters will be seen more often than others, and the larger clusters will not be fully utilized. This could lead to underfitting on the large clusters, and overfitting on the small clusters.

Thus, a **balanced approach** is needed. Moreover, it would be useful to introduce the idea of epochs. This would allow us to easily resume training from a certain epoch, and it would make sure all training data is seen.

4.2.1 Balanced sampling

There are two extremes:

- Original sampling: samples are picked uniformly at random from the training set.
- Round robin sampling: each cluster is represented equally during training.

Sampling can be represented as a probability distribution over the clusters:

- The original sampling strategy samples according to a size proportional distribution. i.e clusters with a larger size have more chance of being picked.
- The round robin sampling strategy samples according to a uniform distribution over the clusters.

Thus we want to interpolate between these two distributions.

4.2.2 Linear

The most easy and intuitive way to this is using linear mixing:

$$p_{mix}(c) = \alpha p_{original}(c) + (1 - \alpha) p_{uniform}(c) \tag{4.1}$$

where

$p_{\text{mix}}(c)$	is the sampling probability for cluster c used by the sampler,
$p_{\text{original}}(c) = \frac{n_c}{\sum_{j=1}^{K} n_j}$	is the size-proportional (raw-frequency) distribution,
$p_{\text{uniform}}(c) = \frac{1}{K}$	is the uniform distribution over clusters,
$\alpha \in [0, 1]$	is the mixing coefficient controlling the balance,
$\alpha = 1$	recovers p_{original} (large clusters dominate),
$\alpha = 0$	recovers p_{uniform} (all clusters equally likely),
$0 < \alpha < 1$	interpolates between the extremes, reducing dominance of large clusters,
n_c	is the number of samples in cluster c,
K	is the number of clusters.

4.2.3 Exponential (Skip-Gram model, word2vec-style)

Another alternative is to use a sampling procedure like the one used in word2vec [19], which employs negative sampling [[24], [19]] to adjust the probabilities of selecting different clusters. When creating word embeddings a similar problem is faced. There are words that appear a lot more than others in corpuses of text and thus are overrepresented. For example, common words like "the" or "and" occur much more frequently than rarer words. This is analogous to our clusters where large clusters may dominate the training process.

When sampling words there are two extremes:

- Sampling uniformly from the vocabulary. Too much emphasis on rare words.
  - This is analogous to original sampling from clusters. Where we sample as much from large clusters as from small ones. Samples from small clusters will be underrepresented.
- Sampling from the vocabulary according to word frequency: frequent words like the will be overrepresented:
  - This is analogous to uniform sampling from clusters. Where we sample as much from large clusters as from small ones. Samples from small clusters will be overrepresented.

$$p_{\exp}(c) = \frac{(n_c)^\alpha}{\sum_{j=1}^{K}(n_j)^\alpha}. \tag{4.2}$$

where

$p_{\exp}(c)$	is the sampling probability for cluster c used by the sampler,
n_c	is the number of samples in cluster c,
K	is the number of clusters,
$\alpha \in (0, 1]$	tempering exponent(e.g., $\alpha = \frac{3}{4}$),
$\alpha = 1$	recovers $p_{\text{original}}(c)$ (large clusters dominate),
$\alpha \to 0$	tends to $p_{\text{uniform}}(c) = \frac{1}{K}$ (all clusters equally likely),
$0 < \alpha < 1$	reduces dominance of large clusters and boosts small ones.

This technique is very much an empirical one, there is no true mathematical proof why this works better to the best of our knowledge. The authors found that a value of $\alpha = 0.75$ works well in practice for word embeddings, and this was also used as a starting point for our experiments.

One possible intuition is that when this form of mixing is smoother and more gradual than the linear one. Since in the distribution each cluster size is divided by the sum of all cluster sizes raised to the power of α. This means that large clusters will have a different scaling behavior compared to small clusters. Making it more smooth.

4.2.4 Implementation

To implement this a custom PyTorch data sampler was created. It can be found in `DGAD/diffab-main-edit-1/diffab/datasets/sampling.py`.

The sampler has the following features:

- Coverage guarantee: every sample is seen at least once per epoch.
- Oversampling: large clusters are sampled more often, according to a tunable probability distribution.
- Flexible mixing: choose between linear interpolation and exponential scaling for cluster probabilities.
- Fixed epoch size: total samples per epoch $=$ multiplier $\times$ dataset size.
- Reproducible sampling: uses local random generators and a seed per epoch.

Additionally a custom EpochTracker was build that allows the use of steps, and more control over the training process while using the classic *for epoch ; for batch in data* loop. It wraps the dataloader and provides an iterator for the epochs all the rest is handled in the background. See sampling.py for more details.

The sampler uses sampling_balance as a hyperparameter:

- `sampling_balance` (float): Controls balance between original and uniform sampling.
  - 1.0: Original cluster distribution (proportional to cluster sizes).
  - 0.0: Uniform distribution across clusters.
  - 0.75: Recommended starting point (Word2Vec standard for exponential).

4.3 Base model characteristics

The following section briefly reviews the characteristics of the base model.

Aside from the new split, and sampling strategy the base model was used for the other techniques is trained by only noise/denoising the CDRH3 region. Since within this work we only focus on this region for simplicity. Sometimes the CDRH3 might not be present for example in the case of nanobodies. Then a fallback mechanism is used, where the model can use the CDRH1 or CDRH2 regions instead. If those are unavailable the LCDR regions are used. Augmentation is also used during training by slightly altering the CDR lengths. This is done by randomly adding or removing residues from the CDR region. This makes sure the model does not overfit on a certain length, and can generalize better to different lengths. The augmentation was also present in the original model.

Finally, a version with all CDR regions noised/denoised was also trained for comparison. Covered in the results chapter.

The characteristics can be found in table 4.1.

TRAITS	Optimization of selected CDR loop (small local changes) for structure and sequence. • Mask desired CDR region; keep length fixed; rest of structure unchanged. • Diffusion model: noise/denoise positions, orientations, and sequence.
EXPERIMENTS	Focus on CDR3 results; sampling only for CDRH3 region.
SPLIT	**Train (0.90)**: 8558 **Val (0.05)**: 439 **Test (0.05)**: 644
SAMPLING BALANCE	Mode: linear / exponential Range: [1, 0] (1 = original cluster distribution, 0 = uniform) Recommended: 0.75 (Word2Vec exponential standard)
SEED	70

Table 4.1: Base model characteristics

4.4 Negative Guidance for DGAD

Negative guidance is a technique used to steer the generative process away from undesired outputs. In the context of DGAD, we employ negative guidance to improve the quality of the generated antibody structures by discouraging the model from producing unrealistic or low-quality designs.

In the paper "Self-Improving diffusion models with Synthetic data (SIMS)", the authors propose a method to enhance the training of diffusion models using self-synthesized data. This approach provides negative guidance during the generation process, helping to steer the model away from non-ideal synthetic data and towards the ground truth data distribution.

Recall that our hypothesis was if the technique proposed in SIMS is also applicable to generative antibody design.

In the following section we will explore how this technique was applied to the Diffab model.

4.4.1 General

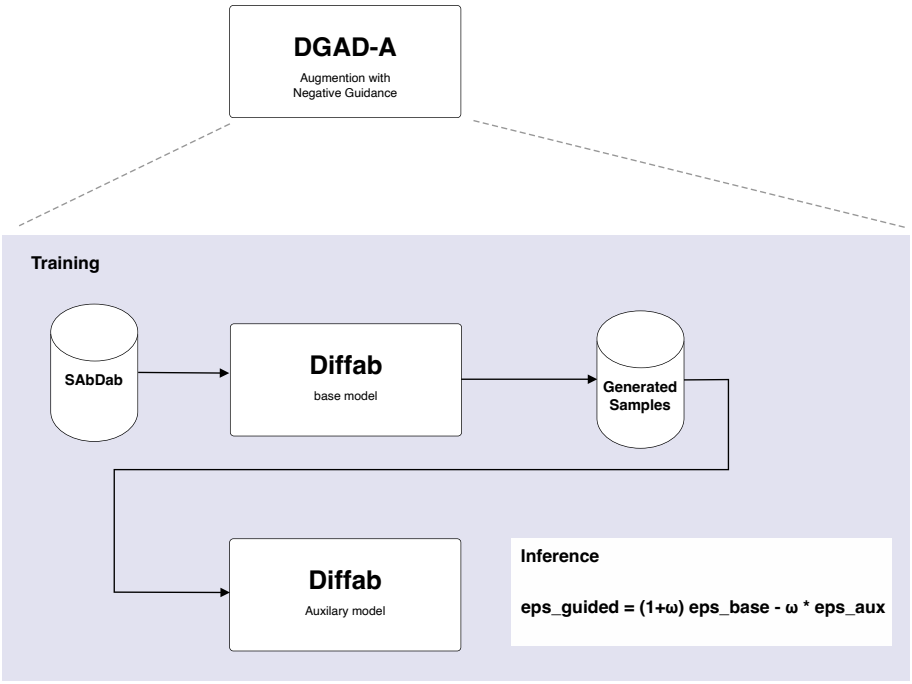


Figure 4.2: DGAD-A is the framework that employs negative guidance to improve the quality of generated antibody structures. The diagram shows the training procedure of training an auxiliary model with self-synthesized data and the inference phase where guidance is applied by using both models.

The method involves training two models: a base model and an auxiliary model. The base model is trained on the original dataset, while the auxiliary model is trained on synthetic data generated by the base model. During inference, the predictions from both models are combined to guide the sampling process. The combined prediction is given by:

$$\epsilon_{guided} = \epsilon_{base} - \omega(\omega\epsilon_{aux} - \epsilon_{base}) \tag{4.3}$$

$$\epsilon_{guided} = (1 + \omega)\epsilon_{base} - \omega\epsilon_{aux} \tag{4.4}$$

Rewriting it in the last way leads to a more efficient implementation.

Note that in the SIMS paper they do this on the score function instead of the noise prediction. This is equivalent since the score function can be expressed in terms of the noise prediction (by scaling with a linear factor). A score function is an alternative parameterization of diffusion models that directly estimates the gradient of the log probability density function of the data. This parameterization is not used in this work.

4.4.2 Creating the auxiliary model

A naive way to train an auxiliary model is to generate with the base model a new sample and pass it to the auxiliary model to train. You could either generate a new sample each time (this is shown in figure 4.3) or form a dataset at the same time and reuse. The first approach makes training slow. The second although faster isn't ideal either since two models need to be loaded and there is still overhead.

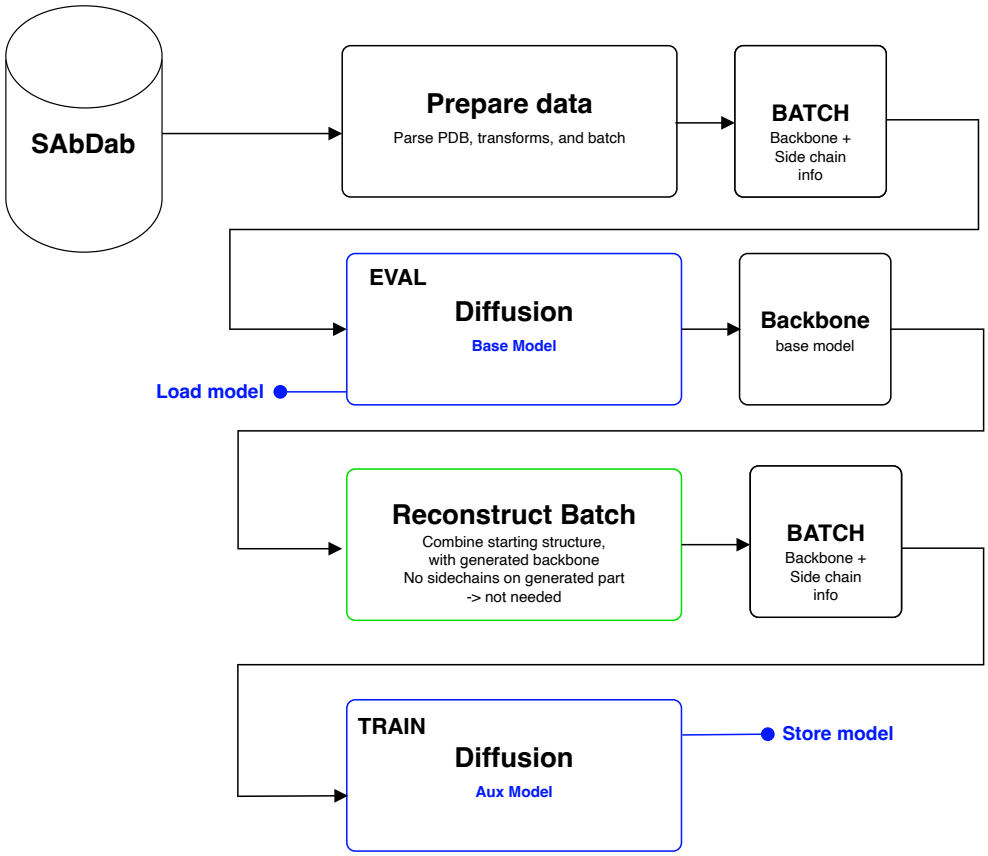


Figure 4.3: The training procedure for the unique auxiliary model. Each sample is generated from the base model during the training of the auxiliary model. This technique works but makes training too slow.

Therefore, to train the auxiliary model a two step approach was used:

1. First the base model is used to generate synthetic samples. These are saved in a data base. Rather than saving the processed PDB files, the direct tensor representation used by the model is saved. This eliminates the need to reprocess the data each time. It also avoids processing artifacts. This saves processing time when training the auxiliary model and also makes sure that data is not organized differently by processing artifacts. The data is saved in LMDB [8]. LMDB is a key-value database that uses memory mapping to store data on disk. This makes it very fast to read data during training.

    - Note internally the model operates on backbone representation (position, rotation and sequence). When a sampling step is done during training diffab expects the full structure (including sidechains, masks, etc) as input. However after sampling the model only returns the backbone, afterwich it is reformatted to the full structure. Therefore an adapted sampling procedure was created that outputs the correct tensor representation for input into the model. This way the output of one model (the baseline) can be used directly as input for the auxiliary model.

2. Next the auxiliary model is trained on the synthetic data. A special dataloader is used that will read the data from the LMDB database and reconstruct the tensor representation used by the model.

    - When training the auxiliary model two different regimes can be used. It can either be trained from scratch so with random initialized weights. Or it can be trained starting by initializing it with the weights from the base model. The authors of SIMS [3] set forward this method as the best one. They say to train for around 40% of the total steps used in the base model. In our case this is 80K steps.

This approach is outlined in figure 4.4.

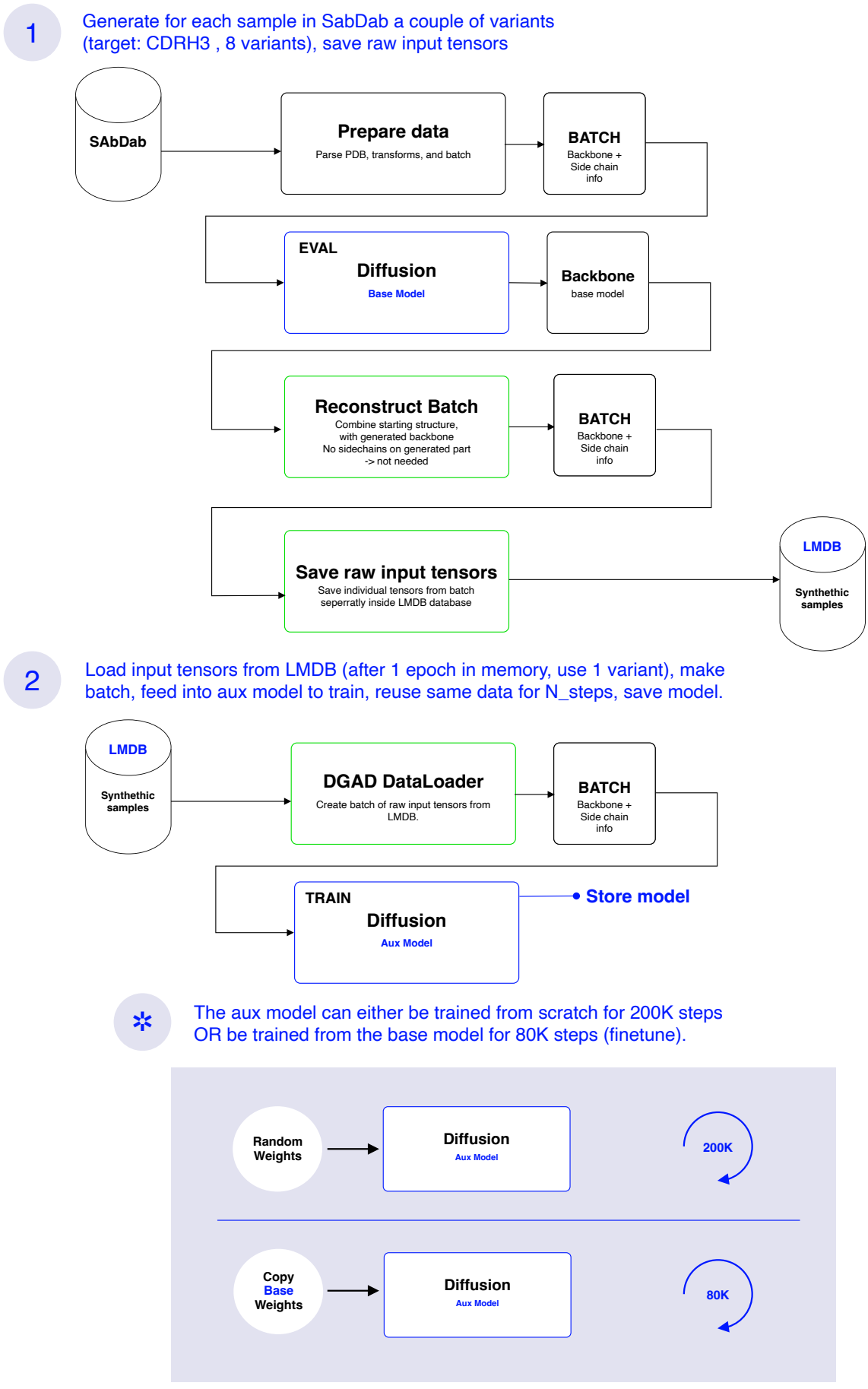


Figure 4.4: The training procedure for the fixed auxiliary model. The auxiliary is trained on a pre-computed dataset of synthetic samples.

4.4.3 Guidance on the position

To evaluate the effectiveness of SIMS, the technique was applied on the position of each residue. This was chosen since the diffusion process on the position is similar too the diffusion process on images. The position is represented as a (x, y, z) coordinate in 3D space. Applying guidance on the predicted noise for the position is a euclidean operation.

Within the diffab implementation the EpsilonNet predicts both position, rotation and amino acid type. During inference two epsilon nets are used, one for the base model and one for the auxiliary model. The guidance is only applied on the position part of the prediction. One drawback is that two models need to be loaded into memory. Before the position is denoised guidance is done on the predicted noise as shown in figure 4.5.

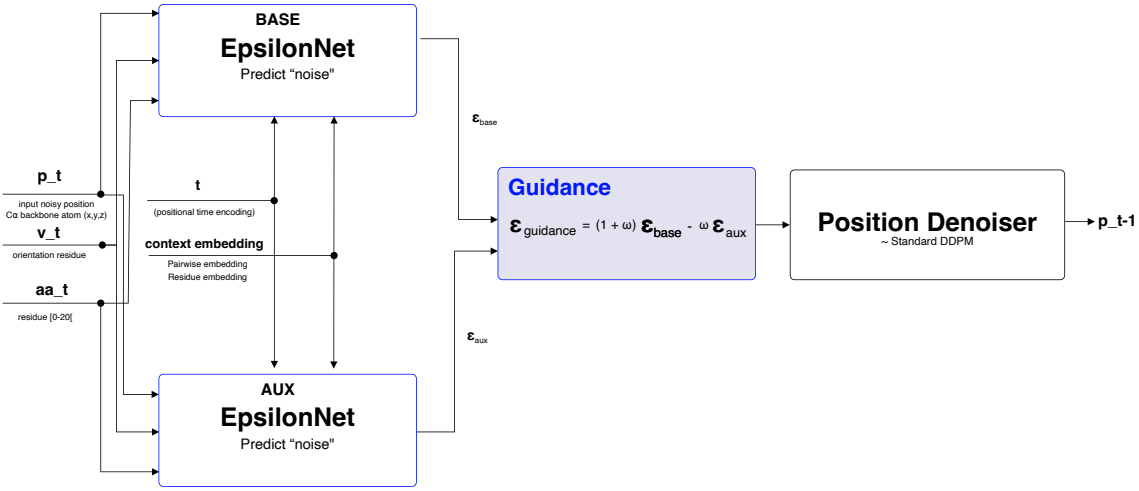


Figure 4.5: The denoising procedure for doing guidance on the position. The previous position p_t is denoised by the base model and the aux model. The outputs of both models are the predicted noise that was added to the previous position. Then this noise is used to compute the current position within the denoiser.

.

4.4.4 Guidance on the sequence

Unlike the position, the sequence is represented as a categorical distribution over the 20 amino acids. Therefore, the guidance needs to be applied on the logits of this distribution or the probabilities.

Listing 1: Code snippet for guidance on the position (simplified)

```
# For the base model
base_v_next, base_R_next, base_eps_p, base_c_denoised = self.base_eps_net(...)  # (N, L, 3), (N, L, 3, 3), (N, L, 3)
# For the aux model
aux_v_next, aux_R_next, aux_eps_p, aux_c_denoised = self.aux_eps_net(...)# (N, L, 3), (N, L, 3, 3), (N, L, 3)
...
sim_eps_p = ((1+omega_pos_t) * base_eps_p) - (omega_pos_t * aux_eps_p)
...
p_next = self.base_trans_pos.denoise(base_p_t, sim_eps_p, base_mask_generate, base_t_tensor)
...
```

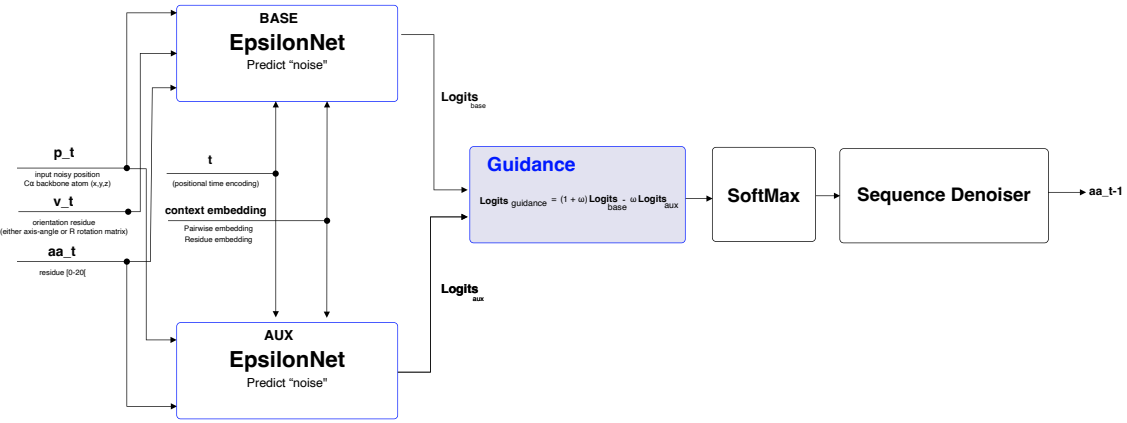


Figure 4.6: The denoising process when doing guidance on the sequence.

For this the epsilon net was adjusted to output logits without a softmax layer. Next the guidance is applied on the raw logits, and then the softmax is applied. Another version that does the guidance directly on the probabilities rather than the logits has also been implemented. The effect that guidance has in this case is that it might shift the distribution to favor certain amino acids more than others when taking the argmax. Figure 4.6 illustrates the guidance process in a similar way as for position. Finally a code snippet is provided in listing 2.

Listing 2: Code snippet for guidance on the sequence (simplified)

```
# For the base model
base_v_next, base_R_next, base_eps_p, base_c_denoised = self.base_eps_net(...)  # (N, L, 3), (N, L, 3, 3), (N, L, 3)
# For the aux model
aux_v_next, aux_R_next, aux_eps_p, aux_c_denoised = self.aux_eps_net(...)# (N, L, 3), (N, L, 3, 3), (N, L, 3)
...
# Guidance on sequence
if self.config.dgad.sims.seq:
    print('guidance on sequence')
    sims_c_logits = ((1 + omega_seq_t) * base_c_denoised) - (omega_seq_t *  aux_c_denoised)
    #now do the softmax
    sims_c = sims_c_logits
    if self.config.dgad.sims.guidance_seq_on_logits:
        sims_c = F.softmax(sims_c_logits, dim=-1)
else:
    sims_c = base_c_denoised
...
_, s_next = self.base_trans_seq.denoise(base_s_t, sims_c, base_mask_generate,  base_t_tensor)
...
```

4.4.5 Guidance on the orientation

Applying rotational guidance is inherently challenging due to the nature of 3D rotations. A naive a approach would be to mix the output of the rotational denoiser of the base model with the output of the auxiliary model just like the positional guidance. However, the position denoiser does not predict the added noise but the next rotation to be applied to the previous one. When denoising the rotations are applied one after the other, in a sequence. Say you have a cube ant it's initial rotation in space, after the first denoising step you will rotate it a bit by that rotation. Next for the second denoising step, you will apply the next rotation to the already rotated cube, and so on. This sequential application of rotations makes it difficult to directly mix the outputs of the two models.

$$R_{final} = R_n \cdot \cdots R_2 \cdot R_1 \qquad \text{sequential denoising of rotation} \tag{4.5}$$

Additionally, mathematically we want to find a mix of two rotations. This can not be done in the same way as in euclidean space for positions. Therefore another approach is needed.

This has not yet been fully explored and remains an open question for future research. However, some progress was already made into how this can be achieved. This is still a conceptual approach and provided as is.

Conceptual approach using Lie algebra. This approach has been an ad-hoc one. It is based on advisory meetings, and was also used to experiment with the current LLM models as a way to assist within a subject that is outside my expertise.

These were not one shot attempts, but rather an iterative process of refining the approach based on feedback and insights gained from discussions with the models. These discussions were always aimed on making sure the methods and techniques were understood.

To ground these models they were conditioned on the work by **Lie Groups for Computer Vision** by **Eade** [11]. Since this mathematical basis forms one of the keys in solving this problem. Additionally, I used context from the codebase of this work to further improve results.

This was very much a new way of working and an interesting experiment. My belief is that in the future this will lead to engineering approaches on a higher level abstract space. However it still required significant effort and is still not ready to be fully trusted.

The thought process is as follows:

- Conceptually we want to interpolate between two rotations. This can be done using lie algebra in tangent space. In this space we can apply guidance as a euclidean operation. Important to note is that the current rotation is applied on the sequence of rotations before it, therefore the predicted rotations need to be anchored at the current rotation.

The rotational guidance does not yet offer any benefits and makes model performance worse to using position alone. Many hypothesis can be made why this is but a few:

- Doing guidance on position moves atoms in space, but doing it on rotation too. They might not be mixing well together.

- The implementation is still lacking some fundamental aspects to making this work.
- The rotation might overshoot.

Doing rotational guidance seems inherently difficult after numerous iterations. Maybe the approach in solving this should be a different one that is less error prone and simpler. One approach that seems interesting to test in the future is: Use latent diffusion [28]. Convert the rotation and/or position to a latent space that noises/denoises from a standard multivariate gaussian. Do the guidance there on the predicted noise.

Listing 3: Code snippet for guidance on the orientation. (concept)

```
def lie_algebra_guidance_so3_anchored_clipped(base_v, aux_v, guidance_strength, R_t):
"""
SO(3) guidance using Lie algebra, anchored at the CURRENT state R_t.
Args:
    base_v: (N, L, 3)  axis-angle of base model's x0 prediction.
    aux_v:  (N, L, 3)  axis-angle of aux  model's x0 prediction.
    guidance_strength (float or tensor broadcastable to (N,L,1)): omega.
    R_t:    (N, L, 3, 3) rotation matrices of the CURRENT noisy state.
Returns:
    guided_v: (N, L, 3) axis-angle of guided x0 prediction (like your original).
"""
# 1) Convert axis-angle vectors to rotation matrices (predictions)
base_R = so3vec_to_rotation(base_v)  # (N, L, 3, 3)
aux_R  = so3vec_to_rotation(aux_v)   # (N, L, 3, 3)
# 2) Express both predictions as displacements from the CURRENT state R_t
#    u_* are axis-angle vectors in T_{R_t}SO(3): u = log(R_t^T * R_pred)
Rt_T = R_t.transpose(-2, -1)
u_base = rotation_to_so3vec(Rt_T @ base_R)   # (N, L, 3)
u_aux  = rotation_to_so3vec(Rt_T @ aux_R)    # (N, L, 3)
# 4) Combine in the tangent at R_t
omega = guidance_strength
u_guided = (1 + omega) * u_base - omega * u_aux
#4.0 clip: ‖u_guided‖ < (1 + omega) * ‖u_base‖.
# u_base, u_guided : (N, L, 3)  axis-angle tangents at the current state R_t
# 4.1) Compute base step length ‖u_base‖ for each (N,L) item.
# .norm(dim=-1)     → sqrt(x^2 + y^2 + z^2) over the 3-vector
# keepdim=True      → keeps shape as (N, L, 1) so it broadcasts cleanly later
# .clamp_min(1e-8)  → replaces values < 1e-8 with 1e-8 (avoids divide-by-zero)
nb = u_base.norm(dim=-1, keepdim=True).clamp_min(1e-8)   # (N, L, 1), radians
# 4.2) Compute guided step length ‖u_guided‖ with the same guards/shapes.
g  = u_guided.norm(dim=-1, keepdim=True).clamp_min(1e-8) # (N, L, 1), radians
# 4.3) Build the target radius tau = (1 + omega) * ‖u_base‖.
# - 'omega' can be a scalar or a tensor that broadcasts to (N, L, 1).
# - Broadcasting: PyTorch auto-expands singleton dims to match (N, L, 1).
tau = (1 + omega) * nb                                   # (N, L, 1), radians
# 4.4) Compute per-vector scale s = min(1, tau / ‖u_guided‖).
# - If ‖u_guided‖ <= tau  → tau/g >= 1 → s = 1 (no change).
# - If ‖u_guided‖ >  tau  → tau/g < 1 → s < 1 (shrink u_guided to lie on radius tau).
# .clamp(max=1.0) ensures we never *increase* the step (s=< 1).
s = (tau / g).clamp(max=1.0)                             # (N, L, 1) in (0,1]
# 4.5) Apply the scale per (N,L,3) tangent vector via broadcasting over the last dim.
# Multiplication is elementwise; s (N,L,1) expands to (N,L,3).
u_guided = u_guided * s
# 5) Map back to SO(3) from the CURRENT state
guided_R = R_t @ so3vec_to_rotation(u_guided)  # (N, L, 3, 3)
# 6) Convert back to axis-angle for downstream use
guided_v = rotation_to_so3vec(guided_R)        # (N, L, 3)
return guided_v
```

4.4.6 Combining position, orientation and sequence guidance

Thus far guidance was explained on the position, orientation and sequence separately. However one of the key aspects of our approach is to combine these different types of guidance.

Individual Guidance Strengths

When guidance is applied the guidance strength ω will determine how much the guidance will influence the model's predictions. When combining the different types, individual guidance strengths can be used:

$$\begin{aligned} \omega_{\text{pos}} &: \text{ guidance strength for position updates} \\ \omega_{\text{rot}} &: \text{ guidance strength for orientation updates,} \\ \omega_{\text{seq}} &: \text{ guidance strength for sequence updates} \end{aligned}$$

It is useful to use individual strengths since each type will operate on it's own scale. For example the position guidance might need a higher strength to have a similar effect as the sequence guidance. When having the same one might hurt the other.

Ablations will be covered on different combinations between position, rotation and sequence, alongside different weight configurations.

Decay

In the beginning of the diffusion model more coarse grained adjustments will be made and the model will be more stable to guidance nudges. At the end of the diffusion process guidance nudges might actually make our solution worse. Therefore a decay factor is used to reduce the impact of the guidance as the diffusion process progresses.

For the decay factor a linear decay is used.

Linear guidance decay. Let T be the total number of diffusion steps and $t \in \{1, \dots, T\}$ the current step (counting down in the sampler, step 0 is the denoised complex). We set

$$\omega_t \;=\; \omega_0 \cdot \frac{t-1}{T-1}, \tag{4.6}$$

so that $\omega_T = \omega_0$ at the start and $\omega_1 = 0$ at the final step.
where

$$\begin{aligned} \omega_0 \in \mathbb{R}_{\geq 0} \quad & \text{is the initial guidance strength,} \\ T \in \mathbb{N},\ T > 1 \quad & \text{is the number of sampling steps,} \\ t \in \{1, \dots, T\} \quad & \text{is the current step (descending in time).} \end{aligned}$$

This is only a hypothesis, the effects will be studied in the result section.

Guidance Schedule

Building on the idea of a simple decay factor, it is possible to take guidance control a step further by introducing a guidance schedule. Such a schedule allows the guidance strengths to be adapted dynamically throughout the diffusion process, enabling more nuanced and flexible control over the influence of each guidance type at different stages.

Instead of decaying, guidance can for example only be applied in the middle of the diffusion process. Or only used on one of the components for a certain period, then switch over to the next component. In this way the components would not play a tug of war between each other.

The possibilities are endless, but one interesting schedule that was explored was first not doing guidance at all for the first 10 steps. Then increasing the guidance on the position for 10 steps keeping it steady for 20 and decreasing it again for 10 steps. Followed by the same scheme but now doing guidance on the sequence and finalizing by 10 steps of doing nothing. The rotation left out of this schedule due to it still being a conceptual approach that could lead to instability. This schedule can be seen in figure 4.7. The rationale behind this: in the first steps the model is left alone so it can establish a good baseline, next when more coarse grained adjustments are still allowed guidance is introduced, first on the positions and then sequence. Position first since the sequence depends on the backbone structure. Finally at the end the model is again left alone to make the final adjustments, since there guidance might be more harmful than helpful.

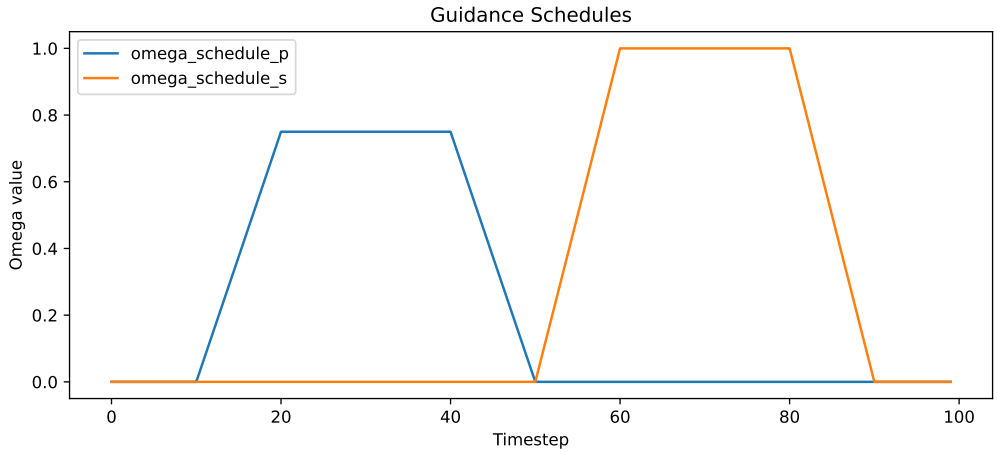


Figure 4.7: Guidance schedule for diffusion process

Chapter 5

Metrics

In this chapter, various metrics will be discussed that can be used to evaluate the performance of the implemented methods. First RMSD is discussed a metric to compare structures, next AAR a metric to compare sequences. Subsequently a discussion about the difficulty of verifying if these structures and sequences are truly of good quality. Finally log likelihood and pseudo log likelihood are covered. These are introduced as a proxy to evaluate the quality of the generated sequences.

5.1 RMSD: Root Mean Squared Difference

Root Mean Squared Difference (RMSD) measures the difference between two structures. The smaller this metric the closer both structures are to each other.

In the case of DGAD we compare the generated (X) and ground truth ($\hat{X}$) structure. Specifically the RMSD is calculated between the ground truth CDRH3 loop and the generated CDRH3 loop. Since the rest of the structure is left unchanged.

The Root Mean Squared Difference (RMSD) between two structures is defined as:

$S = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ and $\hat{S} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_N)$, where both structures have N residues and the i-th residue in S is paired with the i-th residue in $\hat{S}$ (i.e., the order defines the pairing), the Root Mean Squared Difference (RMSD) is defined as:

$$\mathrm{RMSD}(S, \hat{S}) \;=\; \sqrt{\frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2}. \tag{5.1}$$

where

$\mathbf{x}_i \in \mathbb{R}^3$	is the Cα coordinate of residue i in structure S,
$\hat{\mathbf{x}}_i \in \mathbb{R}^3$	is the Cα coordinate of residue i in structure $\hat{S}$,
N	is the number of paired residues being compared,

$\|\mathbf{x} - \mathbf{y}\|^2 = (x_x - y_x)^2 + (x_y - y_y)^2 + (x_z - y_z)^2$. denotes the squared Euclidean norm in $\mathbb{R}^3$.

Typically before this is calculated both structures are aligned using a Kabsch algorithm [17]. This ensures that the RMSD is not influenced by rotation or translation of the structures. The pairing between residues is done by aligning both sequences. However, the authors of Diffab calculate it a slightly different way. To the best of our knowledge,

they do not align the structures, but instead use a windowed approach and calculate it using a dynamic programming algorithm.

This windowed approach assumes that there is a shorter and a longer sequence (or both the same length). A window of the same length as the shorter sequence is matched to a subsequence of the longer sequence. Meaning that the initial alignment is kept but residues can be skipped.

You do this for all possible windows. You then compute the RMSD for each window. Finally you pick the smallest value. For the implementation the RMSD code of **Luo et. al.** [22] was used and integrated into the benchmark pipeline of DGAD. In this way it was ensured that the results were consistent with the original work.

This approach could lead to some ambiguity. However, the model only optimizes the CDRH3 region and only does small local changes. It does not influence the pose or position of the complex. Moreover, we keep the length the same as the ground truth. Thus, this approach is valid for comparing the generated and ground truth structures.

5.2 AAR: Average Amino Acid Recovery

The Average Amino Acid Recovery (AAR) metric measures how much alike two amino acid sequences are. In the case of DGAD we compare the predicted and ground truth amino acid sequences.

$$\mathrm{AAR} \;=\; \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\; s_i = \hat{s}_i \;]\,. \tag{5.2}$$

where

s_i	is the reference (ground-truth) residue at position i,
$\hat{s}_i$	is the generated residue at position i,
N	is the number of positions being compared,
$\mathbf{1}[\,P\,]$	is the indicator: 1 if predicate P is true, and 0 otherwise.

AAR can be calculated by taking the sum of all amino acid recoveries and dividing it by the total number of residues compared. An amino acid recovery occurs when an amino acid in the predicted sequence matches the amino acid in the generated one. This is done for each position. This metric can be multiplied with 100 to obtain a percentage. By dividing by the sequence length we can compare sequences of different lengths. Since the CDR regions have variable length this is important.

This metric is also reported by **Luo et. al.** [22].

Unlike RMSD this metric was implemented from scratch for this work alongside sequence extraction:

1. Sequence extraction
    (a) The model generates raw PDB files, all chains and there sequences are extracted.
    (b) The CDR regions are extracted according to chothia numbering. During extraction some residues may be missing (PDB files are not perfect, often based on experiments and some do not contain all residues). All of this is booked so calculation can account for this. Missing residues are not taken into account.

   (c) In the end all sequences, residue positions, chain ids, cdr ids, chains, CDR chains are stored in a structured json for each experiment.

2. AAR calculation is done, or any other sequence calculation. AAR is implemented according to the formula provided above.

5.3 Discussion

Although these metrics provide insight into how close we can get to our reference structures in the test set. It does not provide any guarantee into how plausible they are in reality. It does not account for the variability that diffusion models produce. Inherently the generative process of the diffusion model can be 'creative' an may find another interesting solution to the given complex. Even tough it does not match the reference complex it might actually be better since it used the whole training corpus internally and found a better solution. On the other hand the model might create totally infeasible structures. Both might be slightly different from the reference structure but one will be an improvement and the other will not. Determining wether or not a generated antibody antigen is a good candidate is an open problem. It needs to adhere to a lot of constraints like previously discussed. It needs to bind, be thermally stable etc.

Currently there is no easy way to know if the antibody is a suitable candidate. A lot of proxies exist to get insight into this and to get the full picture a combination of methods is possibly needed. The following list are some of these methods:

- Physical constraints: are they satisfied? Such as bond length, angles and steric clashes.
- Energy calculation: for example using PyRosetta. [7]
- Docking score
- Molecular dynamics simulations
- Using another model as a proxy.
  - Evaluate the likelihood according to another model (such as ESM [20])
  - Folding the generated sequence using a model such as AlphaFold [1] and compare the structure to the generated one. A type of self-consistency check.
- In vivo or in vitro testing.
  - This is the gold standard. But very expensive to do. This also exactly what these type of generative models aim to make more efficient.

Many of these would either have taken to much implementation time, background expert knowledge, or computation time. However, one class of evaluation stood out, and that was using a model such as ESM-2 [20] as predictor of the likelihood of certain amino acid sequences being viable. Since ESM-2 [20] is trained on a large corpus of protein sequences, it can provide valuable insights into the potential success of generated sequences. Training such a large language model would be intractable for us to do. However, using the model to provide signal into either our generation process or our benchmark can give valuable insights. The model might also capture other aspects such as if physical constraints are satisfied internally.

5.4 Pseudo Log Likelihood (PLL)

The most naive way to use ESM-2 [20] is to calculate the log likelihood (LL) of the generated sequence. This gives a measure of how likely this sequence is according to the model. The higher the LL the more likely it is. However, ESM-2 has not been trained as a classic autoregressive model, but rather as a masked language model. For each training sample one of the tokens is masked and the model is trained to predict it given the context. Thus we want to have a measure that reflects this. One such measure is the PLL, which is outlined in the Masked Language Model Scoring paper by **Salazar et al.** [30]

The PLL is calculated by first masking the first residue of our CDRH3 region and passing it through our model. Then we look up the probability distribution over all possible amino acids at that position. Then the probability that matches the original masked residue is picked. This process is repeated for all residues in the CDRH3 region. Finally the log of all these probabilities is summed to get the PLL.

Figure 5.1 illustrates this process, this figure was inspired by **Salazar et al.** [30] but adopted for DGAD.

5.5 Pseudo-perplexity: PPPL

Although the PLL is a good measure, it is not very good when measuring a corpus of sequences of different length. This is precisely what will be done. Since the test set is made up of multiple cdrh3 sequences of different length. A measure that addresses this issue is the Pseudo-perplexity (PPPL). This is defined in the paper by **Salazar et al.** [30] as:

$$\mathrm{PPPL}(\mathcal{W}) \;=\; \exp\!\left(-\frac{1}{N}\sum_{w\in\mathcal{W}}\mathrm{PLL}(w)\right). \tag{5.3}$$

where

$\mathrm{PPPL}(\mathcal{W})$	is the pseudo-perplexity of the corpus $\mathcal{W}$ (as in the paper),
$\mathcal{W}$	is the corpus, i.e. the set of sentences $\mathcal{W} = \{w\}$,
w	is a single sentence,
N	is the total number of tokens in the corpus $\mathcal{W}$, i.e. $N = \sum_{w\in\mathcal{W}} N_w$,

The PPPL is computed over the whole corpus of sequences. Please note that within this implementation the PPPL per sequence is calculated:

$$\mathrm{PPPL}(s) \;=\; \exp\!\left(-\frac{\mathrm{PLL}(s)}{N_s}\right). \tag{5.4}$$

where

$\mathrm{PPPL}(s)$	is the pseudo-perplexity of the sequence s,
s	is a single sequence,
N_s	is the total number of tokens in the sequence

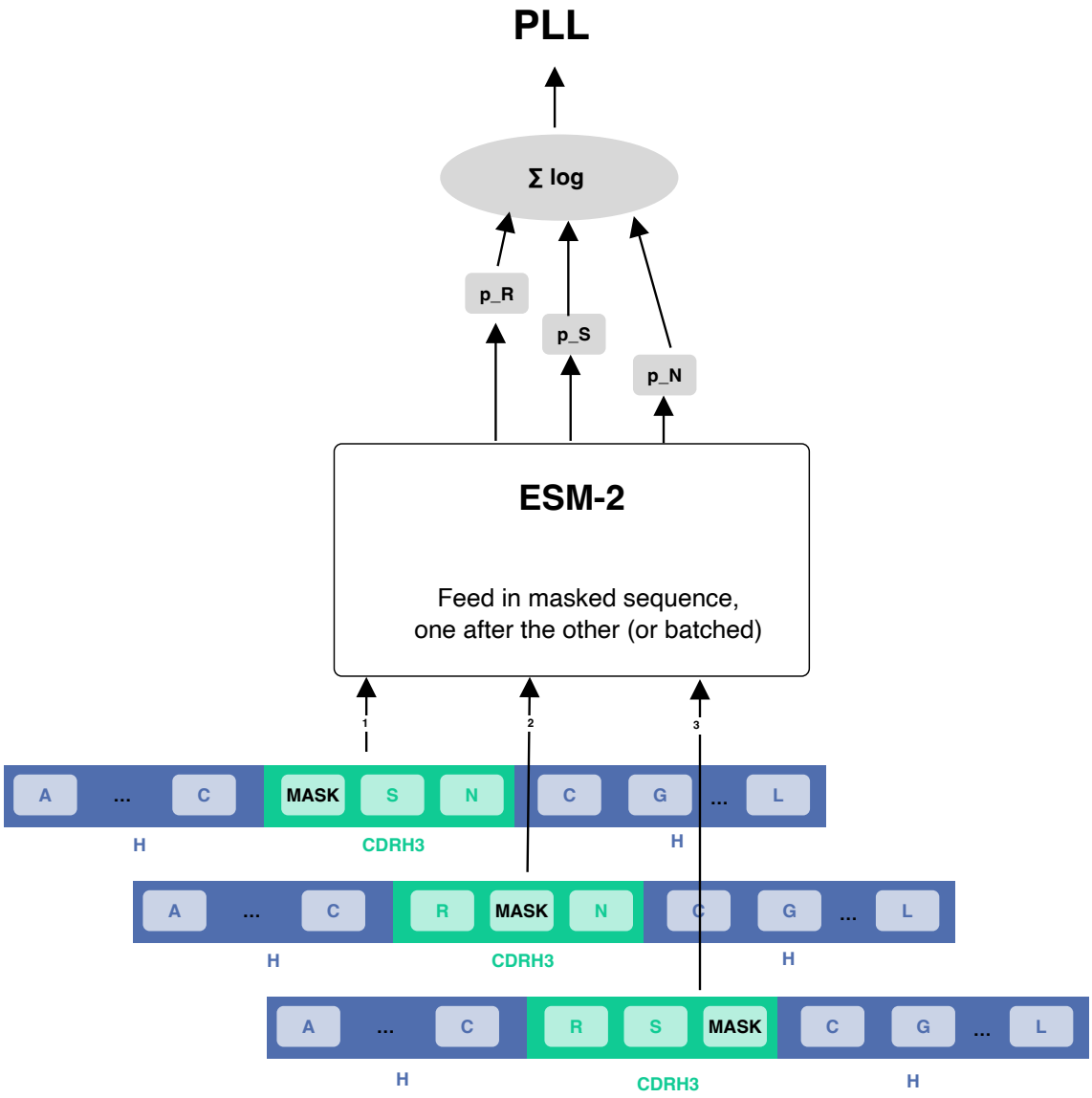


Figure 5.1: The PLL process

When reporting the PPPL per experiment the mean of the PPPL per sequence is taken. It still yields us the normalization benefits but it does not exactly match their definition.

The PPPL will yield a positive number, where a lower number is better. Indicating the model's confidence in the generated sequences while accounting for their varying lengths.

5.6 Implementation

For the implementation of these metrics and the processing of the results themselves the DGAD-B module was created. This module contains all the code needed to run the benchmarks and process the results. Figure 5.2 outlines the architecture.

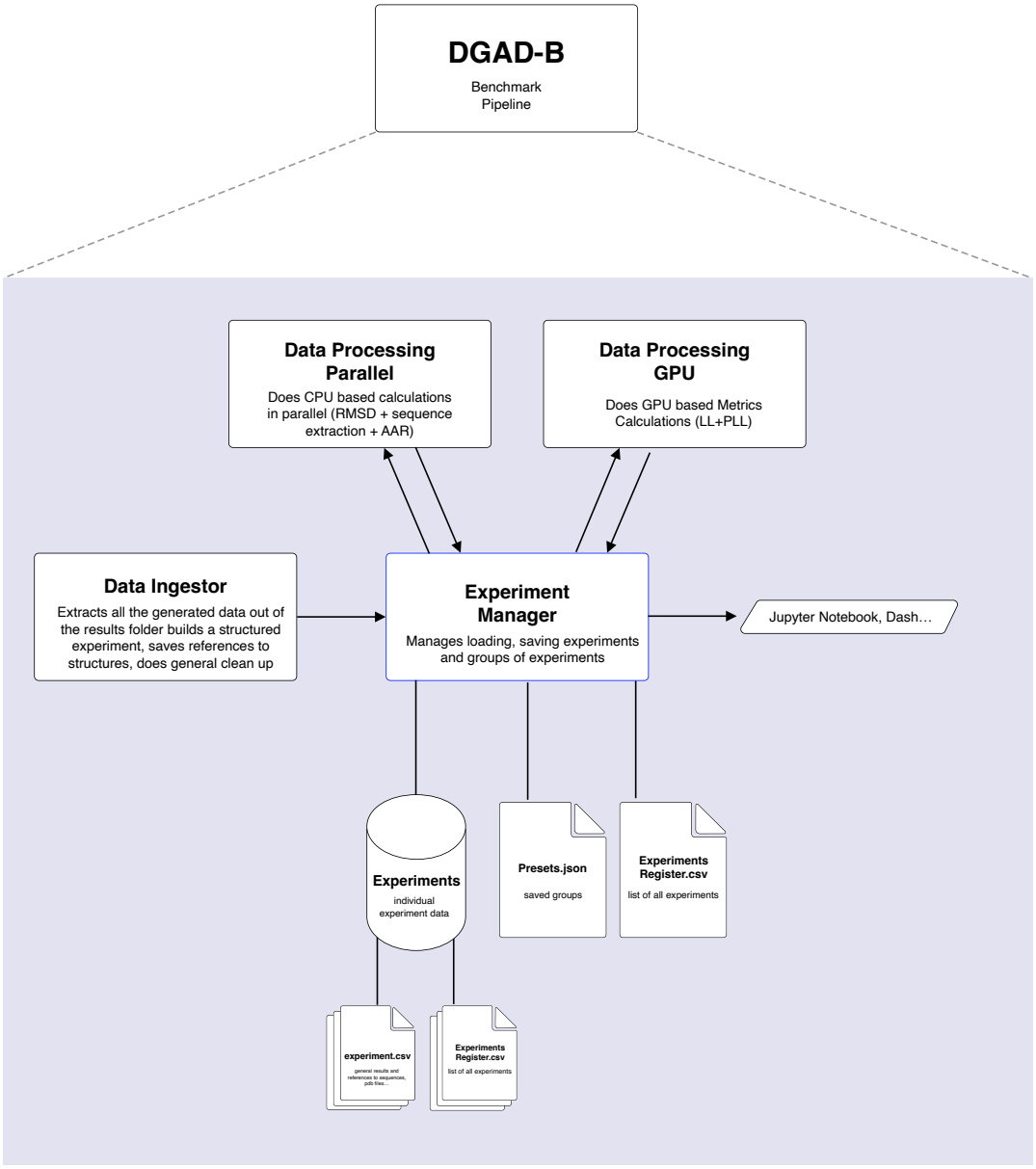


Figure 5.2: DGAD-B architecture

1. First the generated samples are collected using the *data ingestor*. These are stored in a structured way as an experiment and saved by the *experiment manager*.

2. Next data processing can be done, here an experiment or group of experiments can be loaded and either passed to the *parallel data processor* or *data processing GPU*.

    - The *parallel data processor* handles CPU based metrics such as RMSD and AAR, and extracting the cdr3 sequences, etc. It calls the appropriate functions from the DGAD-B module to perform these tasks.
    - The *data processing GPU* handles GPU based metrics such as PLL and LL. It uses batching to efficiently process the data on the GPU.
        – The PLL and LL are calculated using the ESM-2 model [20]. The model is loaded once and then used to process the data in batches. It uses the *esm2_t36_3B_UR50D* variant with 3 Billion parameters (to capture as biological relevance as possible while still being able to be loaded in the GPU efficiently).

3. Finally, the *experiments manager* can be used to load groups of experiments, which can then be visualized or further analyzed.

Chapter 6

Results

The following chapter discusses the results obtained for the implemented methods. It covers the infrastructure setup, the training results and inference results. The inference results showcase the effectiveness of the new techniques.

6.1 Infrastructure setup

Diffusion models require large amounts of compute to train, and inference is also resource intensive. Therefore the GPULab cluster of IDLab was used for running the necessary experiments. The GPULab cluster contains a number of configurable GPU equipped servers (Such as NVIDIA A40, NVIDIA RTX 5090, NVIDIA A100...).

It became quickly apparent that a well thought out AI- and DevOps pipeline was needed to enable efficient development. Therefore development via the remote, submitting jobs and running experiments were automated using pipelines, procedures, automation scripts and architectural changes.

This was one of the most unexpectedly challenging parts in the creation of this work. It took up a large part of implementation time, although it does not contribute to an improvement directly it did so indirectly. Thanks to this setup it was possible to run large amounts of experiments in parallel, iterate, and debug experiments remotely when necessary. Moreover most of the development was done trough a remote connection to the server.

The following iterative process was used throughout implementation:

1. Develop remotely the new feature
2. Debug the feature remotely on a GPU equipped server
3. Prepare the jobs for this new feature
4. Submit jobs
5. Process results through benchmark pipeline.

The setup consists primarily of three parts that work hand in hand together:

1. An automated system for launching a job, and configuring the host and remote to enable development and debugging via VSCode Remote Server (this will also be made available later through Github so other students can benefit when using the IDLab infrastructure.)

2. Start/config scripts to prepare and run jobs that work hand in hand with the centrally stored code, data and results (managed through central IDLab Infiniband storage; each job instance can access this central storage, also configured during step 1).

3. The benchmark pipeline (as discussed in the metrics section).

During the creation of this work over 20 different models were trained, and more than 40 inference runs were performed. Ultimately over 100 000 antibody-antigen complexes were generated over the course of time. Without automation and GPULab this was impossible.

6.2 Experimental Setup

Table 6.1 summarizes the experimental setup.

TRAITS	Optimization of selected CDR loop (small local changes) for structure and sequence. • Mask desired CDR region; keep length fixed; rest of structure unchanged. • Diffusion model: noise/denoise positions, orientations, and sequence.
EXPERIMENTS	Focus on CDR3 results; sampling only for CDRH3 region.
DATASET	SAbDab.
SPLIT	**Train (0.90)**: 8558 **Val (0.05)**: 439 **Test (0.05)**: 644
SAMPLING BALANCE	Mode: linear / exponential Range: [1, 0] (1 = original cluster distribution, 0 = uniform) Recommended: 0.75 (Word2Vec exponential standard)
N_SAMPLES	8 (number of variants generated per complex in the test set).
GPU	NVIDIA A40.
BATCH SIZE	16
LEARNING RATE	$1 \cdot 10^{-4}$
TRAINING STEPS	**Base**: 200,000 **Aux (from scratch)**: 200,000 **Aux (from base)**: 40,000
N_STEPS	100 (number of diffusion steps).
SEED	70
OTHERS	Most parameters from the original DiffAb paper were kept fixed.

Table 6.1: Experimental setup

6.3 Training results

Before covering the inference results, which showcase the effectiveness of the new techniques, the loss curves and validation curves of the trained models are briefly covered here. These primarily were verified to see if the customized models such as base and aux were training properly like the original model.

Figure 6.1 show the loss curves of the original model.

Figure 6.1: Loss curves

There are 4 loss curves:

- loss_pos: The loss of the positions. This has the best convergence.
- loss_seq: The loss of the sequence. Converges ok.
- loss_rot: The loss of the rotations. Here the loss does not drop like the others. This is also reflected in AAR values. Future improvements to the joint sequence generation could help here. This was out of scope for this thesis.
- loss: The overall loss of the model. This is a weighted sum of loss_seq, loss_pos and loss_rot.

The loss curves for a base model that has been trained only on the H_CDR3 region, and with the new data split is shown in Figure 6.2. It shows a similar convergence as the original model.

Figure 6.2: Loss curves

Many variants with different hyperparameters of the base model, aux model were trained. Figure 6.3 shows the loss curves for some of these variants.

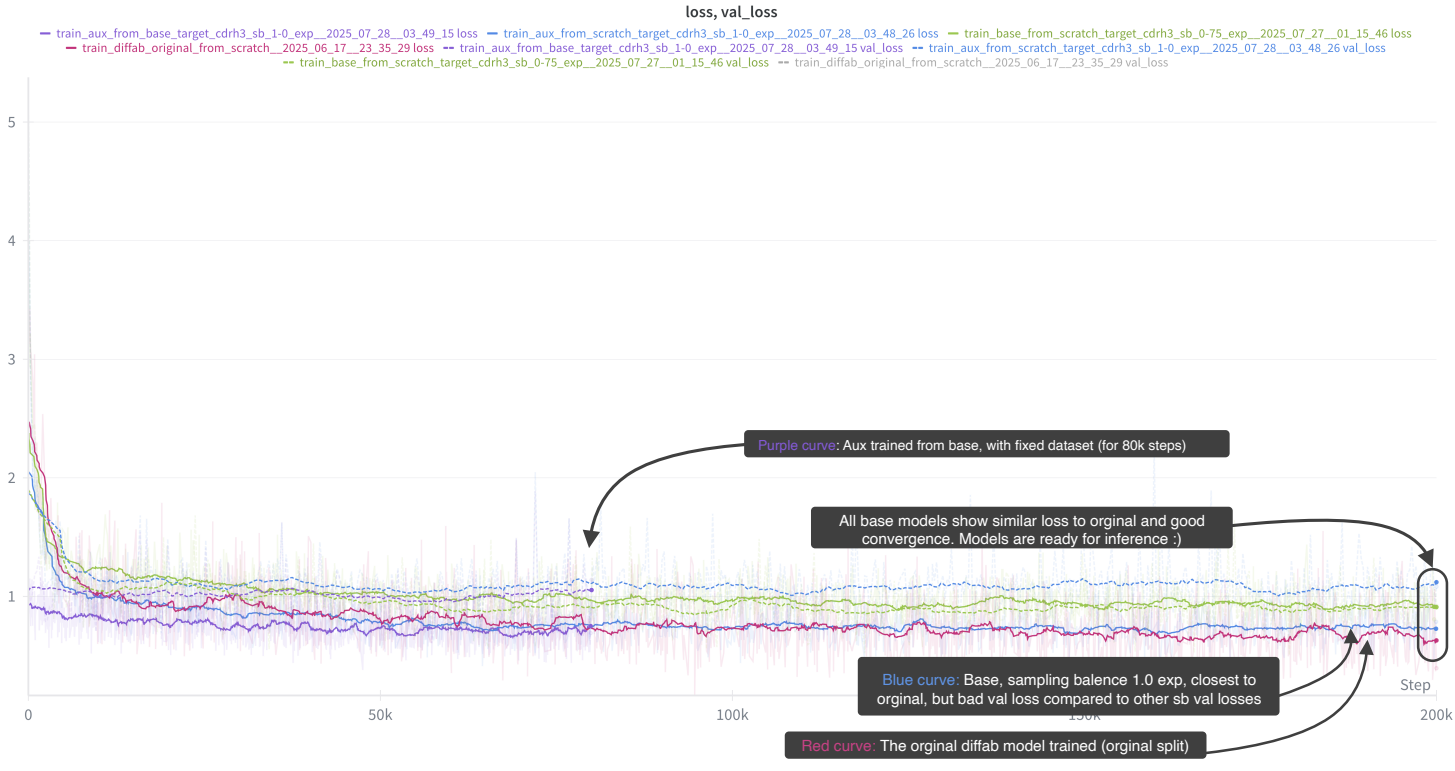


Figure 6.3: Loss curves for base model: Show a similar convergence as the original model.

Notably we trained models with different sampling balances. Interestingly we can see that for a sampling balance 1.0 we get a similar loss as our original model. When we have a more balanced approach the loss is higher. However when we compare the actual validation loss of the one with sampling balance 1.0 vs a more balanced one like 0.75 the picture gets flipped, and the more balanced model does better. This already gives some indication that a more balanced sampling approach is beneficial. In the next section an ablation will be done to verify this.

Finally, Figure 6.4 shows the validation loss curves for the different models.

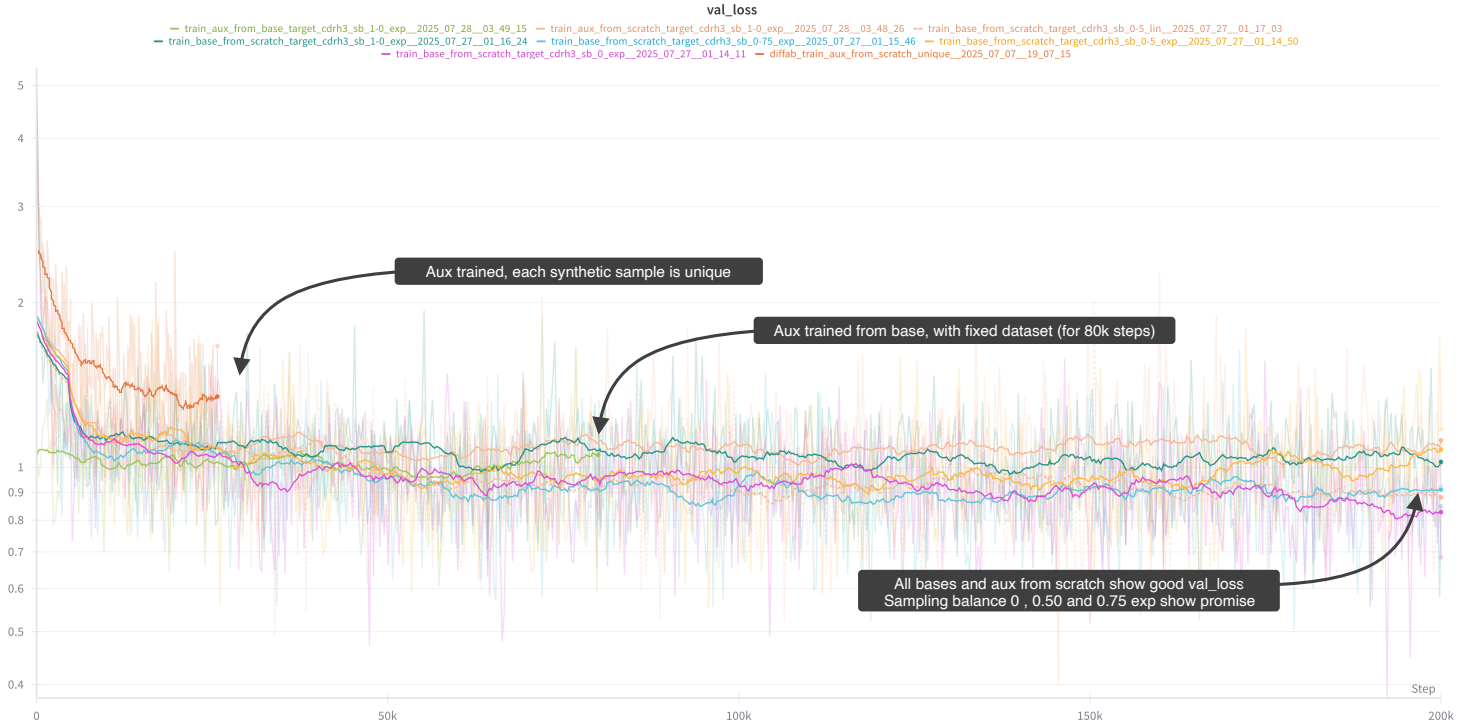


Figure 6.4: Loss curves

Aside from the different variants of the base model with regard to sampling balance, here it is interesting to highlight two versions of the auxiliary model.

Before a fixed dataset, and dataloader were made for the aux model so it could be trained as described in the original SIMS paper [3] a version was created were each sample that went into the model was uniquely generated on the fly by the base model. This was used as a minimal viable product to see if this technique had promise. The loss for this model is considerably higher than that of the fixed dataset version. Additionally whereas typically a training run took around 30 hours this took more than 3 days and only progressed around 40K steps. However, since the aux model primarily provides a shift even an underperforming model like this showed promise.

6.3.1 Summary of trained models

Table 6.2 summarizes the most important trained models.

model	from	dataset	target	steps	sampling method	sampling balance	note
original	**scratch**	**SabDab**	**allcdr**	**200k**	_	_	_
base	scratch	SabDab	allcdr	200k	_	_	_
base	scratch	SabDab	**cdrh3**	200k	_	_	_
base	scratch	SabDab	cdrh3	200k	**lin**	**0.0**	_
base	scratch	SabDab	cdrh3	200k	**exp**	**0.5**	_
base	scratch	SabDab	cdrh3	200k	exp	**0.75**	_
base	scratch	SabDab	cdrh3	200k	exp	**1.0**	_
aux	scratch	**base-unique**	cdrh3	200k	_	_	_
aux	scratch	**base-fixed**	cdrh3	200k	_	_	_
aux	**base**	base-fixed	cdrh3	**80k**	_	_	_
aux	base	base-fixed	cdrh3	80k	_	_	lr: $0.0001 \rightarrow \mathbf{0.0007}$

Table 6.2: Trained model configurations. *from* indicates initialization: `scratch` = trained from random init; `base` = fine-tuned from a previously trained `base` model. Bold entries highlight values that differ from the previous row. A visible _ means “not applicable / not used / not implemented” for that run.

6.4 Inference results

This section covers the inference results of the different models. It can be split into three parts.

First it covers basic results to see if our version produces feasible solutions, and if the proposed techniques have any merit.

In part two the effectiveness of the new techniques will be evaluated. First an ablation study is done on the sampling balance. Subsequently an ablation on SIMS. Evaluating when and in what conditions negative guidance works best.

Part three covers a picked example of a generated antibody-antigen complex, and it summarizes the most important results.

The results are compared using the metrics introduced in the metrics chapter.

6.4.1 Part 1: initial assessment

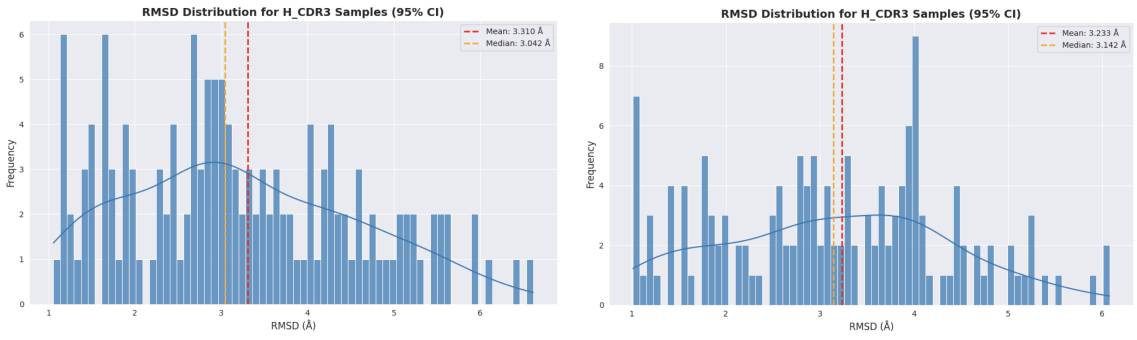
Paper vs Original model

Before starting any changes to the model it is important that with the Diffab codebase similar results can be reproduced as they report.

For this inference was run on the original split of Diffab. Two versions were compared. One with the pre trained published weights of Diffab, this is referred to as paper. The other version is the same model retrained from scratch with the published configuration, this is referred to as original. The results are available in table **??**. Both distributions are visualized side by side in figure 6.5.

model	target	rmsd (mean)	rmsd (std)	rmsd (min)	rmsd (max)
paper	allcdr	3.310	1.616	0.786	10.478
original	allcdr	3.233	1.522	0.781	9.316

Table 6.3: RMSD results base paper vs original



(a)

Paper Distribution for inference on the original test set using pre trained weights.

(b)

Original Distribution for inference on the original test set, with the model trained from scratch

Figure 6.5: Comparing the distributions of the model using the published weights vs retrained from scratch.

The values are slightly different this is possibly due to slight differences in sampling, seed, that they may have used when training their weights. Nonetheless, both results show similar performance, indicating that the original Diffab model can be successfully retrained from scratch. They are however higher than the reported RMSD of 2.89 in the diffab paper [22]. Again possibly due to a different setup they used, or throwing away certain outliers since their appear to be quite a bit of extreme outliers.

This is not necessarily a problem, since in this work the main goal is to see if the guidance technique applied on a model can improve that model. As long as the internal models are compared using similar test set, config and metrics this should be sufficient.

The distributions it self are not very smooth, when these experiments were done this was one of the motivations for increasing the size of the test set. Since doing comparison on a small test set is not very reliable.

Base allcdr vs cdrh3

When training the base model, two versions were trained. One that focuses on all CDR loops, and one that only focuses on the CDRH3 loop. The main focus of this work is on the CDRH3 loop. Therefore it is possible to only train on this loop and not on the others. In the original model during training they randomly choose a cdr region to noise/denoise since you can select which region to denoise. Intuitively training only the CDRH3 region is better. Experiments were done to verify this

When comparing the *base allcdr* model to the *cdrh3* model, we can see that the *cdrh3* model has a lower RMSD value (see table 6.4), indicating that it produces structures that are closer to the ground truth. Since in this work we only focus on the CDRH3 this model will be used for further experiments.

model	target	rmsd (mean)	rmsd (std)	rmsd (min)	rmsd (max)
base	allcdr	5.092	4.045	0.524	29.924
base	cdrh3	**4.050**	2.900	0.596	27.242

Table 6.4: RMSD results base allcdr vs cdrh3

A histogram and KDE of the RMSD values for the base allcdr and cdrh3 models is shown in figure 6.6. We can see that the cdrh3 model has a higher density of low RMSD values, indicating that it produces more accurate structures. The shape also matches the original shape reported in the diffab paper.

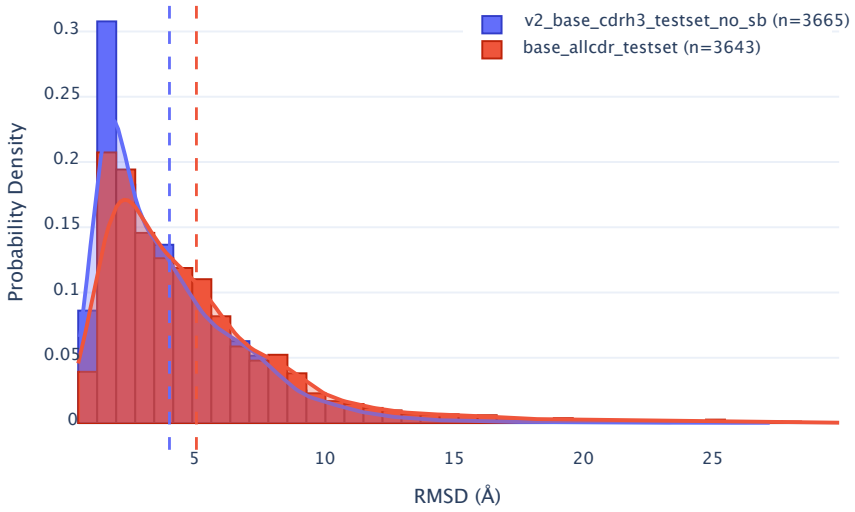


Figure 6.6: Histogram (40 bins) and KDE for allcdr vs cdrh3

6.4.2 Part 2: Ablations

Sampling balance

The following ablations cover results for sampling balance:

	rmsd		aar		pll_perplexity	
	mean ± std	[min, max]	mean ± std	[min, max]	mean ± std	[min, max]
exp sb 1.0 - baseline	4.01 ± 3.19	[0.61, 56.52]	0.29 ± 0.13	[0.00, 0.88]	9.25 ± 3.69	[1.67, 38.14]
exp sb 0.5	4.09 ± 3.29	[0.46, 58.81]	0.28 ± 0.14	[0.00, 0.88]	10.04 ± 3.64	[2.49, 30.56]
lin sb 0.5	3.65 ± 2.52	[0.52, 21.82]	0.28 ± 0.13	[0.00, 0.88]	8.72 ± 3.64	[2.11, 49.67]
exp sb 0.75	3.70 ± 2.48	[0.57, 17.83]	0.28 ± 0.13	[0.00, 0.88]	10.46 ± 3.71	[2.18, 32.75]
exp sb 0.0	3.97 ± 3.16	[0.61, 50.56]	0.27 ± 0.13	[0.00, 0.88]	10.16 ± 4.20	[1.77, 42.63]

Table 6.5: Results of ablation study sampling strategies. The results are obtained on a held out test set of 348 antibody-antigen complexes from SabDab. The RMSD is calculated on the generated CDRH3 region. For each complex 8 samples were generated, the mean is computed over all samples, the std is the standard deviation and [min, max] is the range of values over all samples within the test set. Lower is better for rmsd and pll_perplexity, higher is better for aar.

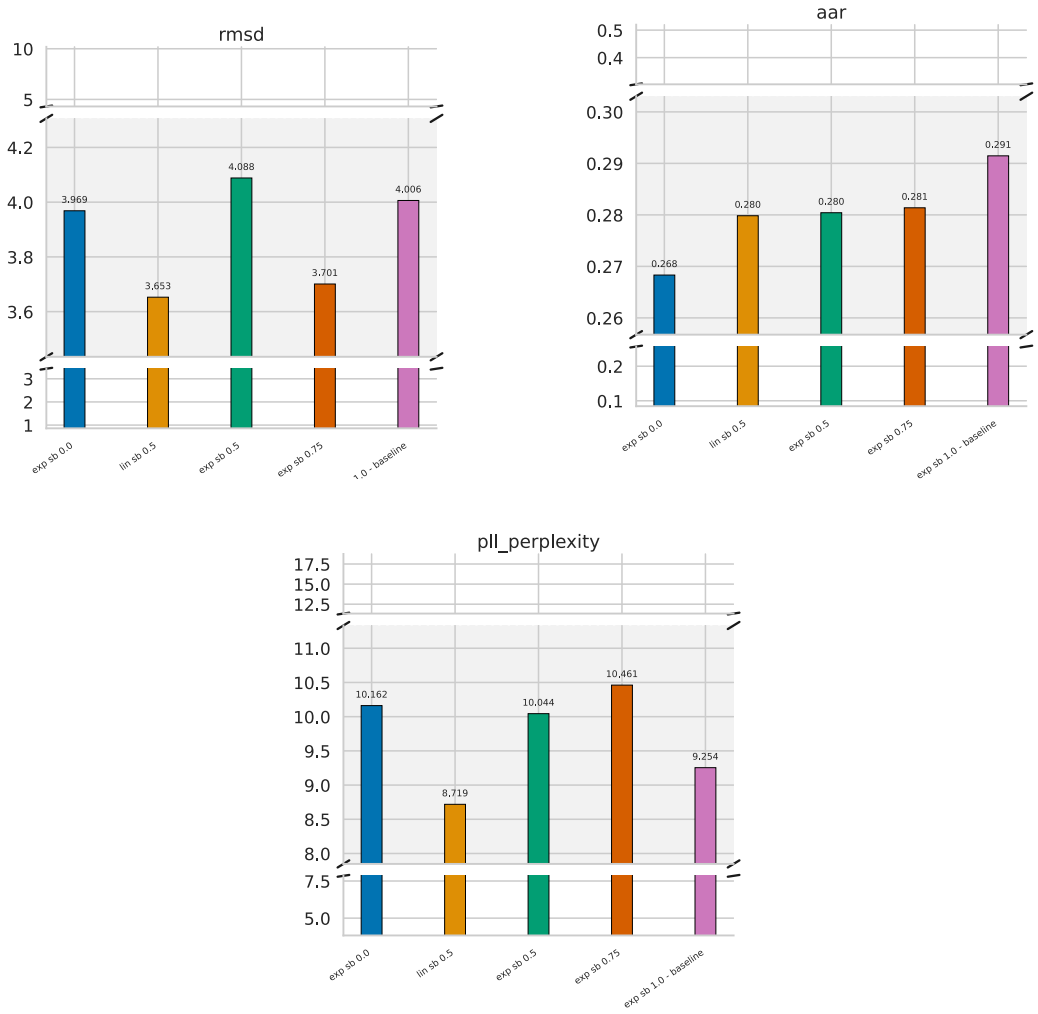


Figure 6.7: The results from the ablation study on sampling strategies visualized.

SIMS: position only

The following ablations cover results for SIMS when applied to the position only:

Table 6.6: Results of ablation study on guidance strength ω. The results are obtained on a held out test set of 37 antibody-antigen complexes from SabDab. The RMSD is calculated on the generated CDRH3 region. The best results overall (taking in to account min/max) are obtained when using a guidance strength of 0.75. For each complex 8 samples were generated, the mean is computed over all samples, the std is the standard deviation and [min, max] is the range of values over all samples within the test set. Lower is better for rmsd.

	rmsd	
	mean ± std	[min, max]
baseline	3.62 ± 3.18	[0.70, 40.12]
from base pos guidance 0.75	3.29 ± 2.03	[0.62, 12.78]
from base pos guidance 1.0	3.30 ± 2.03	[0.62, 13.43]
from base pos guidance 1.5	3.42 ± 2.23	[0.55, 16.05]
from scratch pos guidance 0.5	3.34 ± 2.16	[0.67, 12.79]
from scratch pos guidance 0.75	3.35 ± 2.17	[0.74, 13.26]
from scratch pos guidance 1.0	3.45 ± 2.29	[0.81, 14.11]
from scratch pos guidance 1.5	3.73 ± 2.73	[0.76, 15.72]
from base pos guidance 0.5	3.34 ± 2.14	[0.66, 12.58]

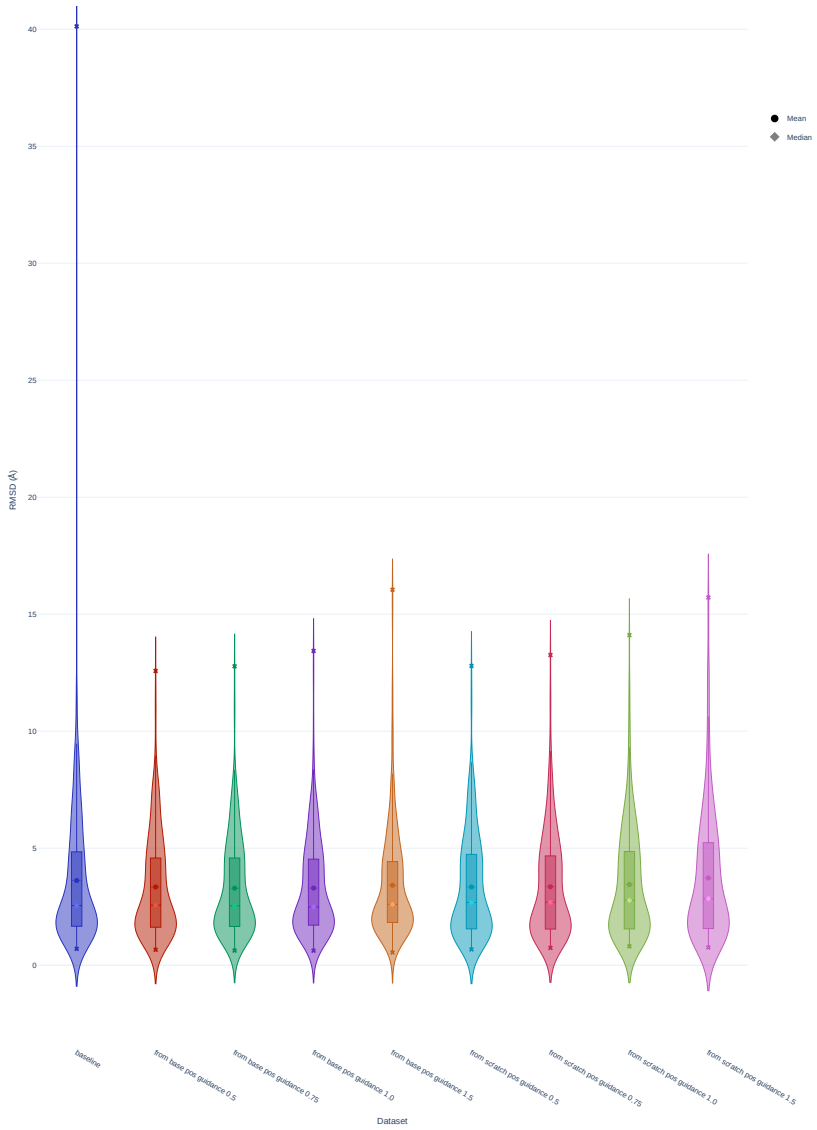


Figure 6.8: Violin plot of the results from the ablation study on guidance strength ω when applying SIMS to position only. The best results overall (taking in to account min/max) are obtained when using a guidance strength of 0.75.

SIMS: sequence only

The following ablations cover results for SIMS when applied to the sequence only:

	rmsd		aar		pll_perplexity	
	mean ± std	[min, max]	mean ± std	[min, max]	mean ± std	[min, max]
baseline	4.13 ± 3.71	[0.60, 62.51]	0.27 ± 0.14	[0.00, 0.88]	9.28 ± 3.34	[2.66, 28.49]
from scratch guidance on seq 1.0	4.51 ± 5.20	[0.59, 88.33]	0.31 ± 0.14	[0.00, 0.88]	6.37 ± 2.30	[1.43, 19.34]
from scratch guidance on seq 0.5	4.49 ± 5.16	[0.57, 88.17]	0.30 ± 0.14	[0.00, 0.88]	7.19 ± 2.62	[1.92, 26.95]

Table 6.7: Results of ablation study on guidance on sequence. The results are obtained on a held out test set of 351 antibody-antigen complexes from SabDab. The RMSD is calculated on the generated CDRH3 region. The best results overall (taking in to account min/max) are obtained when using a guidance strength of 1.0 on sequence only. For each complex 8 samples were generated, the mean is computed over all samples, the std is the standard deviation and [min, max] is the range of values over all samples within the test set. Lower is better for rmsd and pll_perplexity, higher is better for aar.

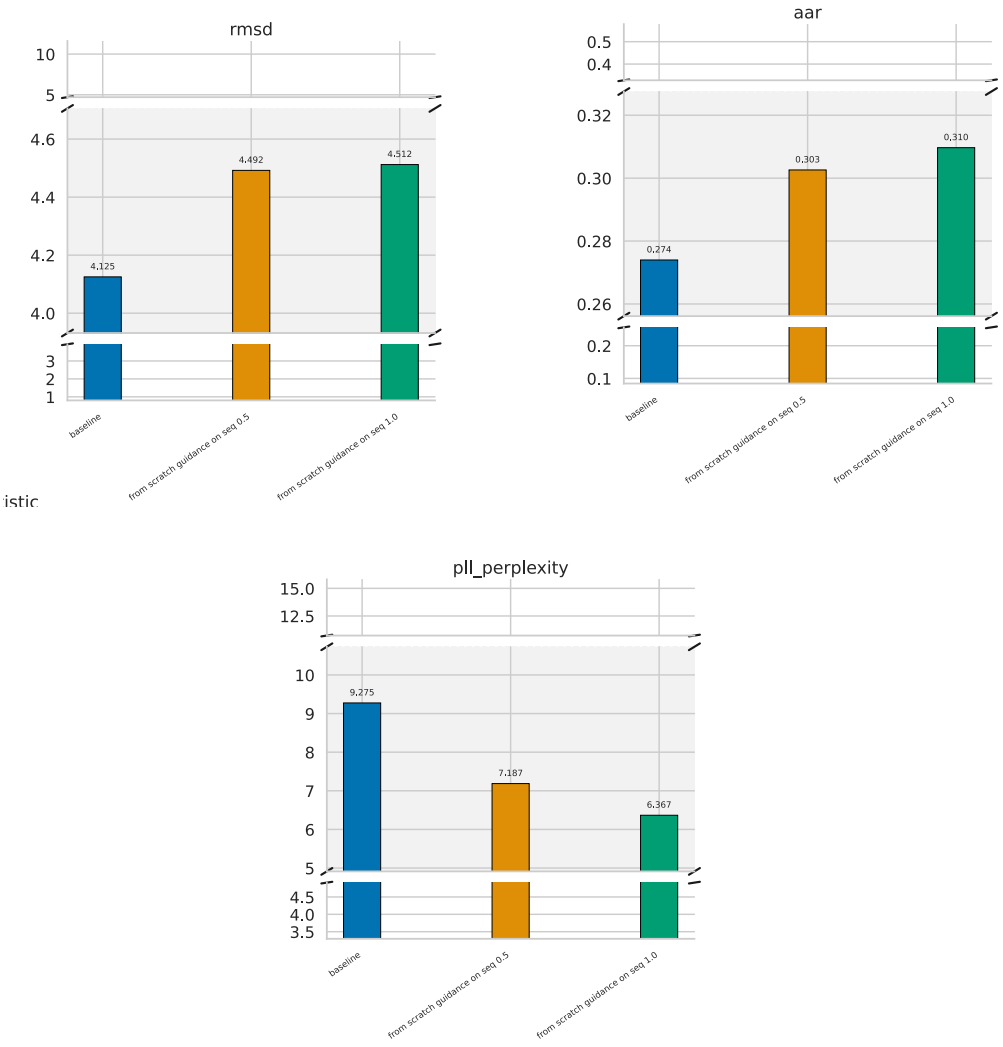


Figure 6.9: The results from the ablation study on sequence visualized. The best results overall (taking in to account min/max) are obtained when using a guidance strength of 1.0 on sequence only. What is interesting is that the AAR performance does not change a lot. However, the PLL perplexity decreases significantly, indicating more biologically plausible sequences.

6.4.3 Part 3: Final results

Table 6.8 highlights the results of applying SIMS to Diffab. The results are obtained on a held out test set of 246 antibody-antigen complexes from SabDab. The RMSD is calculated on the generated CDRH3 region.

	rmsd		aar		pll_perplexity	
experiment	mean ± std	[min, max]	mean ± std	[min, max]	mean ± std	[min, max]
baseline	4.13 ± 3.71	[0.60, 62.51]	0.27 ± 0.14	[0.00, 0.88]	9.29 ± 3.33	[2.66, 28.49]
balanced sampling exp 0.75	3.69 ± 2.48	[0.57, 17.83]	0.28 ± 0.13	[0.00, 0.88]	10.47 ± 3.75	[2.18, 32.82]
sims pos 0.75	3.81 ± 2.80	[0.57, 53.64]	0.28 ± 0.14	[0.00, 0.88]	9.04 ± 3.25	[2.86, 31.12]
sims pos 0.75 (with decay)	4.05 ± 3.40	[0.58, 59.77]	0.28 ± 0.14	[0.00, 0.88]	9.10 ± 3.27	[2.86, 31.12]
sims pos 0.75 (witch schedule)	4.03 ± 3.66	[0.58, 78.62]	0.28 ± 0.14	[0.00, 0.88]	9.06 ± 3.25	[2.64, 30.54]

Table 6.8: Results of applying SIMS to Diffab on a held out test set of 350 antibody-antigen complexes from SabDab. The RMSD is calculated on the generated CDRH3 region. The best results are obtained when using a guidance strength of 0.75 and without using decay or a schedule. This results in a $\sim 8\%$ improvement over the baseline. For each complex 8 samples were generated, the mean is computed over all samples, the std is the standard deviation and [min, max] is the range of values over all samples within the test set. Lower is better for rmsd and pll_perplexity, higher is better for aar.

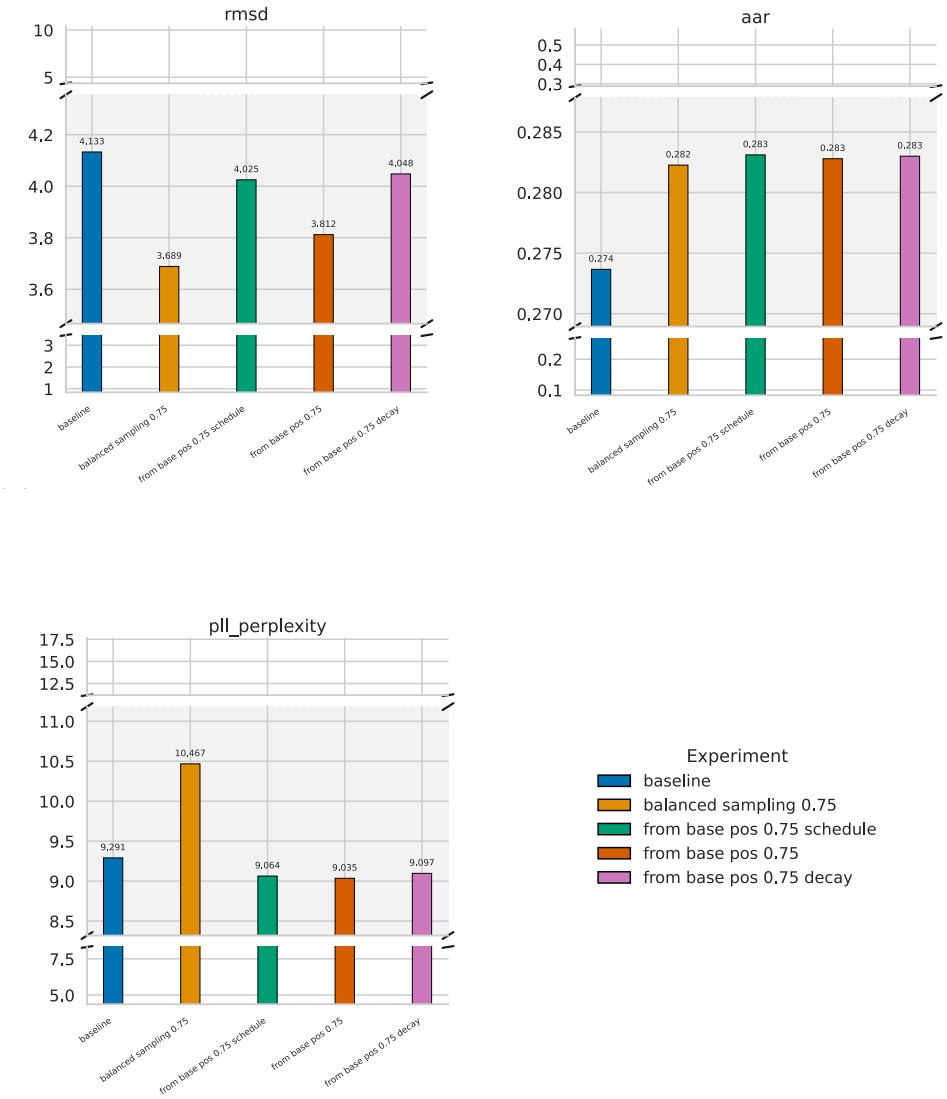


Figure 6.10: The final result visualized.

Discussion

Baseline is the base model trained from scratch using the newly discussed split and only on the CDRH3 region. Balanced **sampling exp 0.75** is the same model but trained with a more balanced sampling strategy (using the word2vec style sampling). **SIMS pos 0.75** is the base model with SIMS applied to position only, with a guidance strength of 0.75. **SIMS pos 0.75 (with decay)** is the same but using a decay. The decay is a linear decay starting from the initial guidance strength towards zero. Finally **SIMS pos 0.75 (with schedule)** is the same but using a schedule. The schedule is outlined in figure 6.11

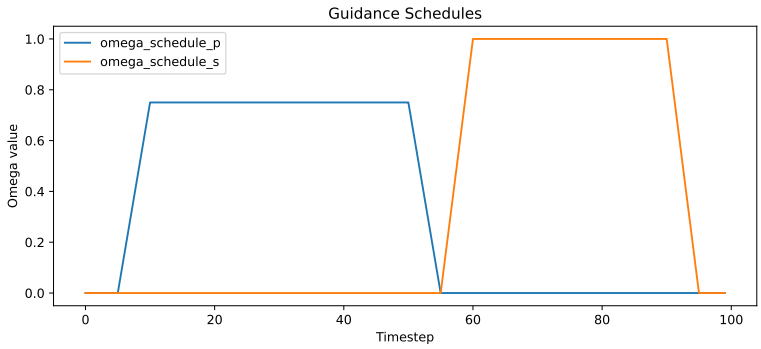


Figure 6.11: Guidance schedule for SIMS. No guidance in the beginning then gradually increase guidance for position keep on the longest, then if turned on apply guidance on the sequence , a bit shorter than the position.

Due to limited training time the balanced strategy could not be combined with guidance (since a new auxiliary model would need to be trained). This could be an interesting avenue for future work. Possibly it would lead to a cumulative effect further improving overall RMSD. Additionally for correctness the sequence guidance is left out here since for the final results runs the results did not match ablations which needs to be investigated first before publishing here. Consequently combing sequence and position did not show improvements over using on the position alone and was therefore omitted here.

Most signal in the results can be obtained by looking at the RMSD since here guidance is only applied on the position which affects the structure. RMSD is better for all our implemented methods (lower is better).

Some small deviations in AAR are visible too this is because a joint epsilon net is used with three different heads so the a change in structure also affects the output of the sequence head of the epsilon net. Additionally we also investigated a change to the sampling balances which obviously affects the whole model. The AAR for our methods (compared to baseline) is slightly higher (higher is better). Consequently the PPPL is also affected. Surprisingly this is higher for the balanced sampling strategy. This indicates that the sequences are possibly less biologically plausible. As of yet an explanation for this was not found. For the guidance methods it's slightly lower (better).

Interestingly the best results are obtained when using a balanced sampling strategy. This confirms the hypothesis that a more balanced sampling strategy is beneficial for generating better antibody structures. In second place is applying the SIMS technique with a guidance strength of 0.75 and without using decay or a schedule. around 8% improvement.

This confirms our hypothesis that using the SIMS technique is effective for improving the quality of antibody structures using diffusion models. However, having good machine learning fundamentals applied to training a model such as sampling in a balanced way is actually crucial for achieving optimal results as well. Often it is better to look first for ways to improve the underlying model architecture and training procedure before applying more advanced techniques such as SIMS.

Chapter 7

Societal Reflection

This thesis studied ways of optimizing the design process for generative antibody design using AI techniques. It focused specifically on making the diffusion models better. By improving sample quality without increasing dataset size. Additionally during the methodology careful attention was spend at creating a robust evaluation framework. Finally machine learning fundamentals were used to come up with a better sampling strategy. This research does not only improve the studied model but can be extended to other generative models within the space and eventually help improve healthcare.

Climate change and the use of energy are important sustainable development goals of the United Nations. It is crucial that we consider the environmental impact of our research. This work uses AI as a means to optimize the design process, we must remain vigilant about the energy consumption of large-scale AI models and strive to minimize their carbon footprint.

In the proposed technique an auxiliary model is used during training, thus increasing the amount of energy needed during sampling of antibodies.

However it is important to balance this with the potential benefits gained from improved antibody generation. One of the other goals is to ensure healthy lives and promote well-being for all at all ages. By improving these models we aim to directly contribute to this goal. Moreover, better models reduce experimental validation time which is also a crucial factor in the effect on energy consumption. As of today typically these models don't always produce high quality structures. Therefore many samples need to be generated and validated experimentally to find a good candidate. By improving the quality of generated structures we can reduce the amount of samples that need to be generated, this would make up for the fact that the model uses more energy since it needs to be used less often to find a good candidate.

Nevertheless, for future work it would be interesting to study ways of incorporating guidance strategies in the generation process without relying on auxiliary models. This way we could potentially reduce the energy consumption associated with training while still benefiting from improved generation capabilities.

As with any powerful technology, there is a risk that it could be used for nefarious purposes, such as designing harmful biological agents. It is essential that we establish robust ethical guidelines and oversight mechanisms to prevent misuse and ensure that our research is conducted responsibly. Moreover even just assuming that these models produce high quality structures can be potentially misleading, as they may not always account for the complexities of biological systems.

Thus it should be noted that although this method is able to improve the quality of generated antibodies, it does not eliminate the need for extensive validation and testing in real-world scenarios. These models currently purely serve as a tool to help researchers explore the vast space of possible antibody designs more efficiently.

In conclusion the benefits of these advancements are significant. By streamlining the design process and reducing the time and resources required for experimental validation, we can accelerate the development of new therapeutics and ultimately improve patient outcomes. It is essential that we continue to explore these possibilities while remaining mindful of the ethical considerations and societal implications of our work.

Chapter 8

Future work and Conclusion

8.1 Future work

Several other avenues where explored during the research process as well as extensions to the current implementation:

- Before committing to applying SIMS to Diffab an alternative was explored. This involved using reinforcement learning to finetune Diffab. As a reward function we would learn a confidence estimator. This confidence estimator predict the quality of the generated structures. The confidence would be a combination of the metrics discussed. Figure 8.1 illustrates how this would work.
  - The confidence estimator would be trained on labeled structures, for which different costly metrics were computed. In this way the estimator could learn to predict the quality of the generated structures.
  - The confidence estimator could then be used either standalone to predict the quality of unseen structures or it could be used as a reward function inside the reinforcement learning framework.
    - Crucially this estimator is differentiable. This would allow us to use algorithms such as PPO (Proximal Policy Optimization) [31] which typically rely on gradients to flow through the reward function.
  - In stead of applying equal guidance on each residue, a mask could be used to apply more guidance on residues that are further away from the ground truth. This mask could be learned.
- Classifier free guidance [15] using a learned guidance mask
  - Omit using an auxiliary model. Instead train the base model with conditional information and with a mask that indicates which residues to apply more guidance on.
- Finally the idea of guidance and reinforcement learning could be combined. Instead of doing reinforcement learning on the model itself, it could be done on the

guidance mask. Doing reinforcement learning directly on a model can take a long time. By doing it on the guidance mask, we could potentially speed up the process and make it more efficient. In this way different guidance masks can be learned that steer the model towards different objectives. For example one mask could be learned to optimize for RMSD, while another could be learned to optimize for physical plausibility. This would allow for more flexibility and adaptability in the model's performance. This opens up a lot new opportunities, and would be interesting to explore.

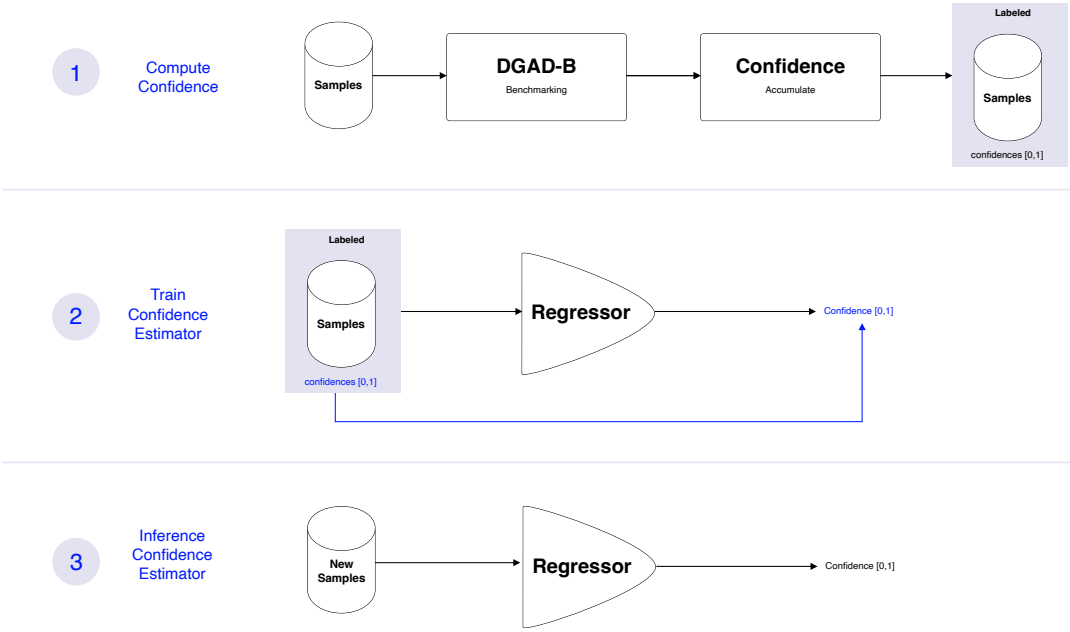


Figure 8.1: Training and using the confidence estimator

8.2 Conclusion

A recently proposed method for enhancing diffusion models through negative guidance called SIMS, was adapted and applied to the Diffab framework. The primary objective was to improve the quality of generated antibody structures without increasing the size of the training dataset.

The auxiliary model was trained using a two step approach to save training time. Guidance was applied on the position of each residue during inference.

Aside from applying SIMS the base model was investigated further to understand its behavior and performance characteristics. It was discovered that a better sampling strategy could be employed to improve the quality of generated structures. This included exploring different sampling techniques and their impact on the generated output.

When combining SIMS with diffab for positions of residues, a $\sim 8\%$ improvement in RMSD on the CDRH3 region on a held out test set of 350 antibody-antigen complexes from SabDab was achieved. Successfully demonstrating the potential of guidance techniques to improve generative antibody design.

Moreover, when exploring different sampling techniques, it was found that certain strategies could further enhance the quality of generated structures. Intensely sampling had a bigger impact on the quality of the generated structures than SIMS. This suggests that sampling strategies play a crucial role in the performance of diffusion models. This confirms the importance of good machine learning fundamentals. Before trying any special techniques on a model it is important to explore the basics first.

Overall, this work demonstrates the effectiveness of negative guidance in enhancing diffusion models for antibody design. Ultimately helping to drive the field of drug discovery forward.

Chapter A

Codebase

A.1 Codebase Overview & Change Summary

Codebase location: see the accompanying folder or the provided link.

This codebase was derived from the original DiffAB model by **Luo et al.** [22]. It is a relatively large code base with a lot of sub layers. To make it easy to see what changed the next section covers the primary changes and additions. Another quick way to understand the differences is to compare this codebase against the original DiffAB codebase using your preferred diff tool. Git was not used for this codebase since a lot of development was done remotely and this further complicated my workflow.

Chapter B

Generative AI Usage Notice

Given the current developments in artificial intelligence, it would have been counterproductive to avoid the use of generative AI tools during the preparation of this thesis. The subject matter explored here lies at the intersection of molecular dynamics, biology, and computer science. To navigate such a broad and technical landscape efficiently, and to progress to a point where a meaningful contribution could be made, generative AI was integrated into the workflow. Moreover, as this thesis itself examines the role of AI in advancing research, it would be somewhat contradictory not to make use of these methods. Responsible use of AI is allowed by the faculty. [34]

The tools primarily employed were ChatGPT and GitHub Copilot, each serving different purposes:

B.1 Literature exploration and methodology

Generative AI was mainly used to break down complex concepts into step-by-step reasoning. For example, when encountering a loss function or an unfamiliar mathematical expression, I would iteratively explore its meaning, applications, and implications with the aid of AI. In some cases, the model was guided by specific research papers to reduce the risk of hallucinations. It is important to note that the actual discovery of literature was carried out through more traditional means such as Google Scholar, YouTube lectures, discussions with my supervisor, and exploratory reading. AI served as a complementary tool for interpretation and clarification, rather than for discovery.

B.2 Code development

AI was used selectively in the implementation of the codebase. Small, modular pieces of code, such as data processing routines or visualization scripts were often generated with the help of Copilot or ChatGPT. However, entire features or critical architectural components were written manually, as relying fully on AI in such contexts proved to be too error-prone. One exception was a highly mathematical but optional feature, which was initially generated with AI assistance. Nevertheless, I validated the requirements, checked the derivations, and verified the implementation myself. Although important this part was

not necessary the core goal of my thesis. This example illustrates how AI enabled me to accelerate time-intensive tasks allowing more energy to be dedicated to the other research contributions.

B.3 Writing

The majority of the text was written by hand. However, AI was occasionally used as a sounding board to test how an explanation might best be structured, or to refine rough drafts into more fluent and readable text. In some cases, bullet-point outlines were expanded into full paragraphs using AI. Formatting tasks, such as LaTeX adjustments, were also supported by generative tools. Despite these uses, the structure, arguments, and flow of the thesis were determined by the implementation process and my own interpretation of the research.

B.4 Reflections on use

Through this process, it became clear that context is crucial when working with AI. A short prompt will rarely yield a useful result; instead, long, detailed conversations where requirements are carefully outlined and iteratively refined. This, proved to be the most productive. In practice, this resembled programming in a higher level abstract language therefore my programming and reasoning skill previously acquired during this program were still very relevant and useful. Once this workflow was established, the true strengths of generative AI became apparent.

Chapter C

Infrastructure

- VSCode for all development — allows local/remote debugging
  - Automated workflow for creating jobs
  - Includes automatic SSH server setup and config, both local and remote
- Compatible with both new nodes (cluster9) and old nodes, CPU and GPU nodes
- All envs (diffab, esm, dgad-b, boltz. . .) preloaded on central storage
- Jobs start using shell scripts
- Remote development using VSCode Server
  - Connect to CPU-only nodes for management, preloading data, fetching data
  - Connect to GPU nodes to allow for complex debug situations

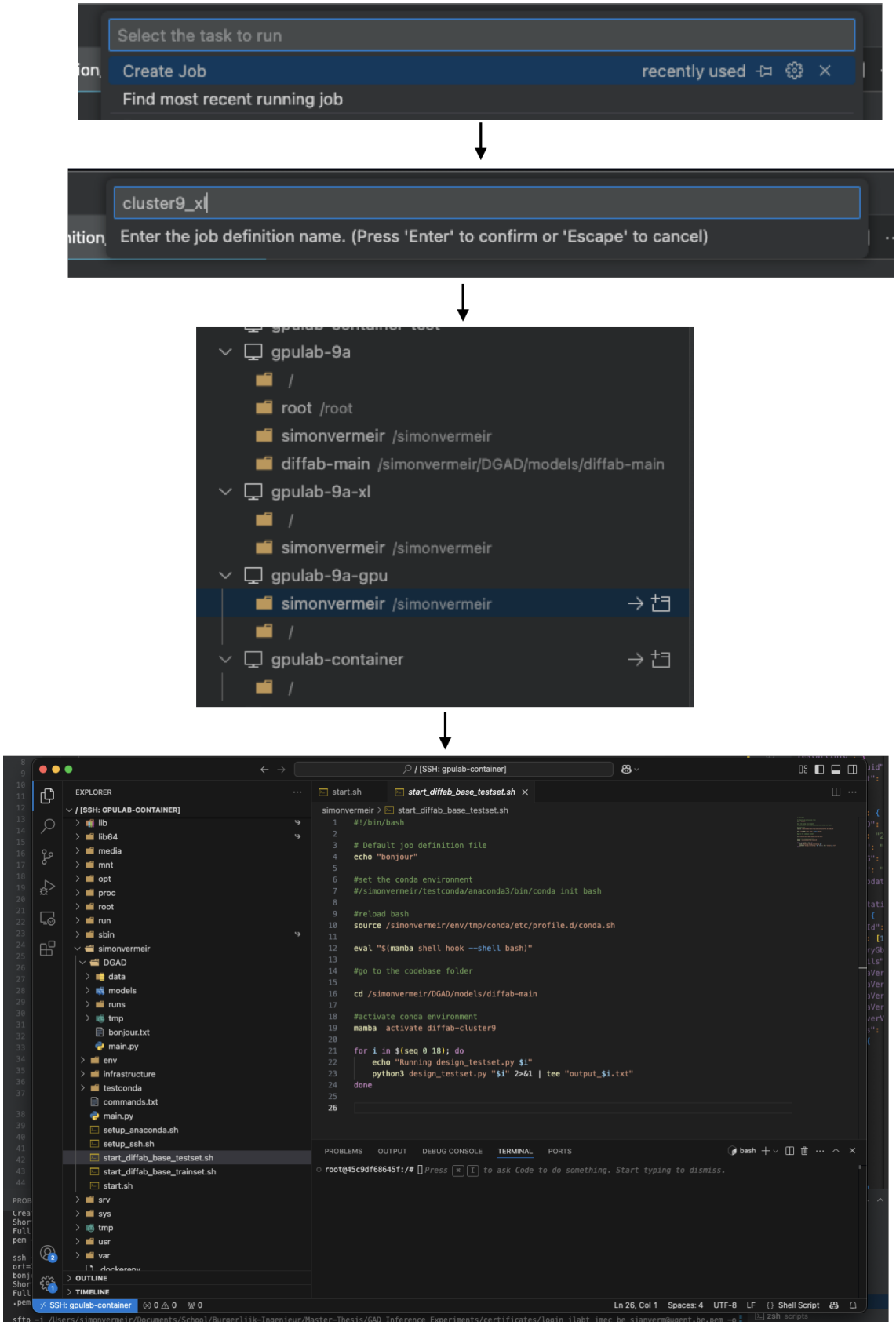


Figure C.1: My typical workflow. Start a job with a specific template. Configure remote/local using my scripts, and connect! That's it, super quick and no time is wasted connecting and configuring the remote.

ID	Project	Username	Nai	Status	GPU's	CPU's	Mem	Cost	Updated	Host
4dd066	students-drugd...	sjanverm	trai...	RUNNING	1	8	256	?	a few seconds ago	8A
084545	students-drugd...	sjanverm	trai...	RUNNING	1	8	256	?	a minute ago	8C
f23f41	students-drugd...	sjanverm	dga...	RUNNING	1	8	256	?	34 minutes ago	9A
27ae6b	students-drugd...	sjanverm	Co...	RUNNING	0	1	16	?	11 hours ago	9A
07c857	students-drugd...	sjanverm	trai...	RUNNING	1	4	32	?	a day ago	8B
c0d495	students-drugd...	sjanverm	trai...	RUNNING	1	4	32	?	a day ago	8B
b14a4a	students-drugd...	sjanverm	trai...	RUNNING	1	4	32	?	a day ago	8C
b8f3f1	students-drugd...	sjanverm	trai...	RUNNING	1	4	32	?	a day ago	8C
b834fc	students-drugd...	sjanverm	trai...	RUNNING	1	4	32	?	a day ago	8C

Figure C.2: This shows multiple parallel training runs running. Easily prepared via my job preparation flow. This flow consists of having organized output folders, start scripts, prepared environments etc. . Thanks to the easy management of the remotes via VScode (see figure C.1) everything can be prepared and pre tested. Once everything is prepared these runs can all be launched in parallel in a matter of minutes.

Chapter D

Extended Abstract

Negative guidance with an auxiliary model improves diffusion models for generative antibody design

Simon Vermeir

September 2025

Abstract

Selecting new drug candidates such as antibodies is a challenging task. In today's drug design tool chain generative diffusion models are employed. However, these models require large datasets to train on. In antibody design this is a problem as the amount of publicly available data is limited. To counter this issue, we propose the use of a negative guidance technique to improve sample quality without increasing dataset size.

Recently a new technique for image diffusion has been proposed that aims to get samples closer to the original ground truth data distribution called SIMS. An auxiliary model is trained on synthetic data from a base model trained on the original dataset. At inference time guidance is done between both. The reasoning behind this is that the bias between base and auxiliary has traits of the bias between base and the ground truth. This is used to shift base closer to the ground truth. They show state of the art FID scores on imagenet. Our hypothesis is that by employing this technique on antibody-antigen diffusion similar improvements can be seen.

The proposed integration of SIMS has been applied to Diffab, an accessible and trainable antibody-antigen diffusion model. When a CDR-region is masked out on an antigen-antibody complex, Diffab is able to generate a new CDR-region that fits the rest of the antibody and the antigen. To verify the technique a base model was trained from SabDab using the Diffab framework. Next an auxiliary model initialized with weights from base was trained with synthetic data from base. To achieve reasonable training times a two step training approach was employed. Finally at inference time guidance is done on the predicted noise ϵ by using both models: $\epsilon_{guided} = (1+\omega)\epsilon_{base} - \omega\epsilon_{aux}$ where ω controls the guidance strength.

On a held out test set of 350 antibody-antigen complexes, the use of SIMS on the (x, y, z) positions has been shown to improve the RMSD of the generated CDRH3 regions by $\sim 8\%$ compared to the base model.

1 Introduction

Antibodies are natural occurring proteins that will identify foreign actors within the body, such as viruses. In immunology, the target that antibodies recognize is called an antigen. An antigen can be the whole foreign element, or just a part of it, such as the spike protein in COVID-19. The antibody will bind to the antigen, forming an antibody-antigen complex.

A new promising technique in the drug design pipeline is in silico generative antibody design. Whereby, we use generative (AI) techniques to create suitable antibodies for a particular target.

Diffusion models are a class of generative models that have gained popularity for their ability to generate high-quality samples from complex distributions. They work by modeling the process of diffusion, where data points are gradually transformed into noise and then reconstructed back into data.

The most well know form of diffusion is image diffusion. Here during training the model will learn to denoise an image that has been corrupted with gaussian noise.

Recently it has been discovered that just like for creating new images from noise, diffusion models can be employed to create novel antibodies. Just like in the case of image diffusion they will during their training learn to reconstruct a corrupted random representation of an antibody. Then during inference novel antibodies can be generated from this noisy state.

1.1 Background

Antibody structure An antibody has a heavy and light polypeptide chain. Each chain has three complementary determining regions. Called CDR1, CDR2, CDR3. When referring to region 3 of the heavy chain we refer to CDRH3. The regions are highly variable loops and determine the binding specificity of the antibody. The rest of the antibody is more conserved. The CDRH3 region is the most variable. When talking about the backbone it refers to the N, C and CA atoms of each residue. The side chains are the rest of the atoms in the amino acid, and determine the amino

acid type and it's properties. [1]

Diffusion models Diffusion models [[2], [3], [4]] are characterized by a forward and backward process. The forward process will gradually add noise until the data is a multi variate Gaussian distribution. The backward process will learn to approximate the ground truth denoiser that will reconstruct the data from noise. Crucially in this work is the denoising step of the backward process. This is done by predicting the noise ϵ that was added to the data. The learned denoiser looks like:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(x_{t-1};\, \mu_\theta(x_t, t),\, \Sigma_\theta(x_t, t)\right) \quad (1)$$

where $\Sigma_\theta(x_t, t)$ is the variance predicted by the model at step t. For the covariance a time dependent profile is used so that this does not need to be learned.

$\mu_\theta(x_t, t)$ can be learned directly or we can learn the amount of noise that was added to x_t. This is done trough $\epsilon_\theta(x_t, t)$

Given the noisier sample x_t and the predicted noise $\epsilon_\theta(x_t, t)$ the next step can be computed:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\,\epsilon_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z} \quad (2)$$

with $\alpha_t = 1 - \beta_t$ A higher α_t means more signal is preserved (less noise added). This is because it has an inverse relationship to β which is the variance of the noise added at that time step. When the variance is low you retain more of the previous step in the forward process (when set to zero for example you copy over the step), when it is high you add more noise. z is a point sampled from the prior distribution (typically a gaussian) σ controls the amount of noise to add to the backward process, this is needed because each diffusion step is a stochastic process. In each forward diffusion step you sample from a conditional gaussian distribution, not only at T! If σ_t would be zero then each step would be deterministic and we would lose sample diversity.

In this work the epsilon formulation will be used. The function to predict the added noise will be called **EpsilonNet** throughout this work. It will play a crucial role in applying the SIMS technique.

2 Foundations

2.1 Diffab

Diffab is a diffusion model that can generate new CDR-regions on an antibody-antigen complex. The model is trained on the SabDab dataset. During training one of the CDR-regions is masked out and the model has to reconstruct it. Diffab is from the paper "Antigen-Specific Antibody Design and Optimization with Diffusion-Based Generative Models" by **Luo et. al** [5].

Classically when diffusion is done for antibody design, the diffusion happens on the backbone representation of the antibody. After the backbone is generated an inverse folding model (such as proteinMPNN [6]) is used to get the full amino acid sequence. Finally the side chains are added using a tool such as PyRosetta [7].

Diffab is different in that it does joint optimization of the structure and the sequence. The model will predict for each residue the (x, y, z) position, rotation and the amino acid type. The model is equivariant to rotations and translations of the input structure.

The following representation is used during diffusion:

- **Position:**
  $p \in \mathbb{R}^3$
  Prior after T steps: $\mathcal{N}(0,\, \sigma_T^2 I_3)$.

- **Rotation:**
  $R \in \mathrm{SO}(3)$
  Prior after T steps: $\mathrm{Unif}(\mathrm{SO}(3))$.

- **Sequence:**
  $s_i \mid \pi_i \sim \mathrm{Cat}(\Sigma;\, \pi_i), \quad s_i \in \Sigma$
  Prior after T steps: $\mathrm{Cat}(\Sigma;\, \frac{1}{K}\mathbf{1})$.

This structure is represented in figure 1
Diffusion happens in the following way:

1. For step t: The position, rotation, sequence, context and time step are passed to the EpsilonNet. The "noise" is predicted for each of the three parts.

2. Each component is denoised using it's own denoiser. Adapted to the data representation. step $t-1$ is obtained.

3. This repeats for T steps until a clean sample is obtained.

The epsilon of diffab has three different heads (the last part of the neural network):

- **Position head**: This head is responsible for predicting the noise added to each position. It thus predicts ϵ like standard diffusion.

- **Orientation head**: This head predicts the next orientation that has to be applied.

- **Amino acid head**: This head predicts a categorical distribution over the amino acid types for each residue. The distribution represents the prior belief of the original distribution. It is a categorical distribution.

Figure 3 available in the appendix illustrates the denoising process for one diffusion step.

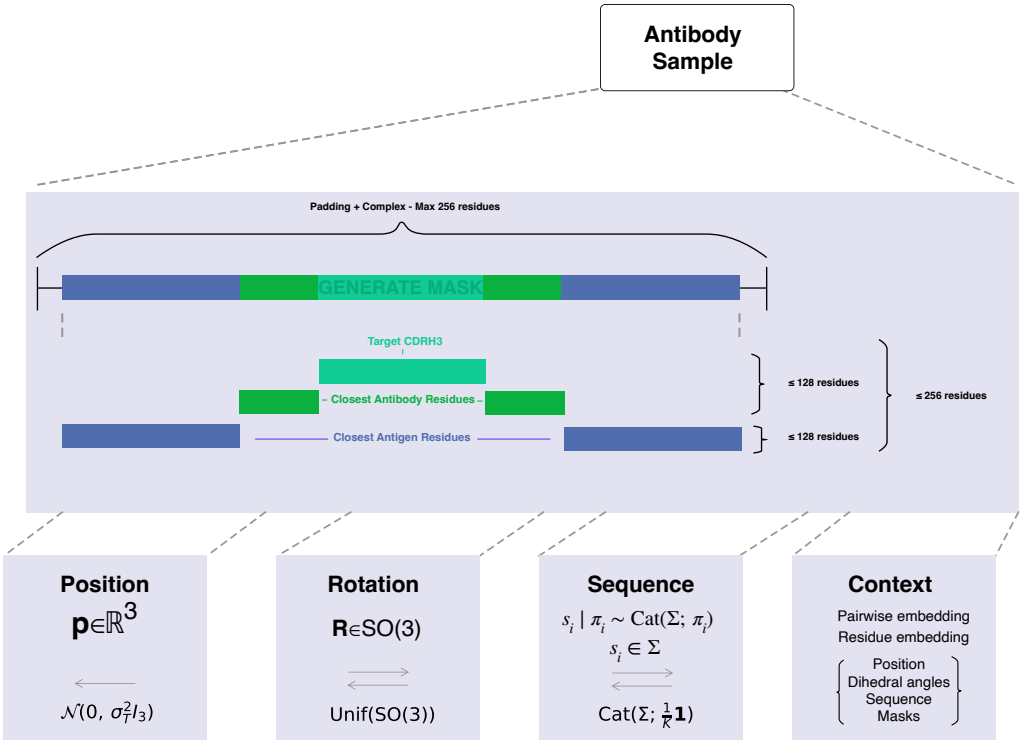


Figure 1: Antibody-Antigen representation for diffusion in Diffab.

2.2 SIMS

In the paper "Self-improving diffusion models with synthetic data" by **Alemohammad et al.** [8] the authors propose a new training paradigm for diffusion models that leverages self-synthesized data to improve the model's performance. The key idea is to use the model's own generated samples as a form of negative guidance during training, steering the model away from non-ideal synthetic data and towards the real data distribution.

The method involves training two models: a base model and an auxiliary model. The base model is trained on the original dataset, while the auxiliary model is trained on synthetic data generated by the base model. During inference, the predictions from both models are combined to guide the sampling process. The combined prediction is given by:

$$\epsilon_{guided} = (1 + \omega)\epsilon_{base} - \omega\epsilon_{aux} \tag{3}$$

The idea is that the bias between the base and auxiliary model has traits of the bias between the base and the ground truth. By subtracting the auxiliary model's prediction, the base model is nudged closer to the real data distribution. The parameter ω controls the strength of this guidance. Figure 4 and 5 illustrates the SIMS idea, they can be found in the appendix.

3 Methodology

3.1 Training the auxiliary model

A naive way to train an auxiliary model is to generate with the base model a new sample and pass it to the auxiliary model to train. You could either generate a new sample each time or form a dataset at the same time and reuse. The first approach makes training slow. The second although faster isn't ideal either since two models need to be loaded and there is still overhead.

Therefore, to train the auxiliary model a two step approach was used:

1. First the base model is used to generate synthetic samples. These are saved in a data base. Instead of saving the processed PDB files, the direct tensor representation used by the model is saved. This eliminates the need to reprocess the data each time. It also avoids processing artifacts. This saves processing time when training the auxiliary model and also makes sure that data is not organized differently by processing artifacts. The data is saved in LMDB [9]. LMDB is a key-value database that uses memory mapping to store data on disk. This makes it very fast to read data during training.
    - Note internally the model operates on backbone representation (position, rotation and

sequence). When a sampling step is done during training diffab expects the full structure (including sidechains, masks, etc) as input. However after sampling the model only returns the backbone, afterwich it is reformatted to the full structure. Therefore an adapted sampling procedure was created that outputs the correct tensor representation for input into the model. This way the output of one model (the baseline) can be used directly as input for the auxiliary model.

2. Next the auxiliary model is trained on the synthetic data. A special dataloader is used that will read the data from the LMDB database and reconstruct the tensor representation used by the model.

This approach is outlined in figure 6, which is available in the appendix.

3.2 Guidance on position

To evaluate the effectiveness of SIMS, the technique was applied on the position of each residue. This was chosen since the diffusion process on the position is similar too the diffusion process on images. The position is represented as a (x, y, z) coordinate in 3D space. Applying guidance on the predicted noise for the position is a euclidian operation.

Within the diffab implementation the EpsilonNet predicts both position, rotation and amino acid type. During inference two epsilon nets are used, one for the base model and one for the auxiliary model. The guidance is only applied on the position part of the prediction. One drawback is that two models need to be loaded into memory. Before the position is denoised guidance is done on the predicted noise as shown in figure 2.

4 Extensions

Aside from simply applying guidance on the position, other types of guidance can be applied. The following extensions were considered:

Guidance on sequence The sequence is represented as a categorical distribution over the 20 amino acids. The EpsilonNet predicts for each residue a categorical distribution over the amino acids. This represents the prior belief of the original distribution. To apply guidance on this distribution we can use the following approach:

The last layer of the sequence head is a softmax layer. This means that the output is a probability distribution over the amino acids. To apply guidance this layer is removed next the logits are obtained. These logits are mixed using the SIMS formula. Next a softmax is applied again to obtain a valid probability distribution. Finally the denoising step is applied using this new distribution.

Guidance on rotation Applying guidance on rotation is not as straightforward as on position. The rotation is represented as a rotation matrix in SO(3). This means that the rotation is not a euclidian space. Therefore the guidance can not be applied directly on the prediction from the epsilon which predicts the next rotation that has to be applied. This way of applying has not been solved yet, however a possible solution could be:

- Conceptually we want to interpolate between two rotations. This can be done using lie algebra in tangent space. In this space we can apply guidance as a euclidean operation. Important to note is that the current rotation is applied on the sequence of rotations before it, therefore the predicted rotations need to be anchored at the current rotation.

Combined guidance Multiple types of guidance can be combined to improve the overall performance. In the hope that combining guidance on position, rotation and sequence will yield better results than just one type of guidance. To have reliable combined guidance it is important that each type of guidance has its own tuned guidance strength ω. This is because the different types of data have different characteristics and scales. For example the position is a continuous variable in 3D space, while the sequence is a categorical variable. Therefore the impact of guidance on each type of data will be different.

Guidance schedule A guidance schedule can be used. This means that the guidance strength ω can be changed during the diffusion process for each type of guidance. For example: at the start no guidance can be applied to let the model 'warm up'. Next positional guidance can be increased in a linear way and kept steady for a few steps. Next rotational guidance can be turned on, and finally sequence guidance can be applied. The rationale is that the in this way the different types wont interfere. At the end you can also turn off guidance to let the model 'polish' the sample.

5 Metrics

Three metrics are used to evaluate the performance of the model: Root Mean Square Deviation (RMSD), which captures the difference between the generated and ground truth structure. Amino Acid Recovery (AAR), which measures the percentage of correctly predicted amino acids in the CDRH3 region. Finally, Perplexity Pseudo Log Likelihood (PPPL) is used to evaluate the biological plausibility of the generated sequences.

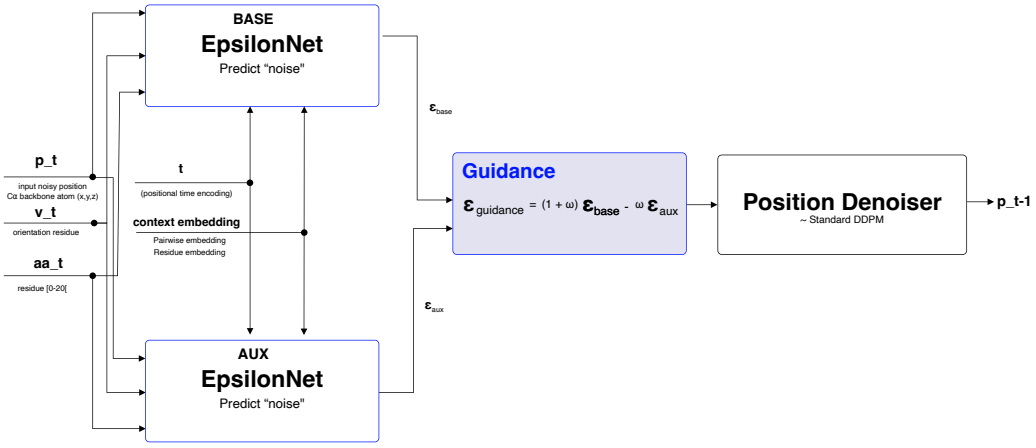


Figure 2: Guidance on position during inference. Two epsilon nets are used, one for the base model and one for the auxiliary model. Guidance is applied on the predicted noise for the position only.

6 Results

Table 1 highlights the results of applying SIMS to Diffab. The results are obtained on a held out test set of 246 antibody-antigen complexes from SabDab. The RMSD is calculated on the generated CDRH3 region. For completeness an ablation study on the guidance strength ω is shown in table 2, provided in the appendix. Additionally study on sequence is shown in table 3. They are provided as supplementary material in the appendix, for more details see accompanying thesis.

Baseline is the base model trained from scratch using the newly discussed split and only on the CDRH3 region. Balanced **sampling exp 0.75** is the same model but trained with a more balanced sampling strategy (using the word2vec style sampling). **SIMS pos 0.75** is the base model with SIMS applied to position only, with a guidance strength of 0.75. **SIMS pos 0.75 (with decay)** is the same but using a decay. The decay is a linear decay starting from the initial guidance strength towards zero. The schedule is: no guidance for 10 steps then gradually increase guidance for position over 5 steps keep on for 40 steps. Then keep off.

The best results are obtained when using a guidance strength of 0.75 and without using decay or a schedule. The 0.75 is obtained via an ablation study which can be found in the accompanying thesis along with more results on guidance on sequence and using different setups. This results in a $\sim$ 8% improvement over the baseline.

Most signal in the results can be obtained by looking at the RMSD since here guidance is only applied on the position which affects the structure. RMSD is better for all our implemented methods (lower is better).

Some small deviations in AAR are visible too this is because a joint epsilon net is used with three different heads so the a change in structure also affects the output of the sequence head of the epsilon net. The AAR for our methods (compared to baseline) is slightly higher (higher is better). Consequently the PPPL is also affected. For the guidance methods it's slightly lower (better).

7 Conclusion and future work

7.1 Future work

- Guidance mask in stead of equal guidance
  - In stead of applying equal guidance on each residue, a mask could be used to apply more guidance on residues that are further away from the ground truth. This mask could be learned.
- Classifier free guidance [10] using a learned guidance mask
  - Ommit using an auxiliary model. Instead train the base model with conditional information and with a mask that indicates which residues to apply more guidance on.
  - This can also be extended to learning the guidance mask trough reinforcement learning processes.

7.2 Conclusion

In this work the SIMS technique was applied to Diffab. The auxiliary model was trained using a two step approach to save training time. Guidance was applied on the position of each residue during inference. This resulted in a $\sim$ 8% improvement in RMSD on the CDRH3 region on a held out test set of 350 antibody-antigen complexes from SabDab. Successfully demonstrating the potential of guidance techniques to improve generative antibody design.

Table 1: Results of applying SIMS to Diffab on a held out test set of 350 antibody-antigen complexes from SabDab. The RMSD is calculated on the generated CDRH3 region. The best results are obtained when using a guidance strength of 0.75 and without using decay or a schedule. This results in a $\sim 8\%$ improvement over the baseline. For each complex 8 samples were generated, the mean is computed over all samples, the std is the standard deviation and [min, max] is the range of values over all samples within the test set. Lower is better for rmsd and pll_perplexity, higher is better for aar.

	rmsd		aar		pll_perplexity	
experiment_id	mean ± std	[min, max]	mean ± std	[min, max]	mean ± std	[min, max]
baseline	4.13 ± 3.71	[0.60, 62.51]	0.27 ± 0.14	[0.00, 0.88]	9.29 ± 3.33	[2.66, 28.49]
balanced sampling exp 0.75	3.69 ± 2.48	[0.57, 17.83]	0.28 ± 0.13	[0.00, 0.88]	10.47 ± 3.75	[2.18, 32.82]
sims pos 0.75 (with decay)	3.81 ± 2.80	[0.57, 53.64]	0.28 ± 0.14	[0.00, 0.88]	9.04 ± 3.25	[2.86, 31.12]
sims pos 0.75 (with schedule)	4.05 ± 3.40	[0.58, 59.77]	0.28 ± 0.14	[0.00, 0.88]	9.10 ± 3.27	[2.86, 31.12]
sims pos 0.75	4.03 ± 3.66	[0.58, 78.62]	0.28 ± 0.14	[0.00, 0.88]	9.06 ± 3.25	[2.64, 30.54]

A Diffab

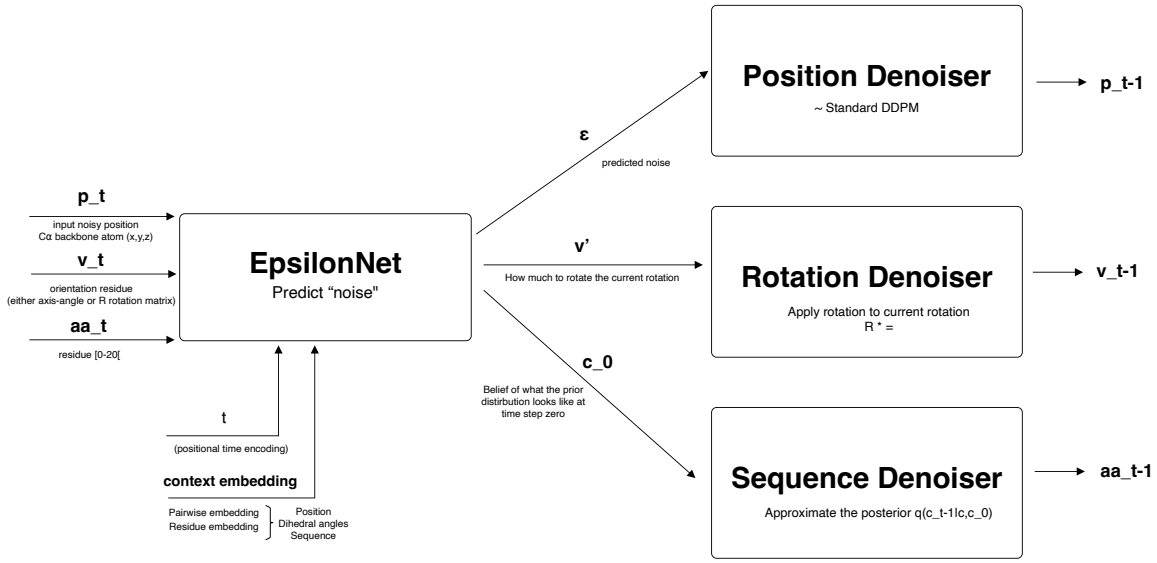


Figure 3: The denoising process for one diffusion step in Diffab. The position, rotation, sequence, context and time step are passed to the EpsilonNet. The "noise" is predicted for each of the three parts. Each component is denoised using it's own denoiser. This repeats for T steps until a clean sample is obtained.

B SIMS

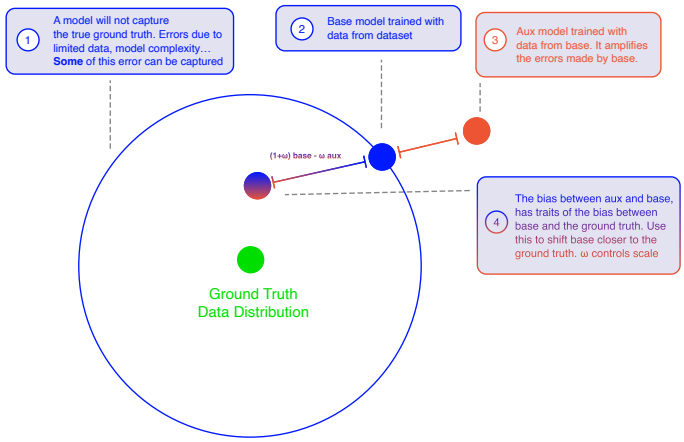


Figure 4: SIMS idea: follow the numbers on the figure for an intuitive explanation.

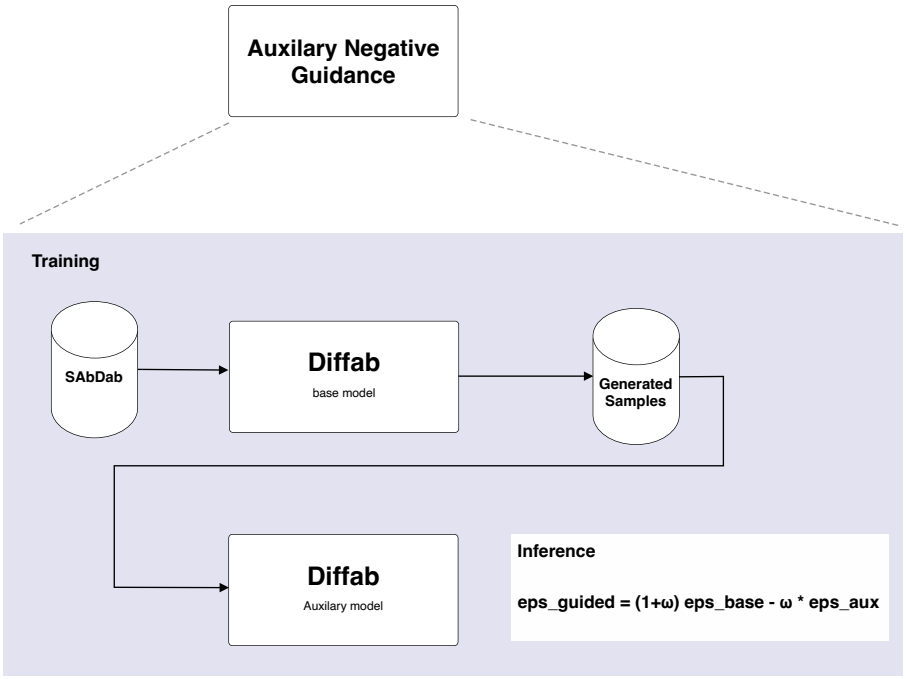


Figure 5: SIMS: The sims procedure, training the auxiliary model and inference with guidance.

C Training of the auxiliary model

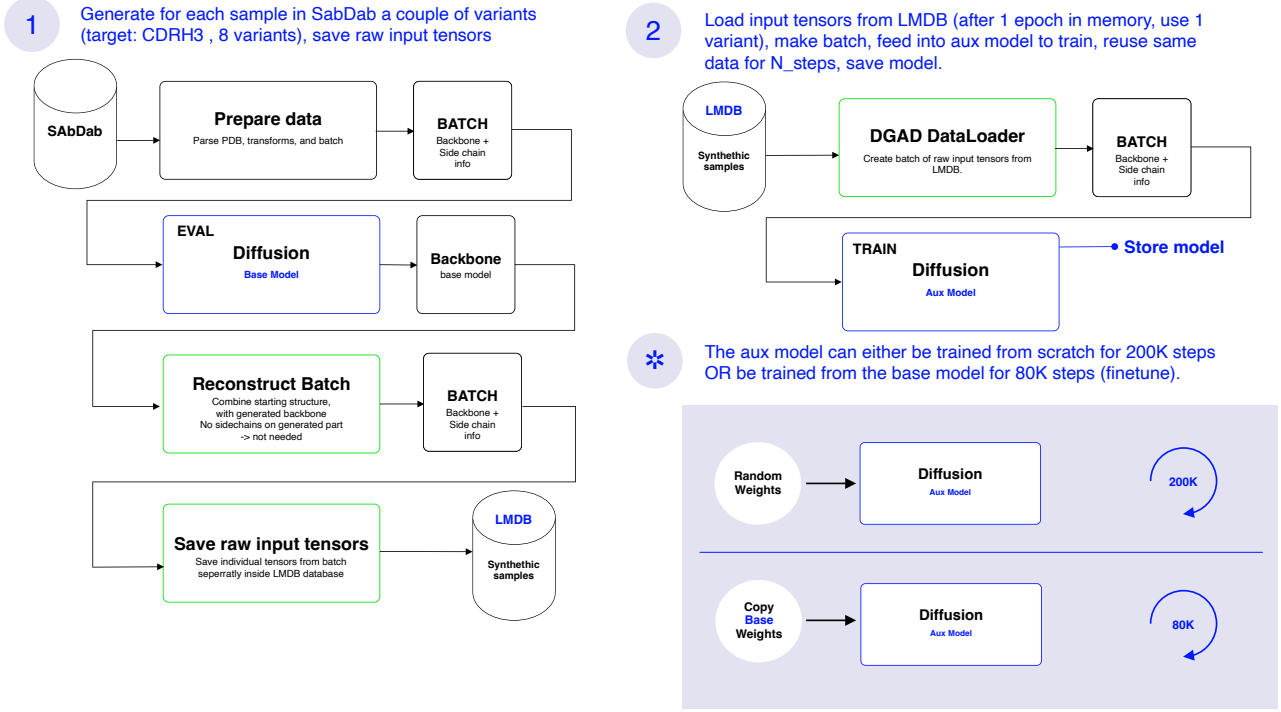


Figure 6: Training the auxiliary model using a two step approach.

D Ablation studies

Table 2: Results of ablation study on guidance strength ω. The results are obtained on a held out test set of 37 antibody-antigen complexes from SabDab. The RMSD is calculated on the generated CDRH3 region. The best results overall (taking in to account min/max) are obtained when using a guidance strength of 0.75. For each complex 8 samples were generated, the mean is computed over all samples, the std is the standard deviation and [min, max] is the range of values over all samples within the test set. Lower is better for rmsd.

	rmsd	
	mean ± std	[min, max]
baseline	3.62 ± 3.18	[0.70, 40.12]
from base pos guidance 0.75	3.29 ± 2.03	[0.62, 12.78]
from base pos guidance 1.0	3.30 ± 2.03	[0.62, 13.43]
from base pos guidance 1.5	3.42 ± 2.23	[0.55, 16.05]
from scratch pos guidance 0.5	3.34 ± 2.16	[0.67, 12.79]
from scratch pos guidance 0.75	3.35 ± 2.17	[0.74, 13.26]
from scratch pos guidance 1.0	3.45 ± 2.29	[0.81, 14.11]
from scratch pos guidance 1.5	3.73 ± 2.73	[0.76, 15.72]
from base pos guidance 0.5	3.34 ± 2.14	[0.66, 12.58]

Table 3: Results of ablation study on guidance on sequence. The results are obtained on a held out test set of 351 antibody-antigen complexes from SabDab. The RMSD is calculated on the generated CDRH3 region. The best results overall (taking in to account min/max) are obtained when using a guidance strength of 1.0 on sequence only. For each complex 8 samples were generated, the mean is computed over all samples, the std is the standard deviation and [min, max] is the range of values over all samples within the test set. Lower is better for rmsd and pll_perplexity, higher is better for aar.

	rmsd		aar		pll_perplexity	
	mean ± std	[min, max]	mean ± std	[min, max]	mean ± std	[min, max]
baseline	4.13 ± 3.71	[0.60, 62.51]	0.27 ± 0.14	[0.00, 0.88]	9.28 ± 3.34	[2.66, 28.49]
from scratch guidance on seq 1.0	4.51 ± 5.20	[0.59, 88.33]	0.31 ± 0.14	[0.00, 0.88]	6.37 ± 2.30	[1.43, 19.34]
from scratch guidance on seq 0.5	4.49 ± 5.16	[0.57, 88.17]	0.30 ± 0.14	[0.00, 0.88]	7.19 ± 2.62	[1.92, 26.95]

References

[1] B. Alberts, A. Johnson, J. Lewis, D. Morgan, M. Raff, K. Roberts, and P. Walter, *Molecular Biology of the Cell*, 6th edition. New York: Garland Science (Taylor & Francis Group), 2014, International Student Edition, ISBN: 9780815344643.

[2] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., series Proceedings of Machine Learning Research, volume 37, Lille, France: PMLR, Jul. 2015, pages 2256–2265. [Online]. Available: `https://proceedings.mlr.press/v37/sohl-dickstein15.html`.

[3] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., volume 32, Curran Associates, Inc., 2019. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2019/file/3001ef257407d5a371a96dcd947c7d93-Paper.pdf`.

[4] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., volume 33, Curran Associates, Inc., 2020, pages 6840–6851. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf`.

[5] S. Luo, Y. Su, X. Peng, S. Wang, J. Peng, and J. Ma, "Antigen-Specific Antibody Design and Optimization with Diffusion-Based Generative Models for Protein Structures," *Advances in Neural Information Processing Systems*, volume 35, pages 1–13, 2022, ISSN: 10495258.

[6] J. Dauparas et al., "Robust deep learning–based protein sequence design using proteinmpnn," *Science*, volume 378, number 6615, pages 49–56, 2022. DOI: `10.1126/science.add2187`. eprint: `https://www.science.org/doi/pdf/10.1126/science.add2187`. [Online]. Available: `https://www.science.org/doi/abs/10.1126/science.add2187`.

[7] S. Chaudhury, S. Lyskov, and J. J. Gray, "Pyrosetta: A script-based interface for implementing molecular modeling algorithms using rosetta," *Bioinformatics*, volume 26, number 5, pages 689–691, 2010. DOI: `10.1093/bioinformatics/btq007`.

[8] S. Alemohammad, A. I. Humayun, S. Agarwal, J. Collomosse, and R. Baraniuk, "Self-Improving Diffusion Models with Synthetic Data," 2024. arXiv: `2408.16333`. [Online]. Available: `http://arxiv.org/abs/2408.16333`.

[9] H. Chu, *Lightning memory-mapped database (lmdb)*, `https://www.lmdb.tech/doc/`, Symas Corporation; OpenLDAP Project. Accessed 2025-09-08, 2011.

[10] J. Ho and T. Salimans, "Classifier-free diffusion guidance," *arXiv preprint arXiv:2207.12598*, 2022.

Bibliography

[1] Josh Abramson et al. “Accurate structure prediction of biomolecular interactions with AlphaFold 3”. In: *Nature* 630.8016 (2024), pp. 493–500. ISSN: 14764687. DOI: 10.1038/s41586-024-07487-w.

[2] Bruce Alberts et al. *Molecular Biology of the Cell*. 6th ed. International Student Edition. New York: Garland Science (Taylor & Francis Group), 2014. ISBN: 9780815344643.

[3] Sina Alemohammad et al. “Self-Improving Diffusion Models with Synthetic Data”. In: (2024). arXiv: 2408.16333. URL: http://arxiv.org/abs/2408.16333.

[4] Helen M. Berman et al. “The Protein Data Bank”. In: *Nucleic Acids Research* 28.1 (Jan. 2000), pp. 235–242. ISSN: 0305-1048. DOI: 10.1093/nar/28.1.235. eprint: https://academic.oup.com/nar/article-pdf/28/1/235/9895144/280235.pdf. URL: https://doi.org/10.1093/nar/28.1.235.

[5] Democratizing Biomolecular et al. “Boltz-1”. In: (2024), pp. 1–19.

[6] Stephen K Burley et al. “RCSB Protein Data Bank (RCSB.org): delivery of experimentally-determined PDB structures alongside one million computed structure models of proteins from artificial intelligence/machine learning”. In: *Nucleic Acids Research* 51.D1 (Nov. 2022), pp. D488–D508. ISSN: 0305-1048. DOI: 10.1093/nar/gkac1077. eprint: https://academic.oup.com/nar/article-pdf/51/D1/D488/48440533/gkac1077.pdf. URL: https://doi.org/10.1093/nar/gkac1077.

[7] Sumit Chaudhury, Sergey Lyskov, and Jeffrey J. Gray. “PyRosetta: a script-based interface for implementing molecular modeling algorithms using Rosetta”. In: *Bioinformatics* 26.5 (2010), pp. 689–691. DOI: 10.1093/bioinformatics/btq007.

[8] Howard Chu. *Lightning Memory-Mapped Database (LMDB)*. https://www.lmdb.tech/doc/. Symas Corporation; OpenLDAP Project. Accessed 2025-09-08. 2011.

[9] J. Dauparas et al. “Robust deep learning–based protein sequence design using ProteinMPNN”. In: *Science* 378.6615 (2022), pp. 49–56. DOI: 10.1126/science.add2187. eprint: https://www.science.org/doi/pdf/10.1126/science.add2187. URL: https://www.science.org/doi/abs/10.1126/science.add2187.

[10] James Dunbar et al. “SAbDab: the structural antibody database”. In: *Nucleic Acids Research* 42.D1 (2014), pp. D1140–D1146. DOI: 10.1093/nar/gkt1043.

[11] Ethan Eade. “Lie Groups for Computer Vision”. In: *Website* (2014), pp. 1–15. URL: http://ethaneade.com/lie_groups.pdf.

[12] Fabian B. Fuchs et al. “SE(3)-transformers: 3D roto-translation equivariant attention networks”. In: *Advances in Neural Information Processing Systems* 2020-December.3 (2020). ISSN: 10495258. arXiv: 2006.10503.

[13] T Hayes et al. “Simulating 500 million years of evolution with a language model”. In: 0018 (2025), pp. 1–16. DOI: 10.1126/science.ads0018.

[14] Angel Herraez. *Dihedral angles animation [animated GIF]*. https://proteopedia.org/wiki/index.php/Image:Dihedral-angles-anim.gif. Proteopedia - Life in 3D. CC BY-SA 3.0. Accessed 2025-09-02. 2018.

[15] Jonathan Ho and Tim Salimans. “Classifier-Free Diffusion Guidance”. In: *arXiv preprint arXiv:2207.12598* (2022).

[16] Michael Jeltsch. *Antibody and its corresponding antigen [SVG]*. https://commons.wikimedia.org/wiki/File:Antibody_and_its_corresponding_antigen.svg. Wikimedia Commons. CC0 1.0 (Public Domain Dedication). Accessed: 2025-09-02. Mar. 2024.

[17] W. Kabsch. “A solution for the best rotation to relate two sets of vectors”. In: *Acta Crystallographica Section A* 32.5 (Sept. 1976), pp. 922–923. DOI: 10.1107/S0567739476001873. URL: https://doi.org/10.1107/S0567739476001873.

[18] Diederik P. Kingma and Max Welling. “Auto-encoding variational bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings* Ml (2014), pp. 1–14. DOI: 10.61603/ceas.v2i1.33. arXiv: 1312.6114.

[19] Omer Levy and Yoav Goldberg. “word2vec Explained: Deriving Mikolov et al.’s Negative-Sampling Word-Embedding Method”. In: *arXiv preprint arXiv:1402.3722* (2014).

[20] Zeming Lin et al. “Evolutionary-scale prediction of atomic-level protein structure with a language model”. In: *Science* 379.6637 (2023), pp. 1123–1130. DOI: 10.1126/science.ade2574. eprint: https://www.science.org/doi/pdf/10.1126/science.ade2574. URL: https://www.science.org/doi/abs/10.1126/science.ade2574.

[21] Calvin Luo. “Understanding Diffusion Models: A Unified Perspective”. In: (2022), pp. 1–23. arXiv: 2208.11970. URL: http://arxiv.org/abs/2208.11970.

[22] Shitong Luo et al. “Antigen-Specific Antibody Design and Optimization with Diffusion-Based Generative Models for Protein Structures”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 1–13. ISSN: 10495258.

[23] Derek M. Mason et al. “Optimization of therapeutic antibodies by predicting antigen specificity from antibody sequence via deep learning”. In: *Nature Biomedical Engineering* 5.6 (2021), pp. 600–612. ISSN: 2157846X. DOI: 10.1038/s41551-021-00699-9.

[24] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 26. 2013.

[25] NIAID Visual & Medical Arts. *B Cell Receptor*. https://bioart.niaid.nih.gov/bioart/617. NIAID NIH BIOART Source. Accessed: 2025-09-02. Mar. 2025.

[26] NIAID Visual & Medical Arts. *B Cell with IgM Receptors*. https://bioart.niaid.nih.gov/bioart/612. NIAID NIH BIOART Source. Accessed: 2025-08-28. Mar. 2025.

[27] *Research Infrastructure (iLab.t)*. https://idlab.technology/infrastructure/. Accessed: 2025-08-25. Imec, IDLab, 2025.

[28] Robin Rombach et al. "High-Resolution Image Synthesis With Latent Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 10684–10695.

[29] Mariana (LadyofHats) Ruiz Villarreal. *Main protein structure levels (English) [SVG]*. https://commons.wikimedia.org/wiki/File:Main_protein_structure_levels_en.svg. Wikimedia Commons. Public domain (released by the author). Accessed: 2025-09-02. Oct. 2008.

[30] Julian Salazar et al. "Masked Language Model Scoring". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020. DOI: 10.18653/v1/2020.acl-main.240. URL: http://dx.doi.org/10.18653/v1/2020.acl-main.240.

[31] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[32] Martin Steinegger and Johannes Söding. "MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets". In: *Nature Biotechnology* 35.11 (2017), pp. 1026–1028. DOI: 10.1038/nbt.3988.

[33] Masatoshi Uehara et al. "Understanding Reinforcement Learning-Based Fine-Tuning of Diffusion Models: A Tutorial and Review". In: (2024), pp. 1–44. arXiv: 2407.13734. URL: http://arxiv.org/abs/2407.13734.

[34] UGent. *VERANTWOORD GEBRUIK GENERATIEVE AI IN DE MASTERPROEF FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR*. URL: https://www.ugent.be/ea/nl/faculteit/studentenadministratie/masterproef/ai_2425.pdf (visited on 05/03/2025).

[35] Joseph L. Watson et al. "De novo design of protein structure and function with RFdiffusion". In: *Nature* 620.7976 (2023), pp. 1089–1100. ISSN: 14764687. DOI: 10.1038/s41586-023-06415-8.

[36] Xiangxin Zhou et al. "Antigen-Specific Antibody Design via Direct Energy-based Preference Optimization". In: (2024). arXiv: 2403.16576. URL: http://arxiv.org/abs/2403.16576.