Software Engineering in protein design
WASP Cloud Computing and Software Engineering

Sofia Andersson
Lund University
August 30, 2024

Introduction

Ever since the introduction of AlphaFold2 in 2020, the usage of deep learning for protein structure prediction has increased at a high rate[1]. AlphaFold is thought by many to have solved the problem of protein folding, but while it can predict most protein structures at a very high accuracy, it cannot tell us how this folding works. This is especially important for the field of protein design, which had historically relied on the understanding of protein folding to design new proteins. However, with the dawn of deep learning, more and more protein design methodologies are being developed using sophisticated machine learning methods, such as RFDiffusion[2].

An area of the field that is especially interesting for understanding protein folding further is being able to design a protein that can take on more than one conformation (or shape), as the vast majority of designed proteins are very stable and do not possess the inherent flexibility that most natural proteins have[3]. Conformational changes in proteins are vital for protein functions in the body in order for them to catalyse reactions and function properly, yet we are currently unable to reliably design these types of proteins[4]. One attractive prospect for this technology is for drug delivery, where a target molecule is encased within a fold changing protein that only releases the drug when in the presence of the diseased cells. While this dream is far away right now, there is currently a lot of research being made on how to design these types of proteins. My project joins this field of research and aims to design a protein design methodology which produces these multi-state proteins.

Robert lectures

Verification and Validation

In the field of protein design, the ability to computationally evaluate a given design before sending it off for experimental validation is crucial. This is because protein expression is both costly and time-consuming. However, unlike many types of traditional software engineering projects, protein design does not require a super high success rate. While many other fields require high uptime and success rate, the process of protein design is both stochastic and complex, so as long as we can relatively reliably identify whether a design is good or not at the *in silico* stage, it is acceptable if only 1 in 100 designs are useable. We can simply run

the design software until we obtain good designs, as long as there is sufficient proof that the design works, and it is computationally feasible.

Verification and validation are important processes in protein design, though defined in a quite specific way. Verification, in this context, involves checking that the computational methods used to predict protein structures are accurate and reliable. Validation, on the other hand, involves checking that the designed proteins have the desired properties and functions. In protein design, both verification and validation can be challenging due to the complexity of protein structure and function. While computational methods such as AlphaFold are used to predict protein structures and functions, these methods are not always accurate, especially for completely *de novo* structures. Therefore, experimental validation is required to confirm the designs. However, experimental validation can be time-consuming, expensive, and may not always be feasible. Hence, it is crucial to develop reliable computational methods for protein design that can be verified and validated.

"Shy Code" and "Best Practices"

Shy code is considered good practice in software engineering, being the practice of writing code that depends on as few other components as possible. Other good practices are not repeating yourself in code, and writing code that is easy to read and understand. Of course these are generally good practice and make both reading and maintaining code much easier. However, in the field of protein design specifically, this is often very difficult to attain due to the vast number of proprietary models that are all written by scientists instead of programmers.

Many state-of-the-art techniques, such as ProteinMPNN and RFDiffusion, are integrated into existing workflows[2, 5]. However, this integration often leads to being dependent on specific implementations of algorithms and locked Python dependencies. Rewriting these algorithms from scratch is not feasible due to resource and time constraints. Additionally, using these algorithms requires formatting the data to match their requirements and parsing their outputs into appropriate data structures for your needs.

Another challenge in protein design is the lack of updates to the state-of-the-art techniques. Oftentimes, the codebase is released to Github when the paper is published, and then summarily never updated ever again. When a new technique is released and not regularly updated, it can result in complex Python package requirements that take hours or even days to set up from scratch. One such specific instance was when Meta stopped updating ESMFold[6], a protein structure prediction module, which led to me having to rewrite large portions of my code to use a different implementation of ESMFold as it was not compatible with the packages available on a cluster that I was using.

Guest lectures

Generalised loss function (Volvo)

At the Volvo talk, the idea of "generalised loss functions" was introduced. It was based on the idea of "we have general classes for optimizers, why not also for loss?" which, to me, seems like a naive approach which I disagree with. If applied to my field, the idea would be that you could have a generalised loss function that could be applied to any type of protein

design problem. However, in protein design, there are so many different types of metrics and constraints that need to be considered in order to create proteins that look like and behave like real proteins. Depending on the purpose, there is no choice but to write manual loss functions. For example, KL Divergence has been used to compare distribution of coordinate points to compare to distributions of real proteins in order to see if a protein design is realistic. In another project of mine, I wrote a custom loss function which performed matrix operations on a matrix describing the orientation between different protein secondary structure elements. Both of these cases required writing manual loss functions, and there are many other examples that I do not have the time to cover here.

Furthermore, I would argue that we have generalised loss functions to some extent, but not in the way that was described in the talk. For example, in the field of protein design, we have generalised loss functions for predicting protein structure, such as RMSD, which is a measure of the average distance between the atoms of the predicted structure and the true structure. There are also classes in Pytorch which implement common losses, such as root mean squared error, cross-categorical entropy, and more. These are good base classes to work off of and take advantage of when you need to write a custom loss for a more complicated issue, but they are also excellent for simpler models. Imagine if you had to re-implement cross-categorical entropy every time you wanted to create a classification model!

Overinvestment in AI solutions (SAAB)

A topic that was brought up at the SAAB talk, albeit slightly jokingly, was that of "overinvestment in AI solutions". This is especially interesting as someone in the protein design field. While AI has revolutionised the field of protein design, it is not the be-all and end-all. There are many traditional methods that are still used in protein design, such as Rosetta, which is a suite of algorithms for protein structure prediction and design. Rosetta uses energy-based methods to both design and evaluate designs, which leads to long loops of energy minimization, evaluation, and repeating. One example of a protein that was designed using Rosetta is a oligomerisation pH switch, meaning a protein consisting of N subunits that changes to M subunits at a certain pH[7]. This is a very unique design that would be difficult to achieve using AI methods alone. In fact, one of my projects has been to try to recreate this using AI methods, which has proven challenging as AlphaFold cannot predict the original structure at all in either of the two states. While I have been able to make headway in designing candidates that are similar to the original structure, there has still been no experimental validation of my results.

One interesting project that embodies this is Chroma[8]. It is a diffusion protein design model that was hyped up to be an "RFDiffusion-killer", where RFDiffusion is the current state-of-the-art protein design model for generating protein backbones, when used in conjunction with ProteinMPNN to design the sequence, as RFDiffusion only generates structure. Chroma, on the other hand, generates both structure and sequence. It also uses an interesting concept called "conditioners", which are a way to enforce constraints on the structural level to a high degree. One of the main figures in the paper shows that the model is able to generate proteins in the shape of numbers and letters. This is of course a technological feat, but unfortunately, these are not physically feasible proteins. The experimental validation of these results were only for the most basic protein designs, and even then they were not as extensive as they should have been. When I tried to use the model in a project of mine, I ended up spending most of my time trying to make the model not generate poor designs. It is a cool model with interesting features, but it was also made by a company funded by venture capitalists to try to

cash in on the protein/drug design boom that is currently happening. In short, while there are good AI models for protein design, the field is getting flooded with poorly validated models made by programmers instead of biochemists. In my opinion, we should use AI protein design methods in conjunction with older energy-based models to finetune our designs, and then validate our results experimentally using proper protocols.

CAIN conference

Investigating the Impact of SOLID Design Principles on Machine Learning Code Understanding [9]

The SOLID design principles are a set of guidelines for writing clean, maintainable, and scalable code. They are widely used in software engineering to improve code quality and readability. In the paper that I chose, the authors investigated the impact of the SOLID design principles on machine learning code understanding. They conducted an empirical study to evaluate the readability and maintainability of machine learning code written with and without the SOLID design principles. They found that code written with the SOLID design principles was easier to both understand and maintain than code written without them. This indicates that following SOLID design principles can improve the quality of machine learning code and make it more accessible to outside parties.

While I am not a fan of object-oriented programming and would argue that most code written with clear design principles would be more readable than average code, the idea of just having well-defined design principles is particularly interesting in my field, as most of the code I work with is written by other biochemists without any formal computer science or software engineering training. The lack of programming experience often leads to code that is difficult to understand and maintain. On multiple occasions I have had to change parts of an algorithm to work for my specific dataset, which led to me spending a whole afternoon trying to understand why three functions all look nearly identical, with nearly identical names, but have slightly different outputs. By applying consistent design principles to machine learning code in protein design, we could improve the quality and readability of the code. This is something that I have tried to improve on myself as well, but when you are the only person who sees your code on a regular basis, it becomes too easy to take shortcuts.

I am not necessarily a proponent of strictly following SOLID design principles only, but there needs to be some sort of standard of clean and readable code if the field is going to progress as quickly as it is. These complicated trial-and-error code bases that only make sense to the author is making collaboration and wider adoption difficult. In my older code, I use a lot of subprocess calls to invoke these external proprietary scripts, as there was no clear or easy way to integrate them into my code. This has led to a lot of spaghetti code that is difficult to maintain and understand.

As I have come into contact with computer scientists and software engineers through WASP, I have noticed my own programming getting more and more purposeful. Instead of simply creating huge blocks of code without a main function, I see that I break up my code more and more. While I would not say that I am following SOLID principles, as most of my programming is functional rather than object-oriented, I am definitely moving in that direction. I am guilty of making esoteric test functions that I forget to remove sometimes, but as of writing this report, I am not currently in charge of maintaining a state-of-the-art protein design repository.

A Combinatorial Approach to Hyperparameter Optimization [10]

Hyperparameter optimization is a crucial step in training machine learning models. It involves tuning the hyperparameters of a model to achieve the best performance. However, hyperparameter tuning can be a time-consuming and challenging process, as I have personally experienced in multiple projects. In the paper that I chose, the authors proposed t-way testing for hyperparameter tuning, instead of the traditional grid searches and Bayesian optimisations. t-way testing is a combinatorial approach to software testing to identify faults with a test set that covers all possible combinations of t parameters from a total of n. The authors applied this approach to hyperparameter tuning and found that it was more efficient and effective than traditional methods. It is based on the principle that a single input parameter or interactions between two input parameters can cause a fault, and that testing all possible combinations of t parameters can help identify these faults faster than exhaustive searching[11]. They argue that this way of testing also applies to hyperparameter tuning and substantially narrows the search space.

This paper is particularly interesting to me as hyperparameter tuning is a crucial step in training machine learning models for protein design. I have spent countless hours tuning hyperparameters for my models, and it is often a trial-and-error process that requires a lot of computational resources. The more organised approach outlined in the paper seems promising, but there is an immediate caveat that come to mind. Their approach starts with identifying relevant hyperparameters and their important values for a given model. Instead of allowing for a range of values like Random Search and Bayesian optimisation, you need to define a predetermined set of important values for each hyperparameter. This, in and of itself, is a sort of hyperparameter tuning. While the main hyperparameter optimization is more efficient, you still have to do this initial analysis, which requires a sort of initial tuning, as previously mentioned. How do you know which hyperparameters are important without doing a grid search, Bayesian optimization, or random testing?

I might implement a version of this in my own research, as I am already familiar with the hyperparameter space of my current models due to a lot of testing, which means I essentially have the initial step covered. Of course you can cover the initial step with more general hyperparameter values from literature, but then your search will be less focused and less customised to your specific model. I welcome any approach that can save me time and resources, as hyperparameter tuning is often one of the most time-consuming parts of training a model. However, I do not know if this completely covers everything I would want in a hyperparameter tuning method. This approach does often converge to the best possible result, but it is frequently among the slowest, which is interesting when the model is supposed to decrease the search space. To me, that implies increased speed, but perhaps that is an interpretation error on my end. In any case, I am happy to see new developments in the field of hyperparameter tuning, as it is an optimisation rabbit hole that is easy to fall into when developing a model.

References

[1] J. Jumper et al., "Highly accurate protein structure prediction with AlphaFold", Nature **596**, 583–589 (2021).

[2] J. L. Watson et al., *Broadly applicable and accurate protein design by integrating structure prediction networks and diffusion generative models*, preprint (Biochemistry, Dec. 10, 2022).

[3] D. Baker, "What has de novo protein design taught us about protein folding and biophysics?", en, Protein Science **28**, 678–683 (2019).

[4] L. Orellana, "Large-scale conformational changes and protein function: breaking the in silico barrier", Frontiers in Molecular Biosciences **6**, 117 (2019).

[5] J. Dauparas et al., "Robust deep learning–based protein sequence design using ProteinMPNN", Science, eadd2187 (2022).

[6] Z. Lin et al., "Evolutionary-scale prediction of atomic-level protein structure with a language model", Science **379**, 1123–1130 (2023).

[7] R. Lizatović et al., "A De Novo Designed Coiled-Coil Peptide with a Reversible pH-Induced Oligomerization Switch", Structure **24**, 946–955 (2016).

[8] J. B. Ingraham et al., "Illuminating protein space with a programmable generative model", Nature, `10.1038/s41586-023-06728-8` (2023).

[9] R. Cabral et al., "Investigating the Impact of SOLID Design Principles on Machine Learning Code Understanding", in Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI (Apr. 14, 2024), pp. 7–17.

[10] K. Khadka et al., "A Combinatorial Approach to Hyperparameter Optimization", in Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI (Apr. 14, 2024), pp. 140–149.

[11] B. Texas, *Pragmatic software testing: becoming an effective and efficient test professional* (John Wiley and Sons, 2007).