PROTEINSCATTER: Visualizing Fragments of Structurally Similar Proteins with a Scatterplot

DONALD BERTUCCI

Oregon State University, Corvallis, Oregon, USA
bertuccd@oregonstate.edu

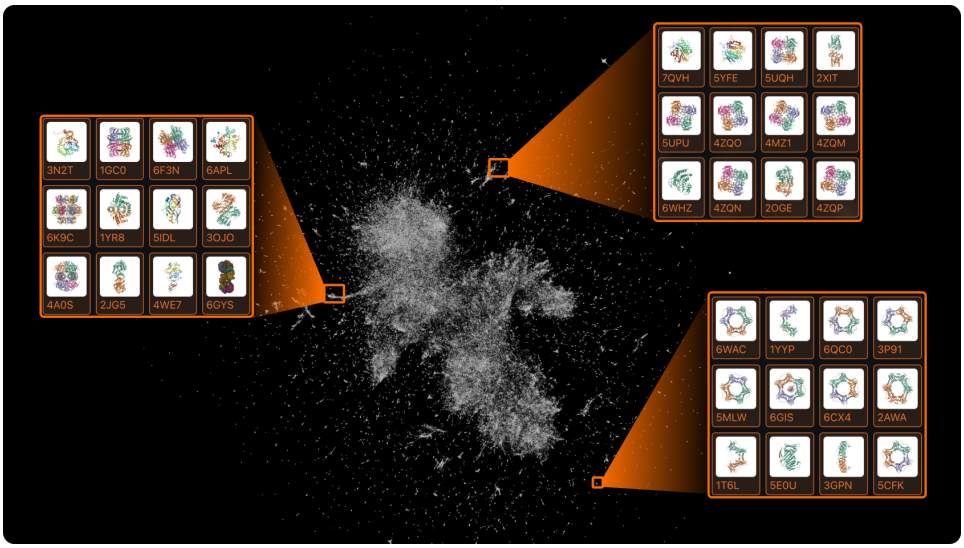

Figure 1: PROTEINSCATTER: each white point represents a protein. Close points represent similar proteins. A select few regions in orange are shown along with their Protein Data Bank [1] protein structures.

Abstract: The Venom Biochemistry and Molecular Biology Laboratory at Oregon State University has hundreds of protein structures, but with completely unknown functions. I've developed a method to map out three-dimensional (3D) protein structures from the Protein Data Bank to compare against their proteins with unknown functions. Specifically, I use a sequence representation that encodes 3D structure (using Foldseek's 3Di model), then I train a deep transformer model (I call 3Di-transformer) on those sequences. Finally, I visualize hundreds of thousands of neural network representations in two dimensions (2D) using the UMAP algorithm and a scatterplot. The PROTEINSCATTER implementation is described in this paper and open-sourced on GitHub https://github.com/xnought/protein-scatter.

1. Introduction

This paper aims to visualize protein similarity as a way to discover similar proteins with known functions. Specifically, this paper visualizes a subset of the Protein Data Bank [1] (PDB) proteins in a map and places proteins from the Venom Biochemistry and Molecular Biology Lab [2] (Venom Lab) within the map.

The final visualization called PROTEINSCATTER (as shown in Figure 1) is analogous to a geographic map, but where regions and clusters represent groups of similar 3D structures of known proteins. By placing our proteins with unknown function in this map, I can find similar structures across regions. Once you find similar proteins, you can then use their known function, or at least more information from the PDB, to start to hypothesize about your own structures.

To convert proteins into this map like in Figure 1, I trained a transformer model on a large

subset of the PDB proteins after first converting them to Foldseek's 3Di sequence [1,3,4]. To put it simply, this model aims to find which parts of the protein's 3D structure are the most important. Then, I can spatially place similar proteins in a 2D scatterplot with the UMAP algorithm [5].

For the exact details, see any one of these sections in the paper:

- The background outlining my thought process which led to the methods in Section 2.
- The methods for how I modeled the 3D structures and transformed them into the ProteinScatter in Section 3.
- The results showing how using ProteinScatter helps identify possible functions of proteins from the Venom Lab in Section 4.
- My concluding remarks and acknowledgements in Section 5.

2. Background

Proteins are variable in length: one protein may have a longer amino acid sequence than another. Unfortunately, Machine Learning (ML) algorithms thrive off of fixed length data structures. Since I want to use ML methods to model 3D protein structure and transform proteins into a 2D scatterplot, I need all of my data to have the same number of dimensions – not variable. In other words, I need a set of vectors with dimension d to represent my proteins.

One great way to convert some data into a fixed size vector of dimension d is with neural network embeddings. I'll give you an intuitive, but off topic, example: I can train a neural network to predict whether a dog image is a golden retriever or not. If the model is trained effectively and performs well, I can assume that the neural network was able to separate the vectors at layer l in the neural network sufficiently to perform well on the task as shown in Figure 2. I can apply the same idea to protein 3D structure to embed the proteins as vectors.
To extract protein embeddings, I first need to model the 3D structure of the proteins. The most effective strategy to date of modeling sequences is with attention based methods such as OpenAI's GPT models [3,6]. These models take a sequence of tokens and asks the model to generate the next token. The attention architecture works by asking what parts of the previous sequence should we pay attention to in order to predict the next token [3]. In the context of protein sequences,

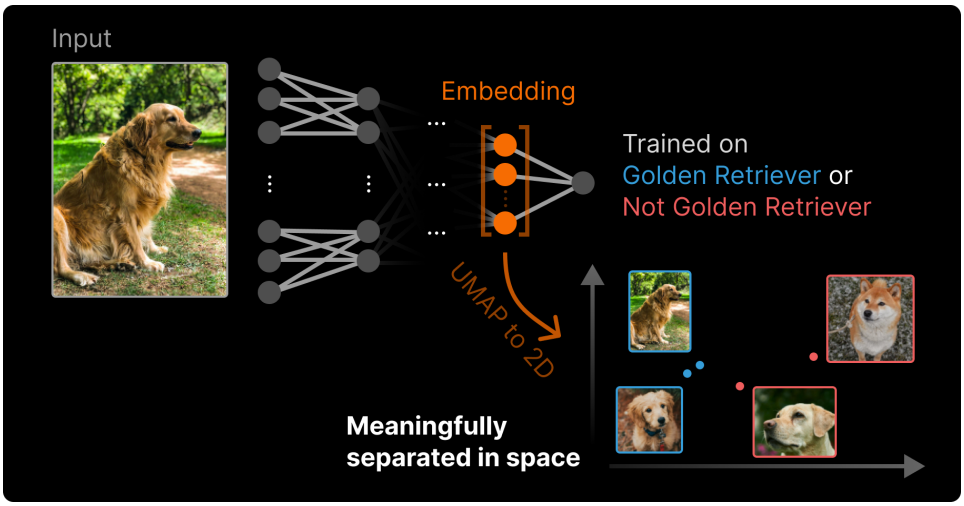

Figure 2: By training a neural network on a meaningful task, an intermediate representation, the embedding vector, separates in space based on similarity.

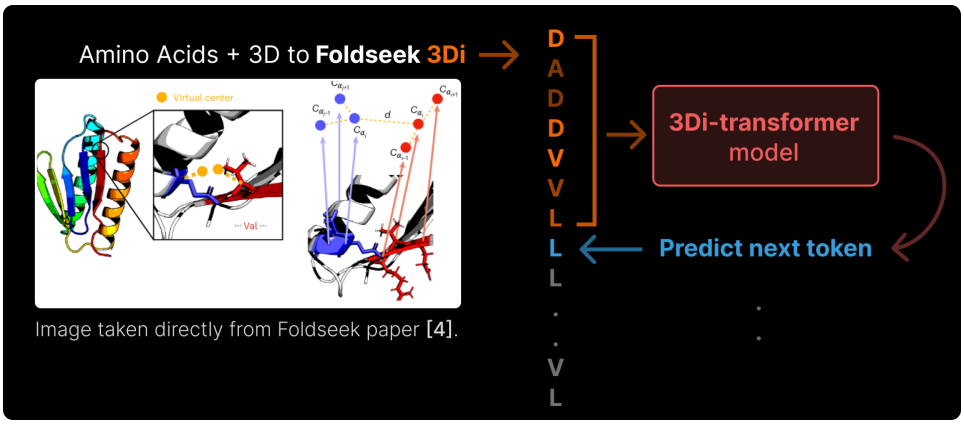

Figure 3: First, I convert the amino acid sequence and 3D structure to the Foldseek 3Di alphabet [4]. Then, I train a transformer model to predict the next 3Di letter in the sequence.

attention would mean asking which 3D structures are the most prominent/important: which 3D structures we should pay attention to.

To infuse 3D information into the amino acid sequence, I'll use the Foldseek 3Di alphabet [4]. I do this by converting each amino acid in the protein to 3Di (Foldseek's 3D code) by extracting angle and distance based information from each amino acid given 3D spatially close neighbors [4]. I've shown a figure directly taken from the Foldseek paper [4] in Figure 3 on the left. This local snapshot of 3D structure (the angles and distances) is then converted into one of 20 letters representing different geometries in the protein [4].

Next, I can train the sequence model and attempt to capture patterns in the 3D structure. As shown in Figure 3 on the right, I can train a model, which I'll call 3Di-transformer, to predict the next 3Di letter by modeling which previous 3Di letters are most important (attention). I can then easily extract embedding vectors for each protein from this 3Di-transformer model.

Note that the 3Di-transformer has a fixed block size that it can look at. In the case of Figure 3, I take in seven previous letters (in orange) as context to predict the next letter. To encode larger sequences than seven, I would have to iterate over the entire sequence in chunks of seven to get multiple embedding vectors to represent one protein. Hence the title "Visualizing **Fragments** Of Structurally Similar Proteins with a Scatterplot." Note that for the actual 3Di-transformer I use a much larger block size (context) of 128.

In essence, by training a model to use a context of 3D structures to predict the next 3D structure on a large database like the PDB, my 3Di-transformer will figure out how to model protein structure and which parts of the 3D structure are most useful. As a result, the embedding vectors will be useful for similarity comparison.

3. Methods

In this methods section, I describe how I trained the 3Di-transformer (Section 3.1). I then encode each protein as embedding vectors that can be reduced to 2D with the UMAP algorithm to create the ProteinScatter (Section 3.2).

3.1. Training the 3Di-transformer

I used PyTorch[1] to implement repeated blocks of multi-headed self-attention exactly like nanoGPT [7]. Based on the the sequence length context of 128, I predicted the next 3Di letter to

[1] https://pytorch.org/

Figure 4: Training and validation loss over 40 thousand iterations logged every 100 iterations. The lower the loss the better the model.

train the model. Since the PDB houses around 200 thousand proteins, I downloaded the entire database for training. First, I converted all the the PDB proteins into 3Di sequence representations with Foldseek [4]. Then, I trained the 3Di-transformer on those sequences.

The transformer model had a block_size of 128, vocab_size of 20 (the 20 possible 3Di letters), n_embd of 256 (embedding size), 8 n_heads per self-attention block, and 20 n_layers of self-attention (blocks repeated). In total there were around 15 million learnable parameters over the 20 layers. For more details on the exact architecture please see the model code[2]. I followed closely how GPT2 and nanoGPT are implemented [6, 7].

I trained the model on an Nvidia T4 GPU for just over four hours straight. I used batches of 64 proteins per forward pass for 40 thousand iterations. During training, the model saw around $64 * 128 * 40000 \approx 327$ million 3Di letters from the PDB. I randomly sampled the PDB proteins each batch, then randomly sampled a 128 block within the indexed protein. To optimize the model, I compared the model's predicted 3Di with the true 3Di in the sequence using Cross-Entropy loss [8] and performed gradient descent with the AdamW optimizer [9]. The loss function output is shown in Figure 4 where a lower loss is better. The validation set used five percent of the PDB. I used the validation set to test how the model does on unseen data. I stopped training once the validation loss flattened out as seen in Figure 4. For exact details, see the training code[3]. The trained weights are linked on the Github repository.

I extracted embeddings just after the last multi-headed self-attention block and just before the final dense layer of prediction probabilities. Since the n_embd parameter was 256, that would be the dimension of the embedding vector. After training, these embedding vectors now effectively represent the most important 3D structures of a given protein and are ready to be transformed into 2D.

[2] https://github.com/xnought/protein-scatter/blob/main/training/model.py
[3] https://github.com/xnought/protein-scatter/blob/main/training/train.py

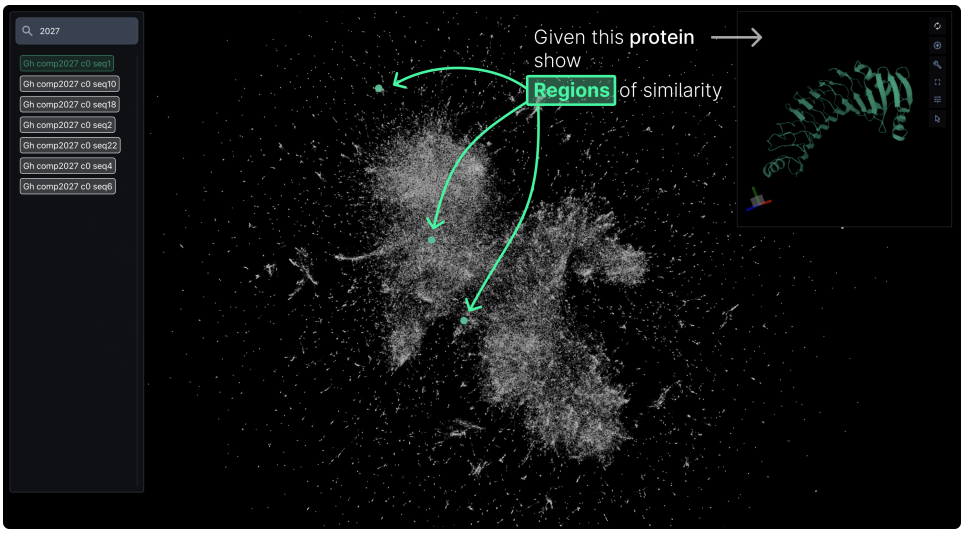

Figure 5: When you select a protein, the ProteinScatter highlights regions where you'll find structurally similar proteins.

3.2. UMAP to 2D

First, I need to iterate over each protein with a block size of 128 to extract the embedding vectors where each is of 256 dimensions. For example, an 80 residue long protein would only have $\lceil 80/128 \rceil = 1$ vector embedding while a 1000 long one would be $\lceil 1000/128 \rceil = 8$ vectors to represent pieces/fragments of that one protein.

I then used the UMAP algorithm [5] with n_neighbors of 8 to generate the visualization as shown in Figure 5. I experimented with higher n_neighbors, but a close local structure of 8 turned out visually the best.

UMAP roughly works by placing points randomly in the lower dimension (in this case 2D), then adjusting those points such that we maintain the same nearest neighbors they had in the higher dimension (in this case 256D). UMAP is actually much more sophisticated than what I've said since they use efficient optimization and graph representations, but you get the point [5]. In some sense we're trying to infuse the 256 dimensions that describe the 3D structure into just 2 dimensions. The output 2D points are linked in the GitHub repository.

For the final visualization in Figure 5, I used 106,193 proteins from nearly 200,000 since I could not fit all of the PDB in 16GiB of RAM for the UMAP computation. Instead, I filtered the number of proteins down to be at least 64 residues long and no longer than 1024 which greatly reduced the number of proteins to map into 2D. Since one protein can map to multiple embedding vectors, I ended up visualizing 271,684 points in total in the ProteinScatter 2D map.

4. Results

To visualize the results, I built a website interface with an interactive scatterplot visualization[4]. As shown in Figure 5, there is a main scatterplot where each white point represents a PDB protein. On the left sidebar, you can select a Venom Lab protein to see regions in space with similar PDB proteins (Figure 5 in green).

In this particular example, I wanted to find similar proteins to a protein from the Ganaspis hookeri parasitioid wasp venom [10]. See the top right in Figure 5 for the structure. I used the

[4]A video demo of ProteinScatter: https://github.com/xnought/protein-scatter/blob/main/README.md#protein-scatter

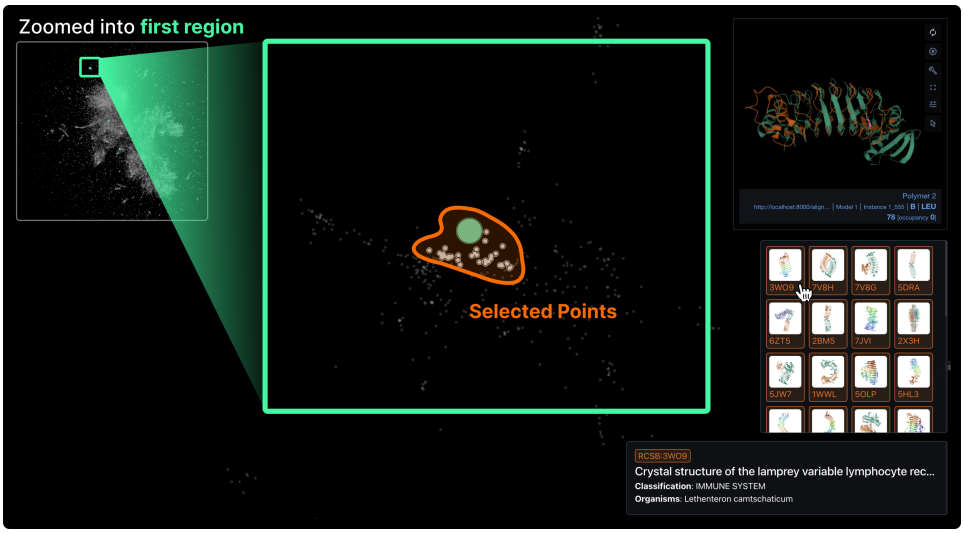

Figure 6: Zooming into the first region, I can select some points and view them on the right sidebar. As you can see many of the neighboring points look almost identical to our protein.

Mol* visualization package [11] to render the protein. To reiterate, the protein structures are from the Venom Lab [2] at OSU where they used AlphaFold [12] to generate predicted structures.

Next, I'll use the highlighted regions to find similar proteins from the PDB with useful information. In the ProteinScatter interface, I can zoom in and select points around those green regions for a closer look.

By checking the top region, as shown in Figure 6, I can see that the neighboring proteins are very similar in shape to our protein. I added the ability to compare proteins on top of each other using US-align [13] to superimpose the PDB protein (orange) on the venom protein (green) in the top right of Figure 6. From my analysis around the highlighted region, I can see that the general shape of the protein matches well with other immune signaling and toxin based proteins from the PDB.

Just like that I can find a starting point for what the function might be for that protein. I can further analyze the linked papers and additional data from the PDB proteins if I want more information.

5. Conclusion

In this paper, I've modeled 3D protein structure to create the 3Di-transformer. Then, I visualized the 3Di-transformer's embeddings in a 2D scatterplot called ProteinScatter. With ProteinScatter you can easily find regions in space with similar proteins from the PDB.

Acknowledgments. I wrote this paper for Oregon State University's BB 499 Molecular Modeling class taught by Dr. Juan Vanegas[5]. Thank you to Juan for the teaching and compute resources.

I was inspired to work with venom proteins because of my prior work with Michael Youkhateh[6] and Dr. Nathan Mortimer[7]. I am thankful they gave me permission to use their proteins in this paper.

[5] https://biochem.oregonstate.edu/directory/juan-vanegas
[6] https://biochem.oregonstate.edu/directory/michael-youkhateh
[7] https://biochem.oregonstate.edu/directory/nathan-mortimer

References

1. H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The Protein Data Bank," Nucleic Acids Res. **28**, 235–242 (2000). https://doi.org/10.1093/nar/28.1.235.
2. Oregon State University Venom Biochemistry and Molecular Biology Laboratory. https://venombiochemistrylab.weebly.com/.
3. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," (2017). https://arxiv.org/pdf/1706.03762.pdf.
4. M. van Kempen, S. S. Kim, C. Tumescheit, M. Mirdita, J. Lee, C. L. M. Gilchrist, J. Söding, and M. Steinegger, "Fast and accurate protein structure search with foldseek," Nat. Biotechnol. **42**, 243–246 (2024). https://doi.org/10.1038/s41587-023-01773-0.
5. L. McInnes, J. Healy, N. Saul, and L. Großberger, "Umap: Uniform manifold approximation and projection," J. Open Source Softw. **3**, 861 (2018). https://doi.org/10.21105/joss.00861.
6. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," (2019).
7. A. Karpathy, "nanoGPT: The simplest, fastest repository for training/finetuning medium-sized GPTs," (2022). https://github.com/karpathy/nanoGPT.
8. PyTorch, "torch.nn.CrossEntropyLoss," https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.
9. I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," (2019).
10. N. Bretz, C. Lark, and N. Mortimer, "Not quite FedEx: How are venom proteins packaged for delivery by the parasitoid wasp Ganaspis hookeri?" J. Biol. Chem. **299**, S804 (2023).
11. D. Sehnal, S. Bittrich, M. Deshpande, R. Svobodová, K. Berka, V. Bazgier, S. Velankar, S. K. Burley, J. Koča, and A. S. Rose, "Mol* Viewer: modern web app for 3D visualization and analysis of large biomolecular structures," Nucleic Acids Res. **49**, W431–W437 (2021). https://doi.org/10.1093/nar/gkab314.
12. J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Highly accurate protein structure prediction with AlphaFold," Nature **596**, 583–589 (2021). https://doi.org/10.1038/s41586-021-03819-2.
13. C. Zhang, M. Shine, A. M. Pyle, and Y. Zhang, "Us-align: universal structure alignments of proteins, nucleic acids, and macromolecular complexes," Nat. methods **19**, 1109–1115 (2022).