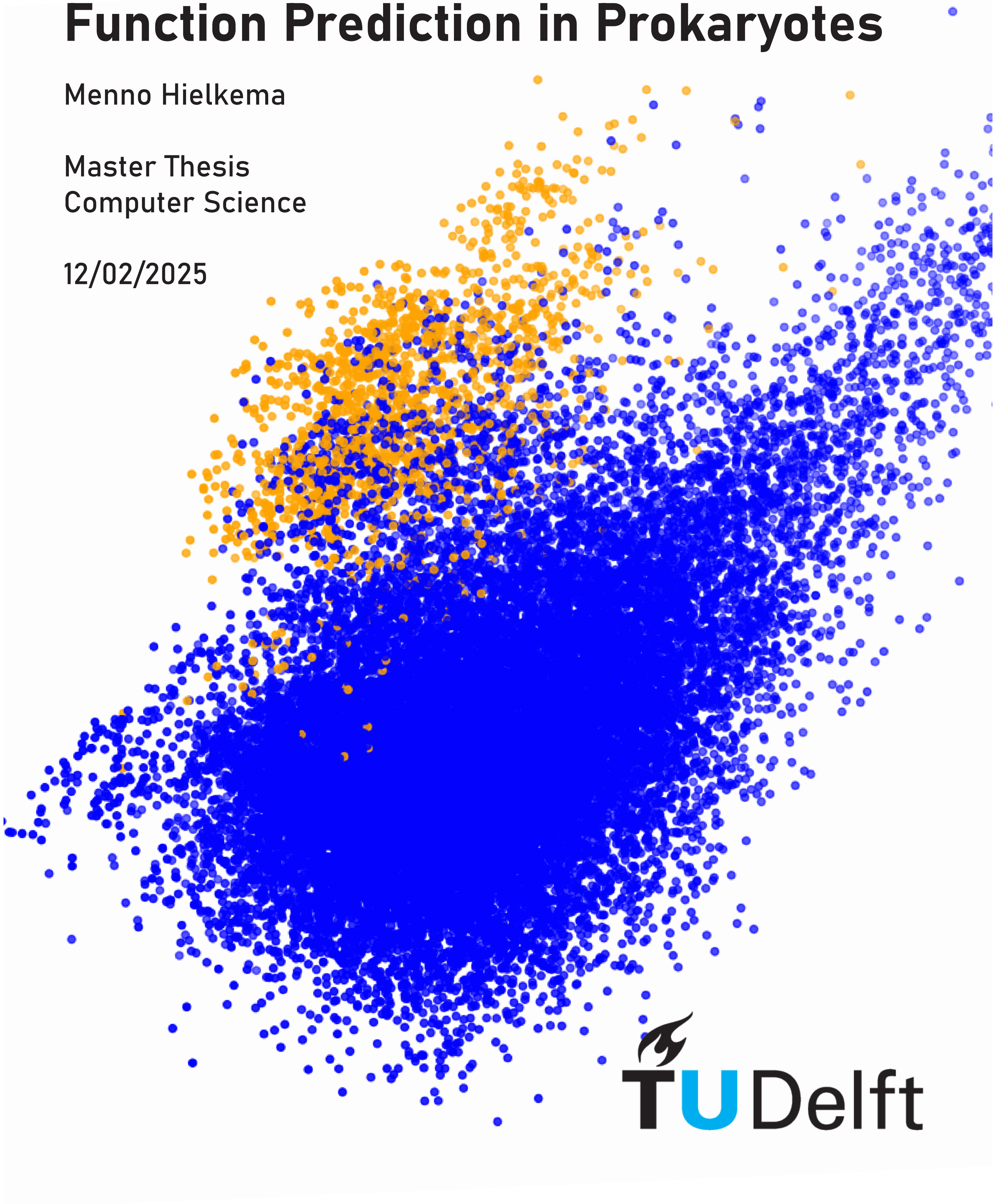
Embedding-Based Multi-Paradigm Protein Function Prediction in Prokaryotes

Menno Hielkema

Master Thesis
Computer Science

12/02/2025



Contents

1 Abstract **5**

2 Introduction **5**

2.1 Automated Functional Annotation 6
2.2 Functional Prediction Strategies 7
2.2.1 Homology inference and Functional Prediction 7
2.2.2 Synteny-based Functional Prediction in Prokaryotes . . . 9
2.2.3 Other Strategies 9
2.3 Gene Ontology 10
2.4 The CAFA Challenge 11
2.5 Established methods 12
2.6 SAFPred 13
2.6.1 CD-HIT and Filtering 14
2.6.2 Train-Test Split 14
2.6.3 Create Syntenic Blocks 15
2.6.4 K-nn training 16
2.6.5 Train SAFPred-Synteny 16
2.6.6 Combine and Infer 17
2.6.7 Evaluation Metrics 17
2.7 Problem Statements 17
2.7.1 Research Question 1 17
2.7.2 Research Question 2 17
2.7.3 Research Question 3 18
2.7.4 Research Question 4 18
2.7.5 Research Question 5 18

3 Materials and Methods **20**

3.1 Datasets 20
3.1.1 SwissProt for Benchmarking Datasets 20
3.1.2 GO Database for Functional Labels 21
3.1.3 NCBI Genomes for Gene Loci 22
3.2 Methods 22
3.2.1 SAFPrednn and goPredSim comparison 22
3.2.2 LLM Embedding Extraction 22
3.2.3 Different amino-acid LLMs 23
3.2.4 Lighter LLMs with Stopping Layers 24
3.2.5 Structural Homology Prediction 24
3.2.6 SBGraph 25
3.2.7 SAFPred-Synteny Reconstruction 26
3.2.8 Multi-Class Complementary Learning Step 26
3.3 Evaluation 29
3.3.1 Maximized F1-Score 29
3.3.2 Estimated F-score Ratio for Robustness Evaluation 29
3.3.3 LLM Stopping Layer Analysis 30

3.3.4 Grid Search Hyperparameter Selection 30
3.3.5 Complementarity Metric 30
3.3.6 Complementary Learning Assessment 31

4 Results **32**
4.1 Reproduction of SAFPred results 32
4.1.1 F_{max} reproducible for BLAST and SAFPred-nn 33
4.1.2 Reproduction issues SAFPred-synteny 33
4.2 Sequence homology inference methods optimally require few matches per query protein 35
4.2.1 BLAST and DIAMOND have a similar performance . . . 37
4.2.2 Restricting matches in DIAMOND makes it almost as powerful as the knn methods 37
4.2.3 goPredSim and SAFPred-nn have similar performance . . 37
4.3 Model application on different SwissProt datasets 38
4.3.1 Models become slightly less accurate with SwissProt-Full instead of SwissProt-SAFPred 38
4.3.2 Pruning non-prokaryote data does not negatively impact model performance 39
4.4 Early stopping layers in embedding models do not negatively impact goPredSim model performance 40
4.4.1 Early stopping layers significantly improve embedding speed and GPU VRAM 42
4.4.2 Using batches in protein embedding can result in a lower performance 43
4.5 New goPredSim-ESM2 does not outperform goPredSim-ESM1b, even with larger models 44
4.6 Proposed structural homology method goPredSim-ESMfoldv1 not viable 45
4.7 Estimated F-score Ratio slightly better for DIAMOND (k=1) . . 47
4.8 Current implementation SBGraph underperforms in comparison to established methods 48
4.9 Combination of methods using TS-Normalization inconclusive . . 49

5 Conclusions **50**
5.1 Research Question 1 50
5.2 Research Question 2 50
5.3 Research Question 3 50
5.4 Research Question 4 51
5.5 Research Question 5 51
5.6 Additional Findings 51
5.6.1 Batch Embeddings 51
5.6.2 Synteny Method Underperformance 51

6 Discussion **52**
6.1 Reproduction Issues SAFPred 52
6.2 Stopping Layers in Other Applications 53
6.3 Taxonomy Level as Supporting Prediction Feature 53

7 Acknowledgments **53**

1 Abstract

In the past decade, protein functional prediction has dramatically shifted towards the usage of large language models (LLMs). In this research, we set out to improve upon the model of SAFPred, a model for prokaryote protein function prediction combining LLM embedding based sequence homology prediction with a synteny aware component. With a new technique referred to as stopping layers, we successfully proved that we can remove layers from pretrained LLMs without sacrificing performance, ultimately giving us the ability to reduce runtime by 70% and required GPU VRAM usage for LLM storage by 72%. Furthermore, we show that we can prune our training dataset by only using prokaryote proteins without any performance impact, reducing the training set by 78%. Additionally, in our evaluated models we show that restricting the amount of training matches per query protein as much as possible is beneficial to model performance.

2 Introduction

In the game of life, proteins are the blocks that both build us, drive us, and in a sense, form us. Having a fundamental understanding of proteins brings us closer to having a fundamental understanding of life itself.

To understand proteins, many researchers with different specialties collaborate to discover the function of hundreds of thousands of proteins. For instance, at release 2024_06 the manually curated protein database Swiss-Prot boasted a figure of 572,619 proteins where, among other relevant features, the function has been manually verified and largely supported by experimental evidence, referring to 300,929 distinct papers. [1]

In comparison, the UniProt protein database, of which Swiss-Prot is a subset, contains 254,254,987 entries of distinct proteins in the same release. That means only 0,2% of all known proteins are manually reviewed. [1] While we have gone a long way to discover and curate the function of proteins, there exist many proteins which are not experimentally verified.

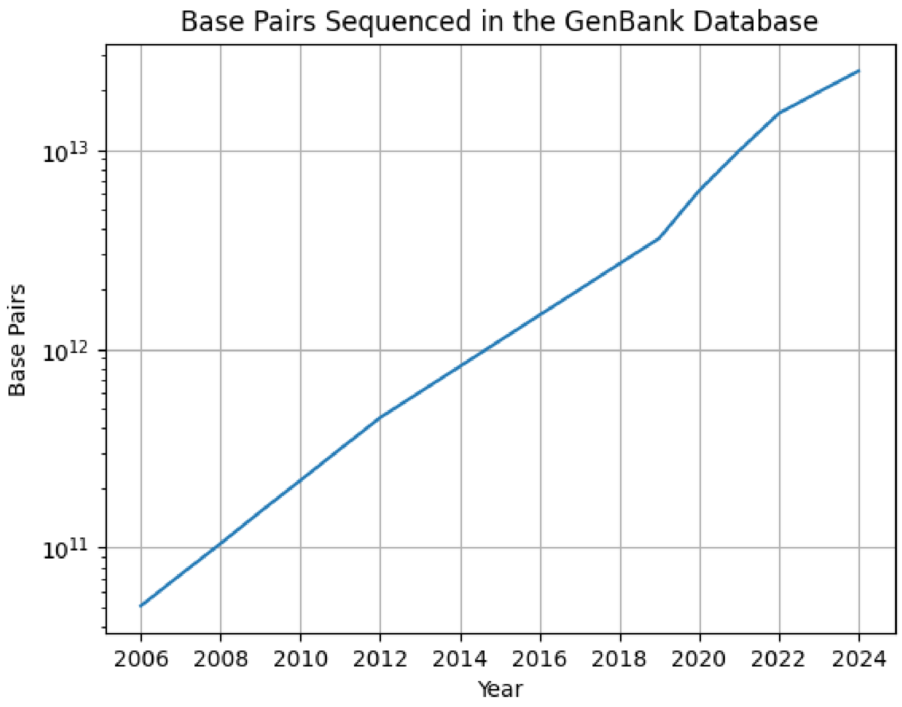


Figure 1: Graph illustrating how many basepairs have been sequenced in the GenBank Database over the years since 2006. The years 2007-2011 and 2013-2018 have been interpolated, the rest are from GenBank publications. [2]

GenBank is a database that stores all publicly available DNA basepair sequences. Figure 1 indicates that the amount of sequenced DNA basepairs is exponentially increasing. The majority of DNA sequences can eventually be transcribed and translated to a protein. Given the establishment of Swiss-Prot in 1986 [3] one can deduce that it will be practically impossible to manually verify both the stored proteins in UniProt and novel proteins that will be discovered in the future.

2.1 Automated Functional Annotation

After learning the defining features of proteins, a comparison using these features can indicate that some proteins are similar enough to predict they share the same function. Thus, if the function of one protein were to be verified, feature similarity might indicate that another protein has the same function.

With the sheer amount of unverified proteins, it is often faster and more practical to use automated methods to detect if unverified proteins have enough shared features with verified proteins to predict that they share certain functionality as well. The process of predicting protein function is referred to as *Automated Functional Annotation.*

2.2 Functional Prediction Strategies

2.2.1 Homology inference and Functional Prediction

A popular and effective way to predict protein function is to infer if proteins are *homologous.* We speak of protein homology if two proteins share a common ancestry. Thus, if two proteins are sufficiently similar, we infer that it is likely that they also share a close common ancestry. [4]

Inferring that protein function has not changed since this common ancestry is not always correct. It is very possible that the protein function between two statistically similar proteins from different species has diverged as their biological 'environment' has also changed. Therefore, methods that focus on homology focus on inferred *orthology.* Orthology is a more constrained version of homology, where two proteins have originated from a common ancestor, only differing in definition due to speciation events. Orthologous proteins generally retain the same function as well. [4, 5]

Homology inference from amino-acid sequence similarity

The protein amino-acid sequence can be represented as a linear 1D string of symbols, where the amount of distinct symbols is the amount of distinct amino-acids. Thus, to evaluate if two proteins are homologous, we could compare these strings of symbols for (dis)similarity.

The low complexity of this feature, the increasing ease of obtaining amino-acid sequences, and the fact that any protein can essentially be reduced to this single 1D sequence, makes it ideal for homology inference. It is therefore no surprise why sequence similarity is broadly used for homology inference.

Homology inference from structural similarity

Generally, protein 3D structure is the feature which confers function. It is the amino-acid sequence which confers 3D structure (assuming the right environmental conditions are met). Therefore, it is often thought that predicting function using 3D structure is more accurate. [6]

However, achieving an accurate representation of protein 3D structure has proven to be a difficult job. While there are numerous significant breakthroughs since 1969, it was only 2021 where the deep-learning paradigm showed its force into the world of protein structural prediction with *AlphaFold.* [6]

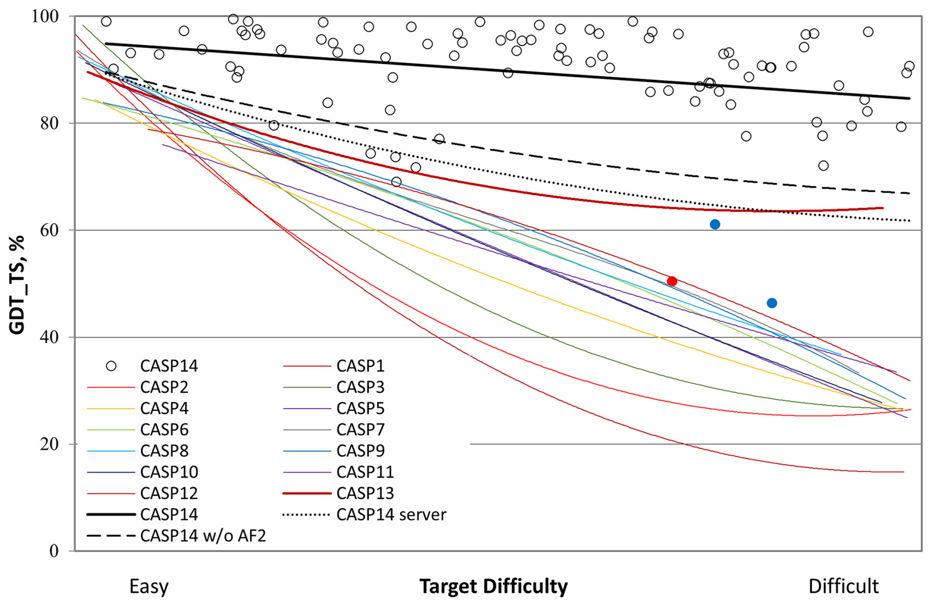


Figure 2: Graph from the original CASP14 paper indicating that CASP14, where AlphaFold was first introduced, boasted a significant outperformance compared to the previous years. On the x-axis we have the GDT_TS, or the Total Score of the Global Distance Test, which the paper describes as the agreement metric compared to experimentally verified structural information. On the x-axis the difficulty of the target protein is indicated. [7]

In CASP14, a biannual assessment of structural protein prediction methods, *AlphaFold* made an unprecedented breakthrough. In Figure 2 we can see how fast this paradigm shift has improved protein structure prediction. [7]

As we continue to improve our prediction of protein structure, more accurate and reliable structure-based homology inference will become available to us. In turn, more reliable structure-based function prediction might be possible.

2.2.2 Synteny-based Functional Prediction in Prokaryotes

Genes are in synteny with each other if their gene order and location is conserved in evolution. [8] It is however not instantly obvious why we can use synteny as a feature for functional prediction. However, in prokaryotes, synteny is intrinsic to **operons**.

Operons and Synteny

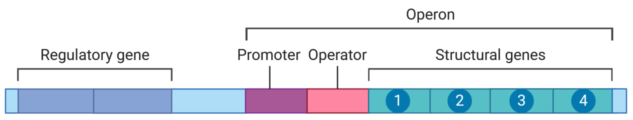


Figure 3: Illustration of an operon. The operon is commonly found on a prokaryotic genome. In this illustration, we see the abstraction of a subsequence of the genome that is related to the operon. Outside of the operon itself we can find regulatory genes. The structural genes can eventually translate into proteins. The Promotor and Operator are involved in the process of making proteins from structural genes. [9]

In prokaryotes, clusters of co-regulated genes are quite common. These clusters are called operons, as we can also see in Figure 3. Genes on an operon often have a related function. [10] One could deduce that genes that are in synteny are likely on the same operon. Thus, they are likely co-regulated and share common functions.

Hence, if one can see that genes encoding for unknown proteins are in synteny with genes encoding for proteins with known function, one could deduce that the unknown genes are on the same operon as the known genes, and thus share functionality. In fact, it has been shown that synteny with the right constraints is an effective predictor for operon presence. [11]

2.2.3 Other Strategies

The aforementioned strategies will be the main focus of this research. However, these strategies do not cover all usable strategies for functional annotation for proteins. For example, proteins could be identified as part of a broader protein family, which also tends to confer shared functionality. In turn, this could give predictive capabilities. [12] Other effective methods might look at protein-protein interactions to predict protein function. [13]

2.3 Gene Ontology

It can be difficult to classify the function of a gene. 'Involved in metabolism' can be a little vague. 'Mutant gene that promotes rapid proliferation of kidney stem cells in mice' might be too specific of a function.

This problem has been identified quite some time ago, and a common language for describing gene functions was made by the Gene Ontology (GO) Consortium. GO defines functional descriptions for three different categories.

- The category **biological process** refers to a biological objective to which the gene or gene product contributes. This could both be quite broad, and quite specific. A protein can be involved in 'cell growth and maintenance' (broad) or 'cAMP biosynthesis' (narrow).
- The category **molecular function** describes the biochemical activity of a gene product. A broad term could be, for example, 'enzyme'. A narrow term could then be 'adenylate cyclase'.
- The category **cellular component** describes the place in the cell where the gene product is active. [14]

GO terms make functional annotation problems into a multi-class classification problem. However, in contrast to a standard multi-class classification problem, the GO annotation labels are hierarchical. To illustrate what this means, we will refer to Figure 4. [15]

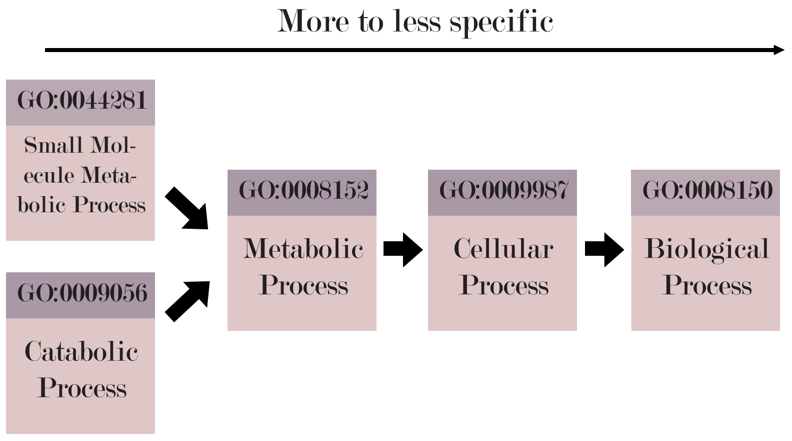


Figure 4: Example hierarchy of GO labels with ancestors, along with a description. In the boxes, the GO label is visible on top, with a description in the bottom part. The arrows point from a GO label to hierarchical parents. [15]

As we can see in Figure 4, one GO label is part of a lineage of several GO labels. What this hierarchy essentially means is if a protein is involved in the catabolic process (GO:0009056), then it is also involved in a metabolic process (GO:0008152) along with all the other GO labels in the given hierarchy. In this example, the directed arrows indicate parent relations of GO annotations.

2.4 The CAFA Challenge

A standard way of classifying genes makes it much simpler to evaluate functional prediction models. To further help standardize models and to stimulate the invention of functional annotation methods, the Critical Assessment of protein Function Annotation algorithms (CAFA) challenge was created. CAFA provides a dataset of genes requiring functional annotation. Models evaluated to the CAFA dataset need to provide the most accurate GO annotations possible [16].

A critical advantage of this challenge is that it offers a solid benchmark for every functional annotation model to pass. The best models can easily be identified, as they demonstrate the best accuracy metrics.

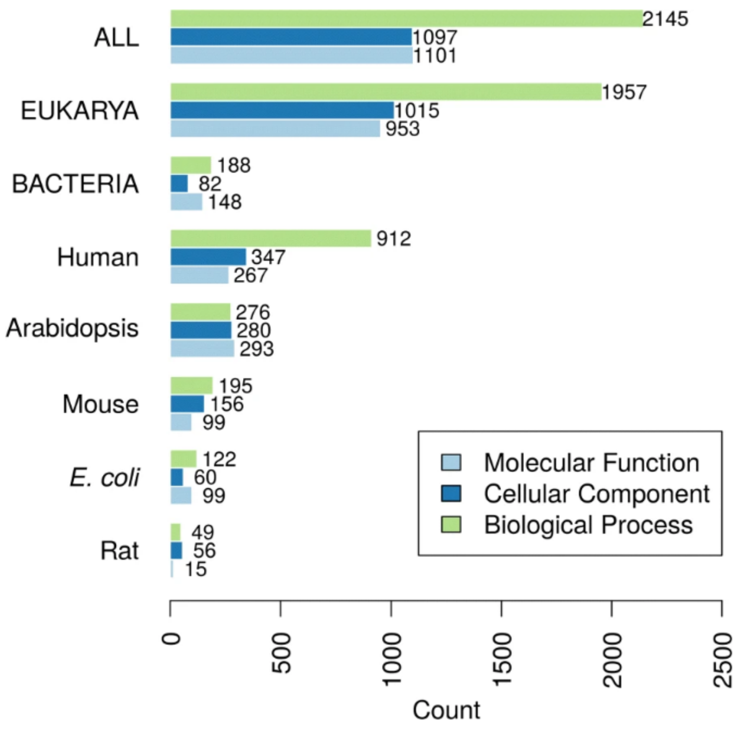


Figure 5: Graph from the original CAFA3 paper showing the amount of proteins for every benchmark species, split by focus ontology category. [16]

However, as we can see in Figure 5, bacterial proteins only make up a small portion of the CAFA dataset. Besides that, about 2/3 of the bacterial proteins in CAFA3 are from one single species, E. coli. While CAFA might work great for benchmarking method accuracy on eukaryotes, it is not quite as applicable for prokaryotes.

2.5 Established methods

Name	Year	Strategy	Description
BLAST [17]	2003	Sequence Homology	BLAST can be used to infer homology by finding sequences that have a high enough sequence identity to the query sequence.
DIAMOND [18]	2015	Sequence Homology	Very similar to BLAST, but faster.
HMMer/PFam [19]	2011	Protein Family Classification	Uses HMMer to identify profiles commonly seen in protein families saved in PFam. Broader than homology inference.
goPredSim [20]	2021	Sequence Homology	Extracts embeddings from protein LLMs and finds sequences with a high enough embedding similarity to infer homology.
DeepGraphGO [13]	2021	Protein Interactions	Puts proteins with known interactions in a graph to train a GNN to infer protein function.
GNNGO3D [21]	2023	Structural Homology	Matches similar experimental 3D protein structures to infer homology.
PFresGO [22]	2023	Sequence & Subsequence Homology	Specifically links function to subsequences in proteins.
SAFPred [11]	2024	Operon Inference	Combines a goPredSim-esque component with a component that infers operons.

Table 1: Functional annotation methods of interest with their respective strategy and description.

In Table 1 we show several examples of functional annotation methods of interest. In this research, we will specifically focus on SAFPred.

2.6 SAFPred

SAFPred (Synteny-Aware gene Function Prediction) is a functional annotation method specifically designed for prokaryotes. SAFPred leverages synteny to predict protein function.

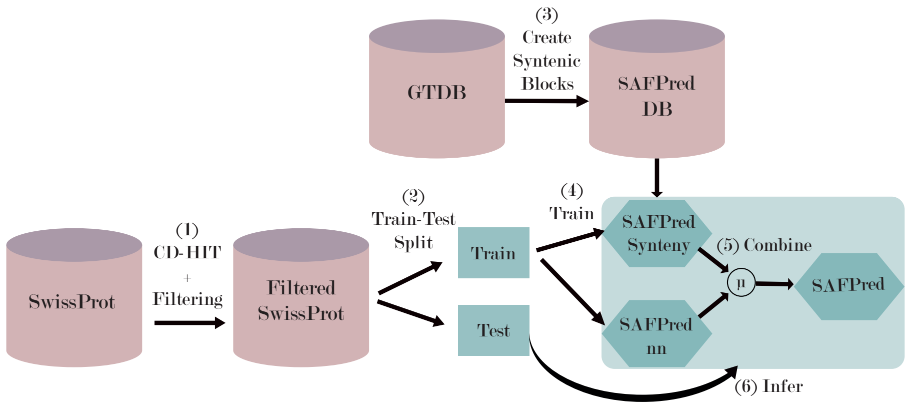


Figure 6: Graphical abstract of training and inference of SAFPred. In step 1, we take SwissProt and combine redundant sequences using CD-HIT. [23] In step 2, we split the data into a train and a test split. In step 3, we create SAFPredDB from representative genomes in GTDB. In step 4, we train the knn method (SAFPrednn) and SAFPred-Synteny. In step 5, we combine the results from both methods into one by taking the mean of predictions. In step 6, we infer on our test set, returning predictions.

The complete training and inference pipeline used for evaluation is illustrated in Figure 6. In the following sections we will break down every step of the process.

2.6.1 CD-HIT and Filtering

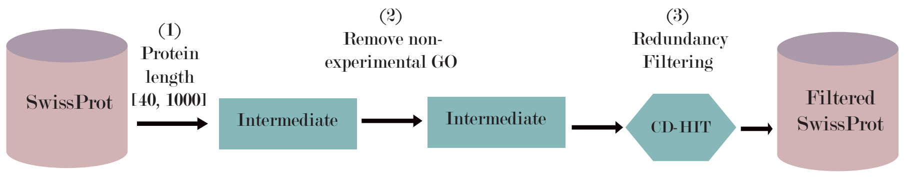


Figure 7: The SAFPred pre-processing pipeline including CD-HIT and filtering. In step 1, proteins between 40 and 1000 are kept, removing the rest. In step 2, we remove all GO labels that do not have experimental proof, also discarding proteins that do not have labels anymore after this step. In step 3, we apply redundancy filtering using CD-HIT.

In this pre-processing step also summarized in Figure 7, first all proteins shorter than 40 amino-acids and all proteins longer than 1000 amino-acids are filtered out of the dataset. Why this filtering step is included is not specifically mentioned, however, it is known that ESM-1b, the amino-acid LLM used in both SAFPred-Synteny and the knn method, only supports proteins shorter than 1024 amino-acids. [24]

Next, only GO annotations with enough manually verified evidence are selected. GO annotations with evidence codes EXP, IDA, IPI, IMP, IGI, IEP, HTP, HDA, HMP, HGI, HEP, IBA, IBD, IKR, IRD, IC, and TAS were selected to be sufficiently supported by evidence. These evidence codes are also provided by the GO resource. [15]

Finally, proteins with 95% similarity according to the CD-HIT redundancy clustering algorithm are grouped and a representative sequence is selected for the cluster. The other sequences are removed as they are deemed redundant. [23]

2.6.2 Train-Test Split

For evaluation, we split the dataset into a training and test set. In the SAFPred paper, one organism is taken out of the original set as the test set.

Furthermore, the training set is compared to the test set using BLAST, which returns sequence identity values between 0 and 100. Looking at the best matches between the preliminary training and test set, we filter out all training sequences exceeding maximum pairwise identity (PIDE) threshold x. The lower x is, the harder it is to find training homologs to test samples. This is done to test model robustness to missing data.

For species we consider Escherichia coli, Mycobacterium tuberculosis, Bacillus subtilis, Pseudomonas aeruginosa and Salmonella typhimurium. For the maximum PIDE we consider thresholds of 40, 50, 60, 70, 80, and 95. Combining both splitting features we are able to make 30 distinct training-test splits.

2.6.3 Create Syntenic Blocks

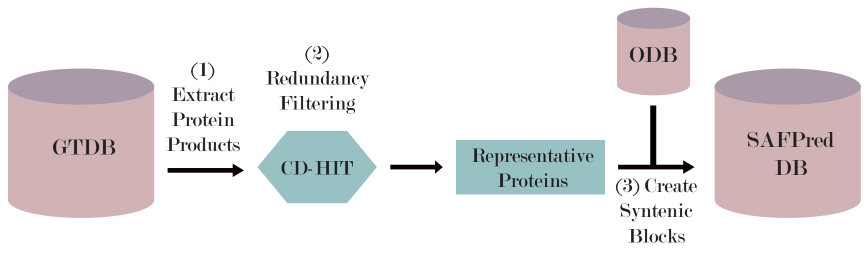


Figure 8: Illustration of the creation of syntenic blocks in SAFPredDB. In step 1, we extract all protein products for protein-coding genes in the Gene Taxonomy DataBase (GTDB). In step 2, we apply redundancy filtering using CD-HIT. In step 3, we create syntenic blocks from these representative proteins using gene loci. SAFPred uses the Operon DataBase (ODB) as a reference to make the syntenic blocks as close to a potential operon as possible.

In step 3 illustrated in Figure 8, all 45,555 representative genomes from the GTDB [25] are used to extract 372,308 distinct representative protein sequences produced by these genomes, again filtered for redundancy using CD-HIT [23], keeping only CD-HIT clusters of at least 10 genes. We also note their location on the genome, and on which DNA strand the gene is found. This process is also mentioned in step 1 and 2 of Figure 8.

In step 3 of Figure 8, we group representative sequences together if at least one CD-HIT cluster member is within 2000 basepairs of another on the same strand and contig. This group is called a syntenic block.

Finally, if the intergenic distance is larger than 300 basepairs, we split up the syntenic block if possible, or remove the block if splitting is not possible.

The proteins in the syntenic blocks are then matched to the non-redundant SwissProt dataset using BLAST to transfer GO annotations to significant hits.

In the paper the Operon DataBase (ODB) was used as a reference to establish the hyperparameters for intergenic distance and maximum syntenic block basepair length. [26] However, since the ODB is manually verified, it only contains 8235 unique operons, while SAFPredDB saves 406,293 distinct syntenic blocks, which we infer to be operons.

2.6.4 K-nn training

For homology inference in SAFPred, SAFPrednn was created. It is however very similar to goPredSim. [20]

SAFPrednn uses a protein LLM to extract protein embedding vectors from the final hidden layer. This protein LLM generates an embedding vector per protein amino-acid. To get a single protein embedding vector, we take the mean over all amino-acid embeddings.

These embedding vectors represent the protein sequence themselves, and using the *cosine similarity* metric in Equation 1, we can compare how alike two embedding vectors are. [11]

$$sim(\mathbf{v_1}, \mathbf{v_2}) = \frac{\boldsymbol{v_1} \cdot \boldsymbol{v_2}}{||\boldsymbol{v_1}|| \cdot ||\boldsymbol{v_2}||} \tag{1}$$

SAFPrednn and goPredSim calculate the cosine similarities between all sequences in the training set and the test set. For every test sequence SAFPrednn takes the values in the top x percentile range of all possible matches, where x is pre-defined. goPredSim takes the K highest matches, where K is pre-defined. The GO annotations of the training set will then be conferred to the test sequences with the cosine similarity as a prediction metric, taking the maximum value for every GO as prediction. [11, 20]

In the SAFPred paper it is mentioned that both ESM1b and T5 were attempted for LLM. Since ESM-1b showed better results in preliminary experiments, it became the LLM of choice. [24]

Why SAFPred adapted goPredSim instead of using goPredSim itself is not mentioned.

2.6.5 Train SAFPred-Synteny

To obtain meaningful functional predictions from the previously created SAFPredDB, the component SAFPred-Synteny is constructed. SAFPred-Synteny works by calculating the embedding vector for every sequence in SAFPredDB. Then, syntenic blocks are summarized by taking the average of all embedding vectors for sequences in these blocks. [11]

To query the best matching syntenic block for test sequences, we calculate the cosine similarity defined in Equation 1 between test sequence embeddings and average syntenic block embeddings. The syntenic blocks in the highest x percentile are the most likely syntenic blocks for the query protein. We multiply the frequency of the GO terms in the syntenic block by the cosine similarity of the test sequence to the syntenic block embedding to calculate the final SAFPred-Synteny prediction. [11]

2.6.6 Combine and Infer

Combining the results of SAFPrednn and SAFPred-Synteny is done by simply adding the results together and dividing the totals by two. We can then infer any dataset by using the training set as a reference in SAFPrednn and SAFPred-Synteny. [11]

2.6.7 Evaluation Metrics

For evaluation on the SwissProt dataset, SAFPred uses the same evaluation methods as CAFA. For metrics, we look at the maximized F1-score (F_{max}) and the minimum semantic distance (S_{min}) described in Radivojac et al. 2013. [27] Additionally, SAFPred looks at the coverage, or the percentage of test proteins annotated with at least one GO term at the threshold which maximizes the F1-score. [11]

2.7 Problem Statements

SAFPred gave us a starting point for a synteny-based prediction method. However, there are several possible avenues of improvement to explore in SAFPred.

2.7.1 Research Question 1

At the core, we would first like to see refine the current model without changing too much fundamentally. This would for example mean looking for better hyperparameters and changing the underlying LLM in safprednn. Therefore, our first research question is formulated as: **How can we refine SAFPred without fundamentally changing SAFPred itself?**

It is hard to really give a hypothesis for this research question. We do expect there to be room for improvement beyond fundamentally changing the model. However, good avenues for refinement would be updating older submodels to newer superior ones, or giving a more comprehensive grid search for hyperparameter selection.

2.7.2 Research Question 2

SAFPred is specifically written for prokaryotes as it leverages the operon structure seen in prokaryotes. However, training the model has been done using the full SwissProt dataset.

It would be interesting to see how prediction would hold up if we prune the SwissProt dataset so that it only includes prokaryote proteins. If this proves effective, we can save a good amount of resources and time for training. We formulate our second research question as: **Can we effectively restrict our prediction to only use prokaryote-sourced data?**

We do not expect the pruned dataset to confer better predictions. However, we expect that the predictions using the pruned dataset are about as accurate as predictions using the full dataset.

2.7.3 Research Question 3

SAFPred makes predictions based on both synteny and sequence homology. However, we have seen in GNNGO3D that structural information can be just as effective. [21] The downside of GNNGO3D is that it requires experimental structural data, which is quite restrictive.

We would like to provide a less restrictive way to predict functionality on structural homology. We formulate our third research question as: **Can we add a functional prediction method utilizing structural homology to improve predictions?**

2.7.4 Research Question 4

SAFPrednn and SAFPred-Synteny are combined by taking the mean of the two predictions. This method of combination is quite simple, but there is no evaluation of the effectiveness of this method. We believe there might be room for improvement.

Therefore, we would like to improve the method by adding a more advanced method of combining predictions. Additionally, we want to properly evaluate if there is any value found in combining two predictions. We formulate our fourth research question as: **Can we create a better method for combining two functional predictions?**

2.7.5 Research Question 5

With the F_{max} from CAFA, we can make quite a good estimate of the predictive potential of a model. However, the F_{max} might not be entirely realistic metric when we would evaluate our model performance on unseen data without a known ground truth.

CAFA does include an evaluation on unevaluated proteins at the start of the challenge. By the end of the challenge, these unevaluated proteins are experimentally checked as well, and the final performance on these unevaluated proteins is checked so we can see how models perform on real unseen data with a pre-set threshold. Unfortunately, this workflow is not available to us, as experimentally evaluating proteins is not part of our research scope.

Therefore, we would like to conceptualize a performance metric that can estimate our model performance on test data, given that the test labels are unknown. We formulate our fifth research question as: **Can we define a metric for evaluating model accuracy on unknown data?**

3 Materials and Methods

3.1 Datasets

3.1.1 SwissProt for Benchmarking Datasets

For benchmarking the models, we use proteins taken from the SwissProt database. [3] We distinguish between three different benchmarking datasets: the 2021_04 release of the dataset (as used on SAFPred) (SwissProt-SAFPred), the new (2024_05) release of the dataset (SwissProt-Full), and the prokaryote SwissProt dataset (SwissProt-Prokaryote), where only prokaryote proteins are included.

For preprocessing and making the train-test splits, we mostly keep the same steps described in Section 2.6. For making the train-test splits, we used DIAMOND instead of BLAST due to its' superior speed.

Organism	SwissProt-SAFPred	SwissProt-Full	SwissProt-Prokaryote
Escherichia Coli	3454	3493	3493
Mycobacterium Tuberculosis	1666	1736	1736
Bacillus Subtilis	1636	1631	1631
Pseudomonas Aeruginosa	1014	1057	1057
Salmonella Typhimurium	774	827	827

Table 2: Amount of proteins in all 5 test sets for every benchmarking dataset. In the first column, the source organism for the test set is denoted. In the next three columns, the benchmarking dataset is denoted.

Organism	40	50	60	70	80	95
Escherichia Coli	92119	97378	100503	102088	103019	104161
Mycobacterium Tuberculosis	98934	103236	105174	105821	105996	106033
Bacillus Subtilis	96624	101679	104341	105516	105903	106068
Pseudomonas Aeruginosa	97172	101683	104588	106069	106563	106687
Salmonella Typhimurium	102299	104256	105296	105858	106212	106916

Table 3: Amount of proteins in the training sets for SwissProt-SAFPred. In the first column, the organism that has been split out of the training set has been denoted. In the other columns, the number of proteins per header-indicated PIDE split is denoted.

Organism	40	50	60	70	80	95
Escherichia Coli	90193	95320	98324	99884	100801	101926
Mycobacterium Tuberculosis	96456	100826	102829	103479	103647	103689
Bacillus Subtilis	94251	99448	102093	103251	103634	103795
Pseudomonas Aeruginosa	94893	99340	102254	103739	104244	104368
Salmonella Typhimurium	99734	101797	105296	103497	103864	104594

Table 4: Amount of proteins in the training sets for SwissProt-Full. In the first column, the organism that has been split out of the training set has been denoted. In the other columns, the number of proteins per header-indicated PIDE split is denoted.

Organism	40	50	60	70	80	95
Escherichia Coli	12002	15195	17515	18911	19810	20936
Mycobacterium Tuberculosis	17447	20453	21938	22495	22657	22699
Bacillus Subtilis	15712	19136	21219	22262	22644	22805
Pseudomonas Aeruginosa	15922	19027	21364	22754	23254	23378
Salmonella Typhimurium	19482	21009	21935	22507	22874	23604

Table 5: Amount of proteins in the training sets for SwissProt-Prokaryote. In the first column, the organism that has been split out of the training set has been denoted. In the other columns, the number of proteins per header-indicated PIDE split is denoted.

Discrepancy Training-Test Splits SAFPred Paper

For the training-test splits in SwissProt-SAFPred the test splits for Escherichia coli and Bacillus subtilis are slightly polluted. For the test split Escherichia coli, 11/3454 proteins actually belong to Mycobacterium tuberculosis, and 1/3454 of proteins actually belong to Yersinia Pestis. In Bacillus subtilis, 3/1636 proteins actually belong to Bacillus cereus.

3.1.2 GO Database for Functional Labels

For functional labels, since the original SAFPred paper never mentioned a GO Database release, we re-used the GO hierarchy obo file provided in the SAFPred repository to match with the 2021_04 release of the SwissProt dataset. For the newer two SwissProt releases, we used GO release 2024-11-03. [28]

Like in the SAFPred paper, we propagate any GO prediction and SwissProt ground truth value to every GO hierarchical parent. Additionally, we only benchmark on every single GO category (Cellular Component, Biological Process, Molecular Function) seperately.

3.1.3 NCBI Genomes for Gene Loci

To obtain gene loci for synteny block creation, we used genomic data from the NCBI datasets. [29] (Accession date: 20/12/2024) From the NCBI datasets we downloaded 20,476 representative genomes for bacteria and 734 representative genomes for archaea.

We chose NCBI datasets instead of GTDB as the NCBI datasets are significantly smaller in size. In SAFPred, GTDB was used, however, the pipeline to infer gene loci using Prokka [30] was unmentioned in the SAFPred paper and not documented anywhere else, making SAFPred-Synteny difficult to retrain properly.

In total, our NCBI genome dataset contains 80.2 million distinct protein-coding genes. Using linclust from the MMSEQS2 package, we clustered all proteins with 95% identity for the smallest sequence together to reduce redundancy. For this step we use linclust instead of CD-HIT as linclust clusters in linear time, making it significantly faster than CD-HIT. It should be noted that linclust is somewhat less sensitive than CD-HIT. However, from 70% sequence identity and up, the sensitivity difference becomes increasingly small. [31]

Since the NCBI dataset is significantly smaller than the GTDB dataset, we set our cluster size threshold lower than 10. Setting our threshold at 10 would net us 170,021 clusters of proteins instead of the 372,308 bacterial representative sequences included in SAFPred. However, setting the threshold at 7 would net us 342,942 bacterial and archaeal representative sequences.

3.2 Methods

3.2.1 SAFPrednn and goPredSim comparison

In the SAFPred paper, the knn method SAFPrednn that is used and evaluated in conjunction with SAFPred-Synteny is mentioned to be designed in a similar manner to goPredSim. However, there is no reason given why to deviate from the K-nn method of goPredSim and to use the percentile method described in SAFPred.

To evaluate if either method has a clear advantage, we provide a comprehensive comparison using relevant evaluation methods outlined in Section 3.3.

3.2.2 LLM Embedding Extraction

In the SAFPred paper, the *bio_embeddings* package had been used to extract embedding vectors from protein sequences. [32] However, we found that this package had not been maintained since 2022, and we experienced quite some compatibility issues.

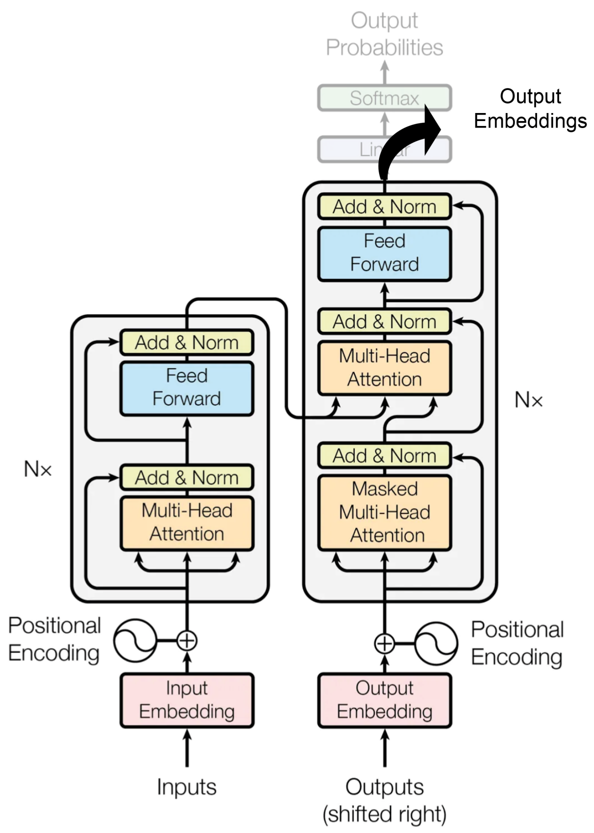


Figure 9: Adapted Illustration from Vaswani et al. 2017, elucidating where we extract embeddings from a protein LLM. [33]

Hence, we made our own module to extract embedding vectors from protein sequences. In Figure 9 we see at which step of the LLM mechanism we generally extract our output embedding vectors.

3.2.3 Different amino-acid LLMs

The underlying LLM that was chosen for the nearest-neighbor methods in SAF-Pred, ESM1b, has been released quite some time ago. [24] A newer version of ESM, ESM2, has been released as well. [34] ESM2 scores relatively high on the Diverse Genomic Embedding Benchmark (DGEB), an amino-acid embedding model benchmark. [35]

We evaluate the performance of ESM2 as an underlying embedding model of goPredSim using metrics outlined in Section 3.3.

3.2.4 Lighter LLMs with Stopping Layers

It is quite common to use the final hidden layer to extract embedding vectors from sequences in LLMs, as is done in goPredSim and SAFPred [11, 20]. However, since every layer of an LLM is trained in unison using back-propagation, earlier hidden layers should have the potential to encode defining features of sequences. Since using earlier hidden layers for embedding vector inference would skip out on multiple layers of calculation, the model becomes significantly lighter.

Note N in Figure 9, indicating the amount of transformer heads that are in succession before reaching the final output. A stopping layer in relation to the transformer architecture would simply be taking a value lower than N as our output transformer head, ignoring the rest of the successive transformer heads.

We will evaluate the most interesting hidden layers to analyze in Section 3.3.3 and we will provide a performance comparison of goPredSim using earlier hidden layers of ESM1b using the methods outlined in Section 3.3.

Additionally, we will evaluate the potential improvement of ESM1b using stopping layers with regards to runtime and storage VRAM usage on the GPU, by plotting the stopping layer in regards to the runtime and storage VRAM on the GPU, and taking a ridge regression to indicate a linear relationship on both plots.

3.2.5 Structural Homology Prediction

GNNGO3D revealed the potential of structural data for homology inference. However, a limitation of GNNGO3D is that it is based on experimentally verified protein structures. [21] These are non-trivial to construct, and as such, not every protein will have a verified protein structure available to us.

Since sequence data is more readily available, we extend structural homology prediction to sequence analysis with ESMfold [34]. Using ESMfold, we extract structural embedding vectors from sequence data. These embedding vectors can then be used in goPredSim.

We evaluate the potential of using ESMfold embedding vectors in goPredSim using the methods outlined in Section 3.3.

3.2.6 SBGraph

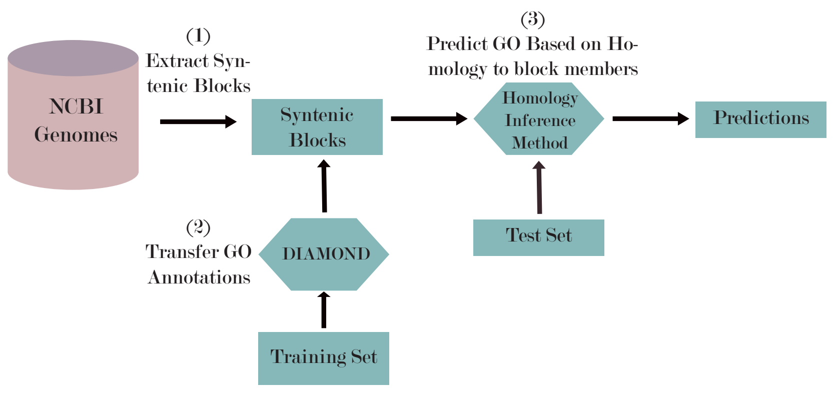


Figure 10: Visual representation of SBGraph. In step 1, we extract syntenic blocks by grouping all genes together that are on the same contig, the same strand, and where the distance between the end of the first gene and the beginning of the second gene (the intergenic distance) does not exceed a set threshold. In step 2, we transfer GO annotations to syntenic blocks by using DIAMOND to find training set homologs in syntenic blocks. In step 3, we predict GO terms by predicting homology to block members.

In SAFPredDB, every syntenic block is identified by the average embedding of all included genes. In SAFPred-Synteny, the best suiting syntenic blocks are then identified by taking the cosine similarity of the test protein to the average embeddings of all syntenic blocks in SAFPredDB.

There is however no reason that indicates the genes in syntenic blocks are similar sequences. Hence, the average embedding could deviate significantly from the embeddings of the members.

Additionally, we introduce SBGraph, a method similar to SAFPred-Synteny. However, instead of taking the similarity of test protein sequences to average embeddings, we take the similarity of test protein sequences to sequences found in the NCBI genome dataset.

Once we infer homology to protein sequences in the NCBI genome dataset, we conclude that this protein sequence is found in a constructed syntenic block that the homolog sequence is a member of. We can then transfer GO terms based on the frequency in the syntenic block.

3.2.7 SAFPred-Synteny Reconstruction

To evaluate the extent exchanging GTDB with the NCBI Genomes dataset, exchanging BLAST with DIAMOND, and exchanging CD-HIT with MMSEQS2-Linclust, we also evaluate a reconstruction of SAFPred-Synteny, aided with SAFPredDB created with these exchanges.

3.2.8 Multi-Class Complementary Learning Step

In SAFPred, the predictions for SAFPrednn and SAFPred-Synteny are combined by simply taking the mean of the normalized predictions for both methods. While this method of combination is simple, there are several potential downsides.

We first define predictions p_1 and p_2 and combined prediction $p_{1,2}$. The integer denotes the underlying prediction method. We establish that the predictions are for the same GO class and for the same test protein.

If we define $p_{1,2} = \frac{p_1+p_2}{2}$ as in SAFPred there are three logical things that can happen:

- Both p_1 and p_2 are non-zero, and $p_{1,2} = \frac{p_1+p_2}{2}$.
- Either p_1 or p_2 is non-zero, $p_{1,2} = \frac{p_1}{2}$ if $p_2 = 0$ or $p_{1,2} = \frac{p_2}{2}$ if $p_1 = 0$
- Both p_1 and p_2 is zero so $p_c = 0$

We also define P_1 and P_2, the set of all GO predictions for every protein given by method 1 and 2 respectively. We establish that predictions are very sparse, as only a fraction of all GO annotations describes one protein at the same time.

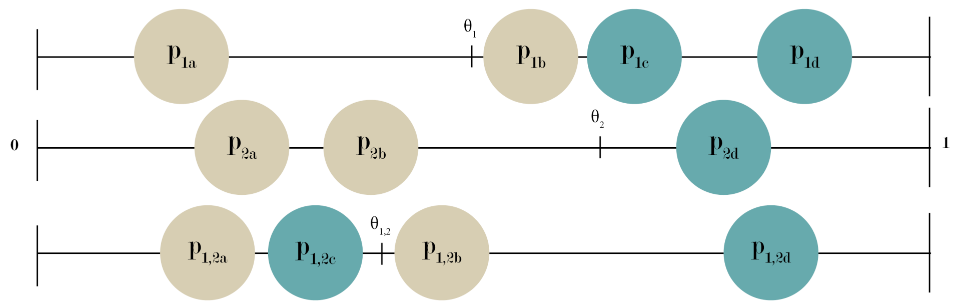


Figure 11: Example to illustrate what can go wrong with taking the mean of two predictions to combine them. There are three example illustrated 1D predictions, all predicting the same GO term. The predictions are linearly rescaled to be between 0 and 1. On the top, hypothetical method 1 is used for prediction. In the middle, hypothetical method 2 is used for prediction. On the bottom, the mean is taken. The integers in the subscript of the predictions indicate the method. The letters in the subscript indicate the protein. The threshold for every method, in this example established in conjunction with other unseen GO predictions, is denoted with θ_n. Predictions where the ground truth is negative are indicated in light brown. The predictions where the ground truth is positive are indicated in blue.

To illustrate what can go wrong with this, we constructed a small example in Figure 11. In this example, the combined method does not perform better than the individual methods for this particular GO. However, if we introduce a different way to normalize the values, we are able to improve the combined method significantly.

The sigmoid function has the following definition:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

The following features of the sigmoid function can be leveraged for our normalisation function:

- The output is between 0 and 1
- The output is centered at $\frac{1}{2}$ (x=0)

To coerce the same shape for every prediction, we require the following features:

- The output is between 0 and 1.
- The output is $\frac{1}{2}$ when the input is equal to our threshold (x=θ) that maximizes the F1 score.
- The output is 1 at our maximum possible prediction.

The following function using the sigmoid has these features:

$$f(p_n, \theta_n, P_n) = \sigma(\frac{p_n - \theta_n}{\max(P_n) - p_n}) \tag{3}$$

We will further refer to this process as Trained Sigmoid (TS) normalization. In Section 3.3.1 we explain how to establish θ_n as well. It should be noted that we do not know θ_n without the ground truth. Therefore, we can not establish θ_n for our test set regularly. We illustrate how to estimate θ_n further in Section 3.3.6.

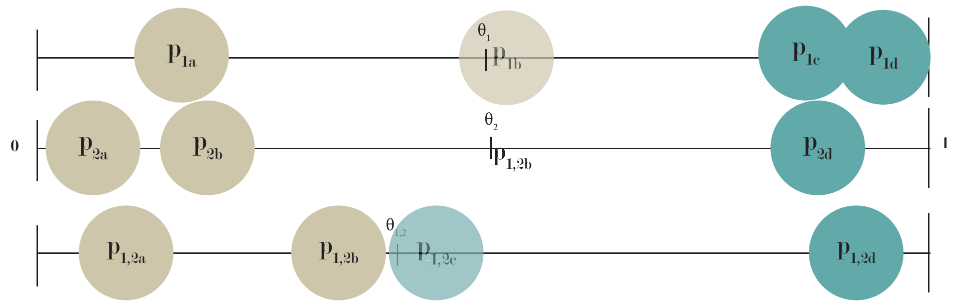


Figure 12: Example to illustrate how TS normalization can improve combined predictions. Figure is the same as Figure 11, however, the axes for method 1 and 2 are rescaled using TS normalization. The predictions of the bottom 1D plot is the mean of the two TS normalized methods above.

As we can see in Figure 12, a perfect prediction suddenly becomes possible through TS normalization as we were able to flip the location of $P_{1,2c}$ and $P_{1,2b}$. Any prediction close to the threshold stays near the threshold, but any prediction far from the threshold gets pushed away.

Please note that the illustrated thresholds and predictions are arbitrary; the ability to flip these two predictions by first aligning the prediction thresholds before taking the mean is central.

In Section 3.3.6 we explain how we evaluate that TS normalization is of added value for combining two complementary predictions.

3.3 Evaluation

3.3.1 Maximized F1-Score

Radivojac et al (2013) [27] defines the maximized F1-Score, or F_{max}, as follows:

$$F_{max} = \max_{\theta} \frac{2 \cdot pr(\theta) \cdot rc(\theta)}{pr(\theta) + rc(\theta)} \tag{4}$$

Specifically for GO annotation evaluation, the following is defined for precision $pr(\theta)$:

$$pr_i(\theta) = \frac{\sum_f I(f \in P_i(\theta) \wedge f \in T_i)}{\sum_f I(f \in P_i(\theta))} \tag{5}$$

$$pr(\theta) = \frac{1}{m(\theta)} \cdot \sum_{i=1}^{m(\theta)} pr_i(\theta) \tag{6}$$

In Equation 5, we calculate the precision for protein i by taking the amount of true predictions and dividing it by the total amount of predictions. Function $I(x)$ is the standard indicator function, which is 1 if logical operation x is true and 0 if not. We sum over every class f to get a total prediction $pr_i(\theta)$ for every protein.

In Equation 6, we take the mean of the test protein precisions, only including test proteins that have one or more predictions at threshold θ.

$$rc_i(\theta) = \frac{\sum_f I(f \in P_i(\theta) \wedge f \in T_i)}{\sum_f I(f \in T_i)} \tag{7}$$

$$rc(\theta) = \frac{1}{n} \cdot \sum_{i=1}^{n} rc_i(\theta) \tag{8}$$

Equation 7 and 8 are very similar to Equations 5 and 6, however, we divide by the amount of true values for calculating rc_i (false negatives + true positives) and for calculating rc we take the mean over every protein, not just the proteins with at least one prediction at threshold θ,

The F_{max} essentially calculates our F1 score in optimal conditions.

3.3.2 Estimated F-score Ratio for Robustness Evaluation

To evaluate the robustness of predictions made by every method, we introduce the Estimated F-score Ratio. In practical terms, we estimate the threshold for our model by looking at the thresholds that maximize the F score in other training-test splits.

$$\hat{\theta}_s = \frac{1}{|S \setminus \{s\}|} \sum^{S \setminus \{s\}} \theta \tag{9}$$

In Equation 9 we estimate θ_s for split s by taking the mean of the θ for other splits, defined as the set $S \setminus \{s\}$. We can then plug in $\hat{\theta}$ in the calculation of the F score to get $F(\hat{\theta}) = \frac{2 \cdot pr(\hat{\theta}) \cdot rc(\hat{\theta})}{pr(\hat{\theta}) + rc(\hat{\theta})}$. Using the equations from Section 3.3.1 we can then get an F-score for our estimated θ.

$$F_{ratio} = \frac{F(\hat{\theta})}{F_{max}} \tag{10}$$

To get the Estimated F-score Ratio, we apply Equation 10.

3.3.3 LLM Stopping Layer Analysis

To evaluate the usability of stopping layers in functional annotation, we calculate the F_{max} for our test splits on goPredSim using ESM1b with a stopping layer for every transformer layer in ESM1b.

To identify changes between embedding outputs from successive layers, we calculate the cosine similarity defined in Equation 1 between embedding vectors given by the last layer and embedding vectors given by earlier layers.

Furthermore, we will plot the embedding runtime and storage VRAM on GPU for every stopping layer on ESM1b to illustrate the required computational load for ESM1b truncated using stopping layers, comparing it with the full model as well. Furthermore, we run a ridge regression [36] on both plots, as they indicate a linear relation.

3.3.4 Grid Search Hyperparameter Selection

For safprednn and goPredSim, we hypothesise that the respective hyperparameters *percentile* and K are of high importance to the final performance of the model. To evaluate this importance, we apply a grid search for the best hyperparameters, looking at the F_{max} and θ that achieves the F_{max}.

3.3.5 Complementarity Metric

To evaluate if two methods are complementary and if there is potential to combine the methods, we define the complementarity metric.

$$l_1 = \frac{1}{n} \sum |p_i - y_i| \tag{11}$$

In Equation 11, we define the L1-loss over a single GO class. With n test proteins, we take the mean over the absolute value of the difference of the prediction and the ground truth.

$$c_g^{f1,f2} = l_1^{f1} - l_1^{f2} \qquad c_g^{f1,f2} >= 0 \tag{12}$$

The complementarity of function $f1$ and $f2$ for a single GO prediction g is found in Equation 12. We clip negative negative values to 0, as we want to evaluate how much of the loss in $f1$ can be fixed by $f2$. If predictions are the same, the loss is the same. However, if predictions are different, one method will be complementary to the other.

$$c^{f1,f2} = \frac{1}{|G|} \sum^{g \in G} c_g^{f1,f2} \tag{13}$$

To get the average complementarity, we take the mean of all GO complementary metrics. To get a better indication of cases where GO predictions can actually be improved, we define G as the set of GO classes g where $c_g^{f1,f2} >= 0$ or $c_g^{f2,f1} >= 0$.

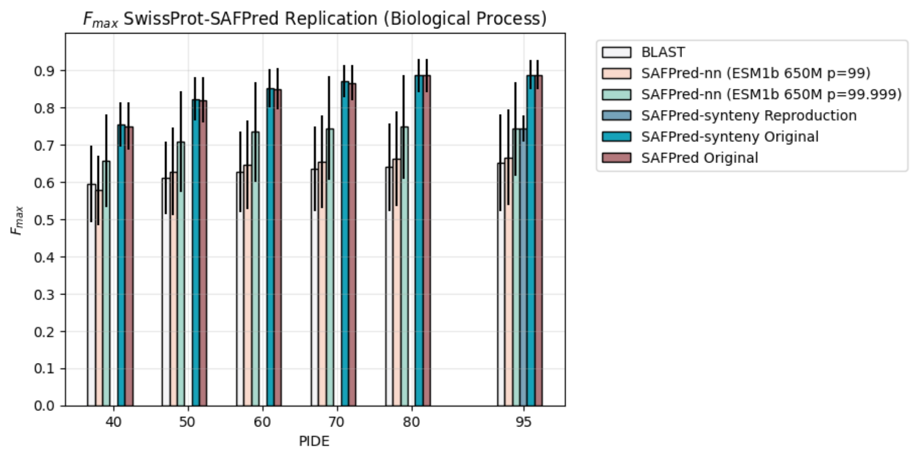
In essence, the higher both $c^{f1,f2}$ and $c^{f2,f1}$ are, the stronger we can conclude that method $f1$ and method $f2$ can fix the loss of one another for more than one GO class, and thus these methods are complementary. If only one of these metrics is high, say $c^{f1,f2}$, then we can only conclude that $f2$ can fix issues in $f1$, but not the other way around.

3.3.6 Complementary Learning Assessment

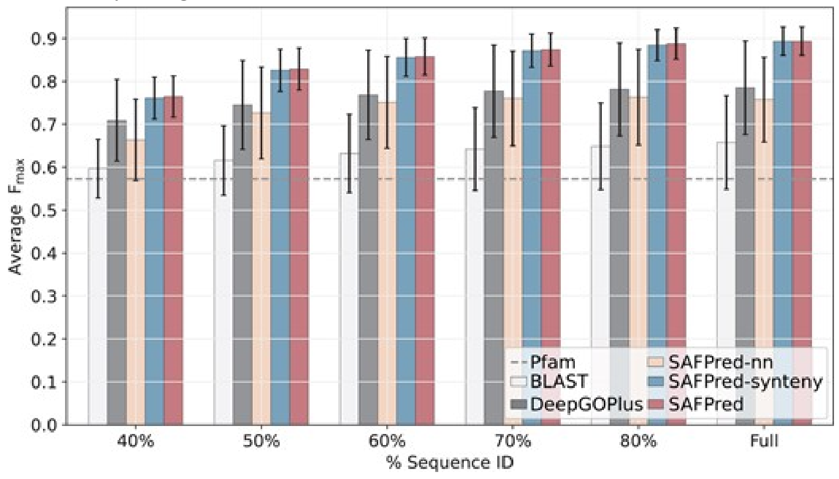
To evaluate if our complementary learning method is superior to taking the mean of predictions, we compare the F_{max} for the standard mean combination where inputs have been minmax scaled with the TS normalized mean combination and the F_{max} for the two uncombined methods.

4 Results

4.1 Reproduction of SAFPred results



(a) Mean F_{max} scores of all test splits for our reproduction of SAFPred. The underlying dataset for this reproduction is SwissProt-SAFPred. For SAFPred-nn, the p variable stands for the percentile, and the underlying LLM is also given. None that the SAFPRed-synteny Reproduction only has a result for $PIDE = 95$. Additionally, we plot our evaluation of the original SAFPred-Synteny and SAFPred predictions, denoted by 'Original'.



(b) F_{max} results of benchmarked methods from Urhan et al. 2024 [11].

Figure 13: F_{max} results for both the reproduction of SAFPred and the SAFPred paper itself. The y-axis indicates the F_{max} for the given method at the given PIDE on the x-axis. The bars are set on the average over all 5 test splits, and the errorbars indicate the standard deviation over all 5 test splits.

In Figure 13a we can see the reproduction of methods also benchmarked in the SAFPred paper. In this benchmark, we have reproduced BLAST, as well as SAFPred-nn with $p = 99$ as mentioned in the paper. We have also reproduced SAFPred-synteny, only for a PIDE of 95. We have also benchmarked SAFPred-nn with $p = 99.999$. Additionally, we have plotted our calculated F_{max} for the predictions made for the original SAFPred paper.

As we did not have direct access to the sequences that were used to build SAFPredDB, it was not possible to reproduce the step to identify training samples too close to test samples with BLAST, and therefore, we could not run SAFPred-synteny for different splits of PIDE. Additionally, the SAFPredDB entries trained on the actual test samples could not be removed either, so we would expect data leakage.

4.1.1 F_{max} reproducible for BLAST and SAFPred-nn

If we look carefully, we can see that for BLAST and SAFPred-nn ($p = 99.999$), the averages line up in Figures 13a and 13b. This indicates that our reproduction of both methods, as well as our F_{max} calculation, should be properly calibrated as well.

However, when we use the $p = 99$ as suggested by the SAFPred paper, we see that the average is significantly lower. This indicates that the suggestion to use $p = 99$ was most likely an inaccuracy.

For future comparison, it is important to verify that we are able to get the same results as the SAFPred paper. Moving forward, we can be confident that we are on the same grounds of comparison for both BLAST and SAFPred-nn, since we are able to get a similar average.

While there are slight differences in standard deviation between Figures 13a and 13b, the highest difference in standard deviation is about 0.05 on $F_{max} = 0.6$ ($PIDE = 40$, for method BLAST). This could be caused by the fact that we use DIAMOND instead of BLAST in an earlier step to remove training entries that exceed the PIDE to a test entry.

4.1.2 Reproduction issues SAFPred-synteny

While BLAST and SAFPred-nn have proven to be reproducible, the performance of the reproduced SAFPred-synteny, as shown in Figure 13a, is significantly lower than originally benchmarked in Figure 13b.

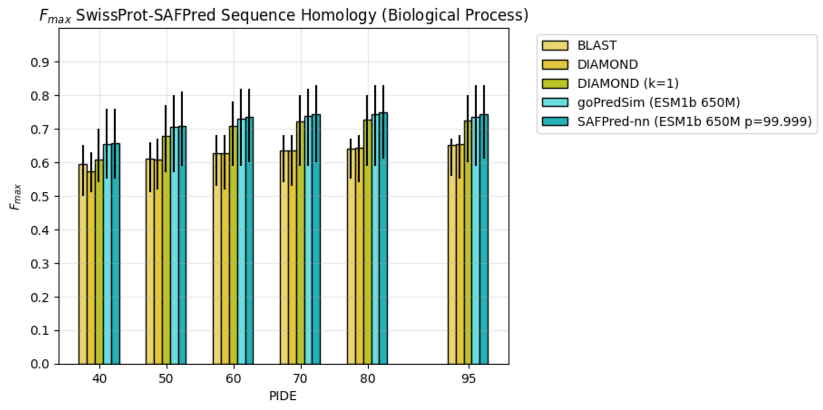
Since SAFPred-synteny is the main contribution of the SAFPred paper, it was an important goal to reproduce SAFPred-synteny. However, despite finding and fixing multiple issues in the SAFPred repository and despite active help from the main author of the SAFPred paper, we have not been able to come close to the original benchmarked performance.

In Figure 13a, we have also plotted our calculated F_{max} for the original SAFPred-synteny predictions, as well as the original SAFPred predictions. Comparing them with Figure 13b, we do see that the predictions that were originally made are actually as accurate as originally evaluated.

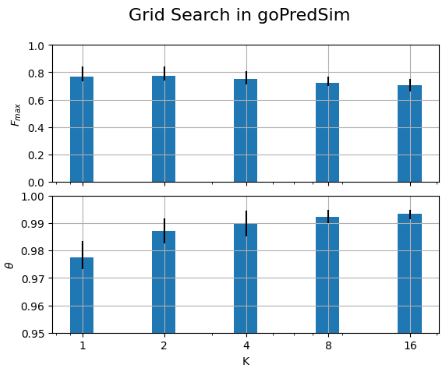
From this, we can conclude that while SAFPred-synteny is not quite reproducible to us, we can rule out that the lower values have to do anything with issues regarding the evaluation method. This also makes it likely that the reproduction issues are mainly on the prediction side.

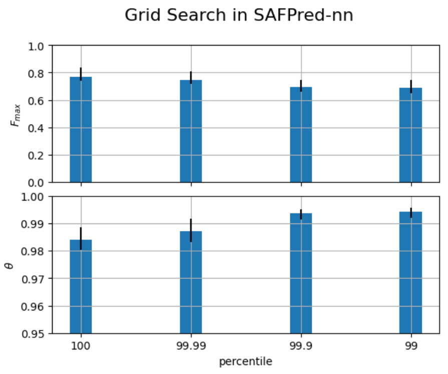
While it is unfortunate that we were not able to reproduce SAFPred-synteny, issues with paper reproduction are not uncommon, and even pervasive in the field of bioinformatics. [37]

4.2 Sequence homology inference methods optimally require few matches per query protein



(a) Mean F_{max} results of several sequence homology methods on the selected SwissProt-SAFPred dataset. For goPredSim, we use a K of 1. Both BLAST and DIAMOND have no restriction to the amount of significant matches per test protein. For DIAMOND (k=1), we restrict the maximum amount of significant matches to 1. The lower and upper errorbars signify the 25th and 75th percentile respectively. On the x-axis, the PIDE threshold for the train-test splitting mechanism is given.





(b) Mean F_{max} and threshold θ for different values of hyperparameter K in goPredSim. In this analysis we excluded E. coli as a test species due to technical limitations.

(c) Mean F_{max} and threshold θ for different values of hyperparameter p in SAFPred-nn. In this analyisis we excluded E. coli as a test species due to technical limitations.

Figure 14: Mean F_{max} results for selected sequence homology methods are plotted in Figure 14a. In Figures 14b and 14c the plots illustrate the effect of different hyperparameters on goPredSim and SAFPred-nn. In all figures the lower and upper errorbars signify the 25th and 75th percentile respectively.

In Figure 14 our main goal was to show that any of the evaluated sequence homology methods is optimized by inferring as few as possible homologs for every test sample. Chronologically, our first experiment was to show that either goPredSim and the similar method SAFPred-nn has a better performance when the amount of matches is low.

To minimize the amount of matches in goPredSim, we take a value of K that is as low as possible. As we can see in Figure 14b, the F_{max} is indeed optimal at K=1, dropping down as we gradually increase the value of K. We also notice that the thresholds that lead to the maximum F-score generally increase as K goes up.

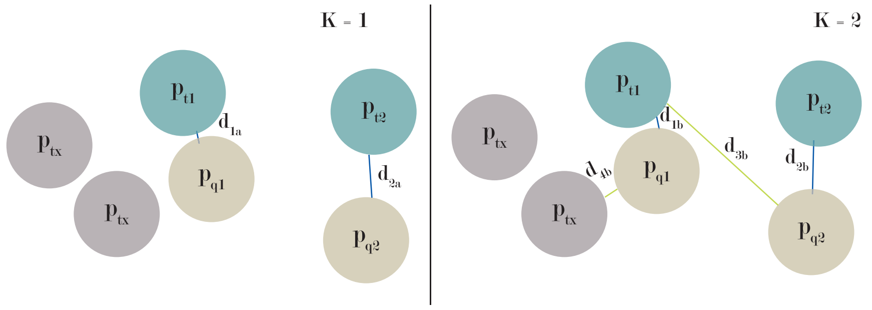


Figure 15: Illustrations to help explain why a higher K in goPredSim can cause both a lower F_{max} and a higher θ. The dots depict proteins in an arbitrary spatial domain. p_{tn} depicts a training protein that confers correct GO labels to their respective query protein. p_{tx} depicts a training protein that confers wrong GO labels. p_{qn} depicts a query protein. d_{na} and d_{nb} represent the distance between a query protein and a training protein.

To help reason why this can happen, we refer to Figure 15. If K=1, we see that the correct training proteins are matched to our query proteins. However, when we set K to 2, we see that p_{q1}, with only one good training example, is linked to a false training protein. p_{q2} is also linked to p_{t1}, which is not exactly a right match.

This should not be a problem in itself. We could simply use a higher threshold to get rid of false matches. However, in this case, we see that d_{1b} is smaller than d_{2b}. In this case, we can not find a threshold anymore that makes the prediction perfect, and our F_{max} suffers.

From this intuition, we speculate that the main reason why goPredSim suffers from a higher K is because query proteins that are close to secondary matches with less accurate GO labels actively hinder the inclusion of primary matches to query proteins that are further away, but still accurate.

From the same intuition, we predict that the threshold has to be set higher at a higher K to compensate the linkage of secondary matches with worse accuracy than the primary match.

In Figure 14c we also see that a higher percentile in SAFPred-nn confers a better accuracy and a lower threshold. Since a higher percentile means more restriction in matches, we can see that this effect is analog to the effect in goPredSim. Due to the high similarity of the two methods, it is likely that the same reasoning holds to this effect as well.

4.2.1 BLAST and DIAMOND have a similar performance

In this section we also wanted to support the use of DIAMOND as an alternative to BLAST. The main reason to use DIAMOND over BLAST is the significant speed advantage. While DIAMOND was praised to be as sensitive as BLAST [18], we needed to show that this is the case for use in functional annotation as well.

In Figure 14a we see that BLAST and DIAMOND have a very similar F_{max} at the mean and both percentiles in the errorbars. This indicates that we should be able to use DIAMOND as an alternative to BLAST with no significant change to performance.

4.2.2 Restricting matches in DIAMOND makes it almost as powerful as the knn methods

Extending the earlier findings regarding the restriction of the maximum amount of matches per query protein, we wanted to see if this holds in DIAMOND as well. Since our findings indicate that we want to restrict the amount of matches as much as possible, we tried to use DIAMOND with only 1 match per query protein at most.

In Figure 14a we see that when we apply this restriction in DIAMOND, the F_{max} gains a significant boost to such an extent that it becomes only slightly worse than goPredSim and SAFPred-nn.

4.2.3 goPredSim and SAFPred-nn have similar performance

From Figure 14a we infer that goPredSim and SAFPred-nn with the given hyperparameters have a highly similar performance. When comparing these two results to DIAMOND (k=1), we see that while DIAMOND (k=1) attains a similar performance at a higher train-test split PIDE, at lower PIDE values the outperformance of SAFPred-nn and goPredSim methods is distinctly visible.

Moving forward, we will use goPredSim as our reference knn method of choice, as we have experienced it to be faster and easier to work with.

4.3 Model application on different SwissProt datasets

4.3.1 Models become slightly less accurate with SwissProt-Full instead of SwissProt-SAFPred

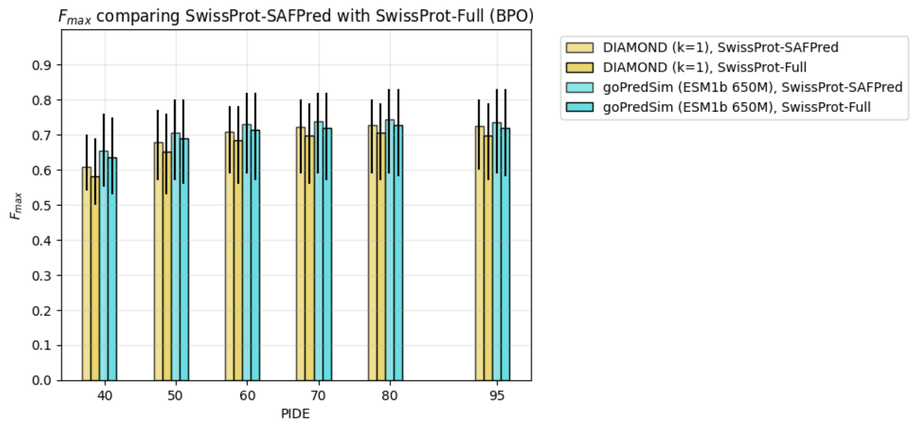


Figure 16: Mean F_{max} comparison of DIAMOND (k=1) and goPredSim on the SwissProt-SAFPred and the SwissProt-Full dataset. The x-axis denotes the PIDE for the train-test splitting mechanism. The y-axis denotes the F_{max}. The height of the bars indicate the mean over all test splits at the given PIDE. The lower and upper errorbars indicate the 25th and the 75th percentile respectively.

Before we can make an accurate evaluation of SwissProt-Prokaryote, we must first establish whether our baseline using SwissProt-Full is any similar to using SwissProt-SAFPred.

In Figure 16, we see that for both DIAMOND (k=1) and goPredSim predictions have gotten slightly worse. However, as this is possibly due to some random fluctuation as both the SwissProt dataset and the GO labels have evolved over three years, we believe this is not something we can prevent.

4.3.2 Pruning non-prokaryote data does not negatively impact model performance

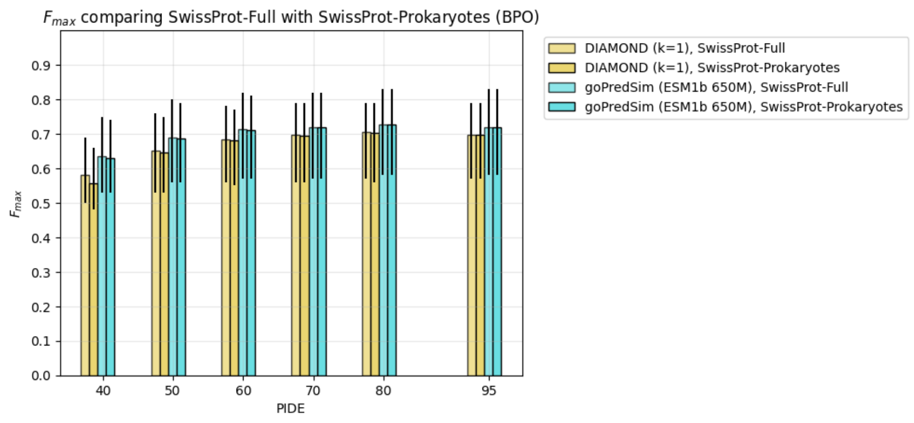


Figure 17: Mean F_{max} comparison of DIAMOND (k=1) and goPredSim on the SwissProt-SAFPred and the SwissProt-Full dataset. The x-axis denotes the PIDE for the train-test splitting mechanism. The y-axis denotes the F_{max}. The height of the bars indicate the mean over all test splits at the given PIDE. The lower and upper errorbars indicate the 25th and the 75th percentile respectively.

In Figure 17 we see that both evaluated methods enjoy a similar performance when we prune non-prokayote protein data from our dataset, as we do in SwissProt-Prokaryote. The only noticable difference is seen at PIDE 40 for DIAMOND (k=1), where it is possible that the prokaryote proteins removed by the training-test splitting mechanism had to be substituted by similar eukaryote/viral proteins.

This result is helpful, as we can essentially remove about 78% of our data without expecting detrimental effects. The pruning of our training set ensures that the models need less computational actions to train and infer functional annotations.

4.4 Early stopping layers in embedding models do not negatively impact goPredSim model performance

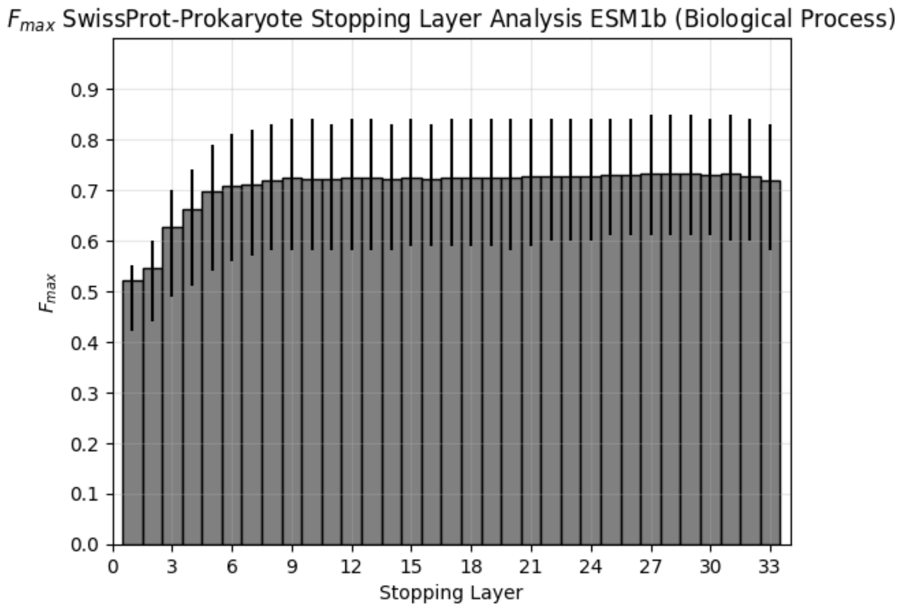


Figure 18: Mean F_{max} for goPredSim on the SwissProt-Prokaryote dataset. In this figure, we vary the transformer output embedding layer (stopping layer) in ESM1b before use in goPredSim. The x-axis denotes the stopping layer for ESM1b. The y-axis denotes the mean F_{max}. The height of the bars indicate the mean over all test splits at the given PIDE. The lower and upper errorbars indicate the 25th and the 75th percentile respectively.

In Figure 18 we can see that from stopping layer 9 and up both the F_{max} means and the errorbars stabilize around the same accuracy. At this point we identify two potential causes: the output of layer 9-33 is virtually the same, or the output of layer 9-33 is not the same, but the changes in the embedding space do not change labels too much anymore.

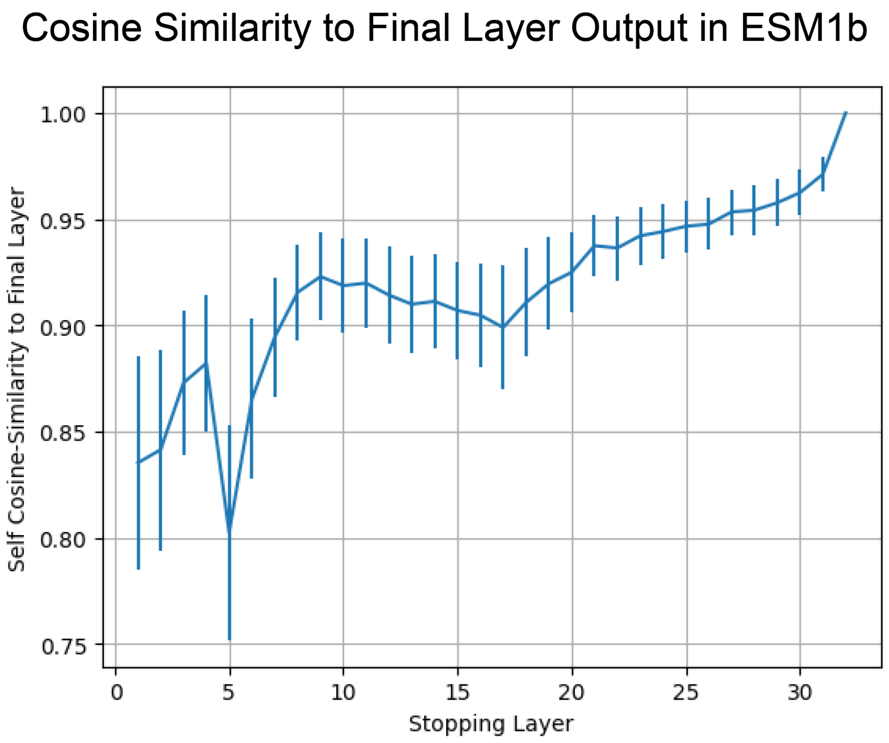
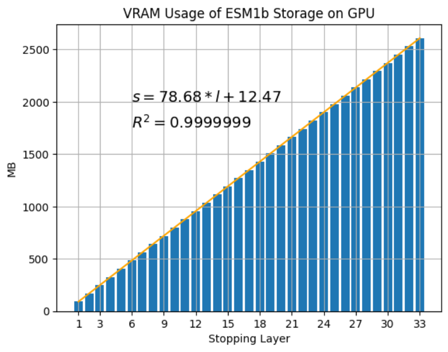


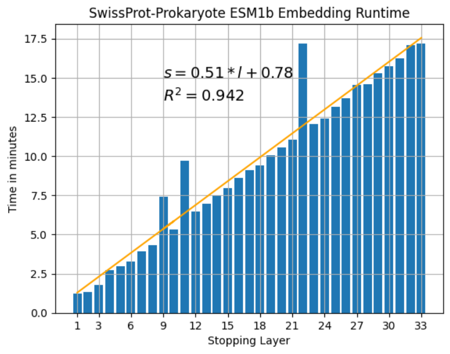
Figure 19: Cosine similarity of 200 random protein embeddings made at the stopping layer given at the x-axis compared to the protein embedding output at the final (33rd) embedding layer.

Figure 19 suggests that the embedding vectors do indeed change for the outputs at layer 9-33 in ESM1b. So, from the information of Figure 18 and 19, we observe that the embedding space changes in these layers are present, but do not actually change the accuracy of goPredSim.

4.4.1 Early stopping layers significantly improve embedding speed and GPU VRAM



(a) VRAM usage of ESM1b on the GPU given the stopping layer on the x-axis. The y-axis indicates the MB amount of VRAM on the GPU.



(b) Runtime in minutes for creating an embedding vector for all proteins on SwissProt-Prokaryote given ESM1b with a stopping layer on the x-axis. The runtime in minutes is indicated on the y-axis.

Figure 20: The runtime and GPU VRAM usage for model storage of ESM1b with SwissProt-Prokaryote. In both plots, a ridge regression is overlaid to highlight the linear relationships between the axes.

Figure 20 highlights what we stand to gain by applying stopping layers in ESM1b for use in goPredSim. In essence, if we use a stopping layer of 9, as Figure 18 indicates to give no impact in performance, the ridge regression indicates that we would need 720,59 MB of VRAM to store the model instead of 2608,91 MB, and generating the embedding vectors would take 5,37 minutes instead of 17,61 minutes. This cuts runtime to 30.4% and storage VRAM to 27,6% compared to the model without a stopping layer ($l = 33$).

It is important to note that model storage is not the only user of VRAM in the GPU. Both inference calculations and input data also take up some space.

This finding does not only help us in making currently available LLM models faster and lighter, but it would make heavier LLM models requiring more runtime and GPU memory available to weaker hardware for the use of embedding extraction. As ESM1b only has $\tilde{6}50$ million parameters, and the current biggest model in the DGEB leaderboard (progen2) has 6,4 billion parameters [35], we see that this 10-fold increase in parameters could be overcome using stopping layers.

Additionally, while 17,61 to 5,37 minutes does not sound like an important reduction, note that we only embed the 24.431 proteins in SwissProt-Prokaryote.

We go from 1.387 proteins per minute to 4.549 proteins per minute, assuming a stable rate with similar protein lengths.

If we wish to embed a dataset that is significantly larger, say the entirety of UniProt, normally that would take us 183.312 minutes, or 127 days. Using the stopping layer $l = 9$, this would take us 55.892 minutes, or 39 days, without a worse performance in GO annotation.

Naturally, we should consider using CD-HIT or linclust as well to reduce the dataset redundancy of UniProt before embedding, which would net us another significant time save.

4.4.2 Using batches in protein embedding can result in a lower performance

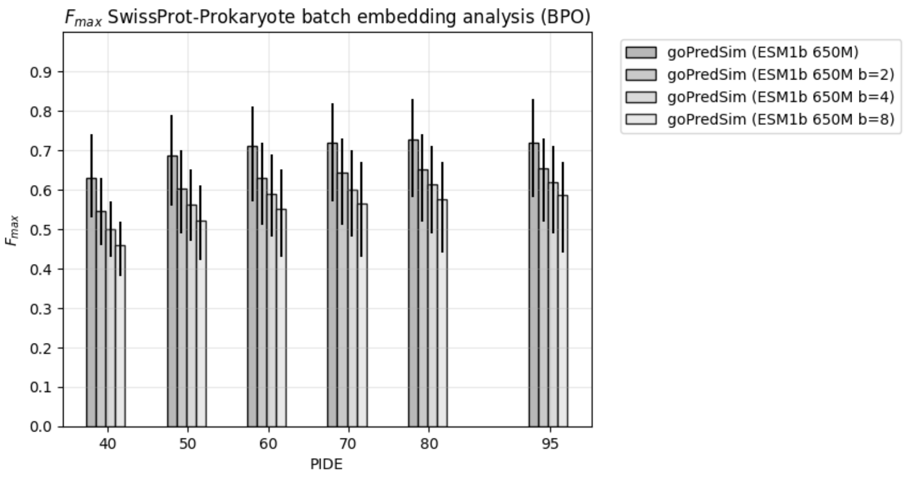


Figure 21: Figure to illustrate the effect on goPredSim of embedding multiple proteins in batches. The x-axis denotes the PIDE for the train-test splitting mechanism. The y-axis denotes the F_{max}. The height of the bars indicate the mean over all test splits at the given PIDE. The lower and upper errorbars indicate the 25th and the 75th percentile respectively.

To look for potential speed optimizations in the embedding process, we tried embedding proteins in batches. However, we never really expected the negative effect batch embeddings could bring us.

In Figure 21 we see what significant negative effect batch embedding has on goPredSim. We see that as we increase the amount of proteins per batch (b), the mean F_{max} drops rapidly.

We suspect this has to do with the fact that ESM1b embedding output is over amino-acids, which we convert to protein embeddings by taking the mean over all amino-acid embeddings. If we implement batching, the transformer input includes padded empty amino-acid characters so that every protein is the same length. If we include the transformer output of the padding in the mean amino-acid embedding, the protein vectors will be skewed by the output caused by the padding.

We predict that we could fix this issue by ignoring the padded additions of the proteins when calculating the mean amino-acid embeddings. However, as unbatched protein embedding achieved quite a good speed in itself, we decided to not investigate this issue further.

4.5 New goPredSim-ESM2 does not outperform goPredSim-ESM1b, even with larger models

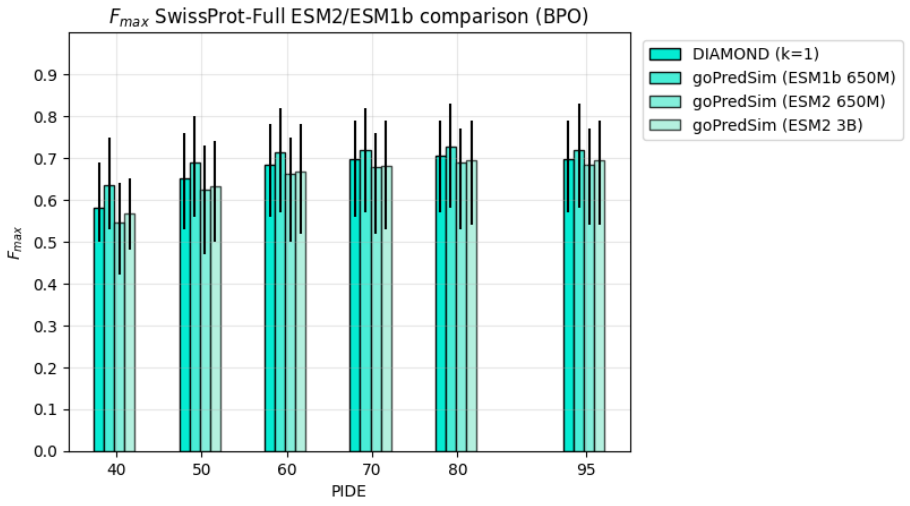


Figure 22: Figure to evaluate the impact of using ESM2 instead of ESM1b in goPredSim. The x-axis denotes the PIDE for the train-test splitting mechanism. The y-axis denotes the F_{max}. The height of the bars indicate the mean over all test splits at the given PIDE. The lower and upper errorbars indicate the 25th and the 75th percentile respectively.

Since previous studies indicate that ESM2 outperforms ESM1b [34, 35], we imagined that using ESM2 instead of ESM1b in goPredSim would be quite a simple improvement for use in functional annotation.

However, Figure 25 suggests the opposite. We see that goPredSim, using ESM1b, is still the best performing model, even compared to goPredSim-ESM2 with 3 billion parameters.

This came as quite a surprise, and we are still not exactly sure why this would be the case. We would have at least expected roughly the same performance, but not worse.

One possible speculation would be that the underlying dataset for ESM2 was somewhat different than for ESM1b. It could be possible that this earlier dataset for ESM1b was better representative for SwissProt-Full compared to the dataset for ESM2.

In any case, this finding highlights that we should not take any previous finding at face value, and if we were to try a different LLM than ESM1b in goPredSim, we should not expect that better LLMs in the literature also perform better in goPredSim.

4.6 Proposed structural homology method goPredSim-ESMfoldv1 not viable

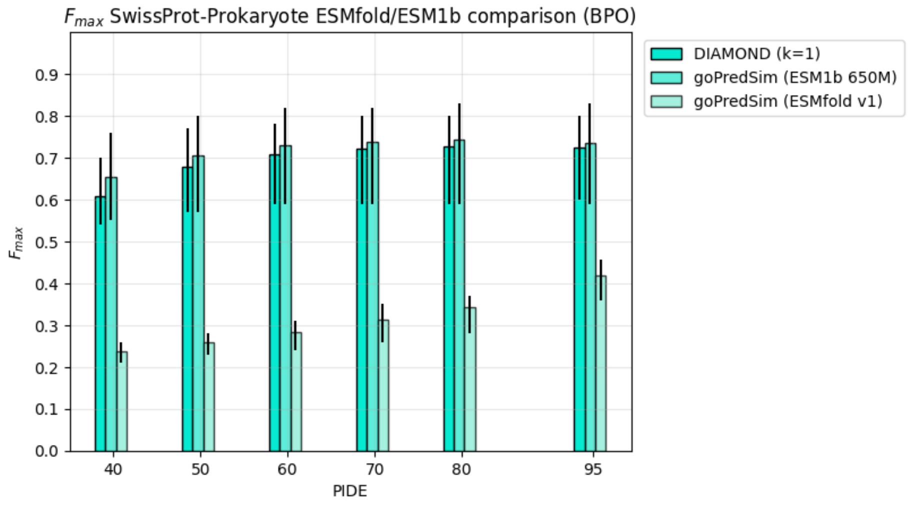


Figure 23: Figure to evaluate the impact of using ESMfold instead of ESM1b in goPredSim. The x-axis denotes the PIDE for the train-test splitting mechanism. The y-axis denotes the F_{max}. The height of the bars indicate the mean over all test splits at the given PIDE. The lower and upper errorbars indicate the 25th and the 75th percentile respectively.

Since GNNGO3D indicated the effectiveness of structural data for functional annotation, we wanted to see if we could expand SAFPred with a component that uses structural data as well. However, the key limitation with GNNGO3D is that we require experimental structural data, which is only possible for properly researched proteins. [21]

We wanted to see if we could use a LLM specifically made for structural inference to extract embedding encoding this structural information. In this way, we could get embedding vectors that could be used for structural homology inference. We chose ESMfold, as it is quite close to ESM1b that we already use in goPredSim.

However, in Figure 25 we see that goPredSim-ESMfold is vastly underperformant in comparison to our established methods for all PIDE.

This result indicates that this proposed strategy of using ESMfold in goPredSim for structural homology inference is not a good strategy for the purposes of functional annotation. A possible interpretation of the underperformance of goPredSim-ESMfold could be that we are essentially predicting on a structural prediction, and predicting the structure of proteins is a hard problem.

We could revisit this strategy as structural prediction methods become more powerful. However, it could very well be that this strategy is mostly fit for sequence-based homology inference.

4.7 Estimated F-score Ratio slightly better for DIAMOND (k=1)

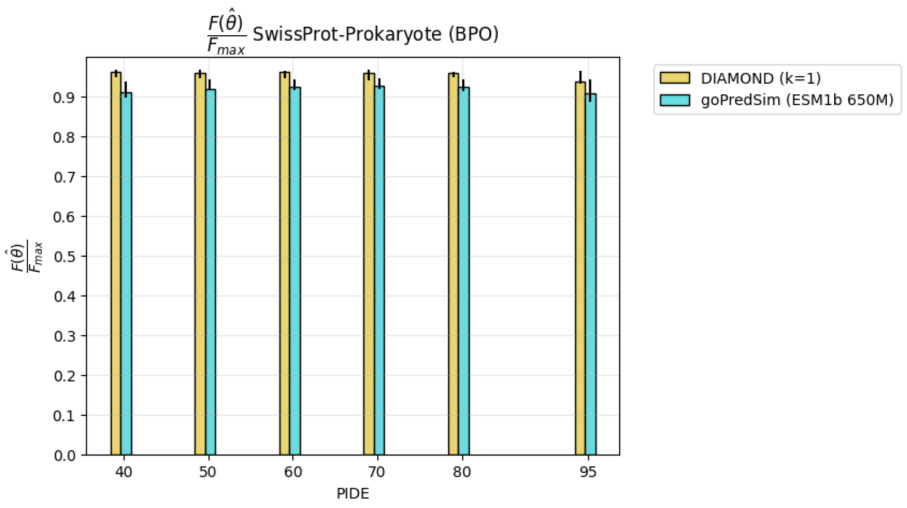


Figure 24: Estimated F-score ratio plotted for all PIDE splits. The methods evaluated are DIAMOND (k=1) and goPredSim (ESM1b 650M). On the y-axis the ratio between the F-score using an estimated threshold and the F_{max} is given. On the x-axis the PIDE for the training-test splitting mechanism is given. The lower and upper errorbars indicate the 25th and the 75th percentile respectively.

We were interested to investigate whether the F_{max} was actually well representative for an F-score for any test species. Hence, we calculated the F-score using an estimated threshold, and divided it by the F_{max}.

In Figure 24 we see that both DIAMOND (k=1) and goPredSim give F-scores that are quite close to the F_{max}, suggesting that the F_{max} gives quite a good indication for the actual model performance, with little deviation.

We see however that the F_{max} for DIAMOND (k=1) is still a bit more reliable than the F_{max} for goPredSim, as the ratio for the former is higher than the latter.

However, as both ratios are relatively close to 1, we can safely say that both methods are still quite reliable at predicting an unknown dataset given an estimated threshold.

It would be interesting to apply this analysis to other state-of-the-art functional annotation methods. If the F_{max} of a method appears high, but the θ

that confers the F_{max} is very dependent on the test species, then we might not be able to reach this F_{max} reliably, making the model weaker than it appears.

4.8 Current implementation SBGraph underperforms in comparison to established methods

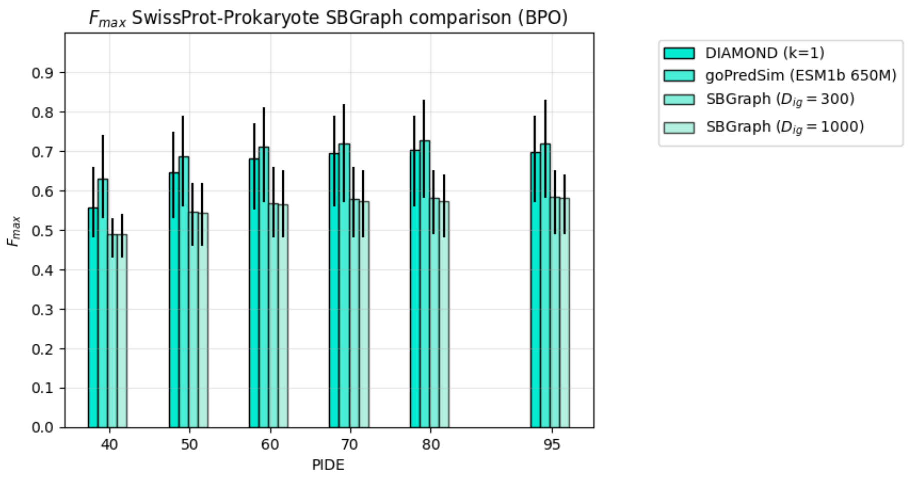


Figure 25: Figure to evaluate the performance of SBGraph in comparison with established methods. The x-axis denotes the PIDE for the train-test splitting mechanism. The y-axis denotes the F_{max}. The height of the bars indicate the mean over all test splits at the given PIDE. The lower and upper errorbars indicate the 25th and the 75th percentile respectively.

Since we were having trouble to reproduce the results for SAFPred-synteny from the SAFPred repository, we figured that, given synteny is an effective method for functional prediction in prokaryotes, we should be able to make a method similar to SAFPred-synteny on our own terms that might overcome some latent issues.

However, SBGraph was not able to perform well in the current implementation. Unfortunately, until we know the root cause of the underperformance of SAFPred-synteny in comparison to the original SAFPred-synteny results, we might not know for sure what the issue is that is holding SBGraph back, and if it a similar issue that is holding SAFPred-synteny back.

4.9 Combination of methods using TS-Normalization inconclusive

f1 / f2	goPredSim (ESM1b)	goPredSim (ESMfold)
goPredSim (ESM1b)	0.00	$4.17 * 10^{-4}$
goPredSim (ESMfold)	$4.53 * 10^{-3}$	0.00

Table 6: Mean complementarity comparing goPredSim (ESM1b) and goPredSim (ESMfold) on E. coli with a PIDE of 95 to the training set. The method in the row is $f1$, the method in the column is $f2$. In this comparison, we estimate the mean of the loss $f2$ can fix in $f1$.

We tried to see if we could combine goPredSim-ESM1b and goPredSim-ESMfold to obtain better predictions by leveraging potential complementarity. In Table 6 we see that, on average, there is not a lot of loss to be solved by either method, suggesting that either method is not complementary to each other.

Method / Organism	Escherichia coli	Mycobacterium tuberculosis	Bacillus subtilis	Pseudomonas aeruginosa	Salmonella ty-phimurium
goPredSim (ESM1b)	0.56	0.58	0.83	0.78	0.85
goPredSim (ESMfold)	0.34	0.23	0.38	0.36	0.69
Combination TS Normalization	0.52	0.50	0.72	0.68	0.82
Combination MinMax Scaling	0.52	0.49	0.69	0.67	0.80

Table 7: F_{max} values calculated for goPredSim (ESM1b), goPredSim (ESMfold), their combination with TS normalization, and their combination with MinMax scaling. The predictions are all for a 95 PIDE train/test split.

When trying to combine the two methods in Table 7, we do see a slight preference towards the TS Normalization combination method. However, overall, we are still better off not combining the methods at all, and sticking with goPredSim-ESM1b as a prediction method.

We can not strongly conclude much yet from the potential of TS-Normalization, as goPredSim-ESM1b and goPredSim-ESMfold are not actually very complementary to each other. It would be important to find two examples which are actually complementary before we can evaluate if TS-normalization has any potential for prediction improvement.

5 Conclusions

We have delved into possible improvements to existing methods pertaining to SAFPred, and there are quite some findings that prove useful to the functional prediction of prokaryote proteins. We will first try to put every finding in relation to our research questions.

5.1 Research Question 1

How can we refine SAFPred without fundamentally changing SAFPred itself? There were various ways which we could refine SAFPred, especially pertaining to goPredSim/safprednn.

Perhaps the most interesting findings here is that we can apply stopping layers to ESM-1b, and possibly to various other LLMs, without loss of prediction accuracy. This means that, for embedding-based inference, we can safely drop 70% and possibly more of the model. This makes for a significantly lighter model and faster model, which also makes it possible to utilize heavier models with less resources.

We have shown that the sequence homology methods DIAMOND, safprednn and goPredSim prefer to have as few as possible matches for every query protein.

We found that safprednn, while interesting, brings not much more to the table than goPredSim already did. These methods can be used interchangably with no severe impact on accuracy.

To our surprise, we found that ESM2, even with larger models, is inferior to ESM1b in the context of embedding-based protein functional prediction. We strongly expected that, due to the boasted performance compared to ESM1b, ESM2 would surely produce better representative embedding vectors than ESM1b.

5.2 Research Question 2

Can we effectively restrict our prediction to only use prokaryote-sourced data? It is safe to say that, for all our benchmarked methods, predictions on the SwissProt dataset pruned to only contain prokaryote proteins are about as good as the predictions on the full SwissProt dataset. This finding helps us dramatically cut down on training set size, which makes benchmarking faster and use less resources.

5.3 Research Question 3

Can we add a functional prediction method utilizing structural homology to improve predictions? The answer to this question is not necessarily no, however, the strategy we attempted was unfortunately ineffective. When we use ESMfold embeddings in goPredSim, we ultimately predict on a prediction, which gives an extra possible layer of failure.

Perhaps when protein structural embedding models become more powerful, this method could have more potential. However, a method more directly predicting on verified structural data might be our best bet right now.

5.4 Research Question 4

Can we create a better method for combining two functional predictions? We still have have too few results to conclude that this method is better than taking the mean of linearly normalized predictions. We should look for more effective methods to try to combine.

5.5 Research Question 5

Can we define a metric for evaluating model accuracy on unknown data? We have defined the Estimated F-score ratio and evaluated this metric on the given methods. From this, we have seen that we can reach quite close to the F_{max} in goPredSim and DIAMOND (k=1) using an estimated θ.

While CAFA gives us good indications for the potential performance of a model, their method of using unverified proteins is not easily reproduced by independent research. Therefore, this metric, or a similar metric with the same idea, gives non-participants to CAFA a good alternative for an F-score robustness check.

5.6 Additional Findings

5.6.1 Batch Embeddings

We never expected batch embedding to give problematic output, and we stumbled on this issue by accident. However, carelessly embedding proteins in batches for use in goPredSim gave us degenerate embeddings that conferred worse predictions than embeddings that have not been created in batches.

5.6.2 Synteny Method Underperformance

We went into this research assuming that synteny was of considerable value in the functional prediction of proteins in prokaryotes. However, every synteny-based method, even the method from SAFPred, gave us results that did not match the performance of the SAFPred paper.

As the original predictions have been cross-checked to be correct by us, it is likely that we are still missing an important piece of the puzzle that made

SAFPred-synteny so well performant. Unfortunately, that missing piece is still unknown.

6 Discussion

6.1 Reproduction Issues SAFPred

Issues around reproducing SAFPred have been a central theme in our research. Sadly, even with collaboration with the author, the reproduction of the original paper has been quite a challenge.

While fixing technical issues within SAFPred have never been a goal of this research, a good portion of the contribution of this thesis has been towards fixing these technical issues. One of the more glaring issues we came across in the SAFPred repository, was that there were multiple bugs in the function for the F_{max}.

The function calculates the F_{max} in a loop for every threshold. There had been a count for the number of proteins (n). At the end, we divide the sum of all protein recalls by the number of proteins to get the mean recall. However, this number of proteins was not re-initialized every threshold, so for higher threshold, the denominator would become extraordinarily high, and our recall would be very low. However, this division is only done on *eval_mode=full*, and the default is *eval_mode=partial*. While there is no indication with which the figures are made, it is likely that only *eval_mode=partial* was used.

The more impactful issue was that in the calculation of the protein-centric precision metric, the amount of true positives was divided by the amount of GO terms, regardless if they exceed the threshold or not. Instead, we should divide by the amount of predictions that exceed the threshold.

However, with the right F_{max} calculation it is still possible to reproduce the metrics for BLAST and safprednn. The impact of this issue might have been limited.

Additionally, there are several instances of files being missing or wrong files that have been placed in place of the missing files. Quite some time has been invested into figuring out why the file that turned out to be from the toy example gave terrible results. As of right now, these files have been recovered, and it should be possible to run the full SAFPred pipeline with minimal tinkering.

Inside the provided files, there are numerous cases where the given arguments do not reproduce the original predictions. In the main script given in the repository, there is a possibility to give the SAFPred-nn percentile as a parameter. However, this parameter never gets passed to the function itself. Instead, the default percentile of 99 gets used, which results in a high amount of predictions and a low performance.

Additionally, the argparse *store_true* had been misinterpreted to 'be true on default'. However, this flag actually means that the parameter is false on default, and true if the parameter is 'stored'. Therefore, the parameters 'norm_sim' and 'keep_singletons' should have been true, but were actually false.

6.2 Stopping Layers in Other Applications

Embedding-based prediction is not only reserved for protein functional predictions. There are quite some methods that use embeddings as features for classification problems, which could benefit by the use of Stopping Layers as well. Stopping Layers would not only make inference faster and less resource intensive, it would make inference on significantly larger models possible on smaller hardware.

There is a variety of applications of using text-embeddings for the purpose of classification. [38] However, the VRAM that is required for these models is increasing rapidly. Take the largest text model, Falcon 180B, which requires up to 320 GB VRAM for inference.

We would require a substantial budget if we wanted to use this model. However, if we only need embedding vectors from this model, we could consider using stopping layers so that we do not need to store the entire model at once. We might be able to cut the model down to 80 GB of VRAM.

However, we would still have to find out if stopping layers are also effective for the task at hand. We could do this by testing on a smaller LLM, and seeing if our accuracy maximizes earlier than at the final layer.

6.3 Taxonomy Level as Supporting Prediction Feature

Since prokaryote proteins can comfortably be predicted by only using other prokaryote proteins as a training set, we could conceptualize a method that pre-filters a training set on taxonomy before applying a model. Perhaps even more specific taxonomies could be effective.

7 Acknowledgments

I would like to take a moment to extend my thanks to everyone that helped me in the process of making this thesis.

Firstly, I would like to thank my supervisor Thomas Abeel, for giving me many good insights and inspiration in all steps of the process. Thomas always took the necessary time to assist, with meetings often taking longer than planned, despite his busy schedule. For that I am very grateful.

I would like to thank everyone at the Delft Bioinformatics Lab for making this project many times more bearable. While I figured things out on my own in the beginning, after a while I realized that it is much better to struggle alongside others.

I would like to thank Aysun Urhan, the main author of the SAFPred paper, for her assistance in the reproduction of SAFPred.

Finally, I would like to thank my friends and family for their continued support despite my substantial absence. I hope my absence is not interpreted as neglect, and if so, I apologize.

References

[1] "UniProt: the Universal Protein Knowledgebase in 2025". In: *Nucleic Acids Research* (2024), gkae1010.

[2] Eric W Sayers et al. "GenBank 2024 update". In: *Nucleic Acids Research* 52.D1 (2024), pp. D134–D137.

[3] Amos Bairoch and Brigitte Boeckmann. "The SWISS-PROT protein sequence data bank". In: *Nucleic acids research* 19.Suppl (1991), p. 2247.

[4] William R Pearson. "An introduction to sequence similarity ("homology") searching". In: *Current protocols in bioinformatics* 42.1 (2013), pp. 3–1.

[5] Kara Dolinski and David Botstein. "Orthology and functional conservation in eukaryotes". In: *Annu. Rev. Genet.* 41.1 (2007), pp. 465–507.

[6] Letícia MF Bertoline et al. "Before and after AlphaFold2: An overview of protein structure prediction". In: *Frontiers in bioinformatics* 3 (2023), p. 1120370.

[7] Andriy Kryshtafovych et al. "Critical assessment of methods of protein structure prediction (CASP)—Round XIV". In: *Proteins: Structure, Function, and Bioinformatics* 89.12 (2021), pp. 1607–1617.

[8] Jessica C Kissinger and Jeremy DeBarry. "Genome cartography: charting the apicomplexan genome". In: *Trends in parasitology* 27.8 (2011), pp. 345–354.

[9] *Illustration of an operon.* `https://www.biorender.com`. Year Unmentioned, accessed February 2, 2025.

[10] Anne E Osbourn and Ben Field. "Operons". In: *Cellular and Molecular Life Sciences* 66 (2009), pp. 3755–3775.

[11] Aysun Urhan et al. "SAFPred: synteny-aware gene function prediction for bacteria using protein embeddings". In: *Bioinformatics* 40.6 (2024), btae328.

[12] Jaina Mistry et al. "Pfam: The protein families database in 2021". In: *Nucleic acids research* 49.D1 (2021), pp. D412–D419.

[13] Ronghui You et al. "DeepGraphGO: graph neural network for large-scale, multispecies protein function prediction". In: *Bioinformatics* 37.Supplement_1 (2021), pp. i262–i271.

[14] Michael Ashburner et al. "Gene ontology: tool for the unification of biology". In: *Nature genetics* 25.1 (2000), pp. 25–29.

[15] Gene Ontology Consortium. "The Gene Ontology (GO) database and informatics resource". In: *Nucleic acids research* 32.suppl_1 (2004), pp. D258–D261.

[16] Naihui Zhou et al. "The CAFA challenge reports improved protein function prediction and new functional annotations for hundreds of genes through experimental screens". In: *Genome biology* 20 (2019), pp. 1–23.

[17] Ian Korf, Mark Yandell, and Joseph Bedell. *Blast.* " O'Reilly Media, Inc.", 2003.

[18] Benjamin Buchfink, Chao Xie, and Daniel H Huson. "Fast and sensitive protein alignment using DIAMOND". In: *Nature methods* 12.1 (2015), pp. 59–60.

[19] Sean R Eddy. "Accelerated profile HMM searches". In: *PLoS computational biology* 7.10 (2011), e1002195.

[20] Maria Littmann et al. "Embeddings from deep learning transfer GO annotations beyond homology". In: *Scientific reports* 11.1 (2021), p. 1160.

[21] Liyuan Zhang, Yongquan Jiang, and Yan Yang. "Gnngo3d: Protein function prediction based on 3d structure and functional hierarchy learning". In: *IEEE Transactions on Knowledge and Data Engineering* (2023).

[22] Tong Pan et al. "PFresGO: an attention mechanism-based deep-learning approach for protein annotation by integrating gene ontology inter-relationships". In: *Bioinformatics* 39.3 (2023), btad094.

[23] Limin Fu et al. "CD-HIT: accelerated for clustering the next-generation sequencing data". In: *Bioinformatics* 28.23 (2012), pp. 3150–3152.

[24] Alexander Rives et al. "Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences". In: *Proceedings of the National Academy of Sciences* 118.15 (2021), e2016239118.

[25] Donovan H Parks et al. "GTDB: an ongoing census of bacterial and archaeal diversity through a phylogenetically consistent, rank normalized and complete genome-based taxonomy". In: *Nucleic acids research* 50.D1 (2022), pp. D785–D794.

[26] Shujiro Okuda and Akiyasu C Yoshizawa. "ODB: a database for operon organizations, 2011 update". In: *Nucleic acids research* 39.suppl_1 (2010), pp. D552–D555.

[27] Predrag Radivojac et al. "A large-scale evaluation of computational protein function prediction". In: *Nature methods* 10.3 (2013), pp. 221–227.

[28] Suzi A Aleksander et al. "The gene ontology knowledgebase in 2023". In: *Genetics* 224.1 (2023), iyad031.

[29] Eric W Sayers et al. "Database resources of the national center for biotechnology information". In: *Nucleic Acids Research* 52.D1 (2024), p. D33.

[30] Torsten Seemann. "Prokka: rapid prokaryotic genome annotation". In: *Bioinformatics* 30.14 (2014), pp. 2068–2069.

[31] Martin Steinegger and Johannes Söding. "Clustering huge protein sequence sets in linear time". In: *Nature communications* 9.1 (2018), p. 2542.

[32] Christian Dallago et al. "Learned Embeddings from Deep Learning to Visualize and Predict Protein Sets". In: *Current Protocols* 1.5 (2021), e113. DOI: `https://doi.org/10.1002/cpz1.113`. eprint: `https://currentprotocols.onlinelibrary.wiley.com/doi/pdf/10.1002/cpz1.113`. URL: `https://currentprotocols.onlinelibrary.wiley.com/doi/abs/10.1002/cpz1.113`.

[33] A Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems* (2017).

[34] Zeming Lin et al. "Language models of protein sequences at the scale of evolution enable accurate structure prediction". In: *BioRxiv* 2022 (2022), p. 500902.

[35] Jacob West-Roberts et al. "Diverse Genomic Embedding Benchmark for functional evaluation across the tree of life". In: *bioRxiv* (2024), pp. 2024–07.

[36] Arthur E Hoerl and Robert W Kennard. "Ridge regression: applications to nonorthogonal problems". In: *Technometrics* 12.1 (1970), pp. 69–82.

[37] Yang-Min Kim, Jean-Baptiste Poline, and Guillaume Dumas. "Experimenting with reproducibility: a case study of robustness in bioinformatics". In: *GigaScience* 7.7 (2018), giy077.

[38] Liliane Soares da Costa, Italo L Oliveira, and Renato Fileto. "Text classification using embeddings: a survey". In: *Knowledge and Information Systems* 65.7 (2023), pp. 2761–2803.