Université Paris Cité

Faculty of Sciences

Long project

Going further in Protein Units Recognition and prediction of the classification using embedding and triplet networks

Presented by:

Anis Merabet

Supervised by:

Dr. Jean-Christophe Gelly **Dr. Gabriel Cretin**

2023-2024

Table of content

1 Introduction

Proteins are large, complex molecules that play crucial roles in the structure and function of living organisms. The primary structure of a protein is the sequence of amino acids. The secondary structure refers to the local folding of the polypeptide chain into alpha helices or beta sheets. Tertiary structure involves the overall three-dimensional shape of the protein, and quaternary structure refers to the arrangement of multiple polypeptide chains in a multi-subunit protein. In the field of structural biology, proteins are usually depicted in two complexity tiers: a basic local level representing secondary structures, and a broader level organized in domains. However, the differences complexity between these descriptions do not allow for a continuous understanding of structural organization.

The introduction of Protein Units (PU) is a paradigm shift in the study of protein architecture. These units serve as a bridging structural layer between secondary structures and domains, offering a more concise framework for analyzing the proteins (figure 1).

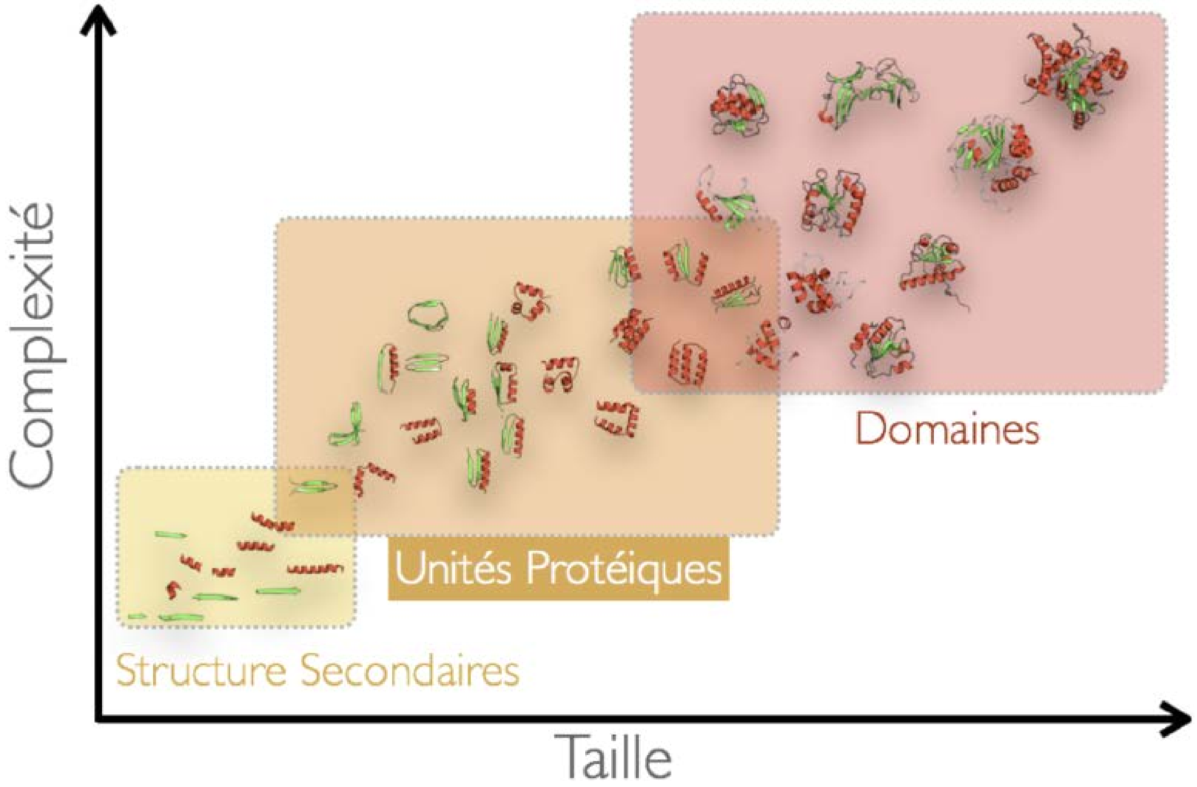


Figure 1: Protein Units serve as an intermediate level between secondary structures and protein domains. source: presentation document of the long project.

The identification of protein units involved a systematic analysis of an evolutionarily non-redundant set of protein structures. Protein units (PUs) are obtained through the Protein Peeling (PP) algorithm [1], [2]. It involves the cleavage of a protein structure. This method relies on an iterative hierarchical segmentation algorithm. The structures were segmented into PUs and then grouped based on their structural similarities, resulting in the formation of large superfamilies of PUs.

PUs were clustered by utilizing the Markov Clustering Algorithm (MCL). MCL operates by identifying groups within a weighted graph through random walks, adjusting both steps and edge weights to highlight frequently traversed segments. Essentially, it interprets the clustering procedure as the identification of communities within a graph, establishing connections among PUs by assessing the percentage of identity. The weighted graph serves as the foundation for MCL to categorize PUs into distinct groups. Those PUs that are successfully classified are referred to as "Core Protein Units" (CPU), while those that remain unclassified, meaning they do not establish any links, are designated as "Non-Core Protein Units" (NCPU).

CPUs are topologically simple, highly compact, recurrent, and energetically favorable. This superfamilly classification provides a comprehensive overview of CPUs and their distribution within proteins. These characteristics contribute significantly to their functional versatility and structural stability. Consequently, understanding the role of CPUs is crucial in elucidating protein dynamics and function in various biological contexts.

The characterization of protein units through recent work has demonstrated the relevance of this approach, opening up new prospects for both theoretical studies and practical applications. A key highlight of PUs lies in their potential for *ab initio* prediction of protein structures. Using mini-threading, PUs can effectively guide structural predictions. Furthermore, the presence of PUs within proteins may offer valuable insights into their functions. Additionally, the mastery of PU structures introduces large possibilities in protein engineering, enabling their use as fundamental building blocks for the creation of thermodynamically stable proteins.

The project aims, firstly, to generate embeddings for PU using a pre-trained model. Subsequently, a model will be constructed to distinguish between CPUs and NCPUs. Following that, fine-tuning of this model will be conducted to predict the classification of CPUs based on various hierarchical levels: Class, Architecture, Topology, and NanoFold. It's important to note that the Family level will not be included in the scope of this project (figure 2). Due to the imbalanced distribution of PUs within the categories of each classification and a limited number of examples for certain categories, the project will adopt a few-shot learning strategy with oversampling of underrepresented categories. This approach is chosen to address the challenges posed by the distribution and scarcity of examples in the dataset, ensuring robust model performance across all classification levels.

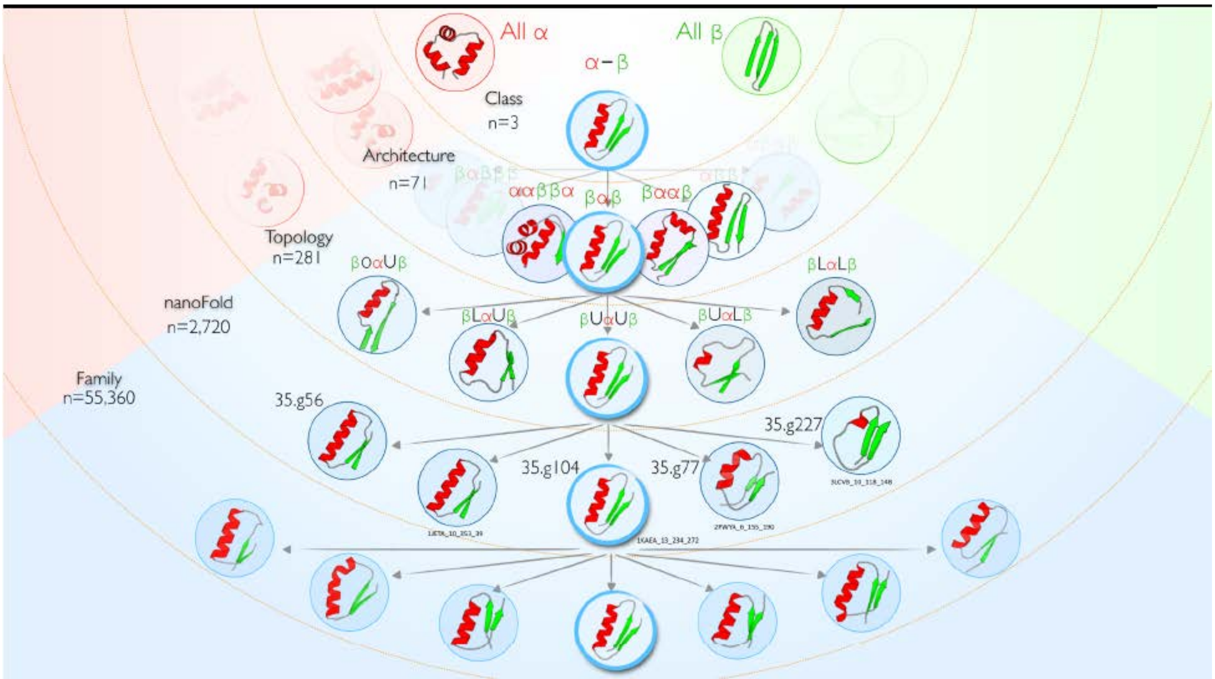


Figure 3: hierarchical classification of Core Protein Units representing the various levels of organization. source: presentation document of the long project.

2 Material and methods

2.1 Data list preparation

Core protein units (CPU) and non-core protein units (NCPU) were initially stored in different lists. The latter were reorganized into a tab-delimited text file then combined into a single file. Before filtering the data, there were 110379 CPU and 2981 NCPU. All CPUs with a single secondary structure were filtered out. In the initial CPU list, some CPUs were simultaneously defined as leaders and examples resulting in duplicates. Only one version was kept in this case. Some CPUs belonged simultaneously to different NanoFold classification. This is due to the uncertainty allowed by the peeling algorithm. These CPUs were kept in the combined data list file. Some NanoFold classes had overlapping classification at the architecture and topology levels, which is abnormal (not expected by using the peeling algorithm). To correct the overlapping issue, new architecture and and topology classes where created by adding “F” for “Fusion” at the beginning of the names of the predominant architecture and topology classes.

PDB IDs of the proteins were converted into UniProt IDs using pdb2uniprot (https://github.com/johnnytam100/pdb2uniprot) [3]. The source code of pdb2uniprot tool was modified to make it avoid duplicates and to adjust the order of the output according to the input. To handle cases where PDB IDs had multiple UniProt IDs, only the first UniProt ID for each PDB ID was retained. All CPU/NCPU where UniProt ID was not found were filtered out.

The complete protein sequences corresponding to the unique UniProt IDs were obtained by querying the UniProt database using the Biopython library. The protein sequences were saved in FASTA format files.

2.2 Global local search for mapping CPU/NCPU sequences to protein sequences and retrieval of nucleotide coordinates

A global local search was conducted using the glsearch36 tool (https://github.com/wrpearson/fasta36) to map each CPU/NCPU sequence to its respective protein sequence [4]. The results of the global local search were processed to extract start and end positions, and percentage identity. The CPU/NCPU with zero identify where filtered out from the dataset.

2.3 Protein Sequence Embedding

The protein sequences were embedded using the "ankh" library (https://github.com/agemagician/Ankh) [5]. This library encompass two pre-trained protein language models: Ankh Large and Ankh Base [6]. In our case, Ankh Base model (has the dimension of 768) was used to generate embedding for all protein sequences shorter than or equal to 10,000 amino acids. This limit was set because of resource limitation in DSIMB server. The final embedding were saved as a ".pt" file.

To get the embedding of CPU/NCPU, the corresponding protein embedding was loaded, then the relevant embedding segment was extracted based on the start and end positions, and saved into a “.pt” file. CPU/NCPU for which embedding was not performed due to large protein lengths were filted out PUs from the original dataset.

2.4 Triplet dataset preparation

To ensure randomness and reproducibility, the dataset was shuffled. Subsequently, it was divided into training, validation, and test sets, with proportions of 70%, 15%, and 15%, respectively.

Triplets, consisting of an anchor PU, a positive example from the same category, and a negative example from a different category, were generated for each dataset (train, validation, test) and for each of the 5 classifications. The algorithm iterates through each row in the dataset, treating each PU as a potential anchor PU for triplet formation. For each anchor PU, a positive example is chosen from the same category. This is achieved by filtering the dataset to include only PUs from the same category and excluding the anchor PU. If no positive candidates are found, the process is skipped for the current anchor PU. Hence, for categories containing only one CPU, no triplet was generated, but the CPU could be chosen as a negative example when constructing triplets for other categories. A single positive example is then randomly sampled from the positive candidates. The negative example is randomly chosen from a different category which is achieved by filtering the dataset to exclude PUs from the same category as the anchor PU. The negative examples did not consider the NCPUs when preparing the datasets for Class, Architecture, Topology and NanoFold classifications.

To address category imbalances in the datasets, the algorithm ensures that each category contributes an equal number of triplets to the overall dataset. The algorithm starts by calculating the frequency of PUs in each category within the dataset to identify the category with the highest number of PUs. Afterwards, the process of generating triplets within each category continues until the number of triplets for that category matches the maximum PU count.

2.5 Model construction and training

For each of the five classifications (type, class, architecture, topology, and nanofold), two models were generated using Ankh-generated embedding or one-hot encoding of the protein sequences, resulting in a total of 10 models.

Each of the 10 models were implemented using the PyTorch library. Each model consists of a triplet network containing a Gated Recurrent Unit (GRU) followed by a fully connected layer. The two layers shares the same weights between the anchor, positive and negative example.

To handle variable-length input sequences, the model employs dynamic padding and packing mechanisms. Sequences within a batch are padded to the same maximum length within each batch, ensuring uniform processing. The pack_padded_sequence function automatically considers the provided sequence lengths and implicitly masks the padded values during computation. The packed sequences are then passed through the GRU layer followed by the fully connected layer. To generate the embedding, the model returns the last hidden state for each of anchor, positive and negative example.

The model was trained using the train and the validation datasets, and optimized on the triplet loss function with category weights, where the loss was calculated based on the cosine distance between embeddings. The cosine distance was computed using the cosine similarity calculated after normalizing the embeddings. The Adam optimizer was utilized to minimize the loss function. The training loop included early stopping where patience was set to 5 epochs to prevent overfitting. In other words, the training was automatically stopped after five epochs if the validation loss was not enhanced. The maximum number of epochs was set to 1000.

The model was saved at the best epoch (lowest validation loss) with its weights and the state of the parameters of its optimizer. The embeddings of anchors belonging to the train dataset were also saved at the best epoch for subsequent analysis.

Model’s hyperparameter optimization was done using 2.5% to 10% (1000 to 4000 triplets) of the total triplet of the train dataset, and was performed only to the model based on Ankh-embedding, dedicated to distinguish CPUs and NCPUs. The following hyperparameters were considered during model optimization: learning rate, batch size, margin value of the triplet loss calculation, presence or absence of dropout layer and dropout rate, number of GRU layers, simple or bidirectional GRU layer, GRU hidden size, output size of the fully connected layer, presence or absence of sigmoid activation function applied to the fully connected layer.

The models designed for the Type classification (distinguish between CPUs and NCPUs) were constructed as described above, but the models designed for Class, Architecture, Topology and NanoFold classifications consisted on fine-tuning the model of the precedent classification level.

The only difference between the models based on Ankh-generated embeddings and the models based on one-hot encoding in the input size that was 768 for models based on Ankh-generated embeddings and 20 (corresponding to one-hot encoding of protein sequences according to the 20 amino acids) for the models based on one-hot encoding.

2.6 Model evaluation

The ten models were all evaluated using the same procedure.

The model's embeddings of the train and the test datasets were visualized using dimensionality reduction techniques, including Principal Component Analysis (PCA), Multi-Dimensional Scaling (MDS), t-Distributed Stochastic Neighbor Embedding (t-SNE), and Uniform Manifold Approximation and Projection (UMAP). It is to note that because of memory issue in DSIMB server, the plots obtained from the train datasets where based on only 50% of the PUs chosen randomly.

Cosine distance was calculated between the anchor and the positive and the anchor and the negative, then a threshold was set to determine true positives, true negatives, false positives and false negatives. To set the threshold, we plotted the distribution of PUs according to cosine distance to visualize the distribution, then we variated the threshold value from 0 to 2 (range of cosine distance distribution) by a step of 0.01 and calculated the Youden's index. The optimal threshold was defined by the value corresponding to the highest Youden's index. Receiver

Operating Characteristic (ROC) Curve was plotted and the Area Under the Curve (ROC AUC) was calculated. Similarly, Precision-Recall (PR) Curve was plotted and PR AUC was calculated.

Using the optimal threshold, Precision (Pr), Recall (SPE), F1-score, Matthew’s correlation coefficient (MCC), and Balance Error Rate (BER) were calculated, Confusion Matrix was plotted. The detailed formula of these metrics is detailed in appendix 1.

To evaluate the capability of the model to predict the categories of unseen PUs during training, we provide the PUs of the test dataset to the model and count the number of correct predictions according to two system modes. In both system modes, for each PU of the test dataset, the cosine distance between the PU and every PU of the train dataset is calculated, then the distances are ordered in an increasing order. In the first system mode, only the lowest cosine distance is considered. The category of the PU of the test dataset is compared to the category of the PU the train dataset. The prediction is counted as correct if they have the same category. The other system mode, it considers the five lowest cosine distances. The prediction is counted as correct if the category of the PU of the test dataset is the same as at least one of the categories of five PUs of the train dataset.

3 Results & discussion

3.1 Data exploration

After data filtration, the number the number of unique Uniprot IDs was 10372, and the total number of PUs was 32936. Among them, 29979 were CPUs and 2957 were NCPUs. Some of the CPUs were had different classifications according to the NanoFold level, so they were duplicated. Then the total number of PUs with duplicates were 59785. These results indicate a large data imbalance between CPUs and PUs.

The CPUs were classified according to different hierarchical levels. Each hierarchical level contains many categories as presented in table 1. We notice a ascending number of categories as we go through the hierarchical level, going from 3 categories for Class level to 1710 categories for NanoFold level.

Table 1: number of categories according to the hierarchical levels

Hierarchical level	Class	Architecture	Topology	NanoFold
Nb of categories	3	80	236	1710

Because of data imbalance observed between CPUs and NCPUs, we also aimed to verify data imbalance among the categories of each hierarchical level for CPUs. As indicated in appendix 2, for each hierarchical level, there were some categories that were overrepresented in comparison to others. To have a better visualization of category imbalance for Architecture, Topology and NanoFold level, we plotted the number of categories according to the number of PUs for each hierarchical level as shown in appendix 3. For the Architecture level, the majority of categories contained between 0 and 50 CPUs, for the Topology level, the majority of categories contained between 0 and 10 CPUs, and for the NanoFold level, the majority of categories contained between 0 and 5 CPUs. However, all categories contained at least 2 CPUs. Nevertheless, these results indicate that not only categories are largely imbalaced, but also, few examples are present for the majority of categories, which justify our strategy of few shot learning adapted in this study.

Now, we wanted to explore the characteristics of PUs in terms of length and compositions, which helps setting the model and understanding its mechanism afterwards. In appendix 4, A, we show the length distribution of the PUs. The majority of PUs had a length between 20 to 40 amino-acids. In second position, PUs had a length between 40 and 60 amino-acids. Finally, only 10 PUs had a length between 80 and 100. In order to see if there is any difference accoding to CPUs and NCPUs, we show in appendix 4, B, the length distribution displayed in percentage comparing CPUs and NCPUs. Although, they have the same tendency, we see a relative difference between them.

To explore the differences in amino-acid composition between CPUs and NCPUs, we show in appendix 4, C, the percentage of amino-acid frequency for CPUs and NCPUs. The results indicate a statistically significant difference between CPUs and NCPUs for 18/20 of amino-acids. We further grouped the amino-acids according to the categories shown in appendix 4, E, we found that CPUs and NCPUs were significantly different for all categories except for the category of small amino-acids (including A, G and S amino-acids). These results allow to hypothesize that there could be some differences in amino-acid composition that impacts their structural stability, allowing them to be redundant for CPUs and non-redundant for NCPUs.

3.2 Datasets and model structures

Using our approach ensuring to have the same number of triplets of each category within the hierarchical level, we constructed 79540 triplets as a train dataset for the model distinguishing CPUs and PUs (39770 triplet for each category), and 61074 triplets as a train dataset for the model distinguishing the categories of Class level (20385 triplets for each of the three categories). The triplet generation for Architecture, Topology and NanoFold levels is still ongoing (table 2). The expected number of triplets for the train dataset for Architecture, Topology and NanoFold levels are 851 040, 1 935 672, 1 201 446 respectively.

Table 2: number of triplets according to dataset and classification

Dataset	Type	Class	Architecture	Topology	NanoFold
Train	79540	61074	Ongoing	Ongoing	Ongoing
Validation	17106	13308	Ongoing	Ongoing	Ongoing
Test	8969	8505	Ongoing	Ongoing	Ongoing

The tuning of the hyperparameters of the 10 models was based only of the tuning of the model designed for distinguishing CPUs and NCPUs based on Ankh-generated embeddings. Hense, the 10 models had the same hyperparameters except for the input dimension that were 768 for Ankh-generated embedding-based models and 20 for one-hot encoding-based models. As a result, the number of parameters were difference between these two model categories.

In appendix 5, we show the model structures and the hyperparameter values. All models were composed of triplet network containing a GRU layer followed by a fully connected layer. According to our hyperparameter tuning procedure. A dropout layer was revealed to not provide an advantage to the model (data not shown). Otherwise, the hidden sizes greater than 32 for the GRU layer and 16 for the Fully Connected (FC) layer was also shown to not be beneficial to model’s performance (data not shown). The same statement was noticed for the batch size.

3.3 Results for models distinguishing CPUs and NCPUs

To distinguish CPUs and NCPUs, we tried two procedures. The first one is based on Ankh-generated embedding and the second one is based on one-hot encoding. In fact, our primary hypothesis is that the model using Ankh pre-trained model to generated the embedding of PUs will preserve the semantic information in amino-acid sequence. Thus, our triplet model could capture this information to distinguish CPUs and NCPUs. However, the one-hot encoding is not able to conserve the semantic information. Hense, the latter is expected to be less performant than the first one.

We trained both models using the same datasets, but for the model evaluation, we selected only 1000 same triplets because of time constrains. In fact, evaluating the model on the whole test dataset would take long time, reason for what we selected only 1000 triplets were selected for this analysis.

The loss curves shown in appendix 6.B indicate that the early stopping was triggered after only few epochs, which. As a result, our strategy of hyperparameter tuning should be reviewed. In fact, because of time constrains, the hyperparameter tuning was conducted using only 2.5% for the majority of hyperparameters.

In table 3, we present the results of evaluation of the two models.

Table 3: evaluation results of the models distinguishing CPUs and NCPUs

Metrics	Ankh-generated embedding based model	One-hot encoding based model
Precision (PRE)	0.7805	0.6074
Recall (SPE)	0.7360	0.6420
F1-score	0.7576	0.6242
MCC	0.5299	0.2274
BER	0.2355	0.3865
First closest mode	Total: 895/1000; R: 0.8950 CPUs: 864/951; R: 0.9085 NCPUs: 31/49; R: 0.6327	Total: 788/1000; R: 0.7880 CPUs: 768/951; R: 0.8076 NCPUs: 20/49; R: 0.4082
5 closest mode	Total: 980/1000; R: 0.9800 CPUs: 937/951; R: 0.9085 NCPUs: 43/49; R:0.6327	982/10000.9820 CPUs: 945/951; R: 0.9937 NCPUs: 37/49; R: 0.7551

BER, Balance Error Rate. MCC: Matthews Correlation Coefficient. R, ratio

As expected, the model based on Ankh-generated embedding performed better than the model based on one-hot encoding. All metrics were better for the first one, except for the ratio of correct prediction of CPUs considering the 5 closest mode. Given that it was not the case for the first closest mode, we could assume that the 5 closest mode strategy is less precise. See appendix 6 for further detail.

Even though, the number of triplets for CPUs and NCPUs used during training was the same, we observe better predictions for the CPUs comparing to NCPUs. This could be explained by two facts. The first one is that the number of CPUs is much higher than the number of PUs (see section 3.1) and our strategy of oversampling even it avoids duplicated triplets, is not flawless. Nevertheless, this strategy seems to be more performant than our primary strategy consisting on implementing weights ranging from 1 to 5 according to the inverse number of counts in each category (data not shown). It worth to mention that by combining both strategies (oversampling and weights) and training the model using 1000 triplets, the results shown a better prediction for NCPUs compared to CPUs (data not shown).

3.4 Results for models distinguishing categories according to “Class” level

In this section we present the results of the performance for the models designed to predict the category of PUs according to “Class” level. Because of time constrain, the model based on one-hot encoding is still running in the cluster M2BI and the results are expected to by ready by 6th February, 2024. We will then only present the results for the Ankh-generated embedding based model.

Otherwise, our primary strategy was to fine-tune the model designed to distinguish between CPUs and NCPUs, but because of time contrains, we changed the strategy to building a model *de novo* (without fine-tuning of the first model).

The table 4 summarizes the results according to the different computed metrics. The recall is very high comparing to the precision, which suggests that the model is good at identifying all instances of the positive class but may have a higher false positive rate. However, it may miss some positive instances. The balance between these two metrics that is given byt F1-score is 0.7419.

Table 4: evaluation results of the models predicting category according to "Class" level

Metrics	**Ankh-generated embedding based model**
Precision (PRE)	0.6057
Recall (SPE)	0.9570
F1-score	0.7419
MCC	0.4100
BER	0.3330
First closest mode	Total: 646/1000; R: 0.6460
5 closest mode	Total: 950/1000; R: 0.9500

Detailed results are shown in appendix 7.

3.5 Results for models distinguishing categories according to "Architecture" level

Results not yet generated. The code is ready, but the limiting factor is the generation of datasets which is still ongoing (table 2).

3.6 Results for models distinguishing categories according to "NanoFold" level

Results not yet generated. The code is ready, but the limiting factor is the generation of datasets which is still ongoing (table 2).

4 Conclusion

In conclusion, our study addresses the challenge of classifying and understanding protein units (PUs) with a focus on distinguishing CPUs from NCPUs. Initial data exploration revealed a substantial data imbalance between CPUs and NCPUs, prompting a comprehensive analysis of hierarchical levels and categories. Notably, the NanoFold level exhibited a remarkable diversity with 1710 categories, emphasizing the complexity of protein classification.

To mitigate data imbalances, we employed oversampling method and to mitigate the fact that the majority of categories containing very few examples, we used a few-shot learning strategy.

We meticulously examined PU characteristics, including length and amino-acid composition. The observed statistical differences in amino-acid frequencies between CPUs and NCPUs hinted at potential structural distinctions that could contribute to their respective roles. Subsequent exploration involved constructing datasets and model structures, with 10 models designed to distinguish CPUs and NCPUs then predict the category of CPUs according to the four hierarchical levels.

Our evaluation of model performance showcased the superiority of the Ankh-generated embedding-based model over the one-hot encoding model. Despite comparable training datasets, the imbalance in predictions between CPUs and NCPUs suggests that oversampling and weighting strategies are efficient, but not perfect. Notably, our approach yielded promising results, with the Ankh-based model demonstrating precision, recall, F1-score, Matthews Correlation Coefficient (MCC), and Balance Error Rate (BER) values that collectively underscore its efficacy.

Furthermore, our investigation extended to predicting categories within different hierarchical levels, such as "Class." While we await the completion of the one-hot encoding model's analysis, the Ankh-based model exhibited high recall but lower precision, emphasizing its proficiency in identifying positive instances at the expense of a potential higher false positive rate. Future analyses targeting "Architecture," "Topology," and "NanoFold" levels remain ongoing, with the code prepared for comprehensive assessments once datasets are generated.

In summary, our study integrates data exploration, model construction, and rigorous evaluation to contribute valuable insights into the classification of protein units. The nuanced distinctions identified between CPUs and NCPUs, coupled with the effectiveness of our models, lay the foundation for advancing our understanding of protein structures and functions. The ongoing exploration of higher hierarchical levels holds promise for extending these findings to a better CPUs prediction and categorization.

5 Bibliography

[1] J.-C. Gelly, A. G. de Brevern, and S. Hazout, "'Protein Peeling': an approach for splitting a 3D protein structure into compact fragments," *Bioinforma. Oxf. Engl.*, vol. 22, no. 2, pp. 129–133, Jan. 2006, doi: 10.1093/bioinformatics/bti773.

[2] J.-C. Gelly and A. G. de Brevern, "Protein Peeling 3D: new tools for analyzing protein structures," *Bioinforma. Oxf. Engl.*, vol. 27, no. 1, pp. 132–133, Jan. 2011, doi: 10.1093/bioinformatics/btq610.

[3] J. Tam, "johnnytam100/pdb2uniprot." Jan. 10, 2024. Accessed: Jan. 28, 2024. [Online]. Available: https://github.com/johnnytam100/pdb2uniprot

[4] W. Pearson, "wrpearson/fasta36." Jan. 24, 2024. Accessed: Jan. 28, 2024. [Online]. Available: https://github.com/wrpearson/fasta36

[5] A. Elnaggar, “agemagician/Ankh.” Jan. 14, 2024. Accessed: Jan. 28, 2024. [Online]. Available: https://github.com/agemagician/Ankh

[6] A. Elnaggar *et al.*, “Ankh: Optimized Protein Language Model Unlocks General-Purpose Modelling.” arXiv, Jan. 16, 2023. doi: 10.48550/arXiv.2301.06568.

6 Appendices

6.1 Results for models distinguishing categories according to “Topology” level

Results not yet generated. The code is ready, but the limiting factor is the generation of datasets which is still ongoing (table 2).

The results are expected to be reasonably good as the preliminary results obtained by training the models using a train dataset containing 2000 triplets and considering weights instead of oversampling, was good (data not shown).

6.2 Appendix 1: model evaluation metric formulas

- Precision (Pr)

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

- Recall (SPE)

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

- F1-score:

$$\textbf{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

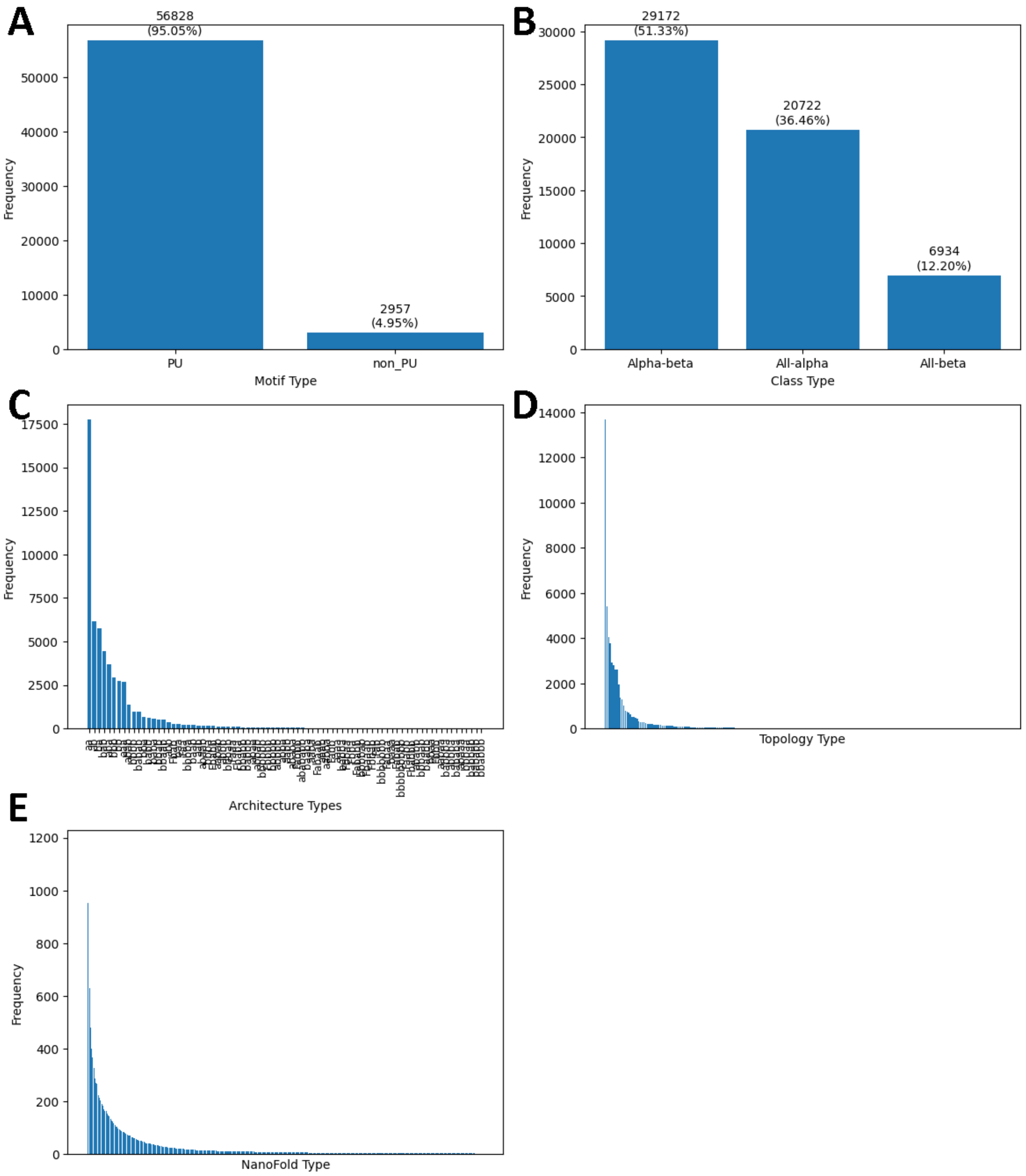
- Matthews Correlation Coefficient (MCC)

$$\text{MCC} = \frac{(\text{TP} \times \text{TN}) - (\text{FP} \times \text{FN})}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}$$

- Balance Error Rate (BER):

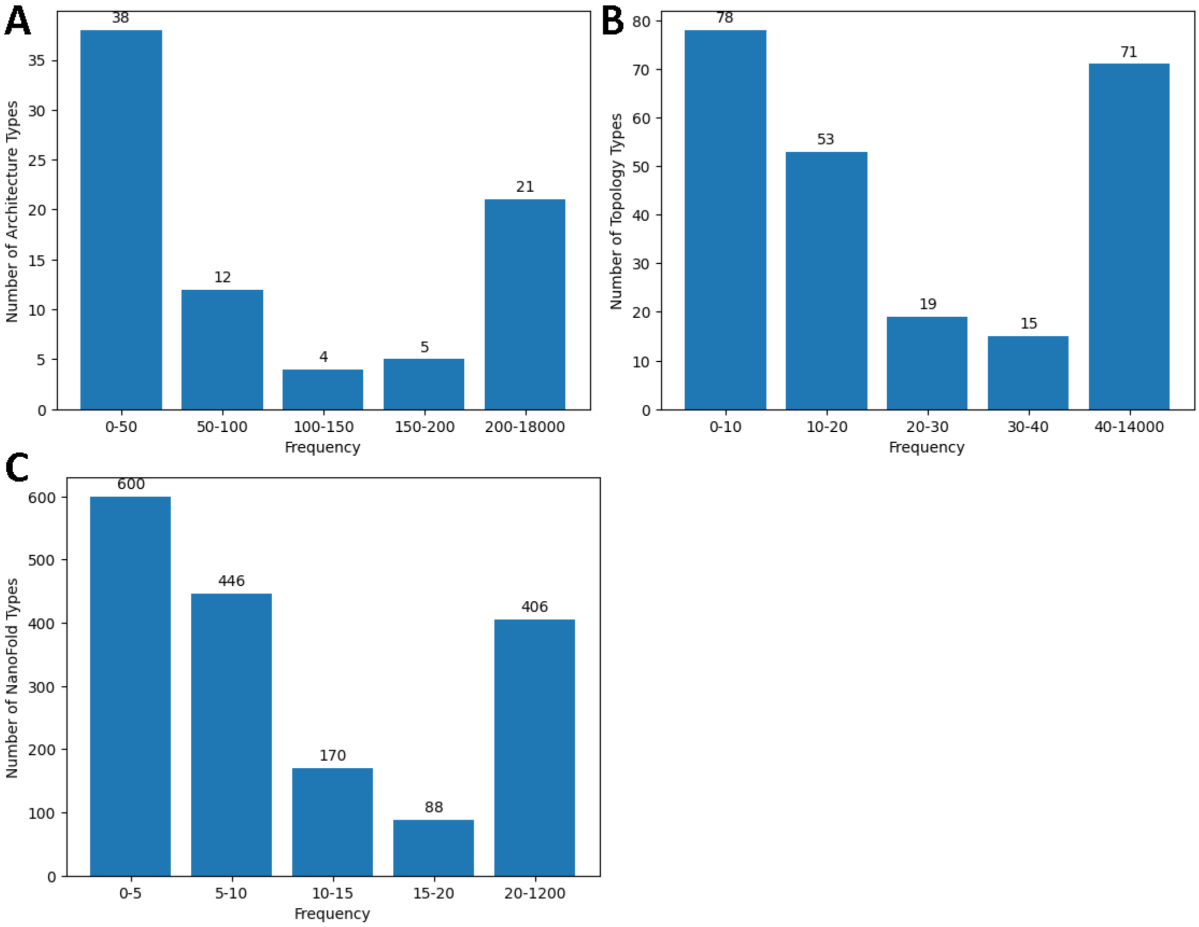
$$Balanced\ Accuracy = \frac{Recall + Specificity}{2}$$

6.3 Appendix 2: number of PUs according to categories for each hierarchical level.



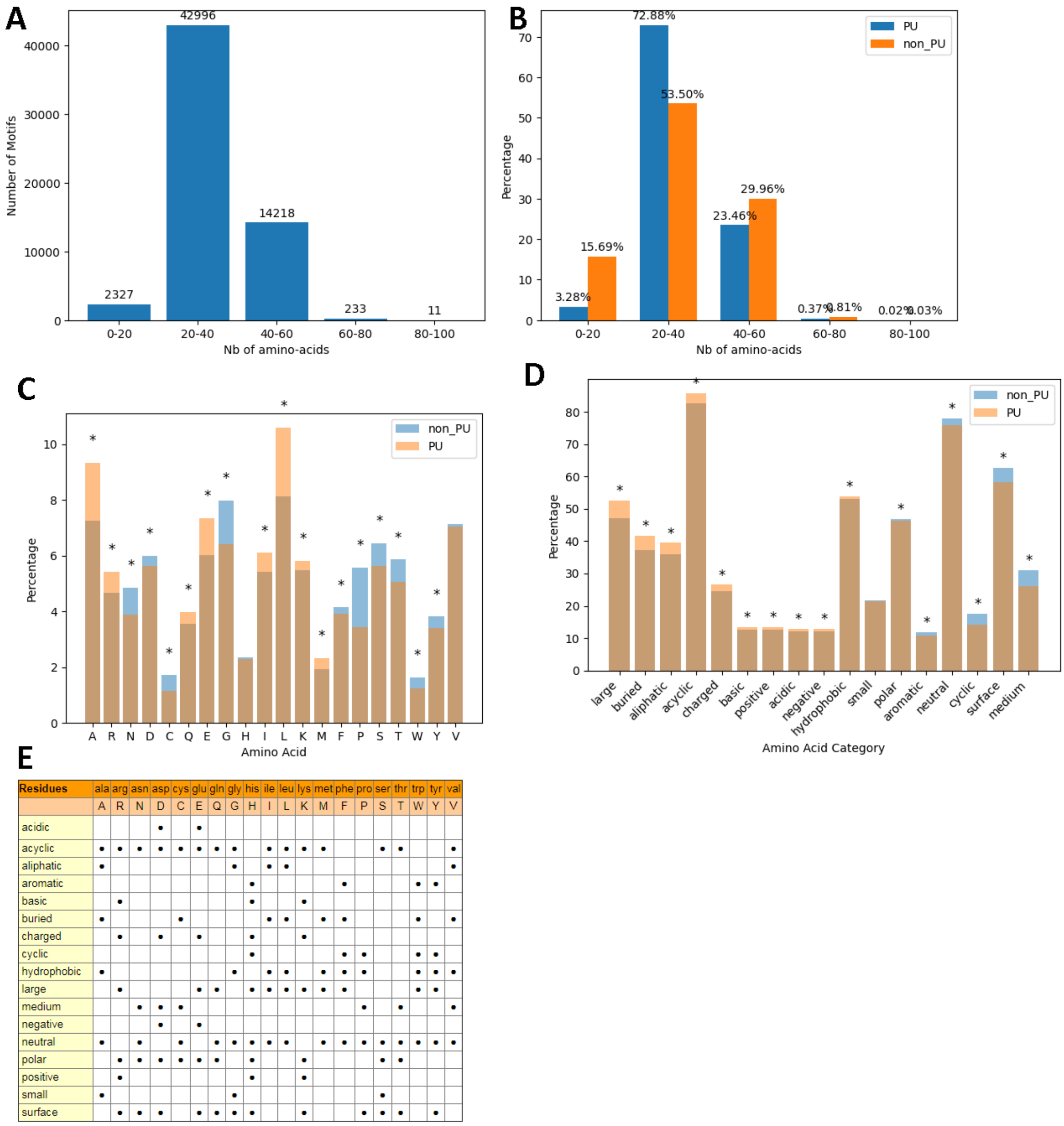
Number of PUs according to categories for each hierarchical level. A, Type. B, Class. C, Architecture. D, Topology. E, NanoFold.

6.4 Appendix 3: number of categories according to the number of Protein Units for each hierarchical level.



Number of categories according to the number of Protein Units for each hierarchical level. A, Architecture. B, Topology. C, NanoFold.

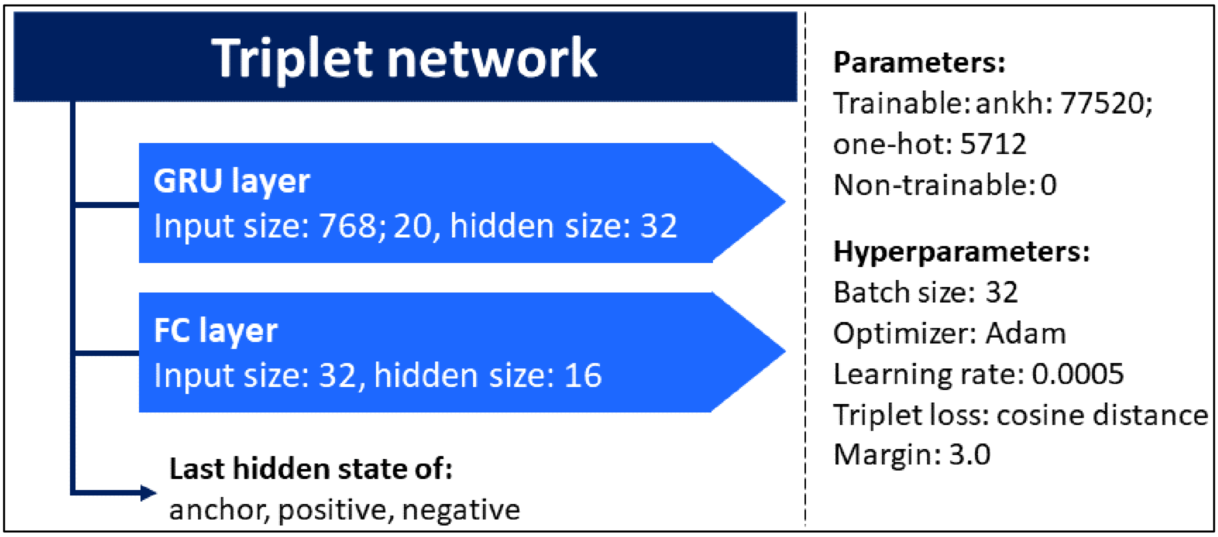
6.5 Appendix 4: length and amino-acid composition analysis of the protein units



Residues	ala	arg	asn	asp	cys	glu	gln	gly	his	ile	leu	lys	met	phe	pro	ser	thr	trp	tyr	val
	A	R	N	D	C	E	Q	G	H	I	L	K	M	F	P	S	T	W	Y	V
acidic				●		●														
acyclic	●	●	●	●	●	●	●	●		●	●	●	●			●	●			●
aliphatic	●							●		●	●									●
aromatic									●					●				●	●	
basic		●							●			●								
buried	●				●					●	●		●	●				●		●
charged		●		●		●			●			●								
cyclic									●					●	●			●	●	
hydrophobic	●							●		●	●		●	●	●			●	●	●
large		●				●	●		●	●	●	●	●	●				●	●	
medium			●	●	●										●		●			●
negative				●		●														
neutral	●		●		●		●	●	●	●	●		●	●	●	●	●	●	●	●
polar		●	●	●	●	●	●		●			●				●	●			
positive		●							●			●								
small	●							●								●				
surface		●	●	●		●	●	●	●			●			●	●	●		●	

A, length distribution of the protein units. B, length distribution displayed in percentage comparing CPUs (PU) and NCPUs (non_PU) according to the number of amino-acids in sequences. C, percentage of amino-acid frequency for CPUs (PU) and NCPUs (non_PU). The star indicates a significant difference between CPUs and NCPU. The statistical test was corrected with Bonferroni. D, percentage of amino-acid category frequency for CPUs (PU) and NCPUs (non_PU). E, amino-acid categories. Source: https://biology.stackexchange.com/questions/71272/reading-an-amino-acid-physicochemical-properties-diagram

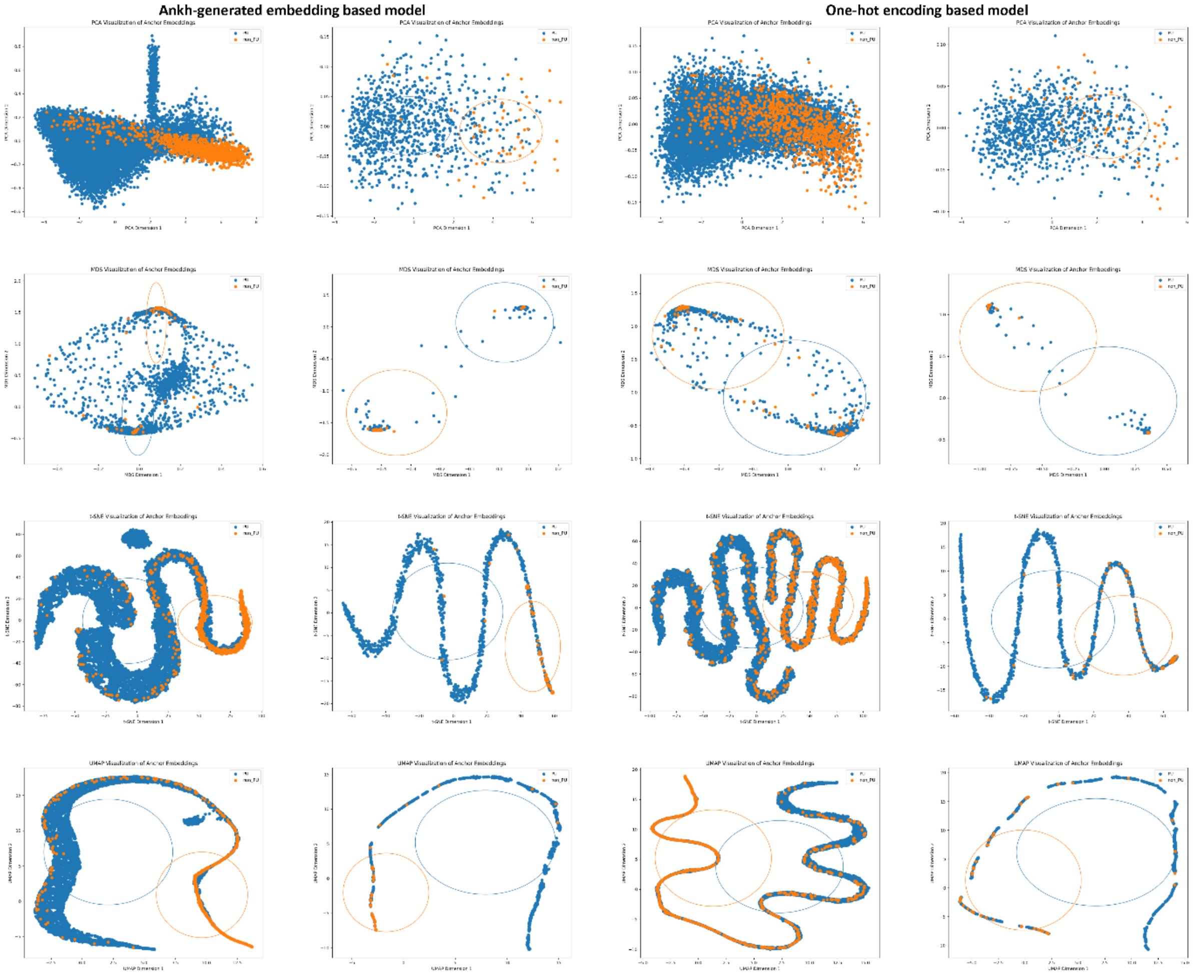
6.6 Appendix 5: model structure and hyperparameter values



It summarizes the model structure and hyperparameters of the 10 models (5 models based on Ankh-generated embeddings and 5 models based on the one-hot encoding) The difference between the two model categories is in the input size of the GRU layer and in the number of trainable parameters. GRU, gated recurrent unit. FC, fully connected.

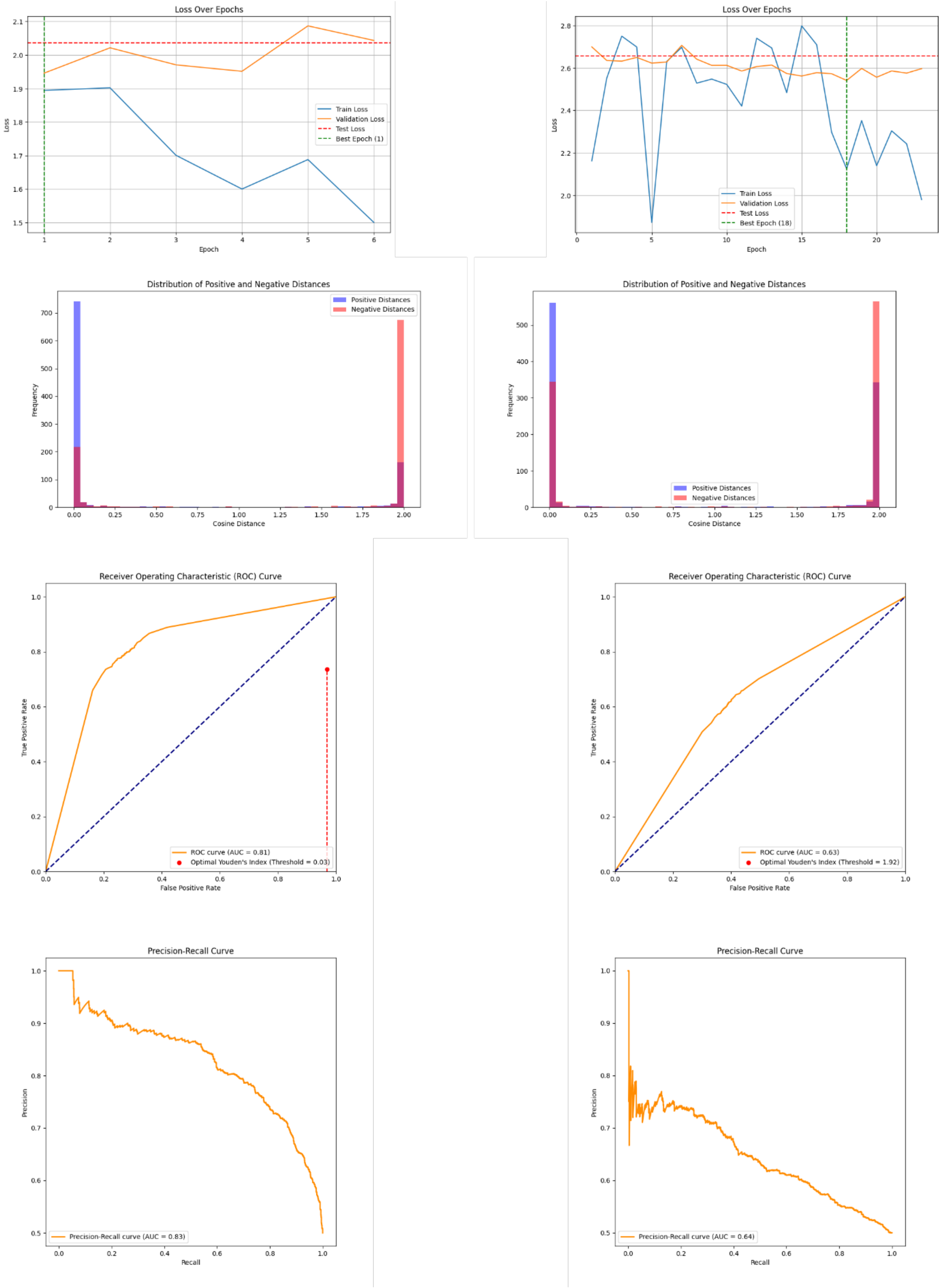
6.7 Appendix 6: supplementary results for models distinguishing CPUs and NCPUs

6.7.1 Appendix 6.A: PCA, MDS, t-SNE and UMAP



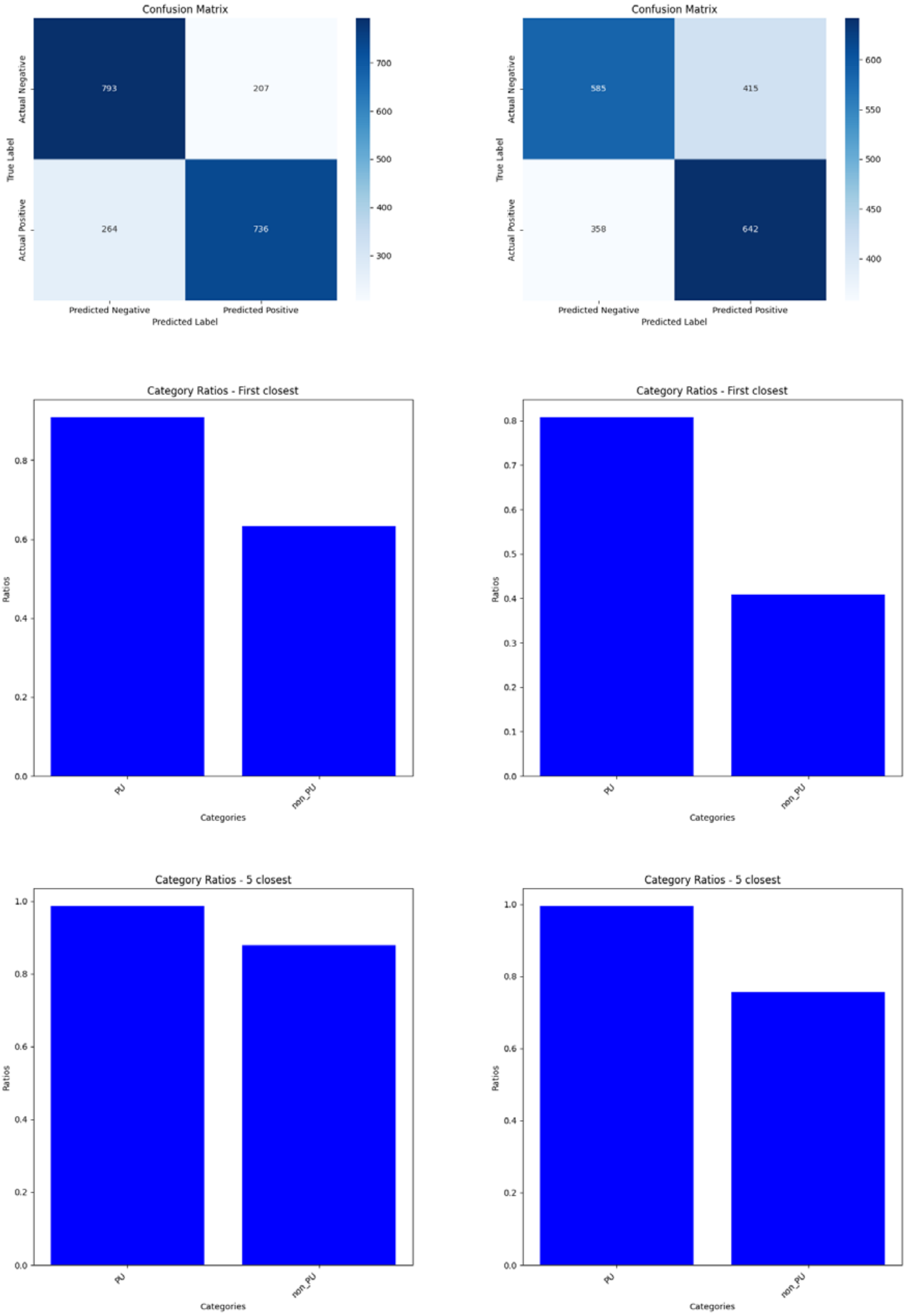
PCA, MDS, t-SNE and UMAP respectively for Ankh-generated embedding based model (left panel) and one-hot encoding based model (right panel). For each panel, results show the plots for the 50% of the train dataset (left sub-panel) and 1000 PUs of the test dataset (right sub-panel).

6.7.2 Appendix 6.B: loss over epochs, distribution of distances, ROC curve, Precision-recall curve



Ankh-generated embedding based model (left panel) and one-hot encoding based model (right panel).

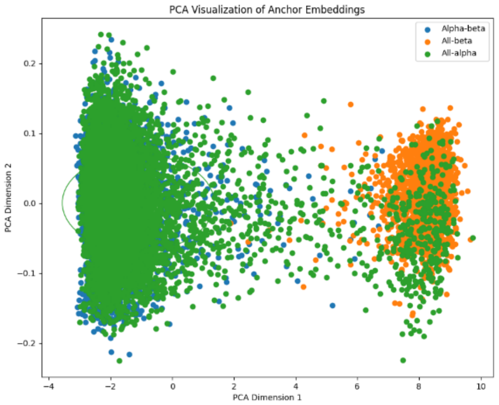
6.7.3 Appendix 6.C: confusion matrix, first closest mode, 5 closest mode

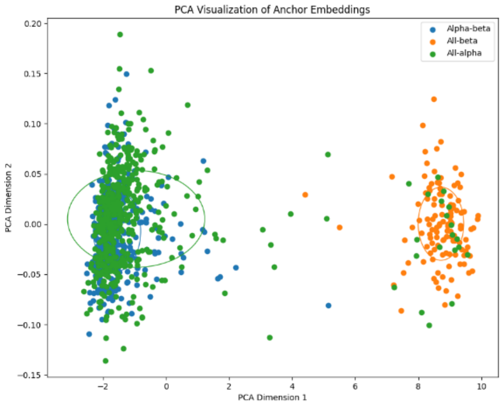


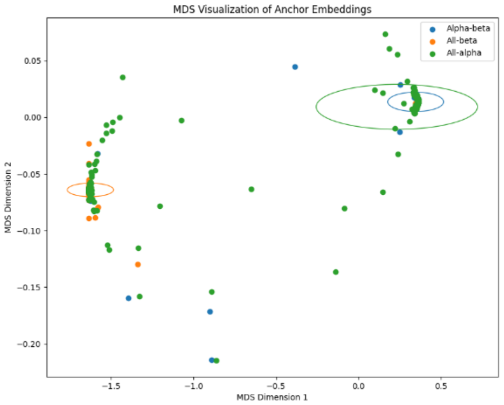
Ankh-generated embedding based model (left panel) and one-hot encoding based model (right panel).

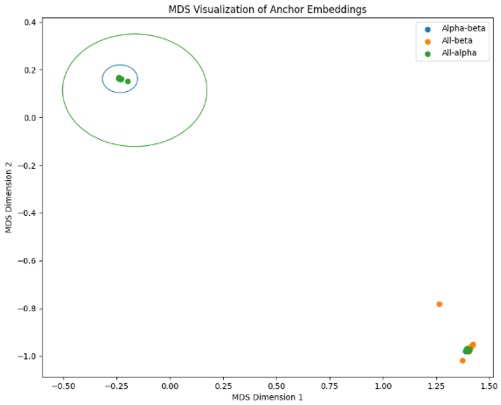
6.8 Appendix 7: supplementary results for models predictingthe category according to “Class” level

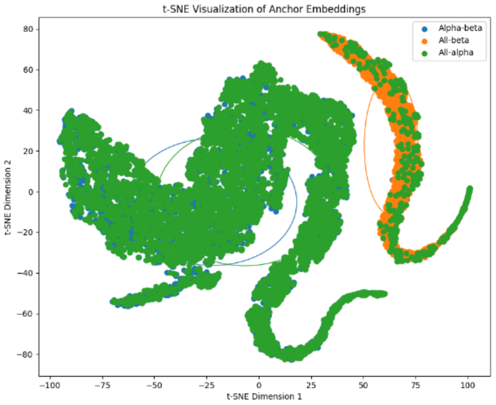
6.8.1 Appendix 7.A: PCA, MDS, t-SNE and UMAP

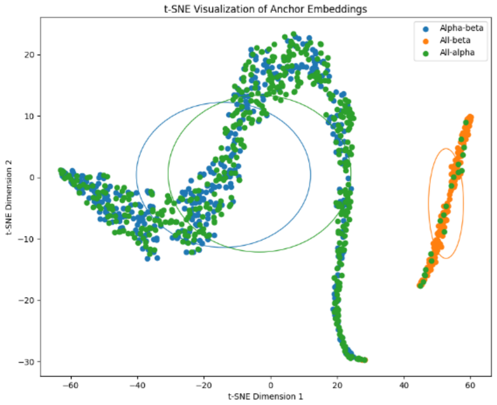

PCA Visualization of Anchor Embeddings
Alpha-beta
All-beta
All-alpha
PCA Dimension 1
PCA Dimension 2
−4
−2
0
2
4
6
8
10
0.2
0.1
0.0
−0.1
−0.2

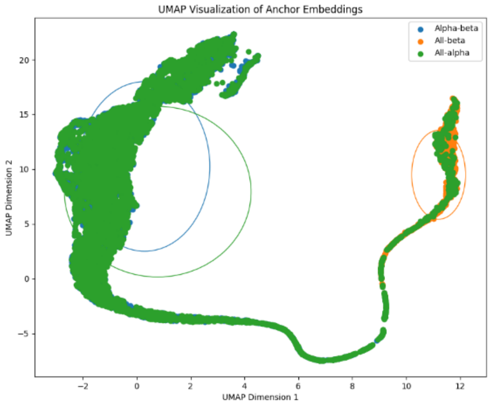

PCA Visualization of Anchor Embeddings
Alpha-beta
All-beta
All-alpha
PCA Dimension 1
PCA Dimension 2
−2
0
2
4
6
8
10
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15

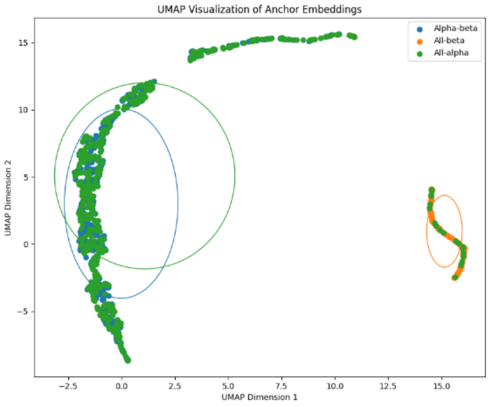

MDS Visualization of Anchor Embeddings
Alpha-beta
All-beta
All-alpha
MDS Dimension 1
MDS Dimension 2
−1.5
−1.0
−0.5
0.0
0.5
0.05
0.00
−0.05
−0.10
−0.15
−0.20


MDS Visualization of Anchor Embeddings
Alpha-beta
All-beta
All-alpha
MDS Dimension 1
MDS Dimension 2
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0


t-SNE Visualization of Anchor Embeddings
Alpha-beta
All-beta
All-alpha
t-SNE Dimension 1
t-SNE Dimension 2
−100
−75
−50
−25
0
25
50
75
100
80
60
40
20
0
−20
−40
−60
−80

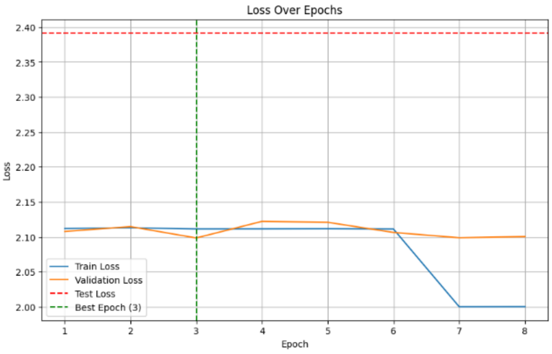

t-SNE Visualization of Anchor Embeddings
Alpha-beta
All-beta
All-alpha
t-SNE Dimension 1
t-SNE Dimension 2
−60
−40
−20
0
20
40
60
20
10
0
−10
−20
−30

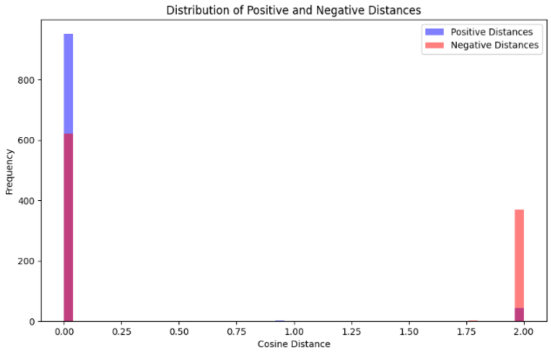

UMAP Visualization of Anchor Embeddings
Alpha-beta
All-beta
All-alpha
UMAP Dimension 1
UMAP Dimension 2
−2
0
2
4
6
8
10
12
20
15
10
5
0
−5

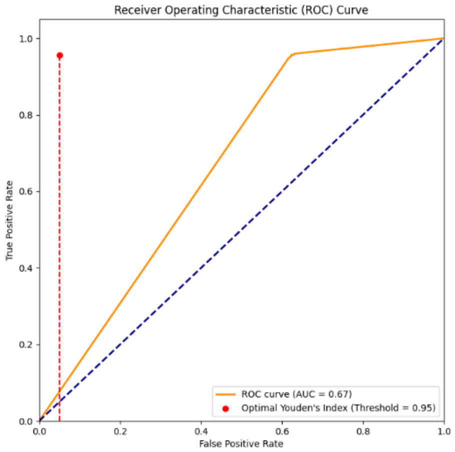

UMAP Visualization of Anchor Embeddings
Alpha-beta
All-beta
All-alpha
UMAP Dimension 1
UMAP Dimension 2
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
15
10
5
0
−5

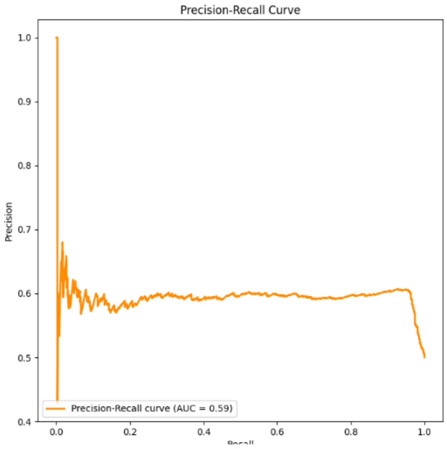
PCA, MDS, t-SNE and UMAP respectively for Ankh-generated embedding based model. Results show the plots for the 50% of the train dataset (left panel) and 1000 PUs of the test dataset (right panel).

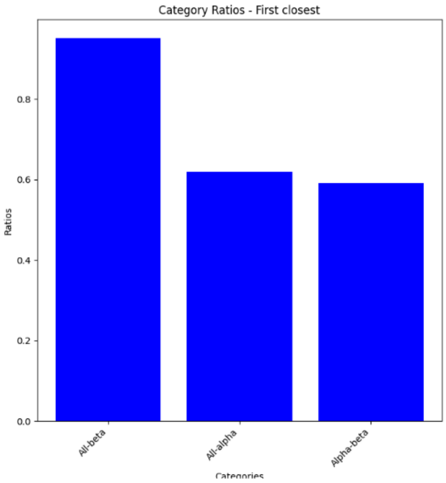
6.8.2 Appendix 7.B: loss over epochs, distribution of distances, ROC curve, Precision-recall curve, confusion matrix, first closest mode, 5 closest mode

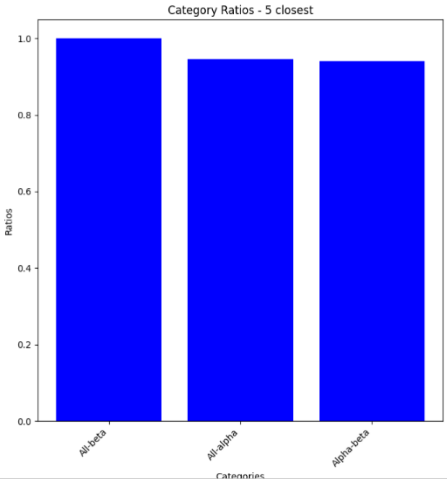

Loss Over Epochs
Loss
2.40
2.35
2.30
2.25
2.20
2.15
2.10
2.05
2.00
Epoch
1
2
3
4
5
6
7
8
Train Loss
Validation Loss
Test Loss
Best Epoch (3)

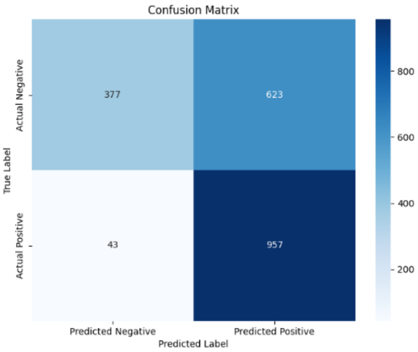

Distribution of Positive and Negative Distances
Positive Distances
Negative Distances
Frequency
800
600
400
200
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Cosine Distance


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (AUC = 0.67)
Optimal Youden's Index (Threshold = 0.95)


Precision-Recall Curve
Precision
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
0.2
0.4
0.6
0.8
1.0
Recall
Precision-Recall curve (AUC = 0.59)


Category Ratios - First closest
Ratios
0.8
0.6
0.4
0.2
0.0
All-beta
All-alpha
Alpha-beta
Categories


Category Ratios - 5 closest
Ratios
1.0
0.8
0.6
0.4
0.2
0.0
All-beta
All-alpha
Alpha-beta
Categories


Confusion Matrix
True Label
Actual Negative
Actual Positive
377
623
43
957
Predicted Negative
Predicted Positive
Predicted Label
800
600
400
200