Advanced Learning for Text and Graph Data (ALTEGRAD) Cellular Component Ontology Classification

Amine ELKARI[1], Mohamed EL-FAKIR[1], Zakaria ECHCHAIR[1] and Marouane NAJID[1]

[1] *Ecole Polytechnique de Paris, IP Paris, Palaiseau, France*

Abstract— In this report, we propose a solution to the Cellular Component Ontology Prediction challenge. Our approach consists of three parts: The first approach involved using a graph neural network (GCN) on the graph created from our dataset, with node features obtained from a pre-trained protein language model (ESM2) applied to the sequence. The second approach involved solely using the pre-trained protein language model (ESM2) on the sequence (purely driven by textual features) and then utilizing diffirent classifiers for classification. The third approach is a combination of both, where we used pre-trained ESM embeddings as inputs for graph-based approaches, such as GATs, GCN, GNN, and GraphSage, along with nodes features. We found that these advanced approaches were more effective for tackling the problem of protein sequence classification. In terms of the data and implementation, we found that the best results were obtained by using data from our pre-trained model and incorporating it into our CatBoost model.

Keywords— Protein, Graphs, GNN, ESM2, Ontoloy, NLP, CATBoost, Pytorch geometric.

I. Introduction

This project aims to apply machine learning and AI techniques to a classification problem in bioinformatics. The focus is on proteins, which are essential biomolecules in living organisms that have various functions such as chemical reactions and structural support. The challenge is to classify 6,111 protein sequences and their corresponding graph structures into 18 different classes based on their location and function using the Cellular Component ontology. The approach is two-fold, using both graph analysis and natural language processing techniques to understand the protein sequences and structures.

The performance of our models will be assessed using the logarithmic loss measure. This metric is defined as the negative log-likelihood of the true class labels given a probabilistic classifier's predictions.

II. Preliminaries

In this section, we start with Graph Representation Learning, followed by setting up our notation and presenting the ideas of message passing neural networks and the main family of GNNs, introducing Neighborhood Sampling in GCNs.

a. Graph Representation Learning :

Graph Representation Learning is a method for integrating information about a graph's structure into a machine learning model. The goal is to encode the structural information of the graph into a low-dimensional vector space (also known as an embedding space) in order to preserve the geometric relationships of the original network. This is achieved by learning a mapping function that embeds nodes or entire subgraphs from the non-euclidean space to the vector space. The key idea is that nodes that are close to each other in the original network should also remain close to each other in the embedding space, while pushing unconnected nodes further apart. This approach allows for end-to-end learning, where features can be learned through a loss function during the training process. The below diagram illustrates the mapping process, where an encoder maps nodes u and v to low-dimensional vectors z_u and z_v. (reference: stanford-cs224w)

b. Notation :

$G = (V, E)$ is an undirected graph consisting of a set of nodes V and a set of edges E. n is the number of nodes in the graph and m is the number of edges. The neighborhood of a node $v \in V$ is denoted by $N(v)$ and the degree of a node v is denoted by $deg(v) = |N(v)|$. The adjacency matrix $A \in \mathbb{R}^{n \times n}$

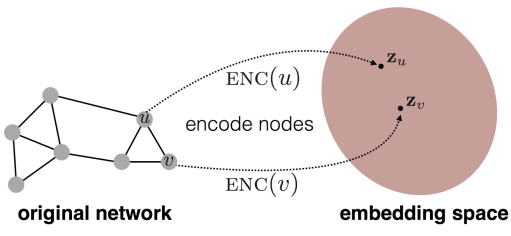

Fig. 1: The mapping process, encoder enc maps node u and v to low-dimensional vector z_u and z_v.

,

of the graph G is a symmetric matrix used to encode edge information, where $A_{i,j}$ is the weight of the edge between the i^{th} and j^{th} node in the graph, or 0 if no such edge exists.

c. Message passing neural networks :

Graph Neural Networks (GNNs) are a class of neural network architectures that are designed to process graph-structured data. These models have gained significant attention in recent years, with the first GNNs being proposed several years ago. The majority of GNNs fall under the category of message passing neural networks (MPNNs). These models use a message passing technique that combines both the graph structure and node features to generate new node representations. Specifically, message passing models iteratively update the representation of each node by aggregating the representations of its neighbors and combining them with the node's previous representation. In other words, message passing GNNs update node feature vectors by aggregating local neighborhood information.

Formally, for a GNN model with T neighborhood aggregation layers, let $\mathbf{h}_v^{(0)}$ denote the initial feature vector of node v, which is the row of matrix X corresponding to node v. In each iteration (t > 0), the hidden state $\mathbf{h}_v^{(t)}$ of node v is updated as follows: $\mathbf{h}_v^{(t)} = \text{agg}\left(\mathbf{h}_v^{(t-1)}, \mathbf{h}_u^{(t-1)} | u \in \mathscr{N}(v)\right)$

Where agg denotes the neighborhood aggregation function and $\mathscr{N}(v)$ is the set of neighbors of node v.

To produce a graph-level representation after T iterations of neighborhood aggregation, GNNs use a permutation invariant readout function such as the sum or mean operator on all the feature vectors of the nodes in the graph. This can be represented as: $\mathbf{h}_G = \text{READOUT}\left(\mathbf{h}_v^{(T)} | v \in \mathscr{V}\right)$

Where readout denotes the readout function (e.g., sum or mean operator) and $\mathscr{V}$ is the set of all nodes in the graph.

d. Neighborhood Sampling :

Lets understand this from the perspective of Graph Convolutional Network diagram (GCNs) described below. GCNs **(Graph Convolutional Networks)** is an algorithm that utilizes both graph topological information, such as a node's neighborhood, and node features to create node representations or dense vector embeddings. The diagram below illustrates the process of GCNs intuitively. On the left side is a sample input graph, with nodes represented by their corresponding feature vectors, such as node degree or text embeddings. The algorithm starts with defining a search depth (K), which determines how much information should be gathered from the neighborhood of a target node. K is a hyperparameter and it also represents the number of layers used in the GCNs.

GCNs starts by initializing all node embeddings to their original feature vectors at K=0. To compute the embeddings for a target node, such as node 0, at layer K=1, GCNs aggregates the feature vectors of all nodes that are at a distance of 1-hop from the target node. For example, at this layer GCNs aggregates the original feature representations of nodes at K=0. GCNs uses a permutation invariant function such as mean aggregator to compute the mean of the neighborhood node features, including the target node's own feature (self-loop). After K=1, the target node now has information about its immediate neighborhood. This process is repeated for all nodes in the graph in order to find new representations for each node at each layer. This is shown in the GCNs image on the right side.

Note: As the search depth increases, the reach of the target node in terms of aggregating features from its local neighborhood also increases. For e.g. at K=1 the target node knows the information about its local neighborhood which are 1-hop distance, at K=2 the target node knows the information about its local neighborhood which are at 1-hop distance and the neighbors of the nodes of 1-hop distance i.e upto 2-hop distance.

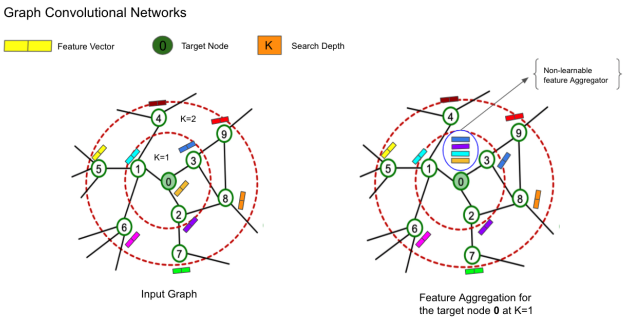

Fig. 2: Diagram illustrates the process of GCNs intuitively.

III. Data Exploration

The dataset used in this study consists of 6,111 proteins, represented as undirected graphs. The protein sequences are provided in the "sequences.txt" file, with each line representing the sequence of a single protein. The edges of all proteins are listed in the "edgelist.txt" file, with each line corresponding to an edge represented by the ids of its endpoints. There are a total of 15,213,222 edges in the dataset.

Additionally, edge attributes are provided in the "edge attributes.txt" file, with each line storing the attributes of a single edge. There are a total of 5 attributes, including the distance between the two connected nodes and binary indicators for whether the edge is a distance-based edge, peptide bond edge, k-NN edge, or hydrogen bond edge.

Node attributes are provided in the "node attributes.txt" file, with each line storing the attributes of a single node. There are a total of 86 attributes, including 3D coordinates of the node, one hot encoding of the amino acid type, hydrogen bond acceptor and donor status, and amino acid features derived from the EXPASY protein scale. There are 1,572,264 nodes in the 6,111 proteins.

Graph indicators are provided in the "graph indicator.txt" file, with the value in each line denoting the graph to which the node with that id belongs. The "graph labels.txt" file contains the names of all proteins along with the class labels of those proteins that belong to the training set. The proteins for which the class label is not available belong to the test set, and the goal of this study is to predict the class label of each one of these proteins.

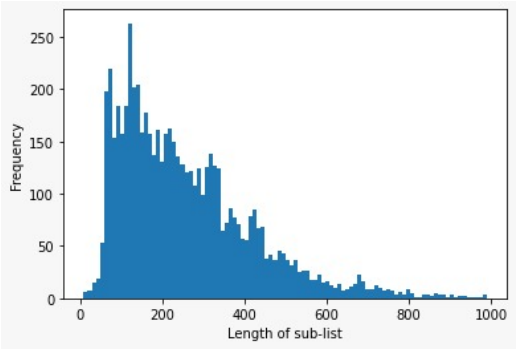

Fig. 3: Distribution of sequence lengths in order to define the input size for models such as transformers that will maintain the semantic relationships between different amino acids.

IV. METHODOLOGY

This section aims to detail the methodology employed in this study to address the problem of protein classification based on graph representations. Firstly, we conduct a literature review to survey the existing methods and their limitations in this field. Following that, we propose three frameworks for protein classification that build upon the concepts of NLP, graph convolutional networks and graph attention mechanisms. These frameworks are designed to effectively capture the structural information of proteins and improve classification performance. The effectiveness of our proposed frameworks will be demonstrated through experiments on a dataset of 6,111 proteins, and their results will be compared to the state-of-the-art methods.

a. State of the art :

A study [1] that was conducted proposed using Language Models (LMs) from Natural Language Processing (NLP) to classify proteins. The team trained several LMs (ProtBert) on a large dataset containing 393 billion amino acids from 2.1 billion protein sequences, using a supercomputer with multiple GPUs. The trained models were able to predict secondary structure, sub-cellular localization, and protein solubility with high accuracy. The team also found that the LMs learned important biophysical properties from the protein sequences. The study demonstrated the success of using large data sets and high-performance computing for training LMs for protein classification, which reduced the gap between models trained on evolutionary information and LMs.

A recent study [2] proposed a two-step approach for protein classification based on graph representations. The approach utilized the AlphaFold model, a well-established model in the field, to predict the 3D structure of a protein from its amino acid sequence in the first step. This step aimed to capture the structural information of the protein, which is crucial for accurate classification. In the second step, the sequence was processed using a transformer-based protein language model and a graph neural network was applied to the graph extracted from the structure. The approach aimed to capture the contextual information of the protein, which can also contribute to the accuracy of the classification. The proposed architecture was evaluated on a standard benchmark dataset and it was found to outperform state-of-the-art methods in terms of classification performance.

A typical approach for classifying protein sequences would involve using a combination of different techniques, such as LSTM [3], ProtCNN [4], word2vec [5], fasttext, and TF-IDF for feature extraction, and models such as XGBoost, SVM, and logistic regression for classification. The LSTM (Long Short-term Memory) is a type of Recurrent Neural Network (RNN) that is particularly well-suited for sequential data such as protein sequences. ProtCNN (Protein Convolutional Neural Network) is a type of convolutional neural network that is effective in recognizing patterns in protein sequences. Word2Vec and Fasttext [6] are word embedding techniques that can be used to convert amino acid sequences into numerical vectors. TF-IDF (Term Frequency-Inverse Document Frequency) is a technique that can be used to weight the importance of amino acids in a protein sequence. Finally, XGBoost, SVM, and logistic regression are models that can be used for classification.

b. Approach :

In this study, we proposed three different approaches to tackle the problem of protein classification. The first approach is based on natural language processing (NLP) and utilizes only the protein sequences as input. This approach aims to capture the contextual information of the protein by processing the sequence using various NLP techniques such as word embedding and language modeling. The second approach is based on graph representation of proteins, this approach aims to capture the structural information of the protein by representing it as a graph and applying various graph neural network models. The third approach is a combination of the first two approaches, where we combine the sequence information with the graph representation of proteins to exploit both the contextual and structural information of the proteins. This approach aims to improve the classification performance by leveraging the strengths of both NLP and graph representation techniques.

1. NLP Approach :

The first approach we proposed in this study is based on natural language processing (NLP) and utilizes only the protein sequences as input. For this approach, we used different embedding methods such as Fasttext and TF-IDF to convert the amino acid sequences into numerical vectors. Additionally, we also used dimensionality reduction techniques such as PCA (Principal Component Analysis) and Auto-Encoder to reduce the dimension of the embeddings. The goal of these techniques is to capture the most important features of the embeddings while reducing the noise.

We then fit these embeddings to various classifiers such as XGBoost, SVM, and logistic regression to classify the proteins. The goal of this approach is to capture the contextual information of the protein by processing the sequence using these embedding methods and then classify the proteins based on their sequence information. The use of dimensionality reduction techniques enhances the performance of the classifiers and reduces the computational cost.

We also employed a LSTM-based approach to classify protein sequences. We used a Long Short-Term Memory (LSTM) network, a type of Recurrent Neural Network

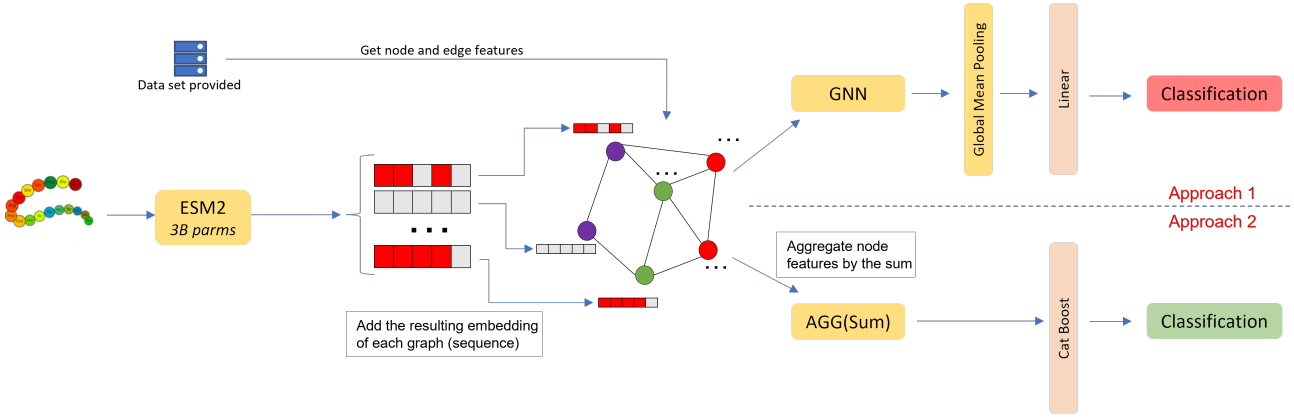

Fig. 4: High-level overview of the proposed framework for classifying the different sequences. The framework consists of two main approaches : The first approach involves using a graph neural network (GCN) on the graph created from our dataset, with node features obtained from a pre-trained protein language model (ESM2) applied to the sequence. The second approach involves solely using the pre-trained protein language model (ESM2) on the sequence and then utilizing Catboost for classification.

(RNN) that is able to capture long-term dependencies in sequential data. We trained the LSTM on our dataset of protein sequences and used it to make predictions about the class of each sequence.

Additionally, we also used a pre-trained embedding model (ProtBERT) to extract embeddings from the protein sequences and fine-tuned it by adding a head of classification to make predictions about the class of each sequence. ProtBERT is a pretrained model on protein sequences using a masked language modeling objective. It is based on the BERT model, which is pretrained on a large corpus of protein sequences in a self-supervised fashion, meaning it was pretrained on raw protein sequences only with no human labeling. This allows it to use a lot of publicly available data and have an automatic process to generate inputs and labels from those protein sequences.

We also trained a ProtCNN model from scratch which uses residual blocks inspired by ResNet architecture which also includes dilated convolutions offering a larger receptive field without increasing the number of model parameters. The amino acids sequences are converted to one-hot encoding with shape (batch size, 100, 21) as input to the model. The initial convolution operation is applied to the input with a kernel size of 1 to extract basic properties. Then two identical residual blocks are used to capture complex patterns in the data which are inspired by the ResNet architecture, this will help us to train the model with more number of epochs and with better model performance. In addition, we also trained a transformer following this architectureers (8 self.attentions layers and 4 heads of attention with an input of size 512), the goal of this approach is to capture the contextual information of the protein by processing the sequence using these models and then classify the proteins based on their sequence information.

On another hand, we used the ESM (Embedding from Language Models) which is a transformer-based language model that uses an attention mechanism to learn interaction patterns between pairs of amino acids in the input sequence. The ESM-2 model is trained on large amounts of protein sequences data to learn the correlations between different sites in a protein, which can be used to predict missing amino acids and ultimately, the protein's 3D structure. The ESM-2 model's performance improves as the scale of the model increases, as it can learn more complex interactions between amino acids. We utilized the final hidden layer of a pre-trained model and combined it with an ensemble estimator (CatBoost) for classification purposes. Additionally, we attempted to improve the model's performance by fine-tuning it with an additional classification head.

2. Graph Approach :

The second approach we proposed in this study is based on graph representation of proteins, this approach aims to capture the structural information of the protein by representing it as a graph and applying various graph neural network models. In this approach, we used node attributes and edge attributes to classify proteins. We used Graph Attention Networks (GATs) which is a specific type of graph neural network that can handle graph-structured data by using self-attention mechanism. Additionally, we also used other graph neural network models such as Graph Convolutional Networks (GCN), general Graph Neural Networks (GNN) and GraphSAGE. However, for the GraphSAGE model, we faced some implementation complications with the neighbor data loader, which may have affected the results obtained from this model.

These models are well-suited for this task as they are designed to handle graph-structured data and can capture the structural information of the protein. The goal of this approach is to capture the structural information of the protein by representing it as a graph and apply these models to classify the proteins based on the structural information. We used the information from both nodes and edges attributes as input to these models and the goal is to improve the performance by leveraging the graph structure information of the proteins.

3. Mixture Approach :

The third approach we proposed in this study is a combination of the first two approaches, where we use both protein sequences and graph structure information to classify proteins. In this approach, we used a pre-trained embedding model (ESM) with different versions in terms of architecture and number of parameters to extract embeddings from the

protein sequences. Specifically, we extracted the last layer of the ESM, as it is believed to contain the most separable representation of the amino acids. We then concatenated these embeddings with the node embeddings to feed them to the graph neural network models used in the second approach. Furthermore, we added some techniques to improve the performance such as mean pooling layer and dropout layers with a probability of 0.2 to prevent overfitting. Moreover, we chose to use 2 layers for the Graph Convolutional Network (GCN) model based on research that suggests that 2 layers is the best architecture for GCN and an increase in the number of layers deteriorates the performance. This approach aims to capture both the sequence and structural information of the proteins and by combining them we expect to improve the performance of the classification.

V. Evaluation

After evaluating our different approaches for classifying protein sequences, we found that using simple embedding methods such as tf-idf and fasttext, and fitting them to traditional classifiers like xgboost, svm, and logistic regression, were not sufficient to achieve good results. Therefore, we decided to explore more advanced approaches, such as training a transformer from scratch, using ProtCNN to extract features from the sequences, and utilizing pre-trained embedding models like ESM. We found that incorporating ESM embeddings as inputs for graph-based approaches, such as GAT, GCN, GNN, and GraphSage, along with nodes features, improved performance. Additionally, we also tried fine-tuning ESM2 by adding a classification head or using only embeddings as input for an ensemble classifier like CatBoost, which performed well. Overall, we found that these advanced approaches were more effective for tackling the problem of protein sequence classification.

In our training process, we utilized the Adam optimizer with a learning rate of $1.e^{-4}$ to minimize the multi-class log loss. The model we trained is a combination of the ESM2 model and the GNN model on a specific dataset. We experimented with different message passing GNN architectures, including GCN, GAT, and GraphSAGE, and found that GCN performed the best in our setting for the first approach. Although GCN is known to have multiple layers, we choose to use only 2 layers for GCN based on our research discussions, as we found that GCN generally reaches its best performance with only two layers. While for the other GNNs we implemented 3 hidden layers. To prevent overfitting, we applied a dropout rate of 0.2 and used a batch size of 32. Our GNN had 2 hidden layers for GCN and 3 hidden layers for the other GNNs with a hidden dimension of 64. We also used dropout, following by layer and elu activation of the GNN.

As can be seen in the table 1 below, our best results were achieved by using the ESM2 with 3 billion parameters and the Cat boost algorithm.

1. Evalution metrics

In evaluating our model, we primarily rely on two metrics: accuracy and logarithmic loss function. The logarithmic loss function, also known as log-likelihood, measures the performance of a probabilistic classifier in predicting the true class labels. It is defined as the negative log-likelihood of the predicted class labels.

Specifically, the multi-class log loss is defined as :

$$\mathscr{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log\left(p_{ij}\right)$$

2. Results

Table 1: Classification results of the diffrents approaches and models on the data set provided. The best performance in typest in **bold**.

Models	ACC Train%	Loss Train	ACC Val%	Loss Val
TF-IDF + SVM	81	0.29	56	1.32
ProtCNN	92	0.62	44	1.42
Transformer	68	0.67	42	1.80
ESM + GCN	79	0.55	70	1.07
ESM2 + Catboost	82	0.40	75	0.75
ProtBert	75	0.49	67	1.43
Lstm	58	1.53	42	1.98

VI. Setup configuration

In order to implement our solution, we utilized a cloud-based configuration that included a powerful 16 GB GPU and 40 GB of RAM. This allowed us to efficiently run our GNN models and process large amounts of data. We chose to use the PyTorch Geometric library for implementing GNNs, as it is a powerful and easy-to-use tool that provides a wide range of functionalities for working with graph-structured data. This library also provides a number of pre-built GNN models, which we could use as a starting point for our own implementation.

VII. Conclusion

In summary, this study aimed to address the problem of protein classification using graph representations. We conducted a literature review to survey existing methods and their limitations in this field. Then we proposed three frameworks that utilize concepts from NLP, graph convolutional networks and graph attention mechanisms to effectively capture the structural information of proteins and improve classification performance. The effectiveness of these framworks was demonstrated through experiments on a dataset of 6,111 proteins, and their results were compared to state-of-the-art methods. Overall, the study showed that utilizing large data sets, high-performance computing, and a combination of different techniques and models can lead to improved protein classification performance.

The code for this project can be found on the following Github repository: [7]

REFERENCES

[1] A. Elnaggar, M. Heinzinger, C. Dallago, G. Rihawi, Y. Wang, L. Jones, T. Gibbs, T. Feher, C. Angerer, D. Bhowmik, and B. Rost, "Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing," *bioRxiv*, 2020. [Online]. Available: https://www.biorxiv.org/content/early/2020/07/12/2020.07.12.199554

[2] A. Qabel, S. Ennadir, G. Nikolentzos, J. F. Lutzeyer, M. Chatzianastasis, H. Boström, and M. Vazirgiannis, "Structure-aware antibiotic resistance classification using graph neural networks," *bioRxiv*, 2022. [Online]. Available: https://www.biorxiv.org/content/early/2022/10/08/2022.10.06.511103

[3] R. C. Staudemeyer and E. R. Morris, "Understanding lstm – a tutorial into long short-term memory recurrent neural networks," 2019. [Online]. Available: https://arxiv.org/abs/1909.09586

[4] M. L. Bileschi, D. Belanger, D. Bryant, T. Sanderson, B. Carter, D. Sculley, M. A. DePristo, and L. J. Colwell, "Using deep learning to annotate the protein universe," *bioRxiv*, 2019. [Online]. Available: https://www.biorxiv.org/content/early/2019/05/06/626507

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[6] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," 2016. [Online]. Available: https://arxiv.org/abs/1607.04606

[7] [Online]. Available: https://github.com/Amine-elk/Cellular-Component-Ontology-Prediction