CS 7643 Final Project Report: Node Classification on Protein-Protein Interactions for Applications in Biochemistry

Jennifer Becker
jbecker68@gatech.edu

Raka Islam
rislam@gatech.edu

Weichen Wang
wwang674@gatech.edu

Yi Shuen Lim
ylim68@gatech.edu

Abstract

The purpose of this project is to explore and advance Graph Neural Network (GNN) architectures to make multi-label predictions on the nodes of Protein-Protein Interactions (PPIs). By testing existing GNN layer types in different architectures and comparing against benchmarks, our study found that a combination of Molecular Fingerprint Convolution (MFConv) and Graph Isomorphism Network (GIN) layers produced our best F-1 scores across validation and test data. This impressive result is analyzed from the theoretical perspective on both architectures in terms of discriminating power.

1. Introduction

PPIs are essential for many biological processes, including cell signaling, metabolism, and gene regulation. Understanding these interactions is crucial for developing new drugs and treatments. PPI networks can also be used to discover disease pathways – sets of proteins that interact with each other and contribute to the development of a disease [1]. By identifying these pathways, we can gain insights into the molecular mechanisms of diseases such as cancer, and develop new treatments.

In this project, we explored different kinds of graph neural networks (GNNs) in the context of multi-label prediction on protein nodes within protein networks [18] [17] [16] by looking at different capabilities of GNNs for multi-label node classification on PPI datasets. Specifically, we wanted to compare the performance of different GNN architectures on the PPI dataset and identify the factors that contribute to the performance of GNNs on this dataset.

Currently, PPIs are typically analyzed using traditional machine learning methods, such as Support Vector Machines and Random Forests [11]. However, these methods are not best-suited for the complex structure of PPI networks. GNNs, on the other hand, are specifically designed to learn from graph-structured data, making them a promising approach for analyzing PPIs [9].

The main limitation of current GNN methods is that they are computationally expensive to train as they need to learn a large number of parameters, which can be time-consuming and require a lot of memory.

Our work has the potential to make a significant difference in the field of drug discovery and development [19]. A deeper understanding of PPIs can lead to the discovery of new drug targets, personalized treatments, and more effective therapies for complex diseases like cancer and neurological disorders.

If we can better understand how PPIs contribute to the development of cancer for example, we can develop drugs that target these interactions and prevent cancer cells from growing. By using GNNs to analyze PPI data, we can accelerate drug development and disease treatment.

2. Dataset

For this project, we used the PPI dataset, which contains undirected graphs corresponding to different human tissues. Nodes each represent a protein while edges represent interactions between proteins. The dataset includes 295 total disconnected graphs, of which 24 consist of more than 2 nodes. Across the 24 larger graphs, the sizes range from 591 nodes to 3480 nodes, making up 56,658 nodes in total. We left out 2 graphs each for validation and testing sets. Each node also has 50 binary features, and the goal is to make predictions on each node's 121 binary labels.

Over the 121 binary labels, there is a 69.5-30.5 split on outcome if we take the mean class balance. The most imbalanced labels have a 91-9 and 11-89 class split. Furthermore, of each nodes' 50 binary features, the value distribution of 0s and 1s ranges from 99.5-0.5 to 85-15. On average, each feature has 2 percent of its values as 1.

This dataset has been used in several studies to benchmark the performance of GNNs. Table 1 summarizes the benchmark F-1 scores of the relevant architectures used in this project.

3. Approach

To study model effectiveness for learning protein interactions in gene ontology, we trained and tested multiple

Architecture	Benchmark F-1 Score
Cluster-GCN (2019)	99.36
GraphSage (2017)	61.2
GAT (2017)	97.3

Table 1: Benchmark F-1 Scores on PPI dataset.

GNN architectures that have existing benchmarks for the PPI dataset. Namely, Cluster Graph Convolutional Network (CGCN), GraphSAGE and Graph Attention Network (GAT) layers. Additionally, we tested Graph Isomorphism Network (GIN) and Molecular Fingerprint (MFConv) layers, which do not have existing benchmarks.

In our very first attempt, we added multiple Graph layers at the head of every model and saw sub-optimal results. We initially encountered an "over-smoothing" problem, which is a common issue in Graph Networks [4]. Some detailed explanation on this problem will be discussed later. Based on "six degrees of separation", we know that a node can reach every other node in a graph with enough aggregation. Thus in practice, when GNNs grow too deep, every node's feature representation aggregates to learn the same effective feature space over time. This is in contrast to convolutional layers in image classification, where deeper nets can capture larger receptive fields and learn from relationships between different blocks of the image.

To combat this, we added several linear layers after the Graph layers, leading to immediate improvement. The Graph layers were able to extract nodes' information from itself and its neighbors, while the fully-connected linear layers with activation functions allowed for better learning for the prediction task. Our findings, as well as architecture-specific tuning steps, issues and solutions, are discussed in more detail in the following sections. To maintain consistency between experiments for comparison between models, the experiments were fixed in the following for all GNN implementations:

- The models were run on a CUDA-enabled 8GB Nvidia RTX-3070 GPU.
- Since the problem is a multi-label classification, we chose F1-score as our primary performance metric due to the imbalanced nature of the dataset's labels, but also observed MCC and AUROC.
- We compiled and trained the model on batches of graph nodes set aside for training with a batch size of 8, chosen to maximize performance while minimizing runtime and CUDA memory size. We then tuned several hyperparameters based on the validation learning curve, and finally ran the model on the unseen test data and used that final F-1 score to compare against benchmarks.
- Our models were written with Pytorch Geometric [14] using only convolution layers implemented in this package.
- The data was loaded as a preprocessed version of the PPI dataset from Pytorch Geometric to maintain a valid comparison against the benchmark models in Table 1.
- The models were tested using learning rates 1e-2, 1e-3, 5e-3 and 1e-4 but had the best performance at 1e-3 while still maintaining time efficiency, so we fixed this hyperparameter for all models.
- We tested the models at 5, 10, 100, multiples of 1000s of epochs before landing on 8000 epochs for all models to maintain time and space constraints. We also found that this also allowed enough training for the models to generalize effectively and exhibit asymptotic performance behavior in the loss curves for both the training and validation data without overfitting.
- We tested multiple loss functions implemented with the Pytorch package (i.e. Binary Cross Entropy Loss, Multi-Label Margin Loss) before landing on using Cross Entropy loss for all models due to its high classification performance.
- We used batch gradient descent to train the model, which sped up the training process as smaller subsets of the data were used to update gradients [2]. However, because the full training data was not used to update gradients all at once, we see spikes in our training loss curves.
- We used an Adam optimizer implemented with Pytorch due to its fast computation time and to minimize the need for aggressive optimizer hyperparameter tuning.

4. Experiments

This section covers the different layers and architectures tested in our study. Our highest F-1 score achieved across all models was 0.951, beating several benchmarks. This study overall increased our understanding of Graph Networks' architectural design and the techniques used for node representation and computation. All layers tested have different methods of capturing the relationships between the features of nodes and their neighbors. The following subsections explain the intuition behind these architectures, in terms of discriminating power and common usages and issues, to provide a theoretical basis on the future design of Graph Networks.

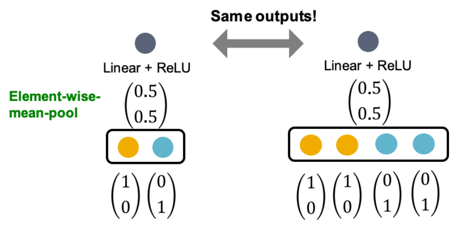
Same outputs!
Linear + ReLU
Linear + ReLU
Element-wise-mean-pool
$\begin{pmatrix}0.5\\0.5\end{pmatrix}$
$\begin{pmatrix}0.5\\0.5\end{pmatrix}$
$\begin{pmatrix}1\\0\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}$
$\begin{pmatrix}1\\0\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}$

Figure 1: Naive GCN fail case

4.1. Naive Graph Convolution Network (GCN)

The naive GCN design follows two key steps: 1.message, 2.aggregation. When feeding each node into the GCN layer, if we look at individual node feature only, we would lose valuable neighboring nodes' feature and those edge connection information will also be missing. It is as if we are feeding isolated node into the GNN. Therefore, the naive GCN will incorporate the neighbor's information by passing their features and aggregating them into the node. In this way, we keep the relationship among the nodes by taking summation of all the nodes connected [6].

Naive version of GCN also suffers significantly drawbacks. For GCN fail case (see Figure 1), for a particular node, if the adjacent nodes contains the same percentage of two different nodes, then the element-wise mean calculation of the nodes feature will produce the same node representation and thus the GCN cannot distinguish two nodes [13]. Therefore, one thing to note is that if we add too many GCN layers into the network, the node representation in the final GCN layer will be the same, regardless of how different the input nodes are. This creates a huge hurdle for the model to learn because the homogeneous input representation makes prediction task extremely difficult (”over-smoothing”).

Even though this Naive GCN does not work well, it indeed provide a important guideline for the following different variations of GCN layer's design. That is, we need to find someway to represent the node and also consolidate neighboring nodes' information. In our experiment, naive GCN is only used in our initial understanding of GNN and thus the performance is not reported here. It is safe to assume naive GCN has the lowest performance on F1 score.

4.2. Cluster GCN

Cluster GCN's idea is to split the graph into k non-overlapping subgraphs [3]. By doing this, the neighbors feature aggregation is reduced and thus in some degree, it increases the discriminating power of the convolutional layer because now the node does not connect to too many neighbors. Only nodes that associate with a dense sub-graph are considered. In this way, even with more layers, the over-smoothing issue will not be so severe as before.

In our experiment, Cluster GCN does perform better with F1 score 0.88 and it boosted the performance way higher than naive GCN. The reason being that, even after too many layers of cluster GCN, the input node representations are still different. However, our Cluster GCN's performance is lower than the paper's reported F1 score of almost 0.9936. Aside from the fact that the hardware is more advanced and training time is longer, the most important difference lies on the deeper layer of Cluster GCN. In our experiment, at most 2 layers has been applied to the PPI dataset while in the paper, the authors used up to 6-layer of the Cluster GCN layers. One thing to note is that as its name suggested, the clustering algorithm is applied to the nodes, and thus this algorithm also consumes more memory for distance calculation in memory, as well as the time taken.

4.3. GraphSAGE

GraphSAGE is a layer that was introduced in 2017 by Hamilton et al. [8], lauded for its ability to generalize well to unseen nodes after only being trained on a subset of a graph. In essence, it aggregates information from a node's nearest neighbors to compute its vector representation [10].

Prior to implementation, the intuition for picking GraphSAGE is that it would work for protein node prediction if a protein's 121 classes are in some way affected by the class labels and feature makeup of the proteins surrounding it. However, a shortfall would be if adjacent nodes contain the same set of different nodes, then any element-wise max pooling calculation of the nodes' features will produce the same node representation and thus the network would not be able to distinguish between two nodes. This is similar to the smoothing problem discussed prior.

The first implementation of the GraphSAGE model began with three consecutive SAGE layers with default hyperparameters followed by two linear layers with ReLU activations in between. The reference study ran their model for 10 epochs to achieve a 0.612 F-1 score; and this model at 10 epochs was only able to achieve 0.504. Although, this was without tuning any of the PyTorch Geometric layers' default hyperparameters.

Hyperparameter tuning was done on runs of 500 epochs. Different aggregation types within each SAGE layer were tested, and the best results were achieved with max aggregation instead of the default mean. Additionally, adding pooling layers in between SAGE layers was tested, a common technique with other geometric and convolutional layers to prevent overfitting, but this only seemed to hurt test F1 scores. This is probably because aggregation is already being performed as part of the process of creating the representations of nodes within the SAGE layers. At 500 epochs, the model's performance achieve about 0.752 F-1 score, and finally achieved and plateaued out around 0.819 F-1 score when run for 8,000 epochs. This is perhaps near the limit of what the model can learn using information from proteins' surrounding nodes. As we will see in our comparison analysis, the SAGE model ran relatively fast compared to other models, because of its ability to estimate nodes' representations from their neighbors.

4.4. Graph Attention Network

Graph neural network is a powerful neural network architecture to learn the importance of different neighbors when aggregating node features. The key idea behind GAT is to compute attention coefficients for each node's neighbors based on their feature similarities. The model was developed by Velickovic et al [15].

In GAT, the attention mechanism is applied to the messages passed between nodes. The attention weights are learned by a neural network that takes as input the features of the nodes and the edges between them. The attention weights determine how much weight each node's message should have when aggregating the features of its neighbors.

During our experiments, we compared different GAT architectures with varying numbers of attention heads and hidden layer sizes. We found that deeper architectures with more attention heads tended to perform better, as they could capture more intricate interactions between nodes. However, we had to be cautious about the risk of oversmoothing, which can occur when the node representations become too similar due to extensive aggregation. To overcome these challenges and improve our GAT model's efficiency, We used a trial-and-error approach. We incorporated dropout regularization and batch normalization into the GAT architecture, which helped to prevent oversmoothing. We also deepened the architecture by adding linear and ReLU layers after GATConv1 and GATConv2, which allowed the model to capture more intricate graph structures. We conducted multiple runs to identify the optimal configuration for attention heads, hidden layer sizes, and learning rate.

However, as shown in Table 2, the GAT model did not perform as well as other models for node classification on PPI datasets. This is likely due to the fact that the PPI dataset is a dense graph, and the attention mechanism may not be able to learn the complex relationships between nodes in such a graph. However, when we ran the GAT model with a learning rate of 0.005, we achieved a slightly better F1 score. We believe that this is because a lower learning rate allows the model to learn more complex relationships between nodes. However, for the sake of comparing apples to apples, we used the same learning rate and epoch across all the models.

4.5. Graph Isomorphism Network (GIN)

GNNs follow a neighborhood aggregation scheme, where the representation vector of a node is computed by recursively aggregating and transforming representation vectors of its neighboring nodes [17]. Compared with previous GCN representations of using mean or max-pooling operator, this one does not define explicit representation on how the message of neighboring node should be passed and aggregated. Instead, GIN uses MLP to learn the best representation of the nodes.

For GIN, it is supposed to be the most discriminating layer. Based on the Universal Approximation Theorem, a 1-hidden-layer MultiLayer Perceptron (MLP) with sufficiently-large hidden dimensionality and appropriate non-linearity $\sigma()$ (including ReLU and sigmoid) can approximate any continuous function to an arbitrary accuracy, as can be seen in equation 1 [13].

$$MLP_{\phi}\left(\sum_{x \in S} MLP_{f}(x)\right) \tag{1}$$

This theoretical analysis also was proven in the performance, where the F1 score on test data set is 0.945 with 8000 epochs of training, which is slightly lower than the best performing architecture. The impressive part of the GIN is that the training time and memory it takes are not significantly higher but it could achieve a relatively good results. This has something to do with the fact the MLP only requires weight matrices and input to be stored for backpropagation, and no additional processing (like clustering in Cluster GCN) is required.

4.6. Molecular Fingerprint Convolution

Molecular fingerprint (MF) convolution was designed by Adams et al. [5] as an attempt to learn fingerprints for molecular feature representations of non-fixed sizes. The network architecture is based on circular fingerprinting, which had been considered state of the art in molecular fingerprinting at the time of Adams' writing. Circular fingerprinting was designed to encode the presence of substructures within molecules by using the same operation everywhere locally, and then combining the information with a global pooling step [7]. MF convolution achieved this in a differentiable way using a convolution layer that trains a distinct weight matrix for every possible degree of every node in the graph (Equation 2), followed by a pooling layer.

$$x'_i = W_1^{(deg(i))} x_i + W_2^{(deg(i))} \sum_{i \in N(i)} x_j \tag{2}$$

Converse to our experiment, this approach was designed for molecular graphs where vertices represent individual atoms and edges represent bonds. Because of this, it had not yet been attempted for learning interactions between higher-level structures such as those in the PPI dataset. Our intuition was that since this convolution layer was proven to learn the features of molecular bonds, such as solubility and toxicity, it would likely have high predictive power for identifying protein interactions due to the overlapping feature space within these fields of study.

We first tested this hypothesis by alternating between MF and linear layers, both with ReLU activations, but found that the lack of normalization led to low performance as compared to our other experiments, likely due to poor model stability. Our next attempt used batch normalization and linear layers to stabilize the model and improve generalization. We found increasing performance with up to three MF layers before hitting the memory limitations of our GPU. With this architecture, we attained the second-highest model performance with an F1-score of 0.938, likely due to the networks ability to learn the feature space on a molecular level. As can be seen in Table 2, this came at the cost of the highest memory in addition to clocking in at the second longest run time. This time-space-performance tradeoff was to be expected due to the large number of parameters stored by each MF layer, which learns edges for the entire neighborhood of each node.

4.7. Combined Models

In an attempt to improve our highest performing models, we decided to combine subsets of different convolution layers described in the previous subsections. Our hypothesis was that by using approaches designed by studies from varying domains, the network would be able to learn a richer feature space. Additionally, since MF was our highest performing model but also the most time and space intensive by a large margin, we decided to replace the MF convolution layers with other convolution layers to relieve these constraints and hopefully maintain similar performance.

The first attempt replaced the middle MF layer with a transformer. The graph transformer layer uses multi-head attention to perform feature and label propagation which generates a transition matrix for message passing [12]. We hoped that this would learn a better representation of the feature and label embeddings, but found that it led to lower performance with the additional cost of a significant increase in time. The drop in performance tells us that the MF layer had better predictive power for our data, and the longer runtime is likely due to the high computational cost of the multi-head dot product attention.

Next, we instead attempted to replace this middle MF layer with a cluster convolution layer as described in Section 4.2. Cluster GCN exploits graph structure by restricting search to neighborhoods within local subgraphs, often improving memory and lowering computational cost [3]. As a result of this, our models runtime and size were roughly cut in half, but this still came at the cost of a moderate drop in performance. This implies that learning feature spaces from neighborhoods (Cluster) of nodes rather than global ones (MF) is less representative of the ground-truth labels for these nodes when the data comes from the gene ontology domain.

Finally, we replaced the second and third layers of our MF architecture with the GIN architecture described in Section 4.3 in hopes of helping the model to discriminate between nodes based on features learned in the first MF layer. The final architecture, shown in Figure 2, was our best performing model with an F1 score higher than both MF and GIN, while still halving time and space as compared to MF. The improved time and space complexity was to be expected, since our original GIN architecture was one of the fastest and most lightweight in our study. From the improved performance, we see that this combined architecture effectively allows the model to first learn a rich feature space, then use discrimination between nodes within local neighborhoods to generalize well to the training data.

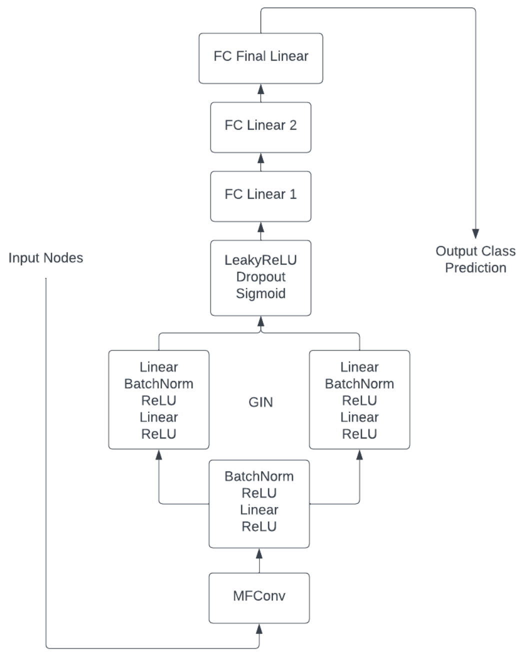
FC Final Linear
FC Linear 2
FC Linear 1
Input Nodes
LeakyReLU Dropout Sigmoid
Output Class Prediction
Linear BatchNorm ReLU Linear ReLU
GIN
Linear BatchNorm ReLU Linear ReLU
BatchNorm ReLU Linear ReLU
MFConv

Figure 2: MFConv + GIN

5. Comparison and Final Results

From Table 2, we can see that MF + GIN had the lowest F1 with MF as a close second, but GIN had the lowest loss and AUROC. We suspect that this is due to the GIN architectures' ability to better distinguish between multiple labels for classification, while MF was better for modeling the molecular graph structure. MF had a very high runtime, while MF + GIN ran in roughly half the time and GIN was a little faster. Lastly, MF + GIN was taking up almost double the amount of space on GPU as compared to GIN.

Model	**Train Loss**	**Train F1**	**Val Loss**	**Val F1**	**Time (s)**	**Memory (MB)**	**Test F1**	**Test MCC**	**Test AUROC**
Cluster	163.6	0.898	157.1	0.867	1218.5	169.4	0.880	0.828	0.886
GAT	167.5	0.771	160.3	0.738	823.2	34.3	0.757	0.645	0.809
GIN	161.5	0.961	154.9	0.932	1003.7	82.6	0.945	0.921	0.961
MF	161.9	0.966	155.0	0.938	2127.7	326.1	0.949	0.927	0.953
MF + CL	163.1	0.936	156.2	0.902	1346.0	183.5	0.916	0.879	0.919
MF + GIN	162.0	0.965	155.0	0.940	1065.3	180.9	0.951	0.929	0.958
MF + T	161.9	0.957	155.9	0.908	2175.6	76.9	0.926	0.894	0.942
SAGE	164.6	0.857	158.8	0.800	856.2	35.4	0.819	0.739	0.840

Table 2: Statistics for final model architectures averaged over five runs. Hyperparameters set to 8000 epochs and a learning rate of 1e-3. Lowest performance is highlighted in red, highest performance in yellow.

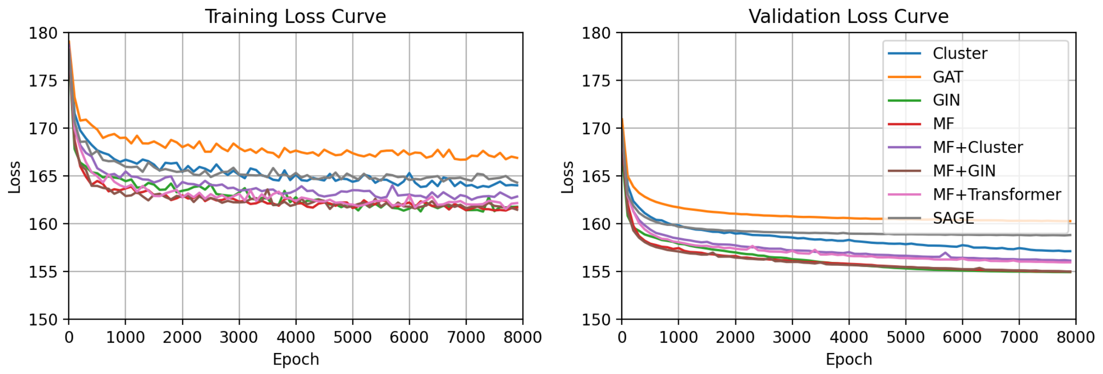

Figure 3: Training and validation loss for final model architectures at every 100 epochs. Averaged over five runs.

These findings were both likely due to the smaller number of parameters in this GIN layers. In conclusion, while MF had great performance for classifying molecular graphs, we see a time and space complexity vs. performance tradeoff where GIN is generally more efficient. By combining the two, we were able to build a model with the best features of both architectures, having high performance while still maintaining low runtime and memory.

6. Conclusion and Future Work

Our best performing model was MF+GIN. We found that the combination of these architectures allowed the model the avoid both overfitting and underfitting while still maintaining time and space efficiency. Our lowest performing model was GAT. From the fast runtime and low space complexity, we can hypothesize that this is due to the shallow architecture.

From Figure 3, we see that Cluster performance may increase significantly with training over more epochs since loss has a steep downward slope. We suspect that with additional training, it may surpass the performance of MF and GIN. Other models like SAGE and GAT show asymptotic behavior and may only show very slight improvements in performance with increased epochs, but there is no sign of overfitting so they can likely be continue to be improved with additional layers and deeper network architectures. We were unable to study these changes because we had been running up against the memory constraints of our GPU, but this could be a venue for future work.

Additionally, we found that adding MF layers increased the predictive power for all combined models. From this we can hypothesize that adding this layer to other GNN architectures may show further improved results and could be another area of future work.

We were additionally limited by time constraints but in future studies we could learn more about the benefits of the MF+GIN model by attempting modifications to our experiments such as a wider range of hyperparameter tuning, different batch sizes, changes in channel sizes or architectural changes such as adding additional pooling, normalization, or linear layers to better learn the feature space. This time constraint also prevented us from running our model additional datasets, but we suspect that it may perform well on other molecular graph datasets such as MoleculeNet, ZINC, AQSOL, Hydronet, QM7B and QM9. We believe with future studies, these results could provide marked improvements in the field of deep learning within the chemical and biological domains.

7. Project Repository

https://github.gatech.edu/wwang674/GNN4CHEM

References

[1] Monica Agrawal, Marinka Zitnik, and Jure Leskovec. Large-scale analysis of disease pathways in the human interactome. In *Pacific Symposium on Biocomputing*, volume 23, pages 111–122, 2018. 1

[2] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction, 2018. 2

[3] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-GCN. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*. ACM, jul 2019. 3, 5

[4] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, and Le Song. Learning steady-states of iterative algorithms over graphs. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1106–1114. PMLR, 10–15 Jul 2018. 2

[5] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *CoRR*, abs/1509.09292, 2015. 4

[6] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*. ACM, jul 2018. 3

[7] Robert Glem, Andreas Bender, Catrin Hasselgren, Lars Carlsson, Scott Boyer, and James Smith. Circular fingerprints: Flexible molecular descriptors with applications from physical chemistry to adme. *IDrugs : the investigational drugs journal*, 9:199–204, 04 2006. 4

[8] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018. 3

[9] Singh H. Jha K., Saha S. Prediction of protein–protein interaction using graph neural networks. *Scientific reports*, 12, 05 20222. 1

[10] mlabonne. Graphsage implementation in pytorch. https://mlabonne.github.io/blog/posts/2022-04-06-GraphSAGE.html, 2022. 3

[11] Juwen Shen, Jian Zhang, Xiaomin Luo, Weiliang Zhu, Kunqian Yu, Kaixian Chen, Yixue Li, and H. Jiang. Predicting protein-protein interactions based only on sequences information. *Proceedings of the National Academy of Sciences of the United States of America*, 104:4337–41, 04 2007. 1

[12] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked label prediction: Unified massage passing model for semi-supervised classification. *CoRR*, abs/2009.03509, 2020. 5

[13] Stanford. Cs224w: Machine learning with graphs. https://web.stanford.edu/class/cs224w/, 2023. 3, 4

[14] PyG Team. Pytorch geometric. https://pytorch-geometric.readthedocs.io/en/latest/, 2023. 2

[15] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. 4

[16] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, jan 2021. 1

[17] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019. 1, 4

[18] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications, 2021. 1

[19] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, jul 2017. 1

Student Name	Contributed Aspects	Details
Jennifer Becker	Model implementation, final analysis	Implemented and analyzed Molecular Fingerprint convolution and combined architecture with other models for MF+GIN, MF+Cluster and MF+Transformer. Trained final models on CUDA-enabled GPU and saved to SQLite database for reproducibility. Assisted with writing readme.txt. Updated main.py for final plots.
Raka Islam	Implementation and Analysis	Worked on implementing Graph Attention Network (GAT). Tried different architectures and tuned hyperparameters in order to build a better model, analyzed the results.
Weichen Wang	Implementation and Analysis	Set up the generic framework for GCN model training and testing codes. Designed and trained the Cluster GCN and GIN networks. Analyzed the results and compared the differences from theoretical perspective (naive GCN).
Yi Shuen Lim	Data Analysis, Model Implementation and Analysis	Worked on data exploration, set up initial framework for GraphSAGE model and tuned hyperparameters for GraphSAGE.

Table 3: Contributions of team members.