A Survey of Graph Neural Networks

Sumi Vora
Pomona College
spva2021@mymail.pomona.edu

Max M. McKnight
Pitzer College
mmcknight@students.pitzer.edu

Abstract

Graph Neural Networks (GNNs) are a type of deep learning model that can operate on graph-structured data, allowing them to capture complex dependencies between nodes and edges. GNNs have become increasingly important in many domains, achieving state-of-the-art results in tasks such as node classification, link prediction, and graph clustering. This survey of GNNs focuses primarily on Graph Convolutional Networks (GCNs). GCNs are a class of graph neural networks that use convolutional operations to aggregate information from a node's local neighborhood in the graph. In this paper, we first provide a comprehensive literature review on graph neural networks and explain the mathematical foundations of GNNs, deriving the necessary formulas for updating node embeddings and minimizing loss, along with an introductory sample implementation. Then, we give an application to Graph Convolutional Networks to recommendation systems. We continue with a discussion of bias and fairness within Graph Neural Network implementations and provide an implementation of a "Fair GNN" generator, and we extend the discussion to explainability methods for GCNs in the context of drug discovery. Finally, we conclude with a discussion on building and working with large graph datasets and the continued challenges associated with graph models.

1 Literature Review

Graph-structured data is ubiquitous across various disciplines and applications, ranging from social networks and biological systems to natural language processing and computer vision. Traditional machine learning and deep learning models, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), are ill-suited for processing and analyzing graph data due to their inherent assumptions about data regularity and grid-like structures. The advent of Graph Neural Networks (GNNs) has revolutionized the way we approach graph-structured data, enabling more efficient and effective learning and analysis of complex relationships and dependencies within graph data.

In this literature review, we provide a comprehensive overview of the development, key architectures, and applications of Graph Neural Networks. We begin with a brief discussion of the foundational work that laid the groundwork for GNNs, followed by an in-depth examination of prominent GNN architectures such as Graph Convolutional Networks (GCNs), GraphSAGE, Graph Attention Networks (GATs), and Graph Isomorphism Networks (GIN). Furthermore, we explore various applications of GNNs across different domains, including social network analysis, recommendation systems, molecular and drug discovery, computer vision, and natural language processing. Finally, we delve into the factors that contribute to the effectiveness of GNNs, including their ability to handle irregular data, capture local and global information, maintain invariance to graph isomorphism, and scale well to large datasets. By providing a comprehensive understanding of the state-of-the-art in GNN research, we aim to inspire further exploration and innovation in this rapidly evolving field.

1.1 Foundational Work

The concept of Graph Neural Networks was first introduced by Gori et al. (2005) [4]. In their paper, "A new model for learning in graph domains," the authors proposed a recursive learning model for processing graphs, laying the groundwork for subsequent developments in the field.

Scarselli et al. (2009) [15] further refined this model in their seminal paper "The graph neural network model." This work established a more general framework for GNNs and provided a thorough analysis of their theoretical properties, such as convergence and computational complexity.

1.2 Key Architectures and Methods

Several GNN architectures and methods have emerged over the years, each with its unique strengths and capabilities:

- Graph Convolutional Networks (GCNs): Kipf and Welling (2017) [7] proposed a simplified and efficient approach to GNNs in their paper "Semi-Supervised Classification with Graph Convolutional Networks." GCNs have since become one of the most popular GNN architectures, as they are well-suited for node classification and graph representation learning tasks.

- GraphSAGE: Hamilton et al. (2017) [5] introduced GraphSAGE (Graph Sample and Aggregate) in their paper "Inductive representation learning on large graphs." GraphSAGE can generate node embeddings for unseen nodes by sampling and aggregating information from a node's local neighborhood, making it effective for inductive learning tasks.

- Graph Attention Networks (GATs): Velickovic et al. (2018) [19] proposed GATs in "Graph Attention Networks." This architecture leverages attention mechanisms to weigh the contributions of neighboring nodes, allowing the model to learn different levels of importance for different neighbors.

- Graph Isomorphism Networks (GINs): Xu et al. (2019) [24] introduced GINs in their paper "How Powerful are Graph Neural Networks?" GINs can effectively learn to distinguish between different graph structures and capture information about the local graph isomorphism.

1.3 Applications

GNNs have been successfully applied to various domains, showcasing their versatility and effectiveness in handling graph-structured data:

In social network analysis, GNNs can be employed to study social networks, identifying influential users, predicting user behavior, or anticipating the formation of new connections. They have shown promising results in tasks such as community detection, link prediction, and social recommendation [29].

Graph-based recommendation systems model user-item relationships as graphs, GNNs can be utilized to generate personalized recommendations with improved accuracy and diversity. GNN-based recommendation systems have been shown to outperform traditional collaborative filtering and matrix factorization techniques in various settings [26].

GNNs have also been used to predict molecular properties, toxicity, and drug-target interactions by analyzing the graph structures of molecular compounds [3]. By capturing the complex dependencies between atoms and chemical bonds, GNNs can facilitate the discovery of novel molecules and drugs with desired properties.

For computer vision, GNNs can help model the relationships between objects and regions in images, leading to more effective scene understanding and object recognition [17]. By representing images as graphs, GNNs can capture both local and global context, enhancing performance in tasks such as object detection, semantic segmentation, and image classification.

Finally, in Natural Language Processing, GNNs can be utilized to model syntactic and semantic relationships between words, phrases, or sentences, improving tasks like sentiment analysis, machine translation, and relation extraction [9]. By representing language as graphs, GNNs can effectively capture the hierarchical and long-range dependencies that are often present in natural language data.

Effectiveness of GNNs

Graph Neural Networks (GNNs) have emerged as highly effective models for handling graph-structured data. The effectiveness of GNNs can be attributed to several key factors that allow them to model complex relationships and dependencies within graph data. Below, we will expand upon these factors in greater detail.

Handling irregular data

Graph-structured data is inherently irregular and unordered, which poses a challenge for traditional deep learning models that require fixed-size inputs and regular grid-like structures, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). GNNs, on the other hand, are specifically designed to process graph data, allowing them to handle irregular structures and varying node connectivity. This enables GNNs to effectively analyze a wide range of data types that can be naturally represented as graphs, such as social networks, biological networks, and transportation networks.

Capturing local and global information

GNNs excel at learning both local and global information from graph data. Local information refers to the relationships and dependencies between neighboring nodes, while global information refers to the higher-level structure and patterns within the entire graph. GNNs are able to aggregate information from a node's local neighborhood iteratively, capturing not only the direct relationships between neighboring nodes, but also the indirect relationships spanning multiple hops. This enables GNNs to model complex dependencies and relationships between nodes, enhancing their performance in tasks such as node classification, link prediction, and graph classification.

Invariance to graph isomorphisms

Graph isomorphism refers to the property that two graphs are considered identical if there exists a one-to-one correspondence between their nodes and edges while preserving the connectivity. GNNs are designed to be invariant to different isomorphic representations of the same graph, meaning that they can recognize identical structures even if the nodes are ordered or labeled differently. This is an essential property for learning meaningful representations from graph data, as it allows GNNs to focus on the inherent structure and relationships within the graph, rather than being sensitive to the specific arrangement of nodes and edges.

Scalability

Many GNN architectures are designed to scale well to large graph datasets, making them suitable for real-world applications with massive amounts of data. Techniques such as neighborhood sampling, subgraph training, and graph partitioning have been developed to address the scalability challenges posed by large graphs. These methods enable GNNs to process and learn from large-scale graph data efficiently, without sacrificing the quality of the learned representations. As a result, GNNs have been successfully applied to various large-scale graph analysis tasks, such as social network analysis, recommendation systems, and biological network analysis.

1.4 Explainability in Graph Neural Networks

As GNNs continue to advance, understanding and interpreting their decision-making processes have become critical for their wider adoption in real-world applications. Explainability, the ability to provide meaningful and transparent explanations for GNN predictions, is a fundamental aspect that enhances trust, allows for domain expert validation, and facilitates the identification of biases or errors.

The concept of explainability in machine learning, including GML, can be traced back to works like Ribeiro et al.'s "Why Should I Trust You? Explaining the Predictions of Any Classifier" (2016) which introduced LIME, a model-agnostic method for interpreting predictions. Although not specific to GML, these works laid the groundwork for explainability in more complex domains like graph data. [14]

Graph Attention Networks (GATs)

One of the first attempts to bring explainability to GML was Graph Attention Networks by Velickovic et al. (2018). [19] GATs use attention mechanisms to weigh the importance of a node's neighbors. The attention coefficients provide some level of interpretability, as they indicate which nodes influenced the prediction most.

GNNExplainer

The need for more explicit explainability methods led to the development of GNNExplainer by Ying et al. (2019). GNNExplainer generates explanations for predictions of Graph Neural Networks (GNNs), finding a compact subgraph and identifying important edges and nodes contributing to the model's prediction. [25]

PGExplainer

PGExplainer (2020) by Luo et al. took a different approach, using a differentiable graph module to learn the importance of each edge. It generates better explanations than GNNExplainer, especially on synthetic datasets. [8]

XGNN

XGNN (2020) by Yuan et al. interprets GNNs from a model-level perspective, aiming to visualize how the GNN model learns to aggregate and transform features over graph structures. [27]

Graph Neural Networks and Causal Inference

Causal inference can help provide explainability in GML. NIPS 2020 paper "Causal Inference using Gaussian Processes with Structured Latent Variables" by Witty et al. proposed a method to exploit the graph structure for causal inference, providing an additional avenue for explainability. [22]

Graph Explainability through Rule Extraction

Rules provide a high-level form of explainability. Rules can be extracted from trained GNNs, a process explored in "RuleMatrix: Visualizing and Understanding Classifiers with Rules" by Ming et al. (2018). [10]

The effectiveness of Graph Neural Networks can be attributed to their ability to handle irregular data, capture local and global information, maintain invariance to graph isomorphism, and scale well to large datasets. These factors have enabled GNNs to excel in a wide range of applications and domains, making them an indispensable tool for processing and analyzing graph-structured data.

In conclusion, Graph Neural Networks have emerged as powerful tools for processing and analyzing graph-structured data, with numerous architectures and applications across various domains. The foundational work by Gori et al. (2005) and Scarselli et al. (2009) paved the way for subsequent developments, including key architectures such as GCNs, GraphSAGE, GATs, and GIN. The effectiveness of GNNs can be attributed to their ability to handle irregular data, capture local and global information, maintain invariance to graph isomorphism, and scale well to large datasets.

2 Theoretical Foundations of Graph Neural Networks

Graph neural networks (GNNs) are deep learning models that are designed to analyze data represented as graphs. We define a **graph** $G = (V, E)$ as a set of objects, called **nodes**, where each node $n_i \in V$, whose relationships are represented by the **edges** E between them. Each node may have a d-dimensional **feature vector** $\mathbf{x}_i \in \mathbb{R}^d$ that denotes the characteristics or properties of a given node. The feature matrix $\mathbf{X}$ is given by $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n)^T$ so that the i-th row of the feature vector corresponds to $\mathbf{x}_i$.

Graphs differ from data passed into ordinary neural networks in that nodes are unordered and each graph has varying topological representations. As such, we must define a message passing operation [20] f such that

$$f(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} g(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}) \\ g(\mathbf{x}_2, \mathbf{X}_{\mathcal{N}_2}) \\ \vdots \\ g(\mathbf{x}_n, \mathbf{X}_{\mathcal{N}_n}) \end{bmatrix}$$

where $\mathbf{A}$ is the adjacency matrix of the graph and g is some local function that operates over a node and its neighbors. Note that to preserve the structure of the graph given various isomorphic representations, f must be permutation invariant, meaning that

$$f(\mathbf{PX}, \mathbf{PAP}^T) = f(\mathbf{X}, \mathbf{A})$$

as well as permutation equivariant, meaning that

$$f(\mathbf{PX}, \mathbf{PAP}^T) = \mathbf{P}f(\mathbf{X}, \mathbf{A})$$

where $\mathbf{P}$ is some permutation matrix. [20] Additionally, g should also be permutation invariant, since it operates over the feature vector of some node and the multiset of the feature vectors of the neighbors of that node. This message passing function will output a matrix $\mathbf{H} = (\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_n)^T$ where each $\mathbf{h}_i$ is a latent vector given by $g(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})$.

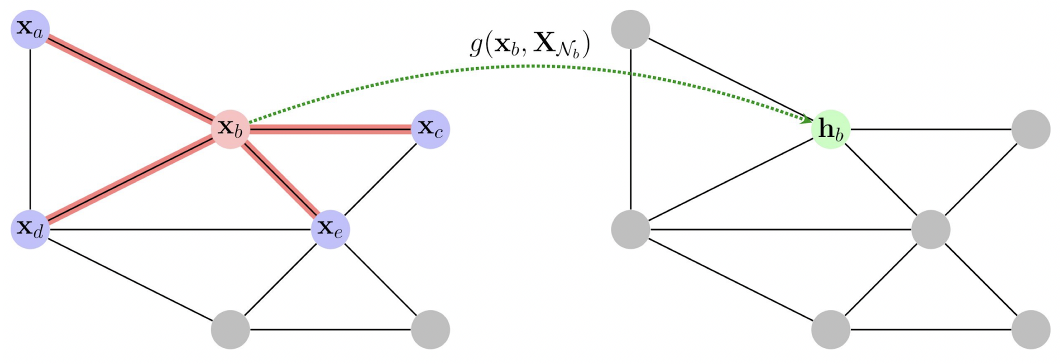

Figure 1: Visualizing $f(\mathbf{X}, \mathbf{A})$ [20]

2.1 Graph Convolutional Networks

For graphs with nodes that are more strongly connected to nodes with similar feature vectors, we use a subset of GNNs called Graph Convolutional Networks (GCNs) to generate node embeddings (i.e. vectors that represent nodes in Euclidean space). Here, the features of neighbors can be aggregated with fixed weights c_{ij} that depend directly on $\mathbf{A}$, so the message passing function is given by:

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right)$$

where $\oplus$ is some permutation invariant aggregation operator, and ϕ and ψ are learnable functions. [20]

The node embeddings are determined iteratively, after k iterations of message passing, in order to minimize some loss function. For generating node embeddings in GCNs (which assume that connected nodes have similar feature vectors), we can use binary cross-entropy loss, which optimize $\mathbf{h}_i$ and $\mathbf{h}_j$ to be nearby if and only if $(i, j) \in E$. The loss is therefore given by:

$$\sum_{(i,j)\in E} \log \sigma(\mathbf{h}_i^T \mathbf{h}_j) + \sum_{(i,j)\notin E} \log(1 - \sigma(\mathbf{h}_i^T \mathbf{h}_j))$$

For graph classification, the cross-entropy loss is calculated similarly:

$$L = -\sum_{i=1}^{m} y_i \log \hat{y}_i$$

where m is the number of graphs, y_i is the ground truth label of graph i, and $\hat{y}_i$ is the predicted probability of graph i belonging to its true class.

GCNs can be powerful for examining networks. We implement a GCN on the PyTorch Cora dataset, which consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the data set is described by a vector of words represented by a value 0/1 that indicates the absence / presence of the corresponding word in the dictionary. The dictionary consists of 1433 unique words.

Our implementation classifies the nodes according to the links in the citation network. We train the model using a negative log likelihood loss computed from the ground truth labels and the boolean training mask from the dataset. (The boolean training mask specifies which nodes in the graph are used for training.) The loss is then backpropagated through the model and the optimizer is updated accordingly. The accuracy was 0.781, which was comparable to other GCN models on this dataset. Figure 2 visualizes the clusters generated by our GCN implementation.

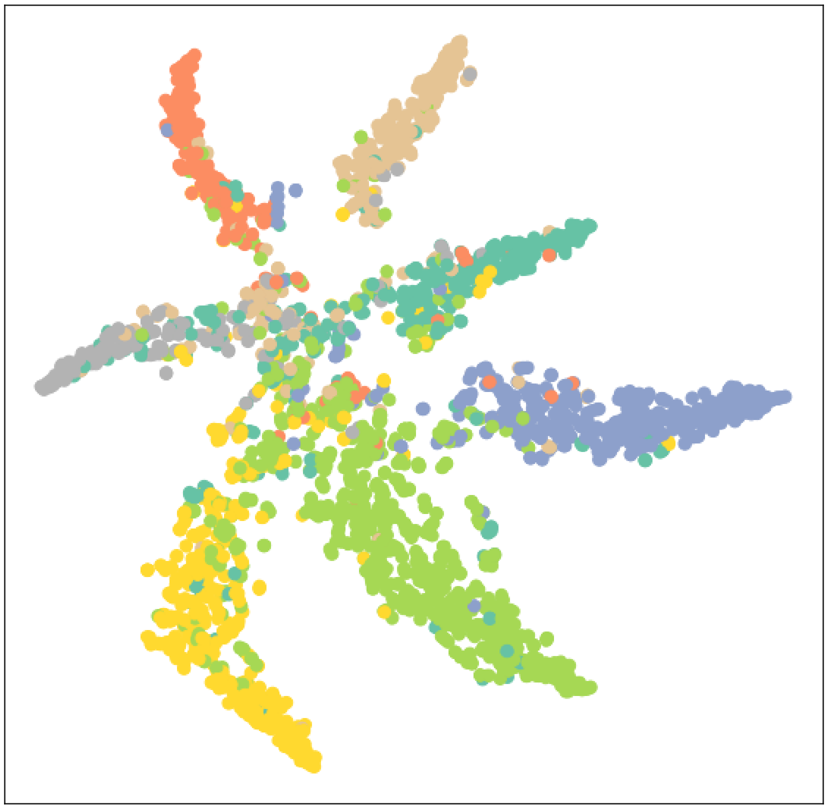

Figure 2: GCN Node Classification on Cora Dataset

3 GNNs for Recommendation Systems

GNNs for recommendation systems can be implemented in a few different ways. If the recommendation is based on a social network, for example a dating app or an employee recommender system, the nodes might be the entities within the social network and the edges might represent relationships. If it is a product representation, it might be more suitable to model the network as a bipartite graph, where one part represents customers and the other part represents products. [16]

In the latter case, the node embeddings determine the similarity between products and the similarity between customers, so the message sending operation and the loss function should not assume that edgewise relationships are the only factor for determining similarity between nodes. Here, we define a more general message passing operation, where nodes work together to determine the updated embeddings at each stage of the iterative process:

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right)$$

After obtaining node embeddings, the prediction will be in the form

$$\hat{\mathbf{y}}_i = r(\mathbf{h}_u, \mathbf{h}_i)$$

where $\hat{\mathbf{y}}_i$ is the predicted rating of item i, $\mathbf{h}_u$ is the embedding of the user u, $\mathbf{h}_i$ is the embedding of the item i, and r is some learnable function that maps the embeddings to the rating score, such as a linear or non-linear classifier.

Finally, the loss function is most accurately determined by the mean squared error, similarly to that of other recommendation systems, and we define the loss for some user-item pair as:

$$L(u, i) = (\mathbf{y} - \hat{\mathbf{y}})^2$$

The MSE will be the average of these losses over all of the user-item pairs:

$$\frac{1}{n} \sum_{(u,i) \in G} L(u, i)$$

For our implementation, we use the Movie Lens 100K dataset, which gives 100,000 ratings by 943 users of 1682 items (movies). Each user has rated at least 20 movies. Since there exist user-item pairs, we create a bipartite graph, with the parts being users and items, and we draw an edge between a user and an item if the user has rated the movie. [13] give each edge a 'size' To start, we use collaborative filtering to build our neural network. Collaborative filtering makes predictions about a user's preferences based on the preferences of similar users or items. For our implementation of this model, we do not take into account the features of the movies and the users – we simply look at the preferences of users who have rated the same movies similarly. Hence, it is appropriate to use a GCN in this case.

The specific architecture of the model that we use is called LightGCN. LightGCN takes advantage of mini-batch positive and negative sampling, where we sample a small subset of positive interactions (i.e., observed interactions between users and items) and a larger subset of negative interactions (i.e., unobserved or negative interactions) from the training data to train the model on these samples. We define a positive interaction as a rating of four or five, and a negative interaction as a rating of 3 or less. We train the model using Bayesian Personalized Ranking (BPR) loss. The loss for a single user is given by:

$$L = \frac{1}{|E(u)|} \sum_{(u,i) \in E(u)} \sum_{(u,j) \in E(u)} -\log(\sigma(r(u, i) - r(u, j))) + \lambda ||\theta||^2$$

where $r(u, i)$ is the predicted ranking score for the observed positive interaction between user u and item i, $r(u, j)$ is the predicted ranking score for a sampled negative interaction between user u and item j. σ is the sigmoid function, λ is a regularization parameter that controls the complexity of the model, θ represents the model parameters and $E(u)$ is the set of pairs (user, item) consisting of positive interactions observed and negative interactions for some user u. The total BPR loss is then given by:

$$\frac{1}{|U|} \sum_{u \in U} BPR(u)$$

where U is the set of users.

Figure 3 on the following page gives the loss curve per training epoch, illustrating how quickly the loss diminishes per epoch. We see that the BPR training loss diminishes quickly at first, but stagnates at a loss of around 0.1. Similarly, Figure 4 shows low metrics for recall and precision, where the maximum recall is 0.3625 and the maximum precision is 0.2876.

A possible extension to this would be to compare this GCN to a more complex model that takes in the feature vectors for each user and item.

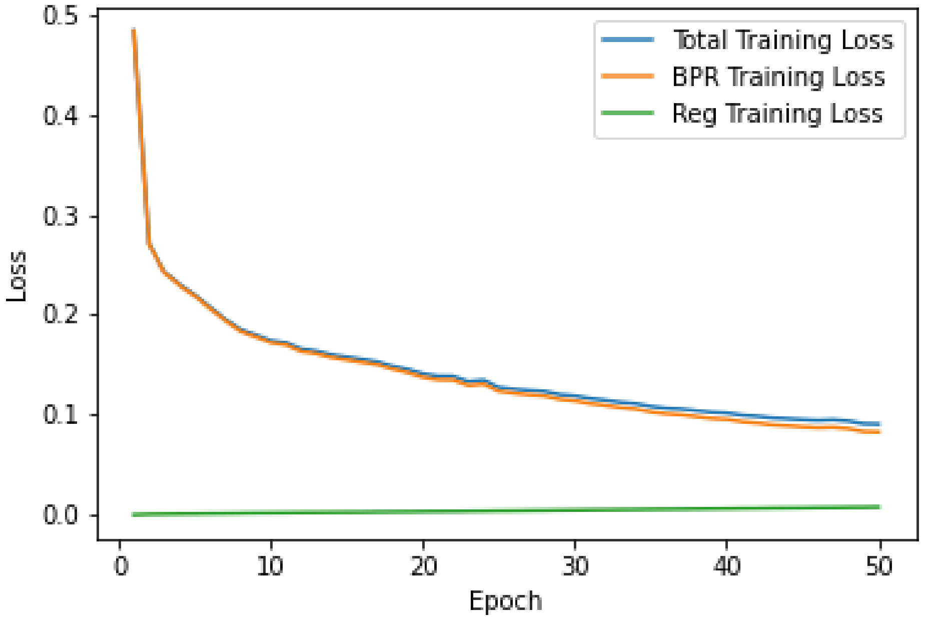

Figure 3: LightGCN loss curves per training epoch.

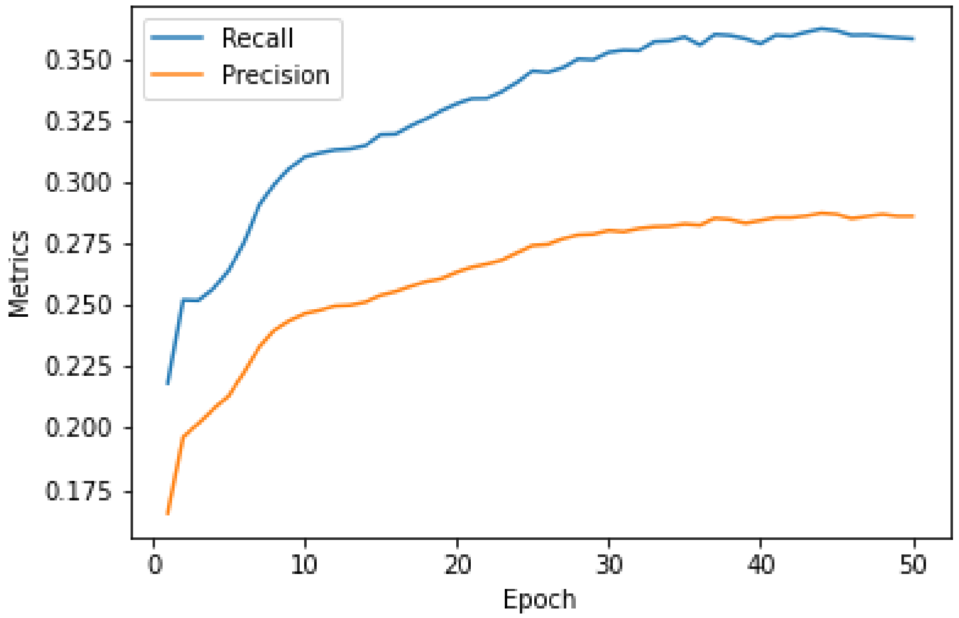

Figure 4: LightGCN Precision and Recall per training epoch.

4 Fairness in Graph Neural Networks

We conclude our paper with a discussion of fairness in Graph Neural Networks. Several studies have shown that data may include discriminatory decisions dominated by sensitive features, e.g. race, gender, religion, sexual orientation, and disability status. In GNNs, node representations may therefore inherit these biases and make predictions or decisions that may be unfair or harmful to certain individuals or groups. Since nodes with similar sensitive features form closer connections to those without these features, predictions are often over-associated with these sensitive features.

Another area of concern is known as sensitive attribution leakage, where some feature channels that have encode less sensitive information may become highly correlated to sensitive ones after feature propagation. For example, suppose a GNN is used to recommend careers to an individual based on their social network connections. Even if the model does not use the gender of the individual as an input feature, it may learn to use other features such as occupation or education level, which are strongly correlated with gender, to make predictions. This can result in biased or unfair decisions that disproportionately affect certain groups.

Following Wang et al. [21], we define the sensitive features $\mathbf{S} = \mathbf{X}[:, s]$ where s-th channel of the node features matrix $\mathbf{X}$ is considered a sensitive feature. The differences of statistical parity (i.e. fairness between groups) and equal opportunity are defined as:

$$\Delta_{\text{sp}} = |P(\hat{y} = 1|s = 0) - P(\hat{y} = 0|s = 1)|$$
$$\Delta_{\text{eo}} = |P(\hat{y} = 1|y = 1, s = 0) - P(\hat{y} = 0|y = 1, s = 1)|$$

Thus, fairer algorithms will have a relatively lower Δ_{sp} and Δ_{eo}. Finally, we denote the correlation between some feature channel and a sensitive feature channel as:

$$\rho_i = \frac{\mathbb{E}_{v_j} \sim V(\mathbf{X}_{ji} - \mu_i)(\mathbf{S}_j - \mu_s))}{\sigma_i \sigma_s}$$

so that a higher ρ_i means that the i-th channel has information that is more highly correlated with sensitive information.

We notice from Figure 5 that even if we mask a sensitive channel, discrimination still exists, but if we mask other non-sensitive feature channels that according to their ρ_i rank, Δ_{sp} and Δ_{eo} decrease.

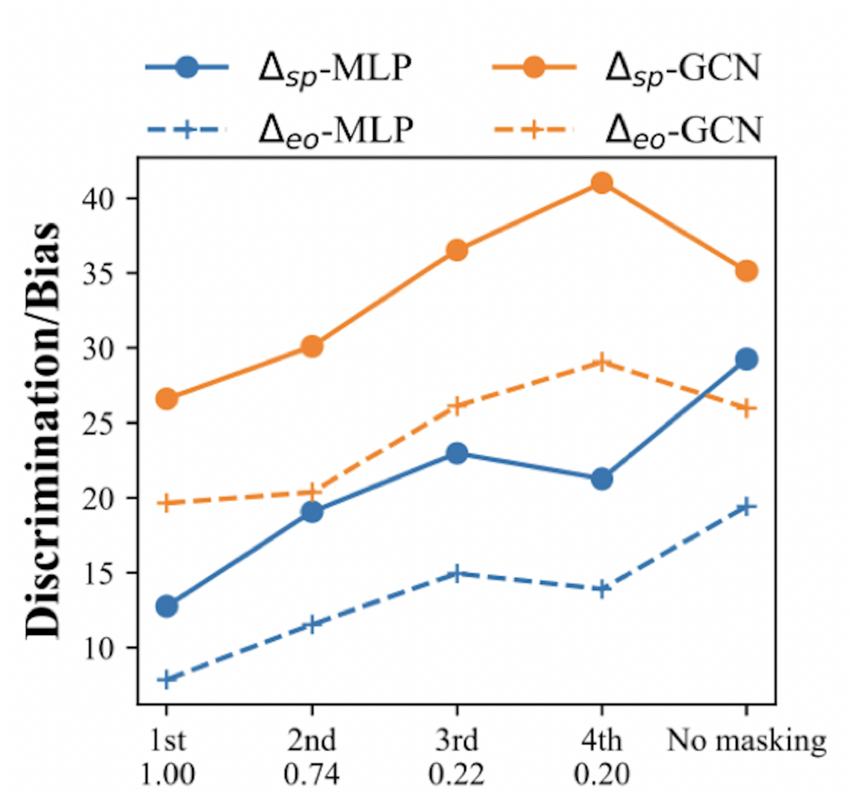

Figure 5: Bias For Masking Features Correlated with Sensitive Channels

This indicates that there exists some sensitive attribute leakage. From this, we might surmise that masking sensitive features and features that are highly correlated with sensitive features will prevent discrimination. However, when we run figure 6 shows that when we perform feature propagation, features that were originally uncorrelated with sensitive features become highly correlated with sensitive features.

The algorithm that we replicate from [21] builds a fair generator $g_{\Theta_g} : g_{\Theta_g}(\mathbf{X}) \to \tilde{\mathbf{X}}$ that expects to preserve task-related information and discard sensitive information such that the node classifier $f_{\Theta_f} : f_{\Theta_f}(\mathbf{A}, \tilde{\mathbf{X}}) \to \mathbf{Y}$ can achieve a better utility vs. fairness tradeoff.

Begin by assuming that g_{Θ_g} is a learnable latent distribution, and sample K different masks from g and give the corresponding views of $\tilde{\mathbf{X}}^k$. We want to update g towards generating less biased views of $\tilde{\mathbf{X}}$. For training, we train a learnable mask $\mathbf{m} = [m_1, m_2, \cdots m_d]$ where each $m_i \in \{0, 1\}$. Then,

$$\tilde{\mathbf{X}} = \mathbf{X} \odot \mathbf{m} = [\mathbf{X}_1^T \odot \mathbf{m}, \mathbf{X}_2^T \odot \mathbf{m}, \cdots, \mathbf{X}_n^T \odot \mathbf{m}]$$

where $\odot$ denotes component-wise multiplication.

Since the masks are discrete, the Gumbel-Softmax trick is used to approximate the continuous Bernoulli distribution of $\tilde{\mathbf{X}}$. This trick and generates n samples $g_1, g_2, \cdots, g_n$ from the Gumbell distribution, defined as $G_i = -\log(-\log(U_i))$, where U_i is a sample from the unform distribution $(0, 1)$. Then, it computes the logits $y_1, y_2, \cdots, y_n$ such that $y_i = \log(\tilde{\mathbf{X}}_i) + g_i$.

This makes it so that gradient descent can be applied to minimize the discriminator d_{Θ_d}, where d_{Θ_d} predicts the sensitive features. We define the loss function for the discriminator as

$$\max_{\Theta_d} \mathcal{L}_d = \mathbb{E}_{\tilde{\mathbf{X}} \sim \mathbb{P}^{\Theta_g}_{\tilde{\mathbf{X}}|\mathbf{X}}} \mathbb{E}_{v_i \sim V} (\mathbf{S}_i \log(d_{\Theta_d}(\tilde{\mathbf{H}}_i^L)) + (1 - \mathbf{S}_i) \log(1 - d_{\Theta_d}(\tilde{\mathbf{H}}_i^L)))$$

where $\mathbb{P}^{\Theta_g}_{\tilde{\mathbf{X}}|\mathbf{X}}$ is the joint distribution of the graph given $\mathbf{X}$ and parameterized by Θ_g and $\tilde{\mathbf{H}}_i^L = f_{\Theta_f}(\mathbf{A}, \tilde{\mathbf{X}}_i)$.

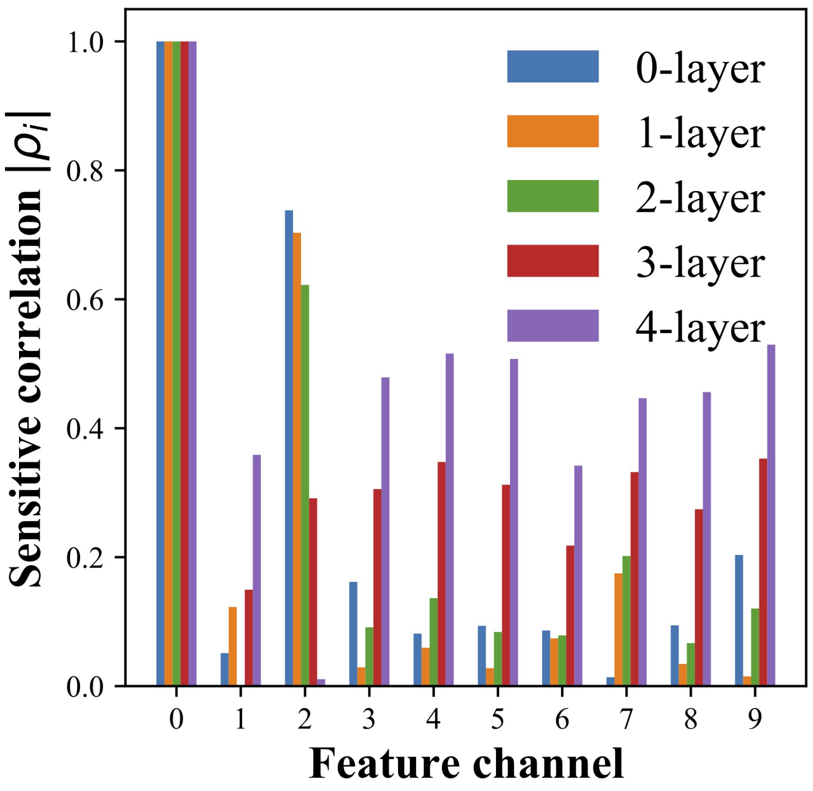

Figure 6: Correlation of Channels with Sensitive Channels During Propagation

In order to make sure that the discriminator reaches the global minimum, we use adaptive clamping, which prevents gradients from becoming too large and causing the optimization process to diverge or oscillate. To do this, we sample K masks in each epoch and compute their mean $\mathbf{p} = \sum_{k=1}^{K} \mathbf{m}^k$, which gives the probability of keeping the feature. Given the learned weights $\mathbf{W}^{f,1}$ given by the first layer of f, we clamp it by assigning:

$$\mathbf{W}_{ij}^{f,1} = \begin{cases} \mathbf{W}_{ij}^{f,1} & |\mathbf{W}_{ij}^{f,1}| < \epsilon * \mathbf{p}_j \\ \text{sign}(\mathbf{W}_{ij}^{f,1}) * e * \mathbf{p}_j & \text{otherwise} \end{cases}$$

where ϵ is selected beforehand. This gives us weights so that if channels have a higher probability of being masked, their overall contributions are lower. Figure 7 shows a summary of the algorithm.

To test our model, we implement on the German datset. The German dataset represents clients at a German bank as nodes, and the node attributes include gender, loan amount, and account details. There is an edge between two clients if their nodes are similar, and the task is to classify the risk as high or low.

We implemented four different types of GNNs on the dataset, and then compared utility metrics (AUC, Accuracy, and F1) and fairness metrics (Δ_{sp} and Δ_{eo}) for the model including the and without the FairGNN. To see the effects of the different parts of the algorithm, we also implemented versions of the without weight clamping and without the generator g.

Our results are summarized in Table 1.

For Graph Convolutional Networks using sparse matrix multiplication (GCN-spmm) the results are as expected: the utility was highest when the FairGNN was not implemented, and the algorithm was the most fair when the full FairGNN algorithm was implemented. However, it is important to note that the variability between the utility metrics was quite low (less than 0.1 standard deviations for the AUC), while the variance in the fairness metrics was relatively high. Therefore, using the full FairGNN algorithm for GCNs may be preferable.

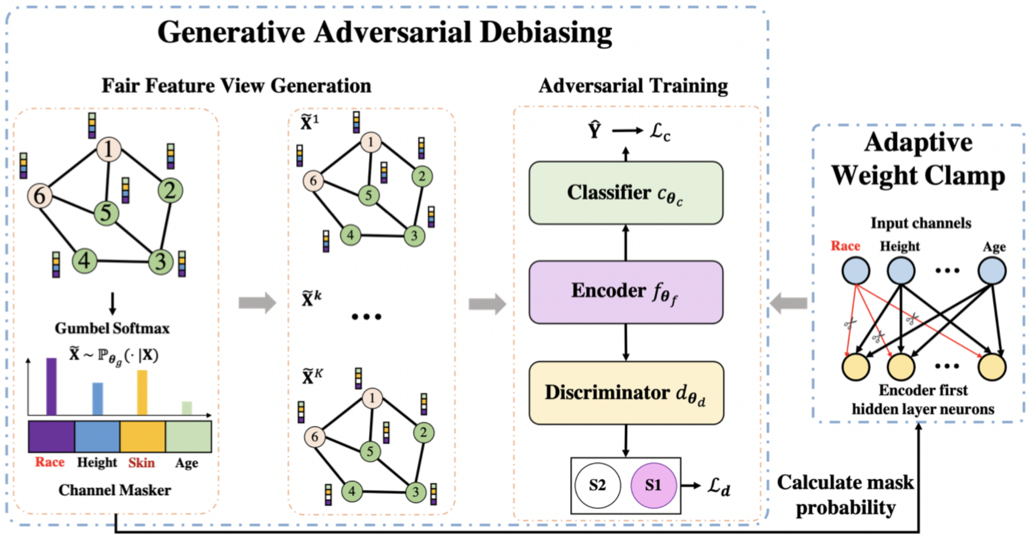
Generative Adversarial Debiasing
Fair Feature View Generation
Adversarial Training
Gumbel Softmax
$\tilde{X} \sim \mathbb{P}_{\theta_g}(\cdot|X)$
Race
Height
Skin
Age
Channel Masker
$\tilde{X}^1$
$\tilde{X}^k$
$\tilde{X}^K$
$\hat{Y} \rightarrow \mathcal{L}_c$
Classifier c_{θ_c}
Encoder f_{θ_f}
Discriminator d_{θ_d}
S2
S1
$\rightarrow \mathcal{L}_d$
Adaptive Weight Clamp
Input channels
Race
Height
Age
Encoder first hidden layer neurons
Calculate mask probability

Figure 7: Algorithm Summary

For both GINs, the full FairGNN actually maximized accuracy and F1, while the FairGNN without the fair generator maximized the fairness metrics.

In the original GIN architecture, the aggregation operation is a sum over the embeddings of neighboring nodes, followed by a MLP with a shared weight for all nodes. This operation can lead to over-smoothing, where the node features become too similar and lose important information about the local graph structure. Therefore, compared to other models, the AUC is relatively low, and models that exclude weight clamping (which further smooth the features) did not perform well on utility metrics. As expected, the fairness is optimized with the full FairGNN implementation.

In GIN-Scatter, the aggregation operation is a weighted sum of the embeddings of neighboring nodes, followed by a fully-connected layer that is applied independently to each node. The weights used in the weighted sum are learned during training and depend on the features of the nodes being aggregated. In the GIN-scatter, the utility overall was higher than the original GIN models. However, the fairness overall was lower. This was not as expected, and we may attempt to use the same models on different datasets to see if this result holds.

Model	Model Variation	AUC	Accuracy	F1	Δ_{sp}	Δ_{eo}
GCN-spmm	Full FairGNN	71.66+0.9	70.0+0.67	81.66+0.55	**2.36+1.81**	**2.73+1.83**
GCN-spmm	FairGNN w/o *g*	72.24+1.34	70.24+0.54	82.0+0.63	3.94+3.74	2.82+2.4
GCN-spmm	FairGNN w/o WC	71.53+2.57	70.4+0.72	82.06+0.3	6.42+9.04	3.7+5.34
GCN-spmm	No FairGNN	**74.38+1.05**	**70.64+0.74**	**82.19+0.82**	6.37+6.23	3.34+4.41
GIN-scatter	Full FairGNN	71.69+3.71	**71.04+1.4**	**82.57+0.41**	6.08+10.28	3.61+6.29
GIN-scatter	FairGNN w/o *g*	69.75+1.6	69.76+0.2	81.71+0.5	**3.01+3.07**	**2.31+2.79**
GIN-scatter	FairGNN w/o WC	70.1+5.16	70.24+1.33	82.23+0.39	4.12+7.41	2.88+4.49
GIN-scatter	No FairGNN	**73.25+0.61**	70.64+0.82	82.12+0.4	7.82+3.98	5.13+3.24
GIN	Full FairGNN	68.68+4.78	70.32+0.47	82.49+0.2	**0.51+0.74**	**0.17+0.34**
GIN	FairGNN w/o *g*	**74.07+0.68**	72.48+0.96	83.28+0.56	7.09+3.12	1.85+1.72
GIN	FairGNN w/o WC	68.41+3.25	70.32+1.06	82.2+0.61	2.71+3.44ee	1.18+1.82
GIN	No FairGNN	73.55+0.8	**72.56+0.82**	**83.39+0.39**	5.61+2.6	1.13+0.86
SAGE	Full FairGNN	73.65+1.13	70.0+0.91	81.63+0.67	4.49+4.47	**2.04+2.03**
SAGE	FairGNN w/o *g*	**73.76+0.69**	**70.72+0.69**	81.34+0.24	3.69+1.92	2.52+2.08
SAGE	FairGNN w/o WC	71.41+2.12	70.16+1.2	**82.05+0.36**	**2.63+2.19**	2.25+1.3
SAGE	No FairGNN	67.02+7.2	69.92+0.16	81.64+1.0	3.43+3.24	2.69+2.81

Table 1: Utility and Fairness Metrics on Several Fair GNN Algorithms

For the SAGE model, the FairGNN without the fair generator tended to maximize utility, while the FairGNN without the weight clamping tended to maximize fairness. SAGE performs inductive learning on graphs by sampling the local neighborhood of each node and aggregating information from the sampled neighbors to generate node representations. Since it uses a fixed-size set of trainable aggregation functions to generate node embeddings based on the sampled neighbors, this result also makes sense, since the sampling performed by g is redundant, thus reducing utility.

5 Explainability in Graph Neural Networks

In this subsection, we explore explainability methods for graph convolutional networks. As GCNs have emerged as a powerful tool for biological data and drug discovery, explainability is an important aspect of evaluating biomolecular results provided by graph convolutional networks.

We test three different approaches to explainability in GCNs: contrastive gradient-based saliency maps, class activation mapping (CAM), and excitation backpropagation.

Saliency maps for GCNs provide a means to assess the importance of individual nodes within a graph. Gradient-based saliency maps compute the gradient of the model's output with respect to the input, and then generate a heat map by taking the norm of the gradient and discarding negative values in order to obtain the parts of the input that positively contribute to the output. [12]

To generate the heat map over a graph, we denote the global average pool for the k-th convolutional feature map as:

$$e_k = \frac{1}{N}\sum_{n=1}^{N} F_{k,n}^{L}(X, A)$$

where L is the last layer before the softmax layer. Following [12], $F_k^l(X, A)$ demotes the k-th feature for the l-th layer for node n. Next, we define the class score y^c as:

$$\sum_k w_k^c e_k$$

where the weights w_k^c are learned from the input and output of the model. Intuitively, the weights represent the importance of the k-th feature for predicting the class c.

Then, we generate the saliency heat map over nodes n from:

$$L_{Gradient}^{c} = [n] = \|\mathrm{ReLU}\left(\frac{\partial y^c}{\partial X_n}\right)\|$$

where X_n is the feature vector for node n, y^c is the class score and e_k is the global average pool feature of the final convolutional layer of the GCN, before the softmax layer.

While saliency maps identify the most important nodes in an input graph, class activation mapping (originally developed for convolutional neural networks) has been adapted to GCNs to identify salient subgraphs or neighborhoods within a graph that contribute to the model's predictions. CAM aggregates the activations of nodes in the graph to generate a heatmap, visually highlighting the discriminative regions or subgraphs. This often gives a more semantically meaningful interpretation of important graph structures that is useful for molecular data.

CAM heatmaps are calculated as:

$$L_{CAM}^{c}[n] = \mathrm{ReLU}\left(\sum_k w_k^c F_{k,n}^{L}(X, A)\right)$$

Finally, excitation backpropagation offers insights into the importance of input features or nodes in influencing the model's predictions. By backpropagating activations through the network, excitation backpropagation accounts for the excitatory impact of each feature or node. This technique reveals the regions or nodes that significantly contribute to the model's activations and outputs.

We calculate the heat map by making backwards passes through the softmax layer, the GAP layer, and the convolutional layers. The equations for the passes through the softmax and GAP layers are:

$$\begin{cases} p(e_k) = \sum_c \frac{e_k \text{ReLU}(w_k^c)}{\sum_k e_k \text{ReLU}(w_k^c)} p(c) & \text{for the softmax layer, and} \\ p(F_{k,n}^L) = \frac{F_{k,n}^L}{N_{e_k}} p(e_k) & \text{for the GAP layer} \end{cases}$$

where $p(c) = 1$ for the class of interest and zero otherwise.

The backwards passes for the convolutional layers are defined as

$$\begin{cases} p(F_k^l) = \sum_m \frac{V_{n,m} F_{k,n}^l}{\sum_n V_{n,m} F_{k,m}^l} p(\hat{F}_{k,m}^l) \\ p(\hat{F}_k^l) = \sum_{k'} \frac{\hat{F}_{k,n}^l \text{ReLU}(W_{k,k'}^l)}{\sum_k \hat{F}_{k,n}^l \text{ReLU}(W_{k,k'}^l)} p(F_{k',n}^{l+1}) \end{cases}$$

where we have decomposed the graph convolutional operator F_k^l into:

$$\begin{cases} \hat{F}_{k,n}^l = \sum_n V_{n,m} F_{k,m}^l \\ F_{k',n}^{l+1} = \sigma\left(\sum_k \hat{F}_{k,n}^l W_{k,k'}^l \right) \end{cases}$$

Then, the heat map on nodes n is given by:

$$L_{EB}^c[n] = \frac{1}{d_{in}} \sum_0^{d_{in}} p(F_{k,n}^0)$$

where d_{in} is the dimension of the input. Note that these explainability methods are outlined in [12].

Excitation backpropagation is particularly useful for molecular data in the field of drug discovery and molecular biology because it enables the interpretation of structure-activity relationships (SAR) in molecular data. By highlighting the influential features or atoms, as it highlights the regions or substructures that have a strong excitatory effect on the model's output, indicating their importance in determining the molecule's activity. [28]

We choose excitation backpropagation over layer-wise relavance propagation and other decomposition techniques because excitation backpropagation provides a pixel-level visualization of the important regions in the input, which is useful in identifying specific atomic or molecular fragments. Also, since excitation backpropagation directly propagates activations backward through the network, there exist structural similarities between the GCN and activation transfer within molecular structures. [23]

Figure 8 gives an overview of decomposition techniques for graph neural networks.

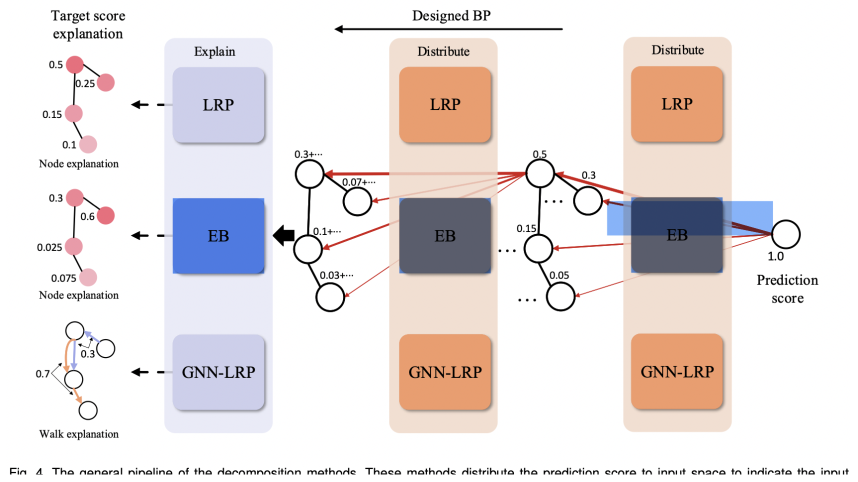

Figure 8: Comparison of Excitation Backpropagation and LRP

As an example, we generate heatmaps for one molecule in the BBBP dataset. The BBBP dataset consists of a collection of chemical compounds along with binary labels indicating their ability to penetrate the blood-brain barrier. The blood-brain barrier is a protective barrier in the central nervous system that restricts the passage of certain molecules from the bloodstream into the brain.

We train the model using an ordinary graph convolutional network. Figure 8 shows the train and test loss per epoch.

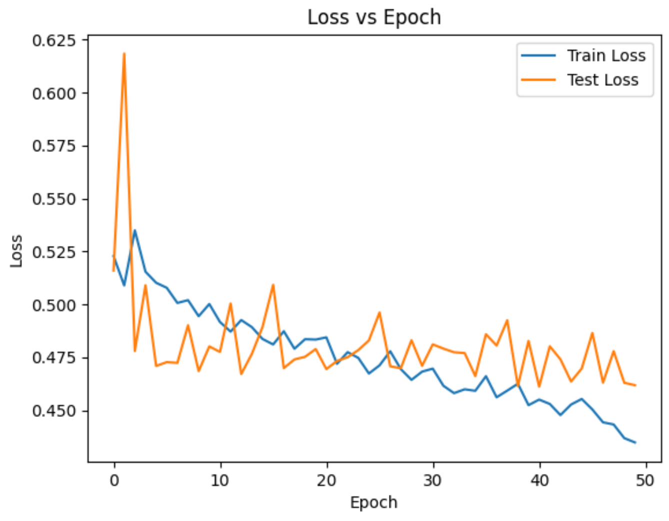

Figure 9: Loss vs. Epoch Plot for Training and Testing BBBP Datasets

As an alternative to ordinary class activation mapping, we use gradient-weighted class activation mapping, which allows the last layer before the softmax layer to not be convolutional. Additionally, it is model agnostic, allowing it to be better applied to graph structured data. [6]

Figure 9 shows the results of applying the three explainability techniques on the molecule.

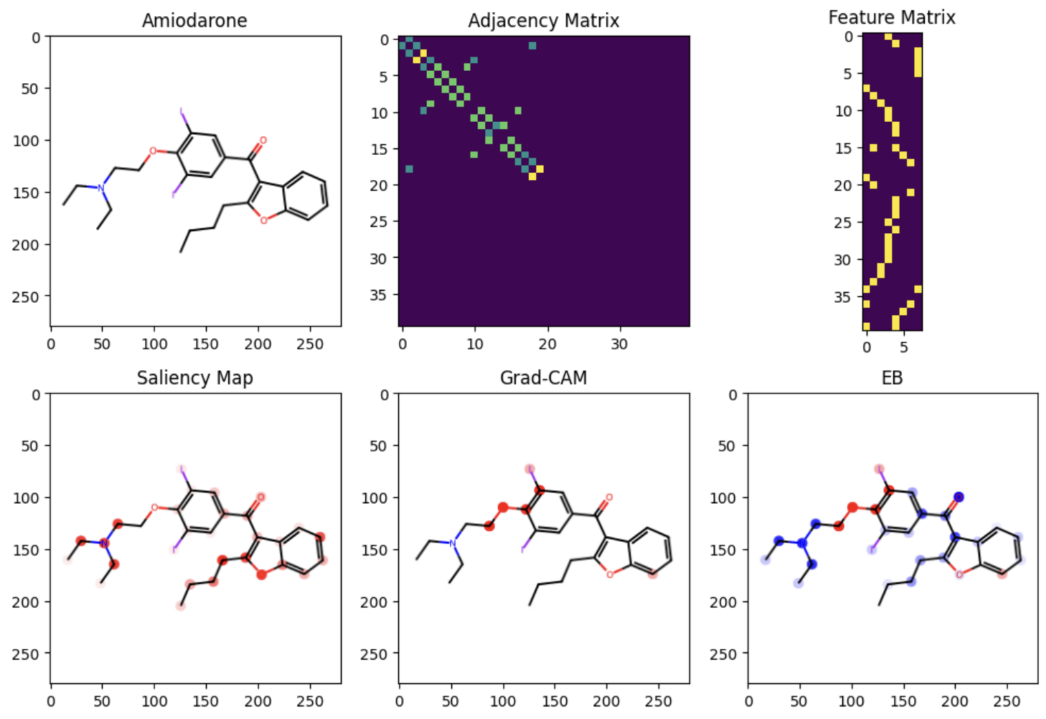

Figure 10: Plots of original molecule, adjacency matrix, feature matrix, and heat maps

Following [12], we quantify three different metrics for evaluating the explainability methods: fidelity, contrastivity, and sparsity.

Contrastivity evaluates whether the explanation captures the distinct characteristics and patterns associated with a particular class while disregarding irrelevant information. It is defined as the ratio of the Hamming distance d_h between the heat-maps $\hat{m}_0\hat{m}_1$ for positive and negative classes, normalized by the total number of atoms identified by either $\hat{m}_0$ or $\hat{m}_1$:

$$\frac{d_H(\hat{m}_0, \hat{m}_0)}{\hat{m}_0 \vee \hat{m}_1}$$

Fidelity, on the other hand, evaluates whether the explanation accurately reflects the importance and influence of nodes or edges as perceived by the GCN during the prediction process. To calculate fidelity, we looked at the difference in accuracy when we hid or removed nodes that had an importance score greater than 0.01 (on a scale from 0 to 1). We repeated this process for each method and calculated an average fidelity score for each class.

Sparsity assesses whether the explanation effectively highlights only the most influential nodes or edges, reducing unnecessary clutter and noise. We prefer sparse explanations as they provide a concise understanding of the most important elements in the graph, and they are useful when graphs are too large to inspect every node. We subtract from 1 the number of nodes in $\hat{m}_0$ or $\hat{m}_1$ that evaluate to 1 over the number of vertices:

$$1 - \frac{\hat{m}_0 \vee \hat{m}_1}{|V|}$$

Our results are summarized in Table 2. We see that excitation has slightly higher fidelity, but Grad-CAM has higher contrastivity and lower sparsity. Since we are working with molecular data, having low sparsity is not very much preferred because it is not difficult to analyze individual atoms within the molecule. Then, we might prefer expectation backpropagation to obtain the fine-grained details and specific features that contribute to the model output, or we might prefer Grad-CAM if we want to prioritize more contrast between important and unimportant subgraphs. In the context of drug discovery, Grad-CAM might be the optimal choice because there is a higher contrast between relevant

	Grad-CAM	**Excitation Backpropagation**
Fidelity	0.16	0.21
Contrastivity	102.31 ± 0.91	52.01 ± 16.35
Sparsity	9.93 ± 5.32	43.50 ± 19.92

Table 2: Comparisons of Explanation Methods for BBBP Dataset

and irrelevant substructures within the molecule, which helps researchers find similar subgraphs in other molecules.

6 Building Large Graph Datasets

Several challenges arise when it comes to finding and working with large graph datasets. Many graph datasets are not publicly available, as they are often owned by specific organizations or researchers who may have restrictions or who just neglect to share the data. Many datasets also require special permissions or collaborations for access. Additionally, several graph datasets contain sensitive information, such as social networks, personal connections, or proprietary data. As such, there is a scarcity of large-scale truthed datasets designed for rigorous evaluation of detection techniques. [2]

In this section, we set out to alleviate several of those problems by making a graph dataset for analyzing the readability of papers. To begin, we use the arXiv dataset (available on Kaggle and other platforms) which contains a sample of the vast collection of scientific papers accessible through the arXiv preprint server. [1] The arXiv dataset includes the paper's arXiv ID and DOI, which can be used to access the paper, as well as the author of the paper, the title of the paper, information about the journal in which the paper was published, the paper's abstract, and arXiv-defined categories and tags, along with some other information. Notably, the dataset includes a "comments" column which usually contains the number of pages and figures used in the paper. We can extract this information using regex.

Next, we use Semantic Scholar to generate the citation information for each article and gather more complete metadata. Semantic Scholar is an AI-powered academic search engine and research paper repository that provides an interface for researchers to search for academic articles, conference papers, and preprints. The Semantic Scholar API allows limited requests without an API key. For each article in the arXiv dataset, we send an API request to retrieve the citation information. We check if each cited article is included in the arXiv dataset and, if so, we add an edge between the two articles. The Semantic Scholar data also contains information about the paper's authors and references. However, requesting a paper's references would frequently lead to API request failures, and as a result, we decided not to do so.

Since the arXiv dataset contains over 1 million articles and the Semantic Scholar API is quite slow and failure-prone, generating the citation network took several days and required manual supervision as we had to restart when the API requests broke. The final network contained over 15 million edges and 2 million nodes. For the first edition of the dataset, we opted to only use publicly accessible methods to access the Semantic Scholar API. However, our request for an API key was later granted, so in future editions of this dataset, we can collect far more data more efficiently by simply downloading the entire Semantic Scholar dataset and extracting the needed data. This will also increase the quality of the data, our previous method caused some requests to be lost due to networking failures. Additionally, the requests reformatted some of the data, making search more difficult. Using the API key in future versions should result in a far cleaner dataset.

6.1 Data Cleaning and Exploratory Data Analysis

The space and time efficiency of storing and processing the data was not ideal, as the network is large and the Pandas library is suboptimal for handling large datasets. Because Pandas could not handle the data efficiently, it was necessary to use Polars, a similar DataFrame API in Python designed with large datasets in mind, to process the data. One reason why Polars is faster than Pandas for handling large datasets is its underlying implementation in Rust. Rust is a high-performance programming language known for its emphasis on memory safety and concurrency. The Rust implementation of Polars allows for efficient memory management and parallel processing, which improves its performance and efficiency. Polars automatically parallelizes any operations possible, distributing the work across multiple threads, to accelerate computations on modern multi-core systems. Additionally, Polars contains optimizations for common big data operations, such as filter, join, and aggregation operations. These optimizations, combined with the benefits of Rust, enable Polars to process the large-scale arXiv dataset more efficiently than Pandas. There are also some algorithmic differences: for example, it employs Single Instruction, Multiple Data vectorization techniques to process data in parallel using CPU vector registers, resulting in faster computation of mathematical and logical operations. [11]

Next, we had to tidy the data. Tidying the data proved a similarly challenging process to collecting it, and some tidying scripts run slowly even when using Polars. This seems to be a pervasive problem when working with graph datasets; as Bliss and Schmidt from MIT note, "storage, representation, and data-access techniques for graphs are often hard-coded, which makes building graph datasets a difficult, error-prone, and timely task." [2]

The next step was including readability scores for each article. We assessed the readability level of each abstract by employing two well-established readability measures: the Flesch Reading Ease (FRE) and the New Dale-Chall Readability Formula (NDC). The FRE score is computed based on the syllables per word and the number of words in each sentence. In contrast, the NDC score is determined by considering the number of words in each sentence and the proportion of 'difficult words'. Difficult words refer to those that are not included in a predefined list of commonly used words. A lower FRE score or a higher NDC score indicates lower readability. [18] Figure 11 shows the readability plots over time.

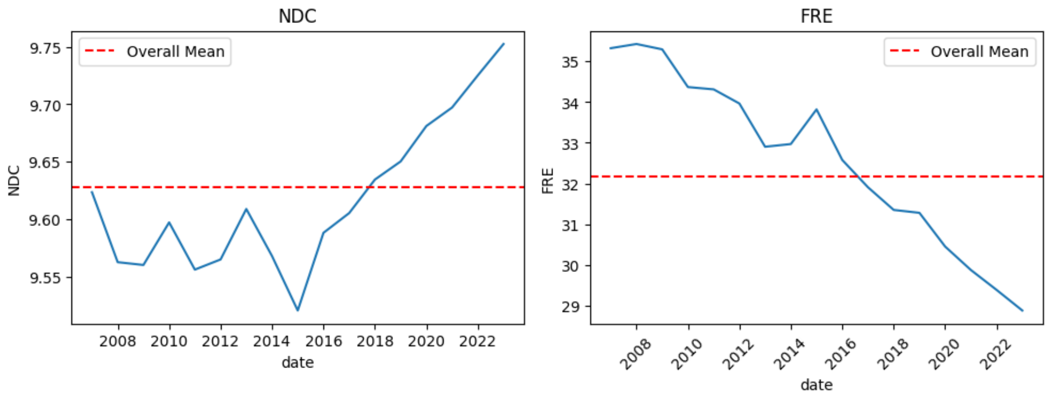

Figure 11: NCD and FRE Plots Over Time

We notice that NDC increases rapidly after 2014 and FRE decreases rapidly after 2014, extending the findings of [18], which uses articles from 2004 to 2015 to find that readability scores are decreasing over time.

Calculating the NDC and FRE for each abstract required running a relatively expensive and non parallelizable algorithm on the entire dataset. As such, it took the longest time compared to any other single step. (We can include this data in the first edition of our dataset for those who are interested in these metrics.) Next, we define a paper as readable if it had an NDC and FRE that were both above the mean. By this metric, approximately 11% of papers are considered readable.

Unfortunately, the storage method of the arXiv dataset and the API request method caused much of the data in nested format to be converted to strings. We wrote custom queries to extract relevant information from the strings. Polars was able to make each query to the dataset relatively short; however, every update to the code required a significant wait in order for the data to process, which made writing the code challenging. We found that a very challenging aspect of big data was that testing code is difficult and timely, even if each operation does not take a long time to run.

Below is a list of the columns in the final dataframe. Note that not every column was used in our analysis:

```
['','paperId','referenceCount','citationCount','influentialCitationCount',
's2FieldsOfStudy','publicationTypes','journal','authors','citations','doi',
'id','submitter','authors_right','title','comments','journal-ref','report_no',
'categories','abstract','versions','update_date','authors_parsed','license',
'NDC','FRE','filtered_citations','nodeID','date','pages','figures','readable']
```

Finally, we created the edge data. Because Semantic Scholar sends the hash that they use for each paper as an ID and sends the citation list as a list of paper IDs, we simply had to map the papers IDs to its node ID, and then remove any values in the citations list that were not contained in our node ID mapping (as this implied they did not link to any node in the dataset). Finally, we expanded each entry in a papers reference list into its own row and used the node ID as the source and reference ID for the destination of our edge list.

Given that a primary objective of this project is to address the challenges associated with finding high-quality, comprehensive graph datasets that are both detailed and readily usable. To achieve this goal, we have included our complete, refined dataset, as well as the raw dataset. By providing both versions, we aim to offer increased flexibility in constructing graphs and facilitate convenient access to the data.

6.2 Analyisis

To confirm that our dataset worked with graph algorithms, we ran a graph convolutional network on the data to classify the nodes as readable or not readable using the citations, influential citations, number of pages pages, number of figures, and reference count. In order to decide weather a paper was readable, we split the data into four categories:

- Bin 0 was for papers with papers with neither NDC nor FRE above their mean.
- Bin 1 was for papers with only NDC above its mean.
- Bin 2 was for papers with only FRE above its mean.
- Bin 3 was for papers with both FRE and NDC above their mean.

12.64% of the articles were in Bin 0, 37.44% were in Bin 1, 38.12% were in Bin 2, and 11.78% were in Bin 3.

Initially, our objective was to determine whether a paper had both its NDC (New Dale Chall) and FRE (Flesch Reading Ease) values above the mean, which would classify such a paper as "readable". This is equivalent to a paper being in bin 3. However, we faced a conundrum when we discovered that only 11.97% of the papers met this standard of readability.

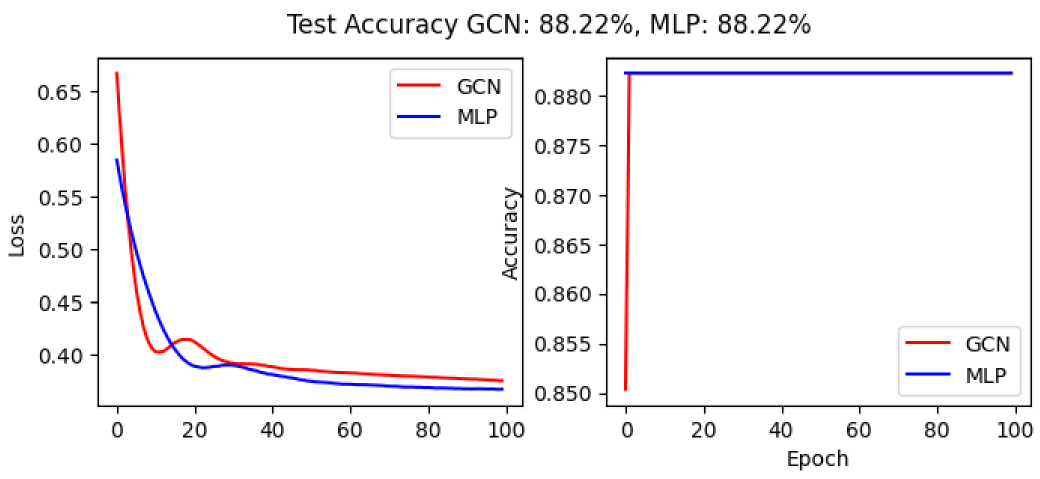

Figure 12: Classifying if a paper was in bin 3 or not

Our initial foray into applying a Graph Convolutional Network (GCN) to the data seemed promising. Both models achieved a plateau with an impressive accuracy of around 88%. Yet, upon deeper scrutiny, we found that the models were simply predicting all instances as zeroes. This skewed result was a consequence of our dataset, where less than 12% of samples were positive. Because of this, the model had insufficient information to derive meaningful patterns.

While a more complex model architecture could potentially overcome this issue, our primary objective was not to experiment with model architectures, but rather to ascertain the utility of our dataset. Thus, we decided to refine our data analysis strategy to create a more balanced categorization.

We proceeded by devising two binary classification tasks aimed at predicting whether the NDC and FRE values would exceed the mean. This approach proved to be far more effective, yielding significantly improved results.

To further validate our dataset and to test its applicability for multi-class classification, we also trained a model to predict which of our four bins a paper would fall into. This additional test allowed us to confirm the versatility of our data in supporting various types of analysis.

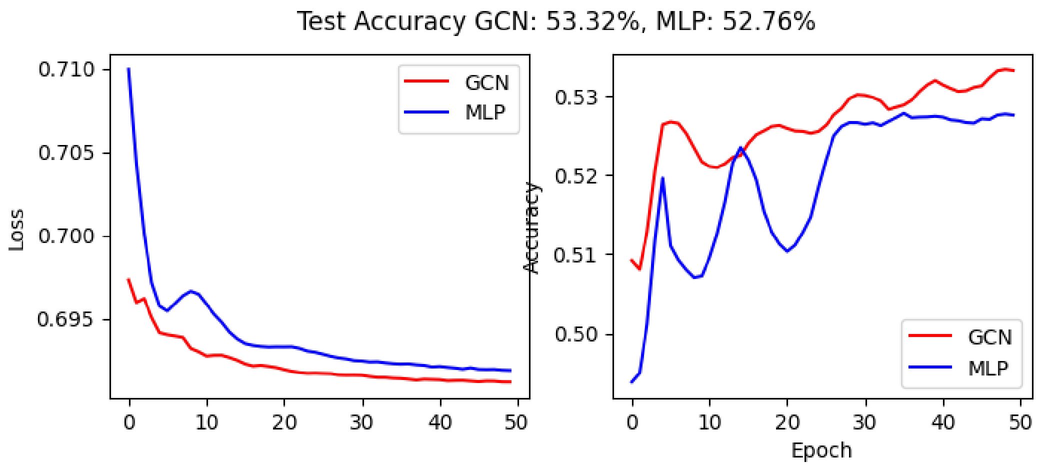

Figure 13: NDC above mean binary classification

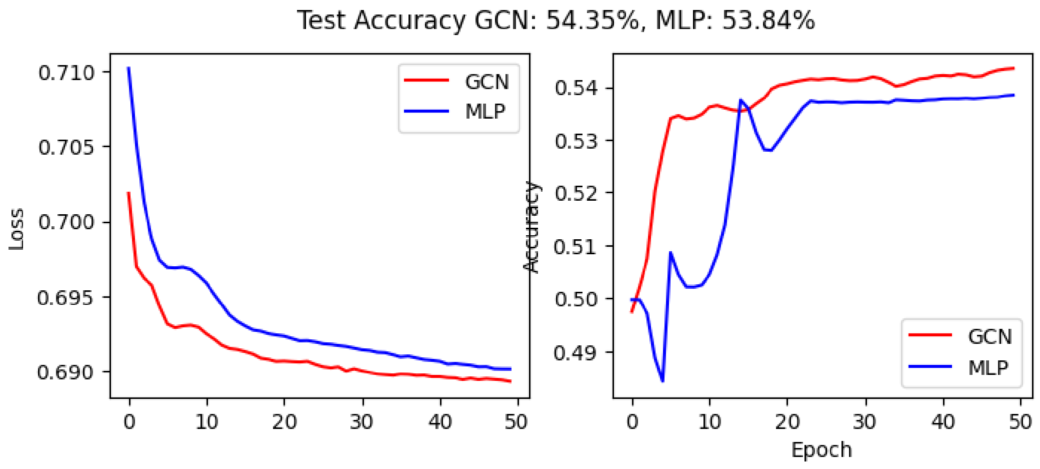

Figure 14: FRE above mean binary classification

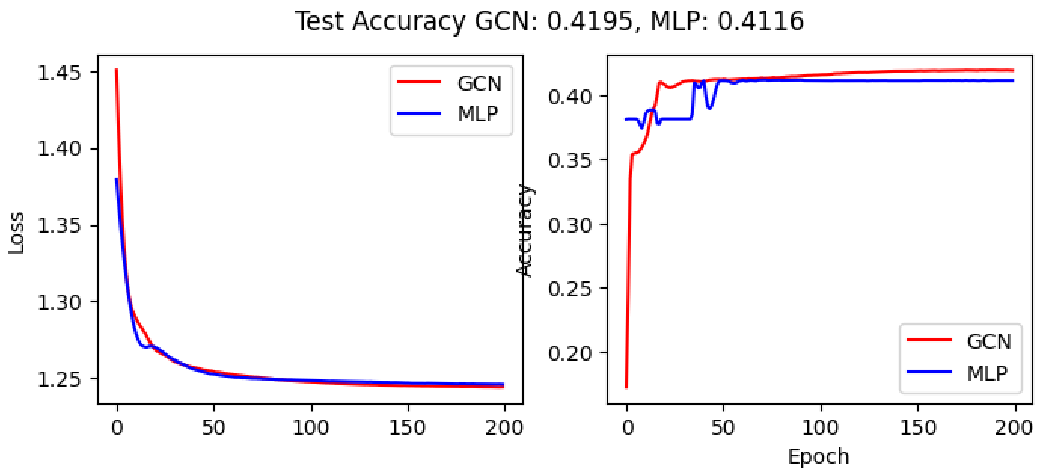

Figure 15: Multi-class model

It is clear that a more even distribution of data enhances the model's training efficiency and predictive capability. The utility of our graph dataset is evident, as the GCN consistently outperformed the Multilayer Perceptron (MLP) across our tests.

We set up the MLP as a to allow us to compare the model utility when the edge data was not used. The number of hidden layers, loss functions, and other hyperparameters were identical between the MLP and the GCN.

An important observation is that none of the models had a particularly high prediction rate (>70%) and the GCN's advantage over the MLP was only marginal. This suggests that the data we used may not have been highly predictive for the readability outcomes, and that the edge data was only mildly predictive.

These findings imply that the complexity of the initial task might have been challenging for the relatively straightforward model architectures we utilized. The difficulty lay in making predictions based on a small subset of data with weak correlations.

6.3 Next Steps

In the future, incorporating the authorship information into the model to make it a heterogeneous graph could reveal patterns and correlations that were not accessible with the document-only data. This approach could potentially improve our model's ability to predict readability scores, enabling it to capture the intricate interplay between authors' styles and text readability.

Our intention is to further enhance the robustness of our model and improve the methods by which we evaluate its effectiveness. The main goal of this phase was to develop a functional dataset, which we have successfully achieved. Potential next steps could be applying explainability algorithms outlined in Section 5 to our dataset and making further improvements to the model.

There are several avenues through which we can achieve this. First, integrating more data types could be highly beneficial. Currently, the features in our model provide limited insights into the actual content of the papers. It would be advantageous to include elements such as fields of study, although this would necessitate additional cleaning and preprocessing of data.

Another significant improvement could be achieved by introducing more complexity to the graph. This could include adding links between nodes representing the same author, as well as incorporating reference data into the dataset as a third type of edge. The inclusion of authorship data could notably enhance the model's performance, as the authorship has a significant influence on the paper's content and overall writing outcomes.

References

[1] Arxiv dataset, https://www.kaggle.com/datasets/cornell-university/arxiv.

[2] Nadya T. Bliss and Matthew C. Schmidt. Confronting the challenges of graphs and networks. *MIT Lincoln Laboratory Journal*, 20(1):1–10, 2013.

[3] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[4] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005.

[5] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the International Conference on Neural Information Processing Systems*, pages 1024–1034, 2017.

[6] T. Kasanishi, X. Wang, and T. Yamasaki. Edge-level explanations for graph neural networks by extending explainability methods for convolutional neural networks. In *2021 IEEE International Symposium on Multimedia (ISM)*, pages 249–252, Los Alamitos, CA, USA, dec 2021. IEEE Computer Society.

[7] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations*, 2017.

[8] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network, 2020.

[9] Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1506–1515. Association for Computational Linguistics, 2017.

[10] Yao Ming, Huamin Qu, and Enrico Bertini. Rulematrix: Visualizing and understanding classifiers with rules. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):342–352, 2019.

[11] Leonie Monigatti. Pandas vs. polars: A syntax and speed comparison. *Towards Data Science*, 2023.

[12] Phillip E. Pope, Soheil Kolouri, Mohammad Rostami, Charles E. Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10764–10773, 2019.

[13] GroupLens Research. Movielens datasets, 2021.

[14] Marco Ribeiro, Sameer Singh, and Carlos Guestrin. “why should i trust you?”: Explaining the predictions of any classifier. pages 97–101, 02 2016.

[15] Franco Scarselli, Marco Gori, Alessandro Crecchi, Lorenzo Garosi, and Alberto Tesi. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[16] Dennis Shen. Recommender systems with gnns in pyg, 2020.

[17] Jin Tang, Chuanren Liu, and Wayne Xin Zhao. Graph neural networks in computer vision. In *Handbook of Graph Neural Networks*, pages 681–714. Springer International Publishing, 2021.

[18] Pontus Plavén-Sigray Granville James Matheson Björn Christian Schiffler William Hedley Thompson. The readability of scientific texts is decreasing over time. *Karolinska Institutet, Sweden*, 2014.

[19] Petar Velivckovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *Proceedings of the International Conference on Learning Representations*, 2018.

[20] Petar Veličković. Graph neural networks, 2021.

[21] Yu Wang, Yuying Zhao, Yushun Dong, Huiyuan Chen, Jundong Li, and Tyler Derr. Improving fairness in graph neural networks via mitigating sensitive attribute leakage. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, aug 2022.

[22] Sam Witty, Kenta Takatsu, David Jensen, and Vikash Mansinghka. Causal inference using gaussian processes with structured latent confounders, 2020.

[23] Wang J. Du H. et al. Wu, Z. Chemistry-intuitive explanation of graph neural networks for molecular property prediction with substructure masking. *Nature Communications*, 14(1):1–12, 2023.

[24] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *International Conference on Learning Representations*, 2019.

[25] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks, 2019.

[26] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*. ACM, jul 2018.

[27] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. XGNN: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*. ACM, aug 2020.

[28] Jianming Zhang, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop, 2016.

[29] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.