technische universität
dortmund

Department of Computer Science
Chair of Data Science and Data Engineering

Ensemble methods for outlier node detection on attributed static graphs

Thi Ngoc Anh Nguyen
Matriculation Number. 230975

Master Thesis in the program Data Science
submitted for the degree of: Master of Science

1. Examiner: Prof. Dr. Emmanuel Müller
2. Examiner: Simon Klüttermann

Contents

Abstract

Graph outlier detection has become crucial due to its wide applicability in representing complex data. Given the absence of ground-truth anomalies in real-world datasets, many deep detectors, powered by graph neural networks, are employed in an unsupervised manner. Ensembling has emerged as a pivotal method for enhancing the accuracy, reliability, and robustness of unsupervised detectors. However, the computational expense associated with fitting deep outlier detectors has limited research in graph ensemble outlier detection. In this work, we present the first attempt to leverage graph sampling techniques, combined with Deep Ensemble Anomaly Detection (DEAN) concepts, to develop a scalable ensemble framework for node outlier detection in attributed graphs. Firstly, our ensemble model has demonstrated superiority over base models in various settings. Secondly, we analyze the optimal settings where the ensemble provides benefits in the graph context. Lastly, our experiments showcase the scalability, accuracy, and robustness of ensemble prediction. The proposed approach exhibits high potential for further improvement due to its flexibility in accommodating different base component architectures, thus motivating future work on graph ensembles with large-scale datasets.

1 Introduction

Graphs have become widely utilized in storing and representing complex data, underscoring their significance across various fields today. This increase in graph usage has led to a growing demand for the detection of graph outliers [1]. These outliers often entail identifying anomalous entities within a graph, such as nodes that deviate significantly, or abnormal connections between nodes, or even anomalous subgraphs [2]. The need for outlier detection spans across a wide range of applications, from ensuring data quality to enhancing security measures.

As graphs continue to evolve dynamically, the necessity for robust and efficient graph outlier detection methods becomes increasingly apparent. Traditional methods may struggle to handle the expanding size of graphs or fail to accurately identify anomaly nodes that exhibit common characteristics and behave similarly to normal nodes [3]. The advancement of graph neural networks [4] has greatly improved graph outlier detection methods [5]. While many of these methods demonstrate robustness and high performance on benchmark datasets, their deep nature often demands substantial computational resources to train deep detectors on large datasets [5]. This highlights the need for more scalable algorithms for graph outlier detection, either through optimizing the implementation of existing methods or the development of new algorithms.

In unsupervised settings, where the ground truth of outliers is unavailable, there is a necessity for highly reliable detectors when applied to real-world datasets [5]. Ensemble methods have proven to be particularly successful in the realm of outlier detection, showing that even the simplest but diverse classifiers can yield high performance when combined [6, 7]. However, ensembling in graph data encounters challenges due to the inherent complexity of graph data, which involves the relational complexities between graph entities and the computational complexities of training a single detection model on large-scale graph datasets. Hence, specialized ensemble techniques tailored to the unique characteristics of graph data are essential. Despite the paramount importance of graph outlier detection and the development of many algorithms for it in recent years, there has been limited research on graph ensemble techniques aimed at enhancing the performance of graph outlier detection methods [5, 8]. To the best of our knowledge, our work represents the first attempt to investigate the effectiveness of graph ensembles and the reliability of ensemble predictions in graph outlier detection.

This work primarily focuses on detecting anomaly nodes on attributed graphs, where anomaly detection becomes more challenging due to the diverse perspectives in defining outliers, ranging from structural and attribute contexts to their normality in local neighborhoods and globally. Utilizing a scalable and efficient outlier detection method based on the Deep Ensemble Anomaly Detection concepts, our analysis suggests that employing efficient ensemble techniques enhances the accuracy, reliability, and robustness of the final outlier prediction.

The following chapters are organized as follows. Section 2 provides an overview of various types of graph outliers, existing works in detecting graph outliers, the background of conducting ensemble outlier detection, and common evaluation methods in an unsupervised setting. Section 3 offers a comprehensive background on graph representation learning and existing approaches in detecting node outliers. Additionally, the Deep Ensemble Anomaly Detection (DEAN) framework is described. Section 4 elaborates on the methodology of conducting ensemble detection following the DEAN concept. It begins with the presentation of a DEAN model utilizing graph neural networks and then explores different techniques for conducting a graph ensemble detection model. Section 5 details the experimentation work, including the choice of baseline methods, benchmark datasets, and the employed DEAN approaches. The experiment results and findings are discussed within this section. Finally, Section 6 summarizes the achievements of this work and presents open directions for future improvement.

2 Related Work

2.1 Graph Outlier Detection

Outlier detection is a technique that aims to identify outlier data points, also referred to as anomalies, exceptions, novelties, etc., that significantly deviate from the standard, normal pattern of the remaining dataset. Depending on the context, a data point may qualify as a global outlier if it deviates from the entire dataset, or as a local outlier when considered within a subset of data. Although outliers, as the name suggests, rarely happen in the real world, detecting them is crucial for uncovering potential issues, errors or unusual patterns that contain critical information in datasets. These techniques are widely applied in real-world tasks across diverse domains, including enhancing data quality, detecting financial frauds, or identifying potential cyber attackers in network security.

Outlier detection can be approached using common statistical techniques, machine learning models (supervised or unsupervised), or clustering-based methods. The selection of a detection method depends on various factors, such as the type of data, data size, availability of real outlier examples, and the need for interpretability of the model output. In supervised settings, when a label for a normal or outlier sample is available, many classification methods can be applied for outlier detection. In contrast, when labelled data is limited, unsupervised approaches are often preferred due to their flexibility and scalability. Unsupervised outlier detection methods are the main focus of this study. These methods generally assume that normal objects form one big group or multiple groups together, whereas outlier objects do not fit into any of the common patterns of those groups. This allows the identification of hidden or unseen outlier patterns. An outlier detection algorithm can produce a score that reflects the level of outlierness; for instance, a higher score indicates a higher likelihood of being an outlier. Alternatively, it may provide a binary label indicating whether a data point is an outlier.

Graphs stand out as one of the most powerful data structures for modelling complex systems. A graph $G = (V, E)$ comprises a set of nodes V and a set of edges E. The connection between node $u \in G$ and node $v \in G$ is denoted as an edge $(u, v) \in E$. The natural variations in the properties of nodes and edges are attributed to different types of graphs. For instance, a plain graph solely contains information about the relationships between nodes. In contrast, in attributed graphs, either nodes or edges include additional values that describe their respective properties.

Graph outlier detection is vital for discovering irregular patterns or identifying abnormal nodes or anomalous connections within graph datasets. While sharing some characteristics with traditional outlier detection methods, a graph outlier detection method should address the unique challenges and characteristics posed by graph-specific challenges. Firstly, it needs to account for various aspects of the graph structure, including its size, the presence of node or edge attributes, whether the dataset is represented as a single graph or composed of many small graphs, and how frequently nodes and edges are updated in the graph. Secondly, graph is often considered as high-dimensional data, graph search space is therefore huge due to a high number of possible graph substructures. Thirdly, graph outliers can take various forms, often derived from downstream application perspectives. Various categories of graph outliers include:

- Node outliers: These are nodes that significantly deviate from others in terms of node-level metrics, such as node degree or centrality. For example, in social networks, a node outlier might be a user with an exceptionally high number of connections.
- Edge outliers: This refers to an unexpected connection between two nodes or an unusual edge's property. For instance, in a citation dataset, an edge outlier could be high-frequency citations between two authors working in different organizations and domains.
- Subgraph outliers: This involves a subset of nodes, or a subgraph, behaving anomalously when considered as a collection, even if each node and edge is viewed as normal. For example, in an e-commerce dataset, bot users may tend to post positive reviews about certain products and negative reviews about many others.
- Structural outliers: These are anomalies apparent only when considering the graph structure. For example, an outlier edge that connects two different communities in graphs.
- Contextual outliers: Nodes or edges that are normal in general but become anomalies under specific contexts or conditions.
- Global outliers: Nodes with attributes significantly different from the overall pattern of the entire graph.
- Community outliers: Nodes with attributes significantly different from their neighbourhood.
- Temporal outliers: In dynamic graphs like transportation networks, unusual events occurring at specific time points may be related to outlier nodes or edges.

As graphs are complex and graph outliers can be defined in different ways, these challenges require a detection algorithm to be not only effective but also efficient and scalable. In addition, because obtaining labelled outliers in real-life datasets is expensive and, even when those labels are available, the number of outliers is significantly smaller than normal data points, many detection methods are developed in an unsupervised manner.

Early work on graph outlier detection often relies on statistical methods to extract features directly from the graph structure. This involves transforming the graph-based structure into multidimensional tabular data and subsequently applying outlier detection algorithms to the new table-based structure. In the context of detecting node outliers, node features can be mined from the neighborhood, including metrics such as node degree or clustering coefficient.

An adjacency matrix A represents the connections between every pair of nodes in a graph. Given a graph with n nodes, A is an $n \times n$ matrix with non-negative entries. The problem of finding edge outliers becomes the challenge of identifying anomalous entries in matrix A [9]. This method makes use of nonnegative matrix factorization [10], involving the decomposition of A into matrices U and V, such that $A \approx UV^T$, to minimize the objective function $\|A - UV^T\|$, with constraints $U \geq 0$ and $V \geq 0$. The low-rank approximation UV^T assumes the modeling of normal data in G; meanwhile, the residual matrix $R = A - UV^T$ serves as a strong indicator for entries that do not adhere to the low-rank assumption. Therefore, the outlier scores for edges are determined by the absolute values in matrix R.

Another variant utilizes the node features matrix $X \in \mathbb{R}^{n \times d}$ to detect outlier edges using shared matrix factorization [11]. The idea involves finding matrices U, V, and W that optimize the approximations: $A \approx UV^T$ and $X \approx UW^T$. Subsequently, anomalous edges can be identified from the residual matrix $R_A = A - UV^T$, while anomalous nodes can be determined using $R_X = X - UW^T$.

An alternative approach [12] utilizes the Minimum Description Length (MDL) and the principle of the SUBDUE system to detect outlier subgraphs. SUBDUE [13] is a method of detecting repetitive patterns in graphs. MDL is the smallest number of bits required to encode a piece of data; SUBDUE approximates this value for any subgraph and finds the best substructures to compress a graph. If S is a repeated substructure in graph G, then G can be compressed by replacing S with a single node. The best substructure S is the one that minimizes the value of $F1(S, G) = DL(G|S) + DL(S)$, where $DL(G|S)$ is the description length of G after compressing with S, and $DL(S)$ is the description length of substructure S. A common subgraph S is expected to have a low value of $F1(S, G)$. Finding substructures that have high values of $F1(S, G)$, which indicates an uncommon pattern, transforms into the issue of identifying subgraph outliers.

In dynamic graphs, outliers can be defined in numerous ways, and their detection is challenging due to the combination of time and structural changes. This involves identifying the time point when unusual events occur and identifying a set of nodes or edges contributing to the abnormality of these events. A common approach is monitoring a graph characteristic over time using some distance measurements and triggering an outlier alert when the change exceeds a threshold. Another set of methods treats temporal graphs as time-series of graphs, representing each graph snapshot as a vector component. The detection is then conducted on the time series of graph vectors using standard auto-regressive moving average (ARMA) techniques for outlier identification.

2.2 Ensemble Outlier Detection

Ensemble analysis is frequently utilized in data mining to reduce the dependence of the final prediction on a specific data locality or a particular model choice. Instead of relying solely on a single model, ensemble methods combine diverse predictions from various algorithms (base models) to generate a final output, often resulting in a more accurate and robust prediction. When certain algorithms excel in one subset of data samples while others perform better in another subset, an outlier ensemble proves beneficial as it mitigates the uncertainty in predictions associated with a single base model.

Ensemble methods are generally classified into two main categories: model-centric ensemble and data-centric ensemble [11]. In the model-centric ensemble category, base models are constructed using various algorithms, different hyper-parameters of the same algorithm, or diverse randomized initializations of the same model. Conversely, in data-centric ensemble, base models are derived from the same algorithm but applied to different variants of the training dataset. Although these categories appear distinct, data-centric ensemble can be considered a subset of model-centric ensemble when the random sampling of data is integrated into the model execution process.

Base models can be implemented independently of each other, with all predictions subsequently combined into the final output. Alternatively, base models can be executed sequentially, wherein each subsequent model refines the predictions of the previous one. In this scenario, the final output may either be a combination of all base models or the result solely from the final base model.

In statistical learning theory, a model is referred to as an estimator $\hat{\theta}$, or an estimated function, of the true underlying process θ that generates training data [14]. Bias and variance are two important properties of an estimator as they are closely related to machine learning concepts of model capacity, overfitting, and underfitting. Bias, $bias(\hat{\theta}) = \mathbb{E}(\hat{\theta}) - \theta$, measures the expected deviation of the estimator from the true value. Variance, $Var(\hat{\theta})$, measures the fluctuations of estimations for a particular data point after many independently resampling training datasets. A high variance estimator is sometimes referred to as an unstable estimator because it tends to produce large fluctuations in predictions with small changes in the input data.

The principle of Occam's razor and bias-variance tradeoff [15] recommends selecting a machine learning model that fits the data well while maintaining low complexity. This principle suggests choosing the simplest among competent hypotheses. A model that inadequately fits the training data, leading to the under-fitting problem, tends to exhibit high bias. While a model that performs poorly on new data suffers from the over-fitting problem, tends to have high variance. The relationship between the bias-variance tradeoff and model optimization, aimed at finding the optimal balance, is depicted in Figure 1. Ensemble analysis can be explained from the perspective of optimizing the bias-variance tradeoff, achieved through methods that either reduce bias or variance.

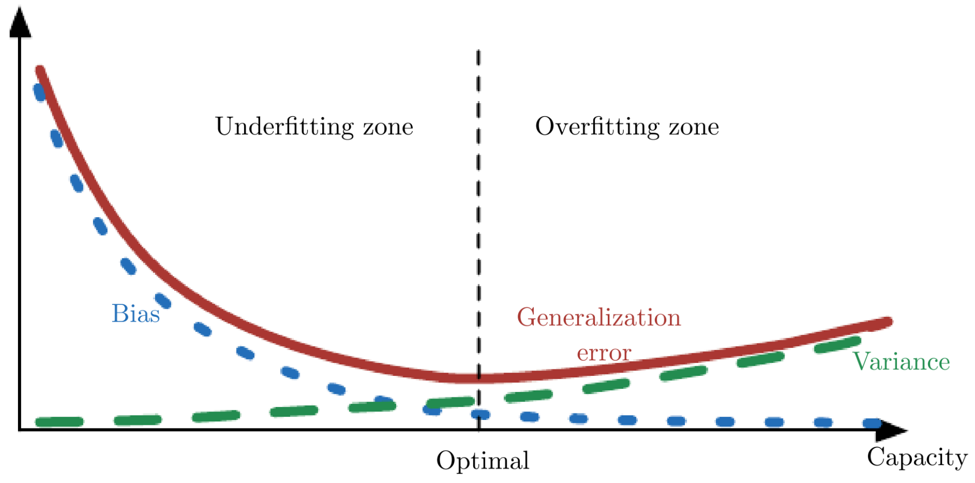


Figure 1: Relationship between bias-variance tradeoff and model optimization [14].

Two popular methods for constructing an ensemble are boosting and bagging. Boosting [16], aimed at reducing bias, iteratively adjusts the base models using the ground truth of the data to improve their performance compared to previous iterations. On the other hand, bagging [17, 18] aims to decrease variance by averaging a set of random variables, such as different predictions from independent detectors. Methods that focus on reducing variance are frequently employed in outlier ensemble techniques for constructing unsupervised methods.

When designing an ensemble model, the initial step involves defining methods for constructing base models. Subsequently, selecting an appropriate combination method that effectively integrates all predictions is crucial.

Variance reduction methods

A typical example where the ensemble generally performs better than most of the base detectors, as the variance in base models is (expectedly) reduced in the ensemble combination, is illustrated in Figure 2. The incremental improvements of the ensemble vary depending on the trade-off between the quality of the base detectors and their stability. In general, base models are expected to perform adequately and exhibit high variance in their predictions, potential conditions that may lead to an outperformed ensemble model.

Parametric ensemble is the most straightforward approach to conduct randomized instantiations of base models, which involves selecting different choices of model parameters to build a base detector. The method is an implicit model-centric ensemble that reduces variance by averaging outlier scores.

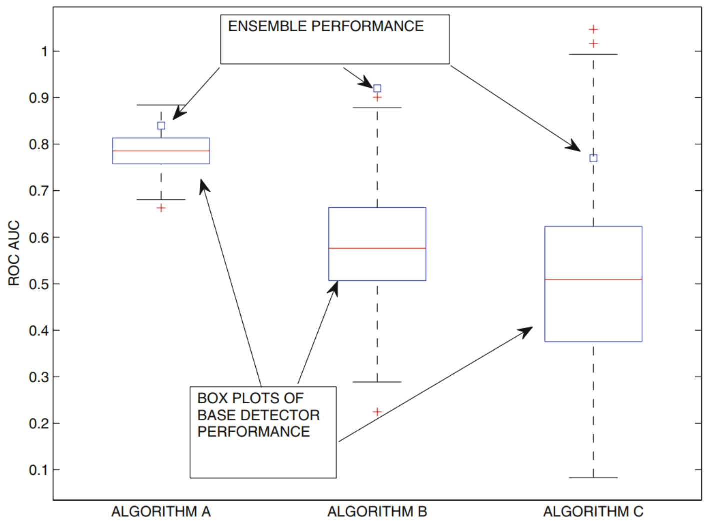


Figure 2: Illustration of the performance of ensembles in relation to the quality and stability of base detectors [11].

Feature-bagging [19] is a popular approach for detecting outliers in high-dimensional datasets. The method randomly samples different subsets of features and model base detectors from the low-dimensional projection. Subsequently, outlier scores are combined using the averaging or maximization function. However, when data features are highly correlated, therefore imposing high correlations between base detectors, may result in a low variance reduction of ensemble combination.

In bagging (Bootstrap Aggregation) [17, 18], different training sets, known as bootstrap samples, are generated from the original dataset with replacement. Each bootstrap sample is then used to train a base detector. Random Forest [20] is one of the most well-known ensemble variants of bagging. Subsampling is another ensemble method involving the creation of multiple data subsets to train base models. Unlike bagging, subsampling iteratively selects a smaller subset of data without replacement.

Combination methods for outlier ensemble

A combination function integrates multiple outlier scores from base detectors into a final score. Different combination functions may yield varied performance across settings due to their differing effects on bias and variance reduction. Methods utilizing the **average** or **median** values are relatively stable and often effectively reduce variance in outlier detection. Conversely, the **maximization** function, while less stable than average and median, performs better when combined with ranking scores. While the maximization function initially appears to overestimate scores, it has been demonstrated with a capacity to reduce bias [11], however, may simultaneously increase variance.

Raw outlier scores can be unstable sources of information due to their outputs stemming from independent processes, often resulting in varying value scales. Consequently, the interpretability of the degree of outlierness cannot be inferred solely from absolute outlier scores. For instance, the method of Local Outlier Factor [21] may yield ∞ score values. In such cases, outlier scores are often normalized before the combination step, using normalization methods such as Min-Max or Z-score [22] or converted to probabilities [23]. Additionally, combining raw scores with rank scores is beneficial because ranks are more robust than raw scores in the sense that they maintain the relative informativeness of outlierness among data samples and do not rely on absolute values.

Intriguingly, the former authors of feature-bagging [19] underscore the conceptual similarity between combining outlier scores and challenges encountered in meta-search engines [24, 25]. They propose combination methods solely based on ranking outputs from multiple base detectors. For example, in the Breadth-First approach, the 1^{st} ranked outliers among all predictions are combined first, followed by the 2^{nd} ranked outliers, and so on.

2.3 Evaluation method

Outlier detection can be viewed from a classification perspective, where a detector model provides binary output for a data point, either classifying it as an outlier (Positive) or not (Negative). Therefore, evaluating the performance of an outlier detection model can be approached using standard evaluation metrics in the classification domain. Given the top-k predictions, *precision@k* emerges as one of the most popular metrics, measuring the proportion of actual true positive predictions among all positive predictions made by the model.

		Predicted	
		Positive	Negative
Actual	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Figure 3: Confusion matrix representation for a binary classification model

Together with *precision*, the true positive rate (TPR) and false positive rate (FPR) are often derived from the confusion matrix as follows:

$$\begin{aligned} Precision &= \frac{TP}{TP+FP} \\ TPR &= \frac{TP}{TP+FN} \\ FPR &= \frac{FP}{FP+TN} \end{aligned} \tag{2.1}$$

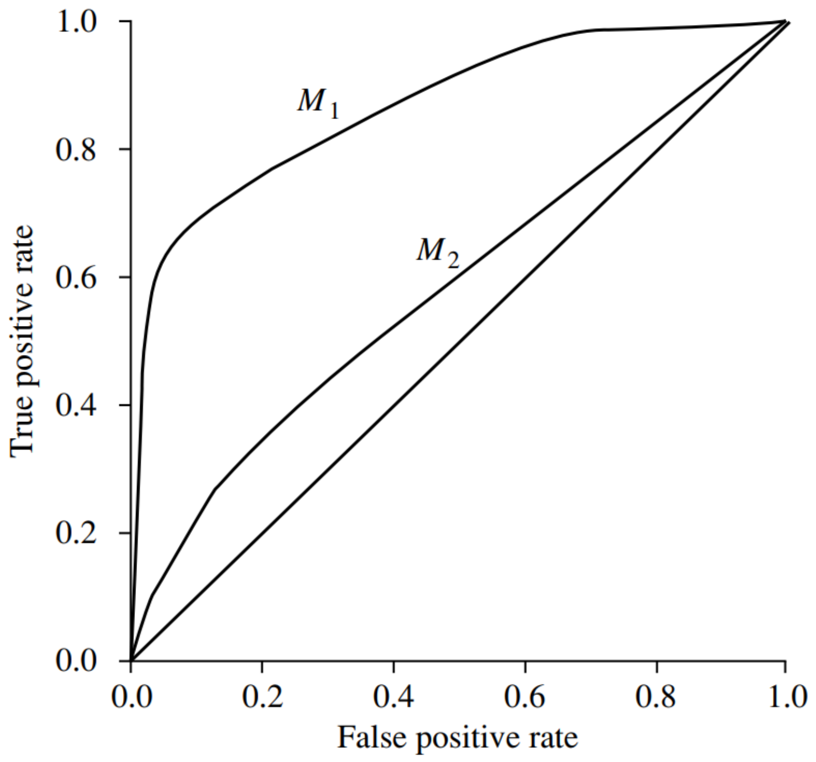


Figure 4: ROC curves of two classification models, M_1 and M_2. The diagonal line indicates a "random guessing" model [28].

However, *precision@k* is sensitive due to the rarity of outliers. As a result, it is applied to highly imbalanced datasets and often penalizes the detection model excessively [26]. For instance, in a dataset of 100 points containing only 2 actual outliers, if an outlier detection method ranks those true outliers at the 3^{rd} and 4^{th} positions, the *precision@2* value will be 0. A common approach is to average the performance of a detector over various values of k.

The receiver operating characteristic (ROC) curve offers a more advanced method of evaluating a model, plotting the true positive rate (TPR) against the false positive rate (FPR). This curve visualizes the trade-off between the model's ability to accurately detect true outliers and its tendency to mistakenly identify normal points as outliers. The area under the ROC curve [27] serves as a measure of model accuracy. The area under the ROC curve (AUC) is utilized to compare the performances of the two models numerically.

The ROC-AUC, computed from predicted outlier scores against truth values, reflects the overall performance of the model on both positive and negative examples. An AUC value of 1 reflects a perfect model that makes completely correct predictions, while an AUC value of 0.5 indicates that the model has no capacity to distinguish between the two classes.

3 Background

3.1 Graph Representation Learning

Graph representation learning (GRL) methods play a crucial role in extracting meaningful information from graph-structured data and addressing complex relationships to enhance the performance of graph-related problems. Machine learning tasks on graphs are often categorized based on the types of problems they aim to solve [29]. Some key graph-related tasks to make predictions at the node or edge level include node classification, link prediction, and graph clustering or community detection. On the graph level, predictions are made for the entire graph, involving classification, regression, or clustering problems. While traditional learning approaches achieve promising results in graph analysis by leveraging well-established graph statistics and kernel functions to extract structured features from graph data, followed by the application of machine learning algorithms on these hand-crafted features to make final predictions [30], they have limitations. Although these approaches can yield accurate predictions, their qualities depend heavily on feature engineering steps, requiring an in-depth understanding of the graph data, the choice of kernel method, and domain knowledge. Consequently, these methods may lack scalability when dealing with large-scale graphs and may not be easily reusable for new graphs.

3.1.1 Node Embeddings

Modern approaches aim to learn graph representation from its structural information, eliminating the need for manual steps. The central idea is to encode a node, subgraph, or graph, as a low-dimensional vector while preserving the structural information, such that geometric distance in the latent space reflects the structure of the original graph [29]. For example, nodes that are directly connected tend to be closer in the latent space. Node embedding is one of the most popular task-independent frameworks for learning node features. The learned features, induced from the graph's structure, are expected to encompass the characteristics and connectivity among entities in the graph. Node embedding can be approached through an encoder-decoder framework, factorization-based methods (deterministic measures), or random walks approaches (stochastic measures of neighborhood overlap).

The encoder-decoder framework utilizes graph structure to derive the learning process, involving two learning phases [29]. Firstly, it learns an *encoder* model that maps each node into a low-dimensional embedding: $ENC : \mathcal{V} \rightarrow \mathbb{R}^d$, where $\mathcal{V}$ is the set of nodes, and d is the dimension of embedding space. Secondly, a *decoder* takes node embeddings as input and attempts to reconstruct information about the immediate neighbourhood of each node. Considering the *encoder* as a mapping function that maps a node $v \in \mathcal{V}$ to its embedding vector z_v: $ENC(v) = z_v \in \mathbb{R}^d = \mathbf{Z}[v]$, where $\mathbf{Z}$ is the embedding matrix (or lookup table) for all nodes. One way to define the *decoder* is as a model that predicts similarity or connectivity between pairs of nodes, represented as $DEC : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+$. Assuming $\mathbf{S}[u, v]$ is the measure of connection between nodes u and v in the original graph structure (e.g., indicating whether u and v are connected or their similarity on a scale from 0 to 1), the learning objective of the *decoder* is to optimize the equation $DEC(z_u, z_v) \approx \mathbf{S}[u, v]$. The empirical loss function is derived by measuring the discrepancy between the output $DEC(z_u, z_v)$ of the *decoder* and the true value $\mathbf{S}[u, v]$ for all training node pairs (u, v).

The encoder-decoder architecture offers a flexible system for defining various node embedding methods based on: (i) a function measuring pairwise similarity, (ii) an encoder function, (iii) a decoder function, and (iv) a learning objective or loss function. Once the learning pipeline is optimized, the trained encoder can generate node embeddings as input features for downstream tasks [31], including various graph outlier detection problems. For instance, node embeddings can be treated as tabular data, and the Local Outlier Factor (LOF) method [21] can be applied to detect node outliers. Alternatively, a prediction model can be built to predict connections between pairs of nodes, serving as indicators for edge outlier detection.

Another perspective, based on matrix-factorization methods which are widely applied for dimensionality reduction problems, can be used to learn low-dimensional node representation [31]. Given a deterministic similarity matrix for every pair of nodes $\mathbf{S} \in \mathbb{R}^{n \times n}$; an *encoder* model ENC, with trainable parameters, that generates node embeddings z; the *decoder* is a function that computes the inner product of two embedding vectors: $DEC(z_u, z_v) = {z_u}^\top z_v$. These factorization-based methods [32, 33, 34] aim to learn node representation in a way that the inner product between node embeddings approximates pairwise similarity measures in the ground truth matrix $\mathbf{S}$, minimizing the difference $|DEC(z_u, z_v) - \mathbf{S}[u, v]|$.

These presented embedding methods are unsupervised by nature because the learning process depends only on the graph itself. The model is trained over a set of nodes in a way that optimizes the reconstruction of pairwise similarity between nodes. However, there are methods that can incorporate external ground truth, such as when labels are available in the node classification problem, to learn node embeddings in a fully supervised manner [35, 36] or in a semi-supervised manner [37].

Node embedding is often categorized as a shallow embedding approach, where the encoder is trained to learn a unique embedding vector for each node in the graph. However, it has limitations in terms of generalization for unseen nodes, especially when the graph is updated with new information or when processing an entirely new graph. This scenario is referred to as the *transductive* learning setting [36], where the trained model can only generate embeddings for nodes that appeared during the training phase. Another limitation is the lack of ability to incorporate existing node or edge features, which often contain rich information, into the learning process, as these approaches consider only structural information from the graph. Lastly, since the encoder model acts as a simple lookup function for all nodes, these methods may not scale well as the number of nodes grows, leading to an increase in the number of parameters and computational expenses [31].

3.1.2 Graph Neural Networks

A Graph Neural Network (GNN) [38] is a general framework that extends methods of deep neural networks (DNNs) to graph-structured data. Although GNN is primarily designed to learn node representations, it can also be used to generate representations at the subgraph or graph level. Moreover, it can collaborate with downstream tasks, taking the entire graph as input and directly outputting predictions for desirable tasks.

Neighborhood aggregation is the main idea behind GNNs [38, 31], the concept is similar to the convolution operator employed in convolutional neural networks (CNNs). Analogous to CNNs learn higher or abstract representations of objects in an image by applying convolution operators to different local parts of the image, GNNs generate embeddings for a node by iteratively aggregating information from its surrounding neighborhood nodes. Each node within a graph is characterized by its inherent features and the set of nodes to which it is directly connected [38].

Message Passing is a generalization framework of node aggregation that defines how nodes in a graph exchange information and update their representation. This idea is originally known as Message Passing Neural Networks (MPNN) [39]. In this framework, information from each node is considered as a message, and a set of connected nodes exchange messages with each other. After receiving and aggregating messages from neighbor nodes, each node updates itself with the new information. A message-passing for a node in the graph consists of three main steps:

- Node message initialization: Information from each neighbor node is considered a message.
- Node aggregation: A node aggregates all messages from its neighborhood, summarizing them using a permutation-invariant function in which the order of messages is irrelevant.
- Node update: After receiving new information, a node updates itself by combining current attributes and aggregated messages.

Intuitively, after each message-passing iteration, each node learns about its neighbors. Then, in the next step, it learns about the neighbors of its neighbors. After many iterations, each node is able to understand its position in the whole graph and repeatedly update itself. This way, the final representation is derived from the internal structure of the graph.

A GNN model can consist of any K layers, where each layer corresponds to a message-passing iteration that runs over all nodes in the graph. Given input with a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and a node features matrix $X \in \mathbb{R}^{|\mathcal{V}| \times d}$. z_u is the final representation of node u after the training process. A message-passing iteration at iteration $k + 1$ can be defined as [29]:

$$\begin{aligned} h_u^{k+1} &= UPDATE^k(h_u^k, AGGREGATE^k(\{h_v^k, \forall v \in N(u)\}) \\ &= UPDATE^k(h_u^k, m_{N(u)}^k) \end{aligned} \tag{3.1}$$

Where h_u^k corresponds to the embedding vector of node u after k iterations. The initial embeddings are default to initialize using node features, $h_u^0 = x_u$. $N(u)$ is the set of neighbor nodes for node u. At iteration $k+1$, the target node u will gather messages passing from its neighbors $N(u)$, compress them into a single message using the $AGGREGATE$ operator, and then use the $UPDATE$ operator to merge aggregated information with its current state h_u^k, forming a new embedding state h_u^{k+1}. The calculations in Equation 3.1 are repeated until the embeddings converge. Once the model has converged, the output of the last layer, iteration K, is output as learned embeddings for all nodes: $Z_u = h_u^K$.

The GNN model requires initial vectors as node embeddings, often leveraging node attributes for this step. However, in circumstances where node attributes are unavailable, various alternatives become available. The original work [38] initializes a random vector for h^0. An alternative option derives node features from the graph structure, using an adjacency matrix A where each row corresponds to the feature vector of each node, or constructing a one-hot encoding vector for each node, with node degree used to fill the non-zero element. A promising approach involves utilizing pre-trained vector embeddings, which contain information about the graph structure, generated by shallow methods such as DeepWalk [40] or Node2Vec [41]. This approach boosts the learning of the GNN model and helps it converge quickly.

The $UPDATE$ and $AGGREGATE$ operators can be any arbitrary differentiable function; it could be as complex as a neural network layer [38, 42], or a non-linear transformation such as the $tanh()$ function, or as simple as the max or $average$ operator. Different choices of aggregation or update functions lead to different variants of GNNs. In general, when defining a GNN architecture, it is necessary to specify the answers to the following three questions: How to prepare a message for a node? How to summarize messages from neighbors? How to update the current state with new information?

The most simplified GNN model, proposed by [38], is defined as:

$$h_u^k = \sigma(W_{self}^k h_u^{k-1} + W_{neigh}^k \sum_{v \in N(u)} h_v^{k-1} + b^k) \tag{3.2}$$

Deconstructing the equation, it shares many similar characteristics with a traditional multi-layer perceptron (MLP) model. The weight matrices W_{self} and W_{neigh}, alongside the bias vector b, constitute the model's trainable parameters. The aggregation function sums up all messages from the neighbor set to generate the message $m^k_{N(u)}$, with $m^k_{N(u)} = \sum_{v \in N(u)} h_v$. Subsequently, this aggregated message is integrated with information from the preceding iteration. The update operator applies a non-linear transformation for the final aggregated message and updates the current state of the target node u. The set of parameters, W_{self}, W_{neigh}, and b, can either be shared across all layers of the model or be separately optimized for each layer. However, in contrast to standard DNN models, the number of parameters and the depth (number of layers) in GNNs do not directly indicate the complexity or computational cost of the model. The complexity of GNNs is derived from the computational graph when performing message-passing for each node at each layer.

Each graph neural network at layer-k can be reformulated from Equation 3.2 in the following matrix form. It should be noted that in the subsequent section, the bias term $\mathbf{b}$ may occasionally be omitted for notation simplicity:

$$H^{(k)} = \sigma(W^{(k)}_{self} H^{(k-1)} + AH^{(k-1)} W^{(k)}_{neigh} + b^{(k)}) \tag{3.3}$$

where $H^{(k)} \in \mathbb{R}^{|\mathcal{V} \times d}$ denotes matrix of node representations at k^{th} layer, $H^{(0)} = X$.

The Graph Convolutional Network (GCN), as introduced by Kipf and Welling [43], stands as one of the most widely adopted and foundational models in the field. Through the utilization of the *self-loop* technique, which implicitly incorporates the update operator within the aggregation function such that $h^k_u = AGGREGATE(\{h^{k-1}_v, \forall v \in N(u) \cup \{v\}\})$. The architecture of GCN models exhibits a notable property of parameter sharing, wherein W_{neigh} and W_{self} are consolidated into one identical weight matrix. The symmetric normalization acts in balancing information aggregation, particularly in scenarios where certain nodes possess significantly high degrees. The message-passing mechanism in GCN is defined as:

$$h^k_u = \sigma(W^k, \sum_{v \in N(u) \cup \{v\}} \frac{h_v}{\sqrt{|N(u)||N(v)|}}) \tag{3.4}$$

In matrix form, the *self-loop* operator is simply implemented by adding the identity matrix I to A such that $\hat{A} = I + A$. Additionally, symmetric normalization is accomplished by multiplying with the diagonal node degree matrix D, resulting in $A' = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. Finally, where $\tilde{A}$ represents the normalized version of A after applying these two operators, a GCN model follows the following layer-wise update rule:

$$\begin{aligned} H^{(k)} &= \sigma(\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} H^{(k-1)} W^{(k)}) \\ &= \sigma(\tilde{A} H^{(k-1)} W^{(k)}) \end{aligned} \tag{3.5}$$

While GCN employs a simple averaging aggregation method, the Graph Attention Network (GAT) [44] adopts attention mechanisms to learn relative weights between two connected nodes. GAT aggregates the weighted sum of information from neighbors, defined as $m^k_{N(u)} = \sum_{v \in N(u)} \alpha_{u,v} h_v$. The attention weight $\alpha_{u,v}$ indicates the connection strength between node v and the neighboring node u.

$$\alpha_{u,v} = \frac{\exp(f(\mathbf{a}, \mathbf{W}, h_u, h_v))}{\sum_{v' \in N(u)} \exp(f(\mathbf{a}, \mathbf{W}, h_u, h'_v))} \tag{3.6}$$

Here, $f(.)$ represents any differentiable function that produces a scalar value, vector $\mathbf{a}$ and matrix $\mathbf{W}$ are trainable parameters of GAT models [29]. Furthermore, various variants implement a multi-head attention mechanism [45] in the graph convolutional operation to increase the representation capacity of GAT models.

During the message-passing process in deep GNNs, aggregating information from a large number of nodes often results in an exponential increase in computational complexity and possibly *over-smoothing* [46], the issue where embeddings for all nodes become identical after too many rounds of message-passing. This issue is known as neighbor explosion.

An alternative to aggregating over the entire neighborhood of a node, GraphSAGE [36], a general inductive learning framework, samples a fixed number of neighbor nodes while performing the graph convolution in a large-scale graph. Besides, it trains a set of aggregator functions that leverage node features to generate embeddings, instead of learning a constant embedding vector for each node, thereby having the capacity to generalize for unseen nodes.

In a different perspective, GraphSAINT [47] utilizes a *graph sampling* technique to train deep GCNs on large-scale graphs, employing an inductive learning approach. In contrast to GraphSAGE, which uses *layer sampling* techniques, GraphSAINT takes a distinct approach. At each iteration, a subgraph is sampled from the original training graph, and subsequently, a full GCN operation is performed on the sampled subgraph. This concept bears resemblance to the fundamental technique of using mini-batch training in DNNs. GraphSAGE and GraphSAINT are state-of-the-art methods to alleviate the "neighbor explosion" problem.

The Graph Autoencoder (GAE) represents another model used for graph representation learning. It extends the autoencoder architecture to generate a low-dimensional representation and incorporates GNNs in the encoder to handle graph input. Given an undirected and unweighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a node features matrix X, the non-probabilistic variant of GAE [48] utilizes a GCN encoder and an *inner product* decoder. It computes the embeddings $Z = GCN(X, A)$, where A is the adjacency matrix, and the reconstructed adjacency matrix $\hat{A} = \sigma(ZZ^T)$. During training, the model optimizes the parameters of the GCN layers to minimize reconstruction errors which are the difference between A and $\hat{A}$.

Node embeddings aim to learn low-dimensional vector representations for individual nodes, enabling the application of off-the-shelf machine learning algorithms to make predictions on graph datasets. In contrast, GNNs define a framework for addressing a wide range of graph-related tasks in an end-to-end manner. In this training approach, the GNN model takes graph data as input and generates predictions for downstream tasks without intermediate steps. This is achieved by defining a suitable loss function based on the specific problem type. Subsequently, the model computes the loss value and performs backpropagation to optimize the model's parameters. Simultaneously, GNNs facilitate the dual processes of learning meaningful representations and making predictions.

3.2 Node Outlier Detection on Graphs

This section presents a comprehensive review of modern techniques that are specialized in node outlier detection. The main focus of this work are unsupervised methods that utilize GNN as the backbone model to handle graph-based structure datasets.

3.2.1 Shallow methods

Shallow methods refer to techniques that involve transferring graph structure to a new representation that can be handled by off-the-shelf outlier detection methods. Initially constructing an outlier detection model with shallow methods is a conventional practice as it typically involves simple models, often avoiding complexities, and can be implemented heuristically with minimal effort.

Statistical approaches frequently leverage common measures [49] to identify outliers and determine node outliers from their mutual interactions with others. *Degree centrality* can be used to identify outlier nodes that have notably higher or lower connections in the graph. *Clustering coefficient* [50] provides insight into the local cohesiveness or connectivity of nodes, uncommon values of clustering coefficients may serve as indications of outliers. *Betweenness centrality* [51] is relevant for identifying important nodes that play significant roles in connecting different portions of the graph, especially in transportation networks. The study in *oddball* [52] presents several statistical features and rules for identifying anomaly patterns in attributed graphs.

In attributed graphs, outlier nodes can also be measured by the deviation in their content compared to other nodes. Various anomaly detection methods can directly apply to the full feature space of X. Another set of methods heavily relies on subspace selection of node features [53, 54, 55, 56]. For example, [56] determines for each node its statistically relevant subset of attributes and attempts to find anomalies in a node feature subspace, allowing the detection of meaningful outlier nodes in local contexts.

The work in [57] proposes a novel framework that integrates graph embedding, graph partitioning, and innovative dimension reduction techniques for detecting structurally inconsistent nodes in large graphs. The objective is to derive an embedding vector for each node, in which each dimension of the vector indicates the similarity of the node to a specific clustered region. Specifically, nodes that are connected exhibit similar values of $Z[i]$ in dimension i, while unconnected nodes show diverse values of $Z[i]$. The embeddings possess a high level of interpretability regarding a node's interaction with different regions in the graph, enabling the detection of nodes with inconsistent structures. The study presents a measure to directly quantify the level of anomalousness of a node based on its embedding vector.

Modern node embedding techniques aim to learn node representations in a latent space, offering convenience by converting graph structures into tabular datasets. The acquired representations are subsequently utilized in conjunction with established outlier detection methods, such as density-based methods [21, 58] or distance-based methods [59], to create a node outlier detector. While there are methods to estimate embedding quality, such as a low reconstruction loss indicating an optimized compression matrix, these methods do not adequately capture the quality of the subsequent modeling step. The work by [60] evaluates the quality of embeddings in encoding elementary properties of a node, such as page rank, degree, closeness centrality, and clustering coefficient. The evaluation of embedding method quality often relies on downstream tasks [61, 62, 63], for example, how well the classifier performs on new representations in a node classification task. Consequently, selecting the most efficient embedding and modeling methods poses a challenge, requiring careful hyperparameter tuning in both the embedding and modeling phases, especially in unsupervised scenarios where truth labels are unavailable to provide an accurate estimate of the outlier detection model for graph datasets.

3.2.2 Graph Reconstruction-based methods

Reconstruction-based methods are popular unsupervised anomaly detection techniques. These methods assume that anomalies do not follow a common pattern in the dataset. When projecting anomalies into a suboptimal space, these data points exhibit a large reconstruction error and are therefore determined as outliers. Dimensionality reduction methods, such as Principal Component Analysis (PCA) or matrix factorization, are heavily utilized in this direction [9, 64]. However, these methods are primarily designed to capture linear relationships between features and can lead to numerous incorrect outliers. An autoencoder [65] is an alternative choice to PCA that uses neural networks to capture the nonlinear correlations between features. Hence, it possesses the ability to compress complex nonlinear data and is likely to be more accurate for anomaly detection [66, 11]. An autoencoder model consists of an encoder and a decoder; the encoder maps data into a low-dimensional representation, and the decoder attempts to reconstruct the original data from the new representation. The autoencoder model is optimized

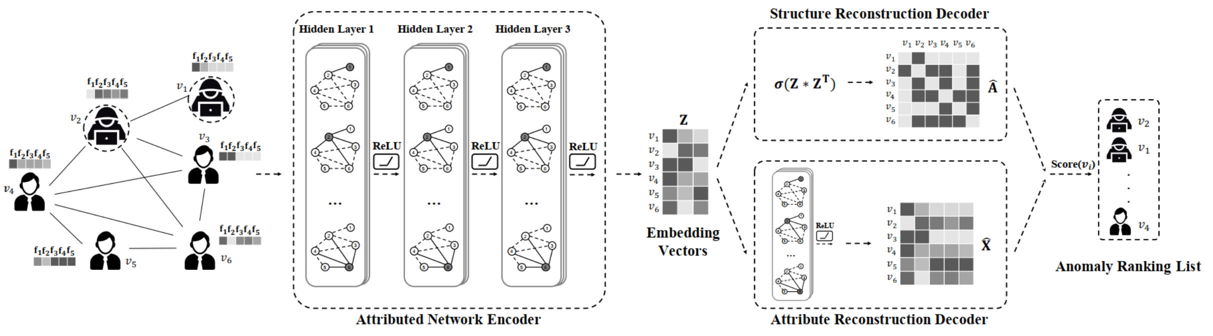


Figure 5: The overall architecture of DOMINANT [69].

by minimizing reconstruction errors. Since the autoencoder provides a flexible framework for defining an outlier detector naturally from the reconstruction error, several works have been proposed in this direction.

DONE [67] introduces an unsupervised deep learning model to detect global, structural, and community anomalies in attributed graphs. The model employs two distinct autoencoders: a structure autoencoder and an attribute autoencoder, both utilizing multiple-layer perception (MLP) architectures in their encoders and decoders. Each autoencoder is trained independently by minimizing reconstruction errors, one on the structure data A and the other on the attribute data X. Nodes that are hard to reconstruct, and thus exhibit higher reconstruction errors, are considered as potential indicators of anomalies. The anomaly scores for nodes are computed based on their reconstruction errors, and the top-k nodes with the highest anomaly scores are determined as anomalies.

GNNs have made significant progress in graph representation learning. Furthermore, the nature of the autoencoder model provides a straightforward indicator of abnormal data points through reconstruction loss. As a result, there is substantial research interest in exploring GNN-based autoencoder architectures to enhance outlier detection performance in graphs. These techniques take the graph structure and node attributes as inputs for the encoder to learn node embeddings. By leveraging GNNs' ability to capture complex relationships within graph-structured, GNN-based autoencoder enhances the robustness of the outlier detection process, making it a powerful tool for anomaly detection within complex graph data. The Graph Autoencoder (GAE), representing one of the simplest approaches within this category, which includes a GCN encoder and an inner-product decoder, can be directly employed for the detection of node outliers. For instance, the work in [68] attempts to convert an original tabular dataset into a knowledge graph and utilizes a GAE model to directly quantify outliers based on the GAE output as reconstruction errors for all nodes.

DOMINANT [69] employs an attributed encoder, a structural decoder, and an attributed decoder. The attributed encoder takes input as graph structure A and node attribute X, using

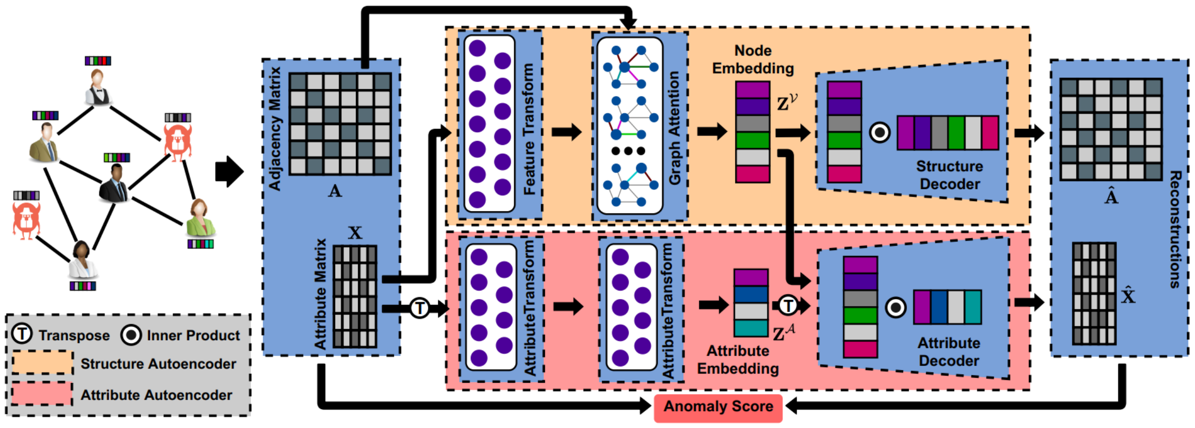


Figure 6: The overall architecture of AnomalyDAE [70].

GCN for node representation learning. For example, $Z = GCN^{+}(X, A)$, where GCN^{+} indicates a GCN model with at least one graph convolutional layer. From the learned node embeddings Z, the structural decoder aims to reconstruct the original graph structure $\hat{A} = \sigma(ZZ^T)$, while the attributed decoder aims to reconstruct node attributes $\hat{X} = GCN(Z, A)$. DOMINANT computes the structure reconstruction error $R_S = A - \hat{A}$ and attribute reconstruction error $R_A = X - \hat{X}$, and defines the learning objective function as $\mathcal{L} = (1 - \alpha)R_S + \alpha R_A$, where α is a weighted parameter. During inference, the anomaly score of each node v_i is subsequently calculated by $score(v_i) = (1 - \alpha)||a_i - \hat{a}_i||_2 + \alpha||x_i - \hat{x}_i||_2$. In the sense that a higher score indicates a greater likelihood of being an outlier, DOMINANT assigns a score to all nodes in the graph, and outliers can be determined either by selecting the top-k nodes with the highest scores or by identifying the set of nodes with scores larger than a predefined threshold. In this way, DOMINANT formulates the task of anomaly detection as a ranking problem, where all nodes are ranked according to their degree of abnormalities.

While DOMINANT employs a single GCN encoder, AnomalyDAE [70] proposes a dual autoencoder consisting of a structure autoencoder and an attribute autoencoder. The dual autoencoder jointly learns the latent representation of nodes and attributes, capturing interactions between the network structure and node attributes during training. AnomalyDAE utilizes a structure autoencoder for network structure reconstruction and an attribute autoencoder for node attribute reconstruction. The structure autoencoder first transforms node attribute X into a latent representation $\tilde{Z}^{\mathcal{V}}$, then employs a GAT layer that takes inputs as node structure A and transformed node attribute $\tilde{Z}^{\mathcal{V}}$ to learn the final node embeddings $Z^{\mathcal{V}}$. Finally, the structure decoder reconstructs structure information $\hat{A} = \sigma(ZZ^T)$. The attribute autoencoder applies two non-linear transformations on node attribute X to learn the low-dimensional representation for attribute embeddings $Z^{\mathcal{A}}$. Then, the attribute decoder takes the learned node embeddings $Z^{\mathcal{V}}$ and attribute embeddings $Z^{\mathcal{A}}$ as inputs to compute the reconstructed node attribute $\hat{X} = Z^{\mathcal{V}}(Z^{\mathcal{A}})^T$.

Analogous to DOMINANT, AnomalyDAE aims to minimize reconstruction errors of both the graph structure and node attributes to optimize the learning process. Subsequently, it measures anomalies in the graph using anomaly scores defined from the reconstruction errors of nodes, considering both structural and attribute perspectives. AnomalyDAE is regarded as an enhanced version of DOMINANT.

Under the intuition that anomalies may appear typical from one view but abnormal from another view, ALARM [71] proposes a deep multi-view framework that incorporates user preferences into anomaly detection on attributed graphs. The input graph is supplied with k-view node attributes $(X^{(1)}, .., X^{(k)})$ that are subsets from the original node attribute X. The encoder applies multiple GCNs to encode the attributed graph under different views separately. Then those extracted view-based node embeddings are unified using a weighted aggregator to generate the final node representation. In subsequent steps, ALARM follows similar approaches to DOMINANT, utilizing both a structure and an attribute decoder to decode original data in A and X from node embeddings. The training objective aims to minimize structure and attribute reconstruction errors. Finally, a weighted scoring function uses those reconstruction errors to spot anomalies.

3.2.3 Adversarial Learning-based methods

A set of methods utilizes Generative Adversarial Networks (GANs) for outlier detection has emerged. GANs provide a flexible approach for outlier detection by learning the underlying data distribution of normal samples with a min-max optimization framework. In the context of outlier detection, the generator synthesizes samples similar to normal data, while the discriminator learns to distinguish between original (normal) and generated (outlier) samples.

AEGIS [72] proposes a GAN-based framework that can be applied in both transductive and inductive settings by training two separate components. Firstly, AEGIS learns anomaly-aware node representations through an autoencoder that employs a Graph Differentiation Network (GDN). The GDN layer utilizes an attention-based mechanism to learn from arbitrary-order neighborhoods. Secondly, a GAN model is trained with the learned node representations to detect outliers. The generator G generates informative potential anomalies, while the discriminator D attempts to distinguish if a representation input comes from a normal node or a generated anomaly. To perform outlier detection on a new node x', the node representation z' is generated with the trained autoencoder, and then an outlier score is computed from the output of the discriminator D as follows: $score(x') = 1 - D(z')$.

In another approach, GAAN [73] employs a generator to create fake nodes, an encoder to learn node representations for both real and fake nodes, and a discriminator to identify whether an edge is from the real or fake connected nodes. The generator G, which utilizes MLP layers, aims to learn the node attribute distribution in X and uses Gaussian random noise to generate fake

X'. Next, an encoder with three MLP layers maps inputs from X and X' to a low-dimensional space, Z and Z'. Subsequently, GAAN tries to reconstruct the original graph structure from low-dimensional representation such that $\hat{A} = \sigma(ZZ^T)$ and $\hat{A}' = \sigma(Z'Z'^T)$. The discriminator is trained to distinguish whether the dot product of any node pair (v_i, v_j) is from $\hat{A}$ or $\hat{A}'$, using cross-entropy loss. During the inference phase of outlier detection, the outlierness of a node v_i is derived as a weighted sum of two scores. The first score, $\mathcal{L}_G$, indicates how well the node attribute x_i is reconstructed by the generator, with a larger value suggesting a higher likelihood of abnormality. The second score, $\mathcal{L}_D$, comes from the discriminator loss, which is the sum of cross-entropy losses for all node pairs (v_i, v_j), with $j = 1..n$, measuring the loss when all node pairs of v are identified as a real edge.

3.3 Deep Ensemble Anomaly Detection (DEAN)

Graph outliers, as presented, can be defined from multiple perspectives and belong to different types of outliers existing within a dataset. Each outlier detection model is tailored to identify a specific type of outlier. Therefore, aggregating the predictions from these models into an outlier ensemble has the potential to significantly enhance the quality of anomaly detection.

Deep anomaly detection models often utilize deep neural networks, which consist of multiple hidden layers, to effectively capture complex patterns within datasets and exhibit the flexibility to adapt to arbitrary feature spaces. While deep models are powerful tools for identifying anomalies, they often encounter challenges regarding the inconsistency of output outlier scores, thereby posing a challenge for comparable scoring of outliers across different deep anomaly detection models. In addition, reconstruction-based models, such as autoencoders, attempt to optimize their parameters to fit as many data points as possible, aiming to achieve the lowest reconstruction loss. This often leads to a model with low variance after training. Because of this, building an ensemble from deep anomaly detection models becomes more complicated.

The main work of this thesis applies the concepts of Deep Ensemble Anomaly Detection (DEAN) [74] in an attempt to build a robust outlier detection model. In short, DEAN defines a general framework for outlier ensembles in which its learning objective function forces the outlier detection sub-model to output consistent outlier scores, thereby enabling the effective combination of them into a deep ensemble model. Firstly, DEAN defines a set of four key properties for each outlier detection sub-model, allowing for the creation of an optimal ensemble model. Secondly, DEAN introduces a trivial learning objective function that enables the comparable scoring of multiple models. Thirdly, DEAN suggests a method to combine the outlier scores from multiple sub-models, incorporating a parameter derived from the training data to make the final outlier score more reliable.

The four key properties that DEAN suggests are:

- **Scalability**: DEAN emphasizes the importance of quickly generating a good sub-model to be integrated into an ensemble, enabling the combination of numerous models and reducing the variance of the ensemble model. Also, DEAN suggests that in situations when limiting the maximum number of models in the ensemble may not be optimal. For example, building an ensemble using multiple algorithms or employing different hyperparameters from one algorithm.
- **Depth**: This property refers to the sufficiently robust performance of a sub-model. A good sub-model should be capable of learning complex patterns, but does not have to capture all the complexity, and can facilitate the identification of non-trivial anomalies when combined with others.
- **Variability**: Each sub-model should demonstrate high variance in its prediction. Identical predictions from sub-models limit the capacity of the ensemble to learn and identify non-trivial anomalies. Thus, maximizing variance between predictions is vital for optimal ensemble performance.
- **Consistency**: DEAN emphasizes the importance of consistency in outputting outlier scores for each sub-model. This entails maintaining similarity in the scores produced by each model, implying constancy in certain variables and uniformity in the scale of model predictions. Consistency is necessary to ensure that each individual model integrates effectively into the ensemble and performs well with the combination function.

DEAN proposes a simple learning objective function: training each model to output a constant value of 1. This trivial loss function enables quick training of a sub-model and facilitates integration with any type of base model that outputs a numeric value, thereby enabling the scalability property of the sub-model. Given an outlier detection model $f()$, the DEAN loss function is defined for each data point x_{train} in the training dataset as follows:

$$l_{DEAN} = (f(x_{train}) - 1)^2 \tag{3.7}$$

DEAN highlights the risk of ending up with a trivial solution like having a constant model always outputting 1. Since neural network layers are typically expressed mathematically as $WX + b$, where W and b are learnable parameters, DEAN proposes that partially, or completely, removing bias term b from the neural network model can lower the likelihood of obtaining a basic model, making it less likely to become a universally approximate function. Removing a parameter also enables the model to converge more quickly and does not significantly alter the final performance of the ensemble model. Additionally, employing ReLU or Leaky ReLU as activation functions, rather than non-linear functions like tanh or sigmoid, can aid in mitigating trivial models. This is because the nature of tanh or sigmoid functions can rapidly drive model outputs towards 1, potentially leading to trivial solutions.

Defined in this way, a DEAN sub-model is expected to produce $f(x) \approx 1$ for normal samples, while a high deviation of $f(x)$ from 1 implies anomalous samples. The DEAN scoring function has the following form:

$$score_{DEAN} = |f(x) - q| \tag{3.8}$$

Defining $q = 1$ as the first simple choice is natural. In addition to that, DEAN suggests that deriving q from the training data is better aligned with data distribution that is learned from the training process, such that $q = mean(f(x_{train})$.

A single DEAN sub-model, $f_i(x)$, represents an outlier detection model that outputs outlier scores for sample x. The final outlier score is aggregated from n sub-models $(f_1, .., f_n)$ as in the following equation:

$$score_{ensemble} = \frac{1}{n}\sqrt{\sum_{i=1}^{n} f_i^2(x)} \tag{3.9}$$

It is noteworthy that DEAN makes the scale of each base model comparable, eliminating the need for a normalization step for outlier scores in the subsequent ensemble combination.

4 Approach

This section outlines the methodology for constructing an ensemble outlier detection model. The base model extends the Graph Neural Network (GNN) architecture to ensure that each sub-model is sufficiently **deep** to accommodate the complexity of graph datasets. Next, each base model is trained using the principles of DEAN to output a constant value, thereby ensuring **consistency** among sub-models' predictions and facilitating the creation of a robust ensemble model.

The graph ensemble is achieved via feature-bagging, where we utilize graph sampling techniques based on inductive learning approaches to preserve the connectivity information in graphs while employing feature-bagging. Specifically, we employ two selected methods, GraphSAGE [36] and GraphSAINT [47], to sample graph data during the training of a single outlier detection base model. Feature-bagging reduces the dimensionality of training data and expedites the convergence of the base GNN model, thereby enabling the training of numerous GNN models with high **scalability**. Additionally, inductive learning methods ensure that the trained base model can generalize effectively on unseen data, guaranteeing a good performance of the base model while maintaining considerable **variability** among their predictions.

4.1 DEAN on Graphs

We extend DEAN to graphs by training a GNN model to output a constant value, providing us with flexibility in manipulating different GNN architectures for the core model. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes in graph $\mathcal{G}$, the first GNN-based DEAN model learns to assign a numeric value to each node $\mathbf{v} \in \mathcal{V}$ such that: $f : \mathcal{V} \to \mathbb{R}^+$.

Specifically, we start with a GCN-DEAN model comprising two GCN layers followed by a multi-layer perception (MLP) layer. The backbone GCN model is aimed at learning a low-dimensional node representation, denoted as matrix $Z \in \mathbb{R}^{|\mathcal{V}| \times d}$. Followed by an MLP layer designed to map the latent node representations to scalar values representing outlier scores for each node, the MLP layer is referred to here as a fully-connected (FC) layer. By employing ReLU as the activation function for each GCN layer, such that $x = max(0, x)$, the forward model f is expressed in the following straightforward form:

$$f(\mathbf{v}) = FC(ReLU(GCN_2(ReLU(GCN_1(\mathbf{v}))))), \mathbf{v} \in \mathcal{V} \tag{4.1}$$

Recall the matrix form of GCN layer-wise in Equation 3.5. Let A be the adjacency matrix representing the graph structure, X be the feature matrix representing node features, and $\tilde{A}$ be the pre-processed symmetric normalized version of A. The example GCN-DEAN model, with an optionally included bias term, can be described as follows:

$$\begin{aligned} f(X, A) &= W^{(2)}Z + b^{(2)} \\ &= W^{(2)}ReLU(\tilde{A}ReLU(\tilde{A}XW^{(0)} + b^{(0)})W^{(1)} + b^{(1)}) + b^{(2)} \end{aligned} \tag{4.2}$$

where $W^{(0)}$, $W^{(1)}$, $W^{(2)}$, $b^{(0)}$, $b^{(1)}$, and $b^{(2)}$ are learnable weight matrices and bias terms, respectively, for the first GCN layer, the second GCN layer, and the fully-connected layer.

After performing the forward phase for all training nodes, we compute the mean squared error (MSE) loss:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(1 - y_i)^2 \tag{4.3}$$

where $y_i = f(v_i)$ is the predicted outlier score for node v_i, calculated from Equation 4.1 or 4.2.

During the backward phase, gradients of the loss with respect to the parameters $W^{(i)}$ and $b^{(i)}$ of the model are computed to update the model's parameters. The model can then be trained using gradient descent algorithm [75] to iteratively adjust the model's parameters and minimize the respected MSE loss function.

Once a single GCN-DEAN model f has been trained, we apply it to infer outlier scores for all nodes $v \in \mathcal{V}$ in the graph:

$$score_{DEAN}(v) = |f(v) - q| \tag{4.4}$$

where q is a user-defined parameter.

After training n GCN-DEAN models $(f_1, .., f_n)$ independently, the final score is combined using DEAN's suggested function in Equation 3.9. Additionally, we employ the "max" aggregation operator for scoring combination, which maintains consistency in outlier scores. The "max" operator simply takes the highest score from all sub-models:

$$score_{ensemble} = max(f_1, .., f_n) \tag{4.5}$$

In the following step, outliers in graph $\mathcal{G}$ can be determined as nodes with the highest top-k scores or nodes with scores higher than a pre-defined threshold τ. Although this step is not included in the training process of the model, with properly fine-tuned values, it can greatly affect the quality of the final prediction and evaluation result.

The flexible nature of DEAN allows us to easily experiment with different architectures of the core detection models. In aiming to find the best parameter settings for GNN-based DEAN models, we are able to conduct experiments covering the following aspects, but not limited to:

- Experimenting with different values of k when pulling messages from neighbor nodes during message-passing forward. Here, the number of GNN layers is chosen between 2 and 4 layers. A very deep GNN model can be computationally expensive and may lead to the problem of "over-smoothing" [46] where all node embeddings end up being identical and not useful for the detection step.
- Experimenting with different graph neural networks, including GCN [43], GAT [44], GraphSAGE [36], and GraphSAINT [47].
- Experimenting with the full and partial usage of the bias term, as well as no bias term.
- Experimenting with different values for the parameter q when deriving outlier scores. The first choice is $q = 1$, and the second choice involves deriving it from the average outlier scores of training samples, such that $q = mean(f(x_{train}))$.
- Experimenting with different combination functions.

Training and inferring a GNN model can be computationally expensive, especially with large graphs. Taking a GCN model as an example, recall that a layer-wise operator of GCN is defined as $H^{(k)} = \sigma(\tilde{A}H^{(k-1)}W^{(k)})$ in Equation 3.5. The computational complexity of GCN is linear in the number of edges in the graph [43, 76]. In which, the cost for *neighbor aggregation* $\tilde{A}H$ is $\mathcal{O}(L|\mathcal{E}|F)$, and the cost for *feature transformation* when multiplying aggregated information with W is $\mathcal{O}(L|\mathcal{V}|F^2)$, where L is the number of layers in the model, and F is the dimensionality of the node feature vector [76]. Even if *feature transformation* can be effectively calculated using parallel multiplication, the main bottleneck arises from performing full neighbor propagation with sparse-dense matrix multiplications $\tilde{A}H$, which remains in a time complexity bounded by $\mathcal{O}(L|\mathcal{E}|F)$. Additionally, space complexity when using a sparse representation for matrix A is $\mathcal{O}(|\mathcal{E}|)$, which can occasionally lead to out-of-memory issues. Hence, while the aim is to create a robust ensemble, it is essential to strike a balance between computational resources and the efficacy of the ensemble model. This involves considering factors such as the number of base models and the depth of the GNN layers within each base model.

4.2 Graph Ensemble

In high-dimensional datasets like tabular data, feature-bagging is often employed to enhance ensemble model diversity. The idea involves repeatedly sampling subsets of dimensions, then training a base model from each subset. However, graph features are more complicated, including node features, edge features, and structural properties. While performing feature-bagging on node features presents the most straightforward approach to conducting an ensemble, it is inefficient due to the computational complexities arising from aggregating neighborhood features when training a single GNN model, which is defined by the graph structure. Conversely, performing feature-bagging on graph structure demands careful design to avoid disrupting crucial relational properties inherent in graphs.

The vanilla GCN utilizes the entire training graph and aggregates information from all neighbors of each node during message-passing. While this approach is crucial for learning a good representation because a node is defined not only by its features but also by its connections to others, it poses limitations for scalability. Among the various ensemble approaches outlined in Section 2.2, graph sampling stands out as preferable due to its ability to mitigate the computational complexity associated with constructing a base detector, particularly when contrasted with the utilization of the full GCN.

To preserve connectivity information crucial for defining a good representation for each node while performing sampling on graphs, we employ two graph sampling techniques that have demonstrated capability in inductive learning. The first technique, known as Layer-wise sampling methods, samples a subset of the neighbors at each layer. The second sampling technique involves sampling a sub-graph and performing full message propagation on the sampled sub-graph across all layers. Even though those methods were originally designed to expedite the training time of a GCN model and scale GCN on large graphs, they establish a framework for performing graph sampling, thereby facilitating the development of a robust graph ensemble outlier detection method.

Instead of requiring all nodes in the neighborhood to be present during message-passing, layer-wise sampling methods sample a subset of the neighbors at each layer. GraphSAGE [36] is the first method in this category that performs layer-wise node sampling and introduces the idea of generalized neighborhood aggregation. On one hand, the method uniformly samples s_n neighbors for each node from the previous layer and generates embeddings only for the sampled nodes. This ensures a maximum sample size, bounds the computational complexity, and enables mini-batch training. On the other hand, instead of learning embeddings for individual nodes in the training graph, GraphSAGE employs a set of learnable aggregating functions. Each aggregator function learns to aggregate feature information for a given node at different numbers of hops to the neighbors. During the inference phase, these aggregator functions are key to generating embeddings for unseen nodes, leveraging node features.

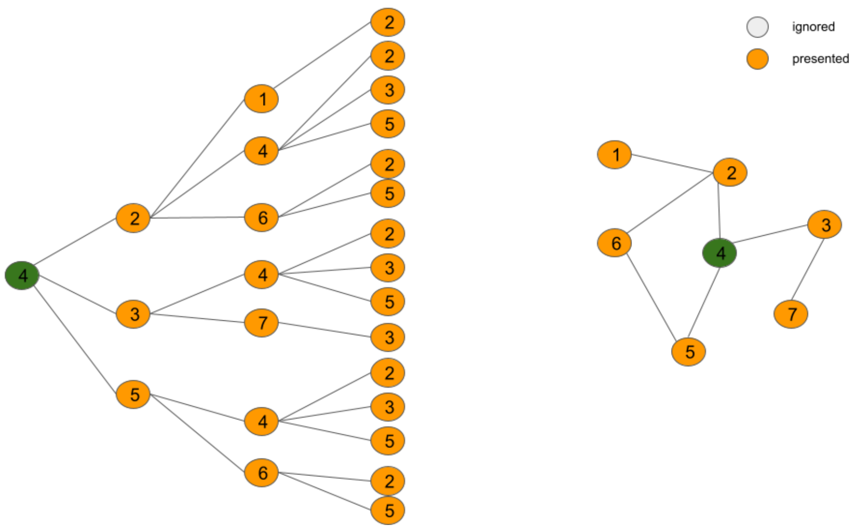
The space and time complexity for GraphSAGE is bounded at $\mathcal{O}(bs_n^L F)$, where L is the number of layers, F is the number of node features, and b is the number of nodes present in an iteration (batch size). The complexity can grow exponentially with L. To limit the exponential expansion, FastGCN [77] and LADIES [78] propose another effective sampling strategy: sampling a fixed number of nodes across all layers, where nodes are sampled according to the probability for the whole layer. These methods reduce the complexity bound to $\mathcal{O}(Ls_n^2 F)$.

In contrast to sampling across the layers, another set of graph sampling methods sample a subgraph from the original one at the beginning and perform full GCN on the sampled subgraph across all layers. Cluster-GCN [79] partitions the original graph into multiple subgraphs using a graph clustering method [80], and subsequently performs full feature aggregation for one subgraph in each mini-batch. GraphSAINT uses a sampler to obtain a properly connected subgraph, enable mini-batch training to perform full GCN on the sampled subgraph. Initially, a set of nodes are sampled, and the sampler returns a subgraph induced from the sampled node. A pre-processing step estimates the probability of a node and an edge being sampled by the employed sampler. Subsequently, the sampled probabilities are used to normalize the neighbor aggregation within the subgraph. The normalization steps aim to reduce the effects of bias in the mini-batch estimator, where nodes with higher degrees are more likely to be sampled in a subgraph.

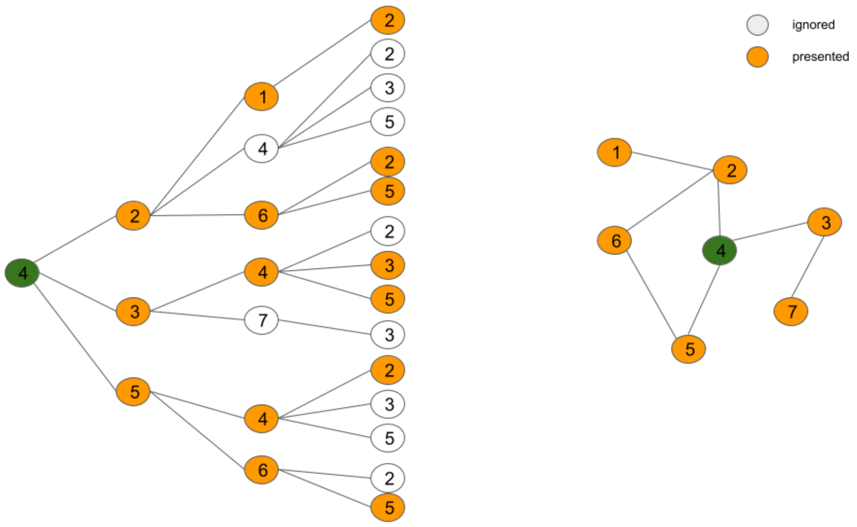
To highlight differences between GCN and graph sampler techniques (GraphSAGE, GraphSAINT) in node sampling and message aggregation, we present an example in Figure 7. In this illustration, we aim to learn embeddings for the target node (**4**). Feature information aggregated from its local neighborhood with a hop number $k = 3$, indicating a GNN model with $L = 3$ layers. The orange circle represents nodes involved in the message-passing process. GCN employs full forward propagation, reflecting the "neighbor explosion" phenomenon with increasing k. GraphSAGE limits the number of sampled nodes at each layer, while GraphSAINT first samples an induced subgraph for the target node (**4**), followed by vanilla GCN processing on the subgraph.

In conducting a graph ensemble, we utilize GraphSAGE and GraphSAINT for graph sampling, which enables the possibility of having numerous graph neural network-based detectors within constrained resources, such as time and memory. During each mini-batch, a base outlier detector is trained on the sampled subgraph following the DEAN framework. Subsequently, each base detector is used to infer outlier scores for the entire original graph, and all predictions are combined to produce final outlier scores for all nodes. This approach shares similarities with the feature-bagging ensemble method. However, it takes a different path than directly applying feature-bagging to node features, which could be seen as an extension of previous research [53, 54, 55, 56]. Yet, exploring ensemble methods in this direction still involves dealing with the high complexity of the vanilla GCN-DEAN model and is therefore not evaluated in this study.

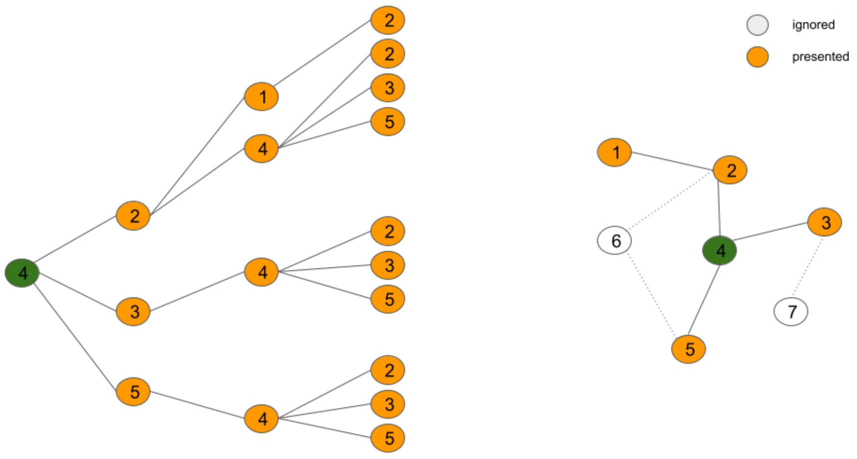
The graph sampling operator, however, introduces bias when learning node representations during each mini-batch [47], consequently impacting the performance of the base detector model. With less information available to train a base model, there is increased uncertainty in the base model's predictions when applied over the entire graph, thereby increasing the base model's variance. We aim to determine the optimal settings for the DEAN model within a graph context to enhance the ensemble model and achieve the greatest improvement. The results of our investigation are presented in the following section 5.



(a) GCN: full propagation over the entire graph.



(b) GraphSAGE: layer-wise nodes sampling.



(c) GraphSAINT: full propagation in a sampled subgraph.

Figure 7: Difference in sampling and aggregation methods illustrated to update node **4**, comparing GCN (a), GraphSAGE (b), and GraphSAINT (c) with $L = 3$.

5 Implementation

In this section, we conduct experiments on node outlier detection to evaluate the performance of the GNN-based DEAN approach against state-of-the-art existing methods. We aim to address the following questions: Does the DEAN approach effectively function in the context of graph data? What are the optimal DEAN settings for graph data? And, how scalable is DEAN when applied to graph datasets?

5.1 Experiment Setup

In this section, we describe the benchmark datasets employed to evaluate the performance of our outlier detector. We then discuss the selection of baseline models, which represent off-the-shelf approaches in the graph outlier detection problem. Finally, we present the settings for our DEAN approach in conducting an ensemble model.

5.1.1 Benchmark dataset

We utilize five datasets from various domains and scales for experimentation. Each dataset includes ground truth anomalies necessary for evaluating the performance of unsupervised detectors. Anomaly nodes are either marked from the original datasets or generated as outliers through different injection methods. We select datasets that are neither too small with only a few outliers, which can lead to biased results when evaluating a model, nor very large-scale datasets, to ensure that most detection methods can be applied feasibly and computational efforts remain reasonable.

Dataset	**Type**	**Outlier Type**	**#Node**	**#Edge**	**#Feature**	**#Outlier**
inj_amazon	undirected	injected	13,752	515,042	767	694
inj_cora	undirected	injected	2,708	11,060	1,433	138
reddit	undirected	organic	10,984	168,016	64	366
weibo	directed	organic	8,405	407,963	400	868
yelpchi	undirected	organic	45,954	3,846,979	32	6677

Table 1: Statistics of Benchmark Graph Datasets.

Cora [81] is a citation dataset where nodes represent publications and edges represent citation links. Node features are sparse bag-of-word vectors extracted from document content. Amazon [82] is an e-commerce dataset where nodes represent products, and nodes are linked if they are frequently purchased together. Node features are encoded as bag-of-word vectors extracted from product reviews. The **Inj_cora** and **Inj_amazon** datasets are modified versions of the original Cora and Amazon datasets using the injected contextual and structural outliers methodology described by [5]. The injection method fixes a set of nodes in the original graph and modifies the existing edges to inject structural outliers, and it also modifies node feature vectors to inject contextual outliers.

Weibo [83] is a social media dataset that connects two users (nodes) who use the same hashtag. Node features are a combination of a location one-hot vector and a bag-of-word vector extracted from post texts. A user who has made at least five suspicious events is regarded as an outlier in the graph.

Reddit [84] is another social media dataset that connects user nodes and subreddit nodes. A banned user is regarded as an outlier in the graph. Each post's text is converted into features. The features of each user node or subreddit node are obtained by summing the features of all posts belonging to that node.

Yelp-Fraud [85], a multi-relational graph dataset extended from YelpCHI [86], contains three heuristic-derived relations between reviews (nodes), for example, reviews posted by the same user or reviews under the same product with the same rating are connected. **YelpCHI** is a spam review dataset containing hotel and restaurant reviews. The dataset is marked with genuine and fake reviews, as well as spammers and normal reviewers. Node features consist of 32 handcrafted features [86] from review raw information (including product and user data, timestamp, rating, and review text).

5.1.2 Choice of baseline methods

We compare DEAN against popular baseline methods, including:

- Autoencoder-based method: Variational Graph Auto-Encoder (VGAE) [48] is applied, and outlier scores are determined by the reconstruction errors of node features.
- Shallow methods: Here, we applied node embedding techniques (Node2Vec [41], VGAE [48]) to learn efficient node representations. In the next step, we apply Local Outlier Factor (LOF) [21] to detect anomalous nodes on the learned node representations.
- State-of-the-art reconstruction-based methods: DOMINANT [69] and AnomalyDAE [70].
- Generative adversarial learning approach: GAAN [73].

Since the baseline methods are not designed for ensemble settings, we report results from fitting each model with each dataset only once. Moreover, reconstruction-based and GAN-based methods are computationally expensive, even for one-time fitting. This high computational cost poses a significant challenge when attempting to construct an ensemble model from these baseline models. However, we conducted one experiment with DOMINANT to demonstrate that building an ensemble does not provide much benefit from those highly stable baseline models.

In addition, for all methods that are backed by GNN layers, such as VGAE, DOMINANT, AnomalyDAE, and GNN-based DEAN, we aim to maintain fairness between experiments by avoiding fine-tuning parameters to fit a particular dataset optimally. Most parameters remain fixed across models, such as the hidden dimension of GNN layers and the number of GNN layers in the backbone GNN model.

5.1.3 DEAN with ensemble settings

The flexibility of DEAN allows us to experiment with different settings to find the best configurations for DEAN in the graph context. As previously described in Section 4.1, DEAN has various parameters, including the number of GNN layers, the type of GNN layer used in the sub-model, whether to include a bias term, the choice of q when inferring outlier scores, and different combination functions for sub-model predictions. Except when explicitly mentioned, all DEAN models in this section refer to an ensemble model.

Experimenting with all combinations of possible values can be exhaustive. Therefore, we constrain the next experiment based on the best findings from previous ones. In the first experiment with GCN-DEAN, where we utilize full propagation in the GCN layer to learn node representations, our goal is to determine which settings are optimal for DEAN in terms of the usage of bias term, parameter q, and combination function.

In the second experiment, we compare GraphSAGE and GraphSAINT sampling techniques in terms of increasing the base model variance and boosting ensemble model performance. Specifically, GraphSAINT works by sampling subgraphs, allowing us to conduct experiments using two approaches. In the first approach, we sample a subgraph and build a base model from it. In the second approach, we adopt a mini-batch training paradigm, where a base model is iteratively trained over many iterations. During each iteration, a subgraph is sampled. This way, a base model is able to see the entire original graph while still performing graph sampling.

Some hyperparameters are employed constantly throughout the experiments:

- number of epochs to train a sub-model: 50.
- number of sub-models: 100.
- number of GNN layers in a sub-model: 3.
- The dimension of hidden GNN layers: due to diversity of node feature dimensions among datasets, we use a dimensionality of 64 when fitting the model with the Amazon, Cora, or Weibo datasets, and 16 for the remaining datasets. This parameter is applied consistently across all baseline models.

Finally, three combination functions are employed: "ssq", the proposed function of DEAN, alongside "mean" and "max" functions.

5.2 Experiment Results

In this section, we present the evaluation results of baseline detectors as well as the DEAN approach for detecting node outliers in graphs. Firstly, we present the performance of baseline models. Then, we showcase the results of GNN-DEAN in various ensemble settings. Finally, we provide additional experiments where we analyze the suitability of DEAN for different types of outliers and investigate whether ensemble settings are suitable for any kind of base model.

5.2.1 Performance of Baseline Model

In the initial set of experiments, we utilize graph autoencoder (VGAE) to directly assess the anomaly degree of each node based on reconstruction errors derived from the node features. In the subsequent set of experiments, we leverage standard graph embedding techniques, Node2Vec and VGAE, to learn node representations. We then apply LOF, a popular outlier detection technique for tabular datasets, to the new representations of nodes obtained from these techniques. The third group of experiments employs more sophisticated reconstruction-based approaches, including DOMINANT and AnomalyDAE. Additionally, a GAN-based method is included to ensure a comprehensive evaluation. The result is presented in table 2, where we highlight cases where the model achieves ROC-AUC scores of at least 0.7, indicating performance better than random guessing.

Dataset	VGAE	VGAE-LOF	N2V-LOF	DOMINANT	AnomalyDAE	GAAN
inj_amazon	**0.74**	0.47	0.41	0.72	**0.75**	0.56
inj_cora	**0.71**	0.68	0.48	**0.77**	**0.71**	0.53
reddit	0.54	0.52	0.47	0.56	0.53	0.55
weibo	**0.92**	0.52	0.54	**0.91**	**0.93**	**0.92**
yelpchi	0.60	0.55	0.52	0.48	_[1]	_[2]

Table 2: ROC-AUC comparison among baseline models.

The selected benchmark datasets pose significant challenges for all methods under consideration. Reconstruction-based techniques exhibit promising performance, particularly on injected datasets. Interestingly, the Weibo dataset emerges as comparatively easier for advanced detection methods. In contrast, both the Reddit and YelpCHI datasets present considerable challenges, attributed either to their scale or the inherent difficulty posed by organic outliers. The graph autoencoder, despite its simple architecture, performs reasonably well compared to other, more sophisticated methods.

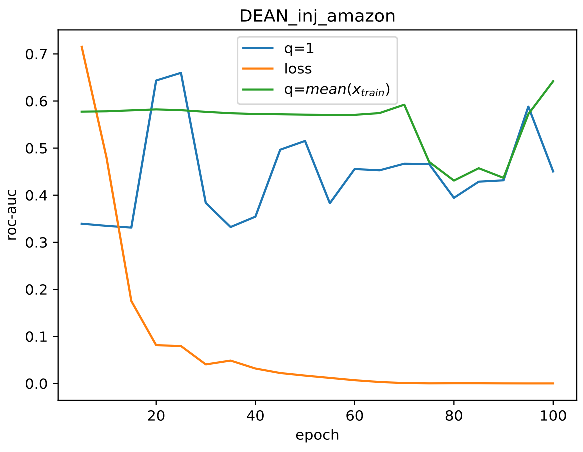
[1]Experiment failed due to memory limitations.

[2]Experiment failed due to memory limitations.

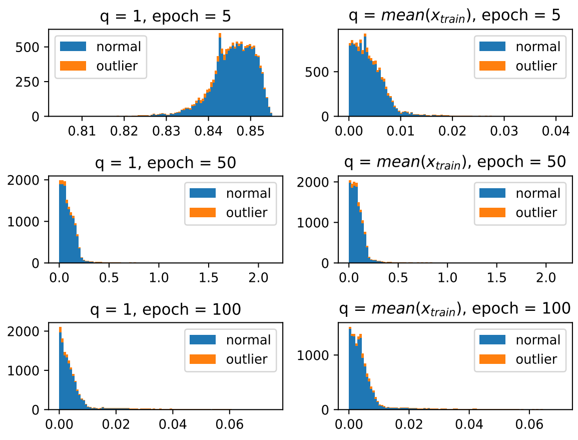
5.2.2 Initial Performance of GCN-DEAN

A single DEAN model

Before conducting an ensemble, we want to investigate how a single DEAN model performs. Figure 8 illustrates the high variance of a single DEAN model when applied to the Inj_amazon dataset.



(a) ROC-AUC scores when fitting a sing DEAN.



(b) Outlier scores distribution changes.

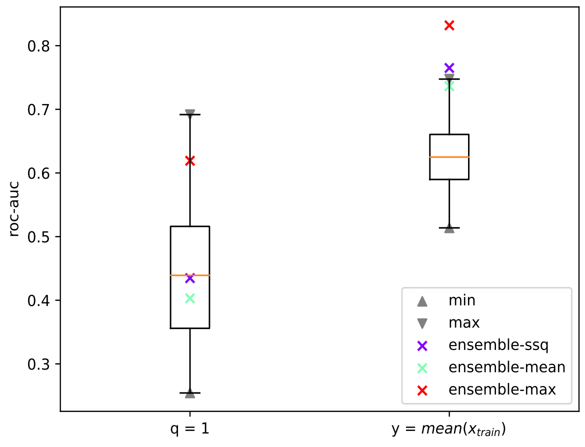
Figure 8: The behavior of a single DEAN model during the training process.

In an unsupervised setting, relying solely on training loss to determine when to stop training presents challenges. The training loss, depicted by the *orange* line in Figure 8a, provides information about how far the model output deviates from a constant value. However, it does not indicate the point at which the model has learned the most useful information before becoming a trivial solution. The *blue* and *green* lines show the evaluation results of the model after training at specific epochs when applying different outlier score inference functions, represented by different parameters of q in Equation 3.8. While there is no strong correlation between loss value and ROC-AUC scores, it is expected that the model becomes more stable after a certain training point. On the other hand, Figure 8b depicts the change in distribution of model output, outlier scores, at different epochs. As training progresses, we observe an increasing number of outlier scores of 0, indicating that DEAN has learned to assign values close to a constant 1 for most nodes in the graph and effectively captured common patterns in the graph.

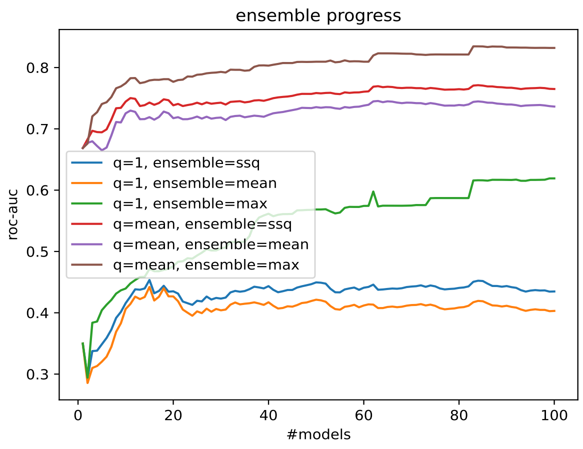
In the first experiment, we utilize a full GCN layer and aim to determine the optimal settings for DEAN: specifically, which combination function performs best among the three employed operators? We investigate whether using a bias term or no bias is beneficial, and evaluate whether an ensemble model outperforms a single DEAN model.

Does ensemble help in improving detection performance?

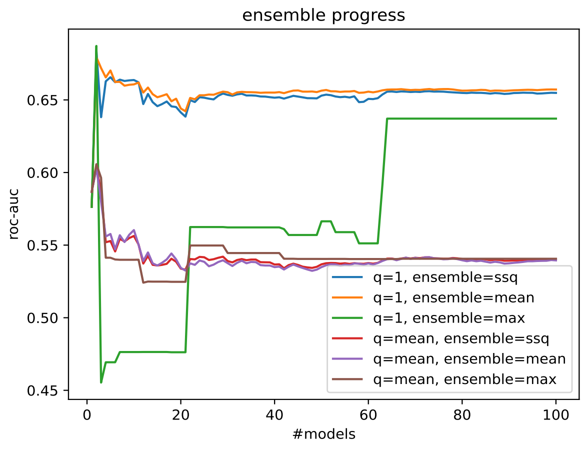
The performance of the ensemble model, although dependent on many factors, has the high potential to create a more robust outlier detector. Figure 9b demonstrates the ensemble's progress in final prediction when adding more base models. While this trend is not consistently strong across all datasets, indicating that the ensemble model is not always superior to all base models, it does show promise in the direction of creating a more stable ensemble model. In addition, although ensemble performance varies depending on different post-processing combination operators, it consistently either shows an increase in progress or stabilizes in its predictions.



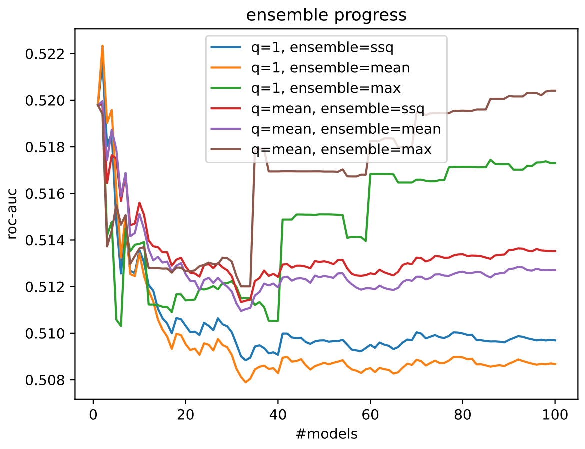
(a) Performance of base models (boxplot) and ensemble model (cross mark).



(b) Progress of ensemble model in Weibo.



(c) Progress of ensemble model in Inj_amazon.

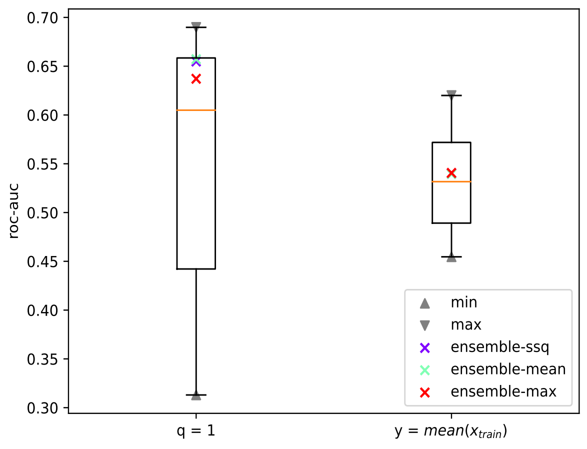


(d) Progress of ensemble model in YelpCHI dataset.

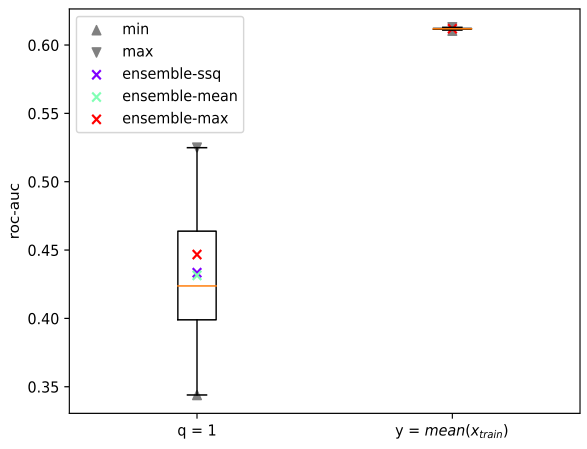
Figure 9: GCN-DEAN performance on the Weibo dataset. Figure (a) illustrates high variance among base models (boxplot) and ensemble performance varying by different settings (parameter q, combination function). Figures (b), (c), and (d) demonstrate the progress of ensemble models as the number of base models increases, respectively, on datasets Weibo, Inj_amazon, and YelpCHI.

Which combination functions are better?

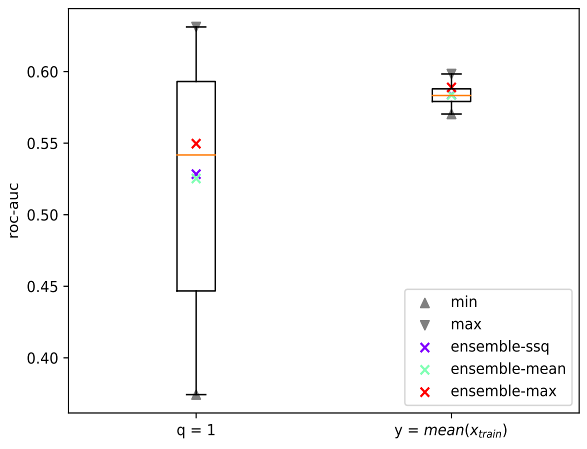
Figure 10 illustrates the performance of the ensemble GCN-DEAN models on injected datasets. Overall, the three combination functions (ssq, mean, max) perform similarly. However, the "max" operator slightly outperforms the others in boosting ensemble performance when combining outlier scores from base models. Additionally, inferring outlier scores using the parameter $q = mean(x_{train})$ tends to produce a more robust base model, enabling better conditions for building a superior ensemble model. Given this observation, in the following results, we report the performance of the ensemble model using only the **"max"** combination operator for consistency.



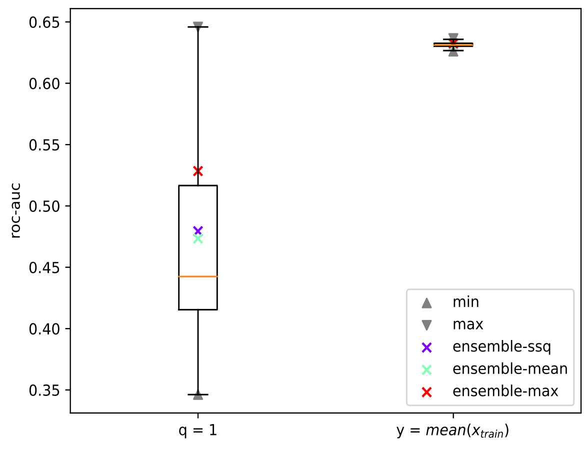
(a) inj_amazon - bias



(b) inj_amazon - no bias



(c) inj_cora - bias



(d) inj_cora - no bias

Figure 10: GCN-DEAN performance on injected datasets: models with bias term (Figure (a) and (c)), models without bias term (Figure (b) and (d)).

Bias or no bias?

Table 3 and 4 provide an overview of the GCN-DEAN model in two different settings: using a bias term or no bias term. The bias term contributes to increasing the variance of base models, and in some situations, it can lead to a better ensemble model. This phenomenon is inferred from the table result by comparing the performance of the worst (**min**) and best (**max**) base model. Additionally, it is depicted by the size of the boxplots in Figure 10. For example, with $q = 1$, the ensemble models with a bias term are slightly better for all datasets. On the other hand, with $q = mean(x_{train})$, there is a significant improvement with ensemble models using a bias term, particularly in the Weibo dataset. Therefore, in subsequent experiments, we only report the performance of ensemble models using the full **bias** term in all model layers.

	q=1			**q=mean**		
	min	**max**	**ensemble**	**min**	**max**	**ensemble**
inj_amazon	0.35	0.54	0.45	0.61	0.61	0.61
inj_cora	0.35	0.65	0.54	0.64	0.64	0.64
reddit	0.49	0.51	0.50	0.49	0.49	0.49
weibo	0.28	0.40	0.51	0.48	**0.70**	**0.72**
yelpchi	0.50	0.52	0.51	0.50	0.52	0.52

Table 3: ROC-AUC scores for GCN-DEAN, **no bias** term. *min*, *max*, and *ensemble* are performances of the worst base model, the best base model, and the final ensemble model. Combination function using *max* operator.

	q=1			**q=mean**		
	min	**max**	**ensemble**	**min**	**max**	**ensemble**
inj_amazon	0.31	0.68	0.64	0.45	0.63	0.55
inj_cora	0.38	0.63	0.55	0.56	0.60	0.59
reddit	0.38	0.61	0.54	0.44	0.59	0.55
weibo	0.28	**0.70**	0.61	0.51	**0.75**	**0.82**
yelpchi	0.48	0.53	0.52	0.48	0.53	0.52

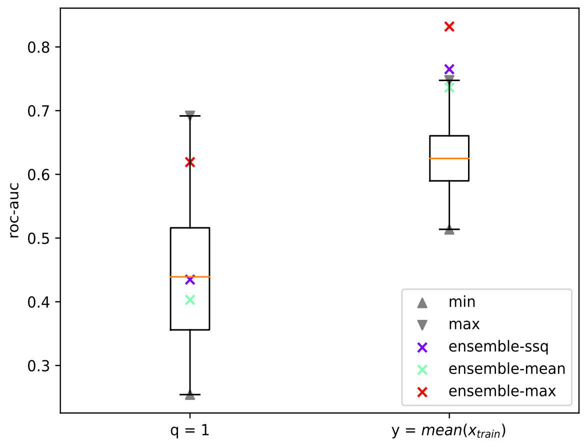
Table 4: ROC-AUC scores for GCN-DEAN, use full **bias** term. *min*, *max*, and *ensemble* are performances of the worst base model, the best base model, and the final ensemble model. Combination function using *max* operator.

While DEAN may not emerge as the top-performing model individually, the initial experiment with DEAN using a standard GCN layer has shown that its simple design holds high potential for achieving superior performance in ensemble settings.

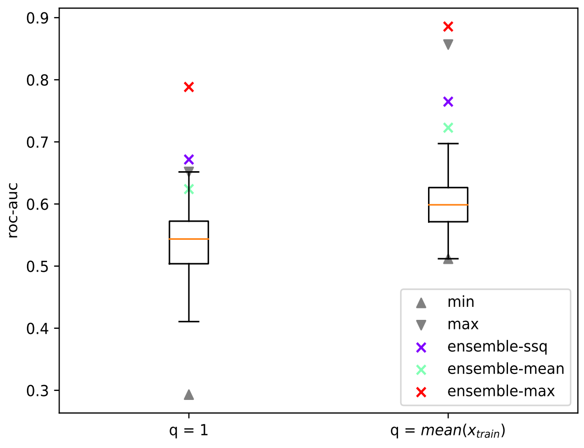
5.2.3 GNN-DEAN Performance with Feature Bagging

In this experiment, our goal is to further optimize DEAN's performance through feature-bagging, employing three different graph sampling approaches. We aim to find the best sampling technique that is more suitable for the complexity of graphs. Specifically, we seek a sampling technique that increases the variance of base models by "hiding" connectivity information in the graph during model fitting, while simultaneously ensuring that the current performance is not decreased too much or is preferably maintained or improved.

GraphSAGE: layer-wise node sampling



(a) GCN-DEAN



(b) GraphSAGE-DEAN

Figure 11: A comparison illustrates a greater improvement in ensemble models when node sampling is applied on dataset Weibo. Specifically, with $q = mean$, despite the base model's lower performance, greater improvement is attained in the final prediction.

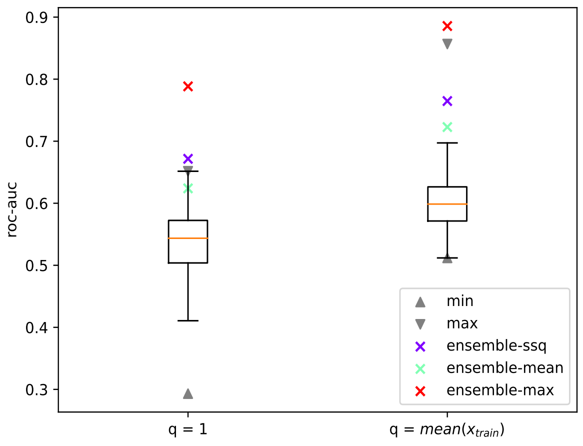
Instead of applying the full GCN during neighbor aggregation, a subset of neighbor nodes is sampled. This approach introduces variation in neighborhood information when learning representations for the target node, which may lead to an unstable and inferior base model. However, it also introduces high variance between base model predictions, reduces computational complexity, and enables faster training.

Figure 11 shows that the performance of the base model decreases when applying node sampling, as expected when less information is present during model training. From the results in Table 5, given the condition that base models perform adequately—achieving ROC-AUC scores higher than 0.6—the ensemble model demonstrates a larger improvement compared to previous settings with GCN-DEAN.

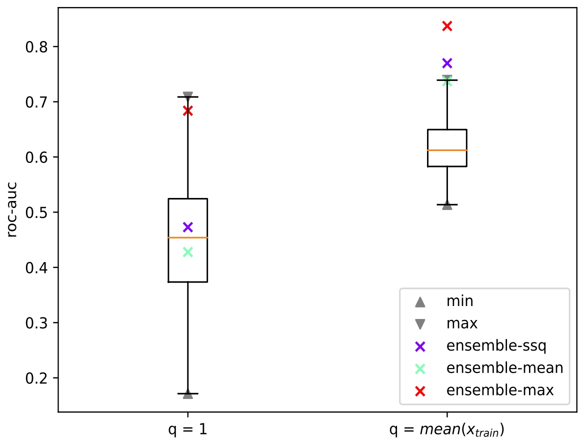
	q=1			**q=mean**		
	min	**max**	**ensemble**	**min**	**max**	**ensemble**
inj_amazon	0.30	0.69	0.48	0.68	0.70	**0.70**
inj_cora	0.42	0.54	0.43	0.42	0.46	0.44
reddit	0.38	0.62	0.52	0.38	0.57	0.44
weibo	0.30	0.67	**0.80**	0.51	0.85	**0.90**
yelpchi	0.41	0.59	0.56	0.45	0.54	0.53

Table 5: ROC-AUC scores for GraphSAGE-DEAN, use full bias term. *min*, *max*, and *ensemble* are performances of the worst base model, the best base model, and the final ensemble model. Combination function using *max* operator.

GraphSAINT: subgraph sampling



(a) GraphSAGE: layer-wise node sampling



(b) GraphSAINT: subgraph sampling

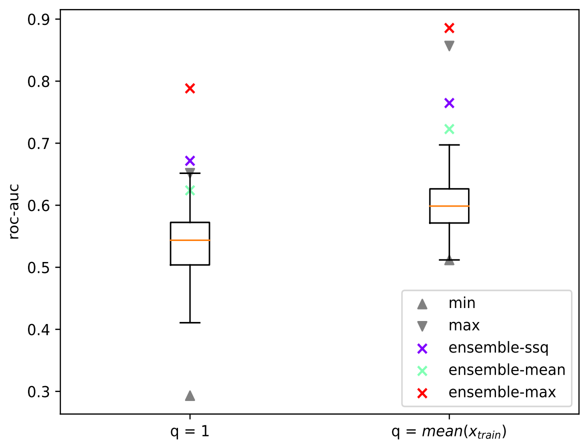
Figure 12: Performance of ensemble model on Weibo dataset. GraphSAINT-DEAN result in less effectively performing of base model ($q = 1$) and lower performance of ensemble model.

In this experiment, a subgraph is sampled to train a base model, meaning that a portion of the original graph is entirely hidden from the submodel. Consequently, the base model exhibits poorer performance when predicting unseen nodes. The experimental results are presented in Table 6. These results indicate that using layer-wise node sampling with GraphSAGE performs better than simply removing a portion of the graph when training a base model.

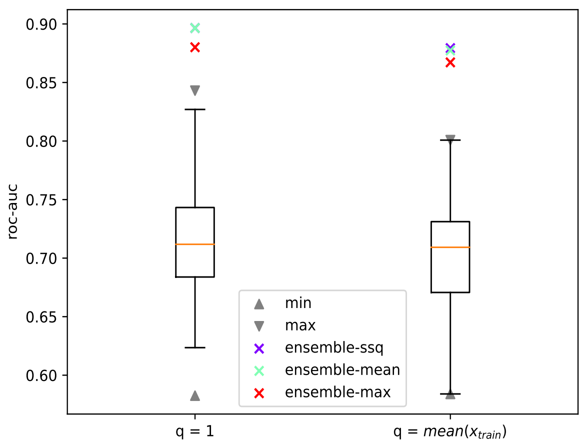
	q=1			**q=mean**		
	min	**max**	**ensemble**	**min**	**max**	**ensemble**
inj_amazon	0.35	0.69	0.51	0.45	0.63	0.55
inj_cora	0.39	0.62	0.50	0.56	0.60	0.58
reddit	0.43	0.60	0.54	0.50	0.59	0.55
weibo	0.19	0.71	0.70	0.50	0.73	**0.83**
yelpchi	0.50	0.53	0.52	0.50	0.53	0.52

Table 6: ROC-AUC scores for GraphSAINT-DEAN, use full bias term. *min*, *max*, and *ensemble* are performances of the worst base model, the best base model, and the final ensemble model. Combination function using *max* operator.

GraphSAINT: mini-batch training



(a) GraphSAGE: layer-wise node sampling



(b) GraphSAINT: mini-batch training.

Figure 13: Illustration for improvement of sampling subgraph in mini-batch training setting, resulting in a better performance of base model and more robust ensemble, regardless of combination functions.

Given the observations from the previous experiment with GraphSAINT, we aim to provide the base model with information about the entire graph while still performing feature-bagging during model fitting. The base model is trained over many iterations, with each iteration involving the sampling of a subgraph to update the model. This approach allows the base model to observe the entire graph structure, but it results in variations due to the randomness of graph sampling at each iteration.

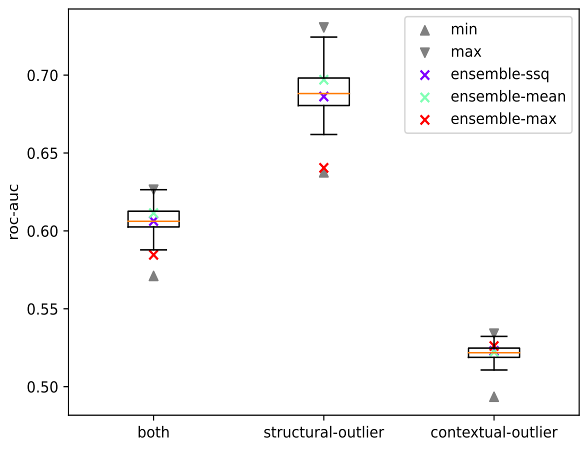
Comparing this sampling technique to the experiment with GraphSAGE, we observe a slight improvement in the performance of the injected dataset Inj_cora, in addition to the improvement on the Weibo dataset, as shown in Table 7.

	q=1			q=mean		
	min	max	ensemble	min	max	ensemble
inj_amazon	0.35	0.65	0.53	0.55	0.64	0.57
inj_cora	0.30	0.69	0.61	0.61	0.69	0.65
reddit	0.42	0.58	0.52	0.42	0.61	0.53
weibo	0.59	0.85	**0.88**	0.58	0.81	**0.87**
yelpchi	0.45	0.56	0.51	0.49	0.54	0.51

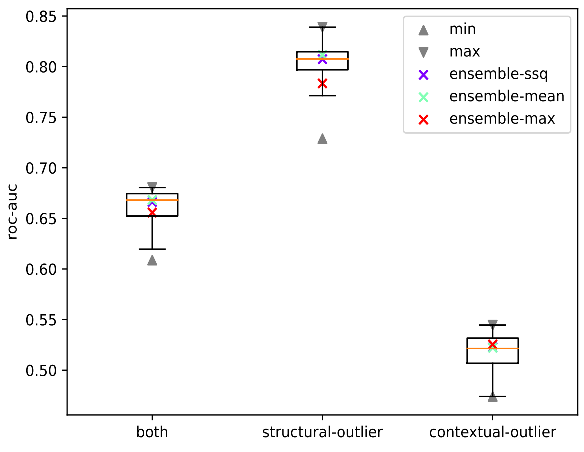
Table 7: ROC-AUC scores for mini-batching training version of GraphSAINT-DEAN, use full bias term. *min*, *max*, and *ensemble* are performances of the worst base model, the best base model, and the final ensemble model. Combination function using *max* operator.

5.2.4 GNN-DEAN Performance on Structural and Contextual Outliers

The GNN-DEAN models in the previous experiments with different graph sampling techniques showed high potential to improve the detection of node outliers. However, they still exhibited only moderate performance on challenging datasets, such as Reddit and YelpCHI. We conducted an additional experiment on injected datasets, where the ground truth of outlier types is available. In this experiment, nodes are categorized as either structural or contextual outliers. The number of contextual outliers and structural outliers is equal in each dataset, with 350 and 70 nodes for each outlier type in Inj_amazon and Inj_cora, respectively.



(a) inj_amazon



(b) inj_cora

Figure 14: Performance Comparison for detecting structural and contextual outliers using GNN-DEAN and GraphSAINT subgraph sampling method.

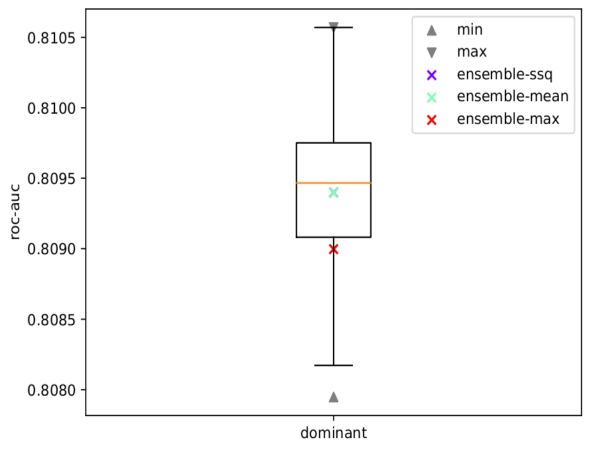
We employ subgraph sampling with GraphSAINT and mini-batch training, as presented in the previous section 5.2.3. Figure 14 demonstrates the superior performance of GNN-DEAN in detecting structural outliers. However, it exhibits poor performance in detecting contextual outliers, suggesting a potential direction for improving the GNN backbone model to enhance overall detection performance on other graph datasets by incorporating a sub-component that can handle node features effectively.

5.2.5 Further Analysis

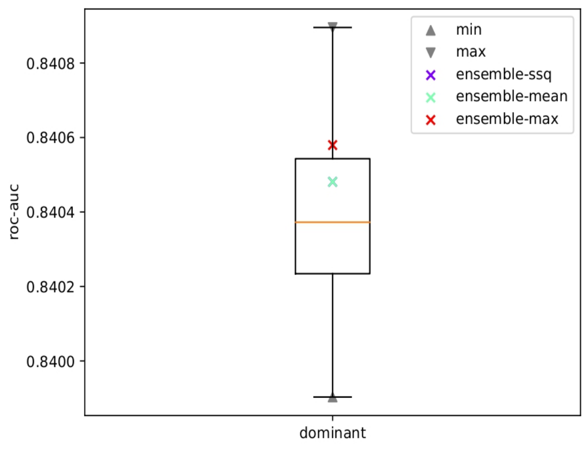
Given the observations that the GNN-DEAN model performs well in ensemble settings, where high variances of the base model contribute to a larger improvement in ensemble prediction, but final outcome is highly dependent on the performance of the base model. Here, we attempt to replace the base model GNN-DEAN using DOMINANT, a robust baseline model, in conducting a graph ensemble model. The experiment is conducted to answer the question: does it always guarantee a robust ensemble given two conditions—having a strong base model and an efficient graph sampling technique?

Ensemble with DOMINANT

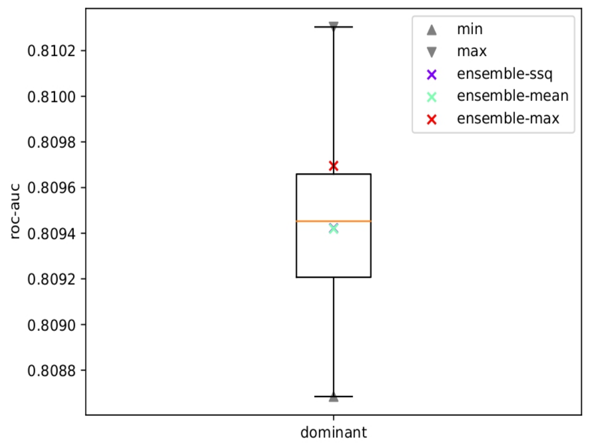
To demonstrate the significance of high variance in base models when creating ensemble models, we conducted an experiment using DOMINANT, a stable and high-performance baseline model, as the backbone architecture for our base model. The results in Figure 15 show that with a highly stable model architecture, there are fewer benefits to conducting an ensemble. Due to the sophisticated design of DOMINANT and the substantial computational resources required for training a submodel in a single iteration, the improvement in ensemble performance is relatively low.



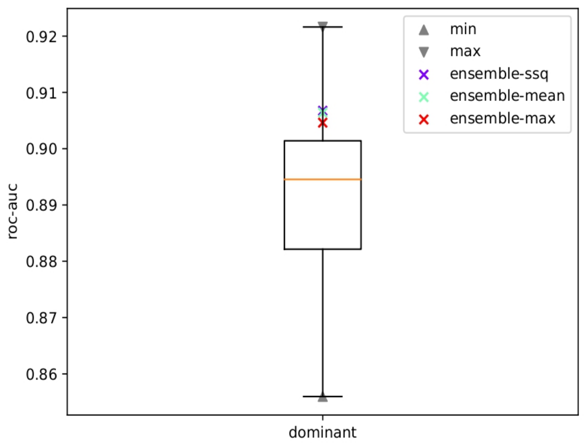
(a) inj_amazon



(b) inj_cora



(c) Reddit

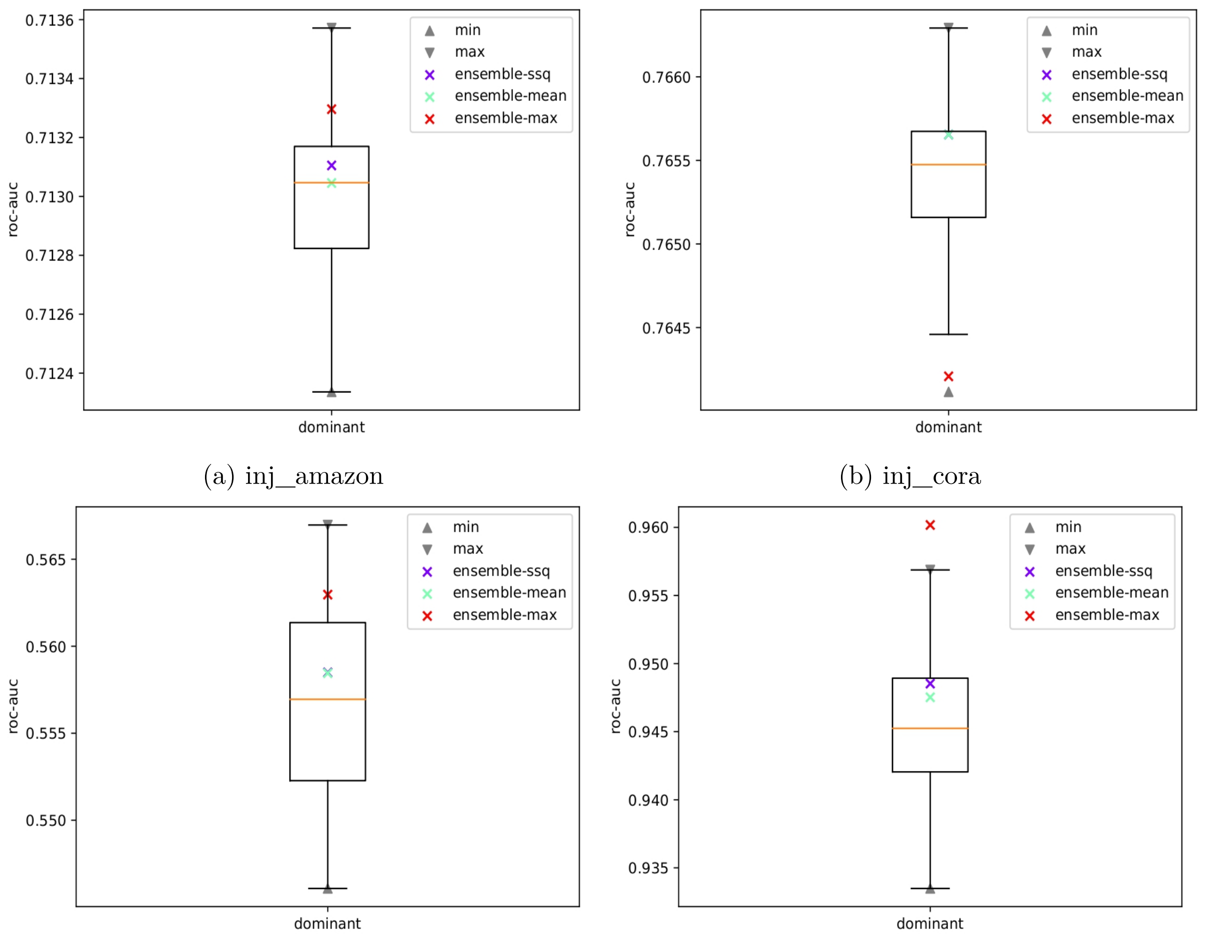


(d) Weibo

Figure 15: Performance of ensemble model with DOMINANT base models.

Perform feature-bagging with ensemble DOMINANT

Here, we perform feature-bagging to allow for higher variance in a strong baseline DOMINANT model. Despite the cost of expensive computation, we conducted an ensemble model using 100 base DOMINANT models, employing GraphSAGE for node sampling at the layer-wise level. The results are presented in Figure 16. While there is an improvement with the ensemble model on the Weibo dataset, the increase in ensemble prediction is moderate for the other datasets. This is due to a significant decrease in the performance of the base DOMINANT model. Since the base DOMINANT model still exhibits low variance, those ensembles are performing poorly.



(a) inj_amazon (b) inj_cora

(c) Reddit (d) Weibo

Figure 16: Performance of ensemble model with DOMINANT and layer-wise node sampling.

The experiment has demonstrated the critical importance of selecting an appropriate base model architecture to fully leverage the benefits of an ensemble detector. This choice ensures both decent performance and high variance in the base model, thereby enabling the potential for a superior ensemble within reasonable computational resources.

6 Conclusion and Future Work

The thesis focuses on enhancing the performance and scalability of an anomaly node detection model on graph datasets. This is achieved through the application of various graph sampling methods and graph ensemble techniques following DEAN concepts. DEAN provides a flexible framework that allows for the creation of an ensemble model from various architectures of base models. Additionally, the GNN-based DEAN model is highly scalable and can easily adapt to different types of graphs of varying sizes.

While the ensemble model has shown high potential to create a more robust outlier detector, its success is highly dependent on several factors in ensemble settings to achieve a superior ensemble that outperforms all base models [87]. Firstly, it requires a decent base model with high variance, contributing to a significant increment in the performance of the final ensemble. Secondly, in the context of graphs, it necessitates a base model that can be trained within reasonable computational effort, as computation explosion can easily occur as the size of the graph increases. A base model that performs reasonably well is a necessary condition for constructing a robust ensemble. However, the choice of base model architecture greatly influences the ensemble model, as the high variance of the base model is as important as its performance.

Graph sampling is not a trivial task, especially in the context of graph data where the connectivity dependency between entities poses challenges for computational complexity and efficient node representation learning. It is crucial to select a sampling technique that allows for the creation of high variance in the base model while ensuring efficient node representation learning from the sampled neighborhood. This once again highlights the uniqueness of graphs, where a node is represented by its own features and its connections to other nodes in the graph. The most effective sampling techniques in graph ensemble are those that satisfy two conditions: on the one hand, they randomly hide some information during neighborhood aggregation, and on the other hand, they occasionally give the model a chance to understand the entire graph information after many sampling iterations.

Throughout the experimentation section, we have demonstrated the effective application of the DEAN concept in the context of graphs. Despite the simple design of its base components, we have shown that a superior ensemble can be achieved through proper settings. Firstly, the "max" combination function, although seemingly biased, tends to yield superior ensemble output. Secondly, inferring outlier scores, such that $q = \text{mean}(x_{\text{train}})$, from the training dataset results in greater consistency between base model predictions and a more reliable ensemble model. Lastly, incorporating the bias term does not result in a trivial solution; instead, it contributes to the high variance in base model predictions, despite the simplicity of DEAN's learning objective function.

We have shown that employing a sophisticated architecture for the base model does not always lead to superior performance in ensemble settings. While a sophisticated design may result in a stronger base model, it often leads to a highly stable but low-variance base model. Consequently, a minor improvement in ensemble performance may not justify the computational expense.

The implemented GNN-DEAN models have shown promising results in detecting structural outliers, which are nodes that exhibit significant deviations in terms of their connections to other nodes. However, there is room for further improvement in detecting contextual outliers. This task holds promise, especially considering that node features are often represented as tabular datasets. The GNN-DEAN base model has numerous possibilities for incorporating its architecture with a component that utilizes off-the-shelf tabular detection methods, providing another source of information about node feature outlierness to the ensemble model.

Moreover, the GNN-DEAN models presented in this work mainly utilize the standard GCN layer for neighborhood feature aggregation, except for experiments with layer-wise node sampling utilizing the GraphSAGE layer. We are optimistic about further improvements to the proposed GNN-DEAN in various aspects. On one hand, optimizing hyperparameters of the DEAN base model, such as using a larger hidden dimension for large-scale graphs, adopting an early-stopping strategy to prevent overfitting of the base model, or finding optimal settings for employing the bias term partially, could lead to enhancements. On the other hand, the DEAN base model can utilize different backbone GNN layers. It is not limited to all base models sharing the same GNN architecture but could involve each base model employing a different core GNN model.

Finally, this work heavily utilizes graph sampling to increase model variance by modifying the graph structure during model fitting using layer-wise node sampling or subgraph sampling. There are open directions to combine graph sampling with node feature-based sampling methods in constructing a graph ensemble model.

Bibliography

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29:626–688, 2015.

[2] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z Sheng, Hui Xiong, and Leman Akoglu. A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12012–12038, 2021.

[3] Srikanth Thudumu, Philip Branch, Jiong Jin, and Jugdutt Singh. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, 7:1–30, 2020.

[4] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[5] Kay Liu, Yingtong Dou, Yue Zhao, Xueying Ding, Xiyang Hu, Ruitong Zhang, Kaize Ding, Canyu Chen, Hao Peng, Kai Shu, Lichao Sun, Jundong Li, George H. Chen, Zhihao Jia, and Philip S. Yu. Bond: Benchmarking unsupervised outlier node detection on static attributed graphs. *Advances in Neural Information Processing Systems*, 35:27021–27035, 2022.

[6] Jinghui Chen, Saket Sathe, Charu Aggarwal, and Deepak Turaga. Outlier detection with autoencoder ensembles. In *Proceedings of the 2017 SIAM international conference on data mining*, pages 90–98. SIAM, 2017.

[7] Charu C Aggarwal and Charu C Aggarwal. *Outlier ensembles*. Springer, 2017.

[8] Yihong Huang, Liping Wang, Fan Zhang, and Xuemin Lin. Unsupervised graph outlier detection: Problem revisit, new insight, and superior method. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 2565–2578. IEEE, 2023.

[9] Hanghang Tong and Ching-Yung Lin. *Non-Negative Residual Matrix Factorization with Application to Graph Anomaly Detection*, pages 143–153.

[10] Daniel Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[11] Charu C Aggarwal and Charu C Aggarwal. *An introduction to outlier analysis*. Springer, 2017.

[12] Caleb C Noble and Diane J Cook. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636, 2003.

[13] Diane J Cook and Lawrence B Holder. Graph-based data mining. *IEEE Intelligent Systems*

and Their Applications, 15(2):32–41, 2000.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[16] Robert E Schapire et al. A brief introduction to boosting. In *Ijcai*, volume 99, pages 1401–1406. Citeseer, 1999.

[17] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[18] Peter Bühlmann and Bin Yu. Analyzing bagging. *The annals of Statistics*, 30(4):927–961, 2002.

[19] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166, 2005.

[20] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[21] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[22] SGOPAL Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.

[23] Jing Gao and Pang-Ning Tan. Converting output scores from outlier detection algorithms into probability estimates. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 212–221. IEEE, 2006.

[24] Steve Lawrence and C Lee Giles. Inquirus, the neci meta search engine. *Computer networks and ISDN systems*, 30(1-7):95–105, 1998.

[25] B Uygar Oztekin, George Karypis, and Vipin Kumar. Expert agreement and content based reranking in a meta search environment using mearf. In *Proceedings of the 11th international conference on World Wide Web*, pages 333–344, 2002.

[26] Erich Schubert, Remigius Wojdanowski, Arthur Zimek, and Hans-Peter Kriegel. On evaluation of outlier rankings and outlier scores. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 1047–1058. SIAM, 2012.

[27] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[28] Jiawei Han, Jian Pei, and Hanghang Tong. *Data mining: concepts and techniques*. Morgan

kaufmann, 2012.

[29] William L Hamilton. *Graph representation learning*. Morgan & Claypool Publishers, 2020.

[30] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

[31] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[32] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48, 2013.

[33] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

[34] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114, 2016.

[35] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[36] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[37] Jiongqian Liang, Peter Jacobs, Jiankai Sun, and Srinivasan Parthasarathy. Semi-supervised embedding in attributed networks with outliers. In *Proceedings of the 2018 SIAM international conference on data mining*, pages 153–161. SIAM, 2018.

[38] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[39] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017.

[40] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[41] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In

Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pages 855–864, 2016.

[42] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[43] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[44] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[46] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[47] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[48] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[49] Mark Newman. *Networks*. Oxford university press, 2018.

[50] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

[51] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, 25(2):163–177, 2001.

[52] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Advances in Knowledge Discovery and Data Mining: 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II 14*, pages 410–421. Springer, 2010.

[53] Emmanuel Müller, Patricia Iglesias Sánchez, Yvonne Mülle, and Klemens Böhm. Ranking outlier nodes in subspaces of attributed graphs. In *2013 IEEE 29th international conference on data engineering workshops (ICDEW)*, pages 216–222. IEEE, 2013.

[54] Patricia Iglesias Sánchez, Emmanuel Müller, Fabian Laforet, Fabian Keller, and Klemens Böhm. Statistical selection of congruent subspaces for mining attributed graphs. In *2013 IEEE 13th international conference on data mining*, pages 647–656. IEEE, 2013.

[55] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. Focused clustering and outlier detection in large attributed graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1346–1355, 2014.

[56] Patricia Iglesias Sánchez, Emmanuel Müller, Oretta Irmler, and Klemens Böhm. Local context selection for outlier ranking in graphs with multiple numeric node attributes. In *Proceedings of the 26th international conference on scientific and statistical database management*, pages 1–12, 2014.

[57] Renjun Hu, Charu C Aggarwal, Shuai Ma, and Jinpeng Huai. An embedding approach to anomaly detection. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 385–396. IEEE, 2016.

[58] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[59] Ville Hautamaki, Ismo Karkkainen, and Pasi Franti. Outlier detection using k-nearest neighbour graph. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, pages 430–433. IEEE, 2004.

[60] Ayushi Dalmia and Manish Gupta. Towards interpretation of node embeddings. In *Companion Proceedings of the The Web Conference 2018*, pages 945–952, 2018.

[61] Neha Yadav and Dhanalekshmi Gopinathan. Node embedding approach for topic search from academic papers. In *2022 8th International Conference on Signal Processing and Communication (ICSC)*, pages 445–450. IEEE, 2022.

[62] Olayinka Adeleye, Jian Yu, Ji Ruan, and Quan Z Sheng. Evaluating random walk-based network embeddings for web service applications. In *Databases Theory and Applications: 31st Australasian Database Conference, ADC 2020, Melbourne, VIC, Australia, February 3–7, 2020, Proceedings 31*, pages 198–205. Springer, 2020.

[63] Kento Nozawa, Masanari Kimura, and Atsunori Kanemura. Analyzing centralities of embedded nodes. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 1046–1049. IEEE, 2018.

[64] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. In *Proceedings of the IEEE foundations and new directions of data mining workshop*, pages 172–179. IEEE Press, 2003.

[65] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[66] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau. Autoencoder-based network anomaly detection. In *2018 Wireless Telecommunications Symposium (WTS)*, pages 1–5, 2018.

[67] Sambaran Bandyopadhyay, Lokesh N, Saley Vishal Vivek, and M Narasimha Murty. Outlier resistant unsupervised deep architectures for attributed network embedding. In *Proceedings of the 13th international conference on web search and data mining*, pages 25–33, 2020.

[68] Wasif Khan, Nazar Zaki, Amir Ahmad, Mohammad M Masud, Romana Govender, Natalia Rojas-Perilla, Luqman Ali, Nadirah Ghenimi, and Luai A Ahmed. Node embedding-based graph autoencoder outlier detection for adverse pregnancy outcomes. *Scientific reports*, 13(1):19817, 2023.

[69] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. Deep anomaly detection on attributed networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 594–602. SIAM, 2019.

[70] Haoyi Fan, Fengbin Zhang, and Zuoyong Li. Anomalydae: Dual autoencoder for anomaly detection on attributed networks. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5685–5689, 2020.

[71] Zhen Peng, Minnan Luo, Jundong Li, Luguo Xue, and Qinghua Zheng. A deep multi-view framework for anomaly detection on attributed networks. *IEEE Transactions on Knowledge and Data Engineering*, 34(6):2539–2552, 2022.

[72] Kaize Ding, Jundong Li, Nitin Agarwal, and Huan Liu. Inductive anomaly detection on attributed networks. In *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*, pages 1288–1294, 2021.

[73] Zhenxing Chen, Bo Liu, Meiqing Wang, Peng Dai, Jun Lv, and Liefeng Bo. Generative adversarial attributed network anomaly detection. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1989–1992, 2020.

[74] Benedikt Böing, Simon Klüttermann, and Emmanuel Müller. Post-robustifying deep anomaly detection ensembles by model selection. In *ICDM*, 2022.

[75] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[76] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Scalable graph neural networks via bidirectional propagation. *Advances in neural information processing systems*, 33:14556–14566, 2020.

[77] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[78] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in neural information processing systems*, 32, 2019.

[79] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019.

[80] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[81] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[82] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[83] Tong Zhao, Chuchen Deng, Kaifeng Yu, Tianwen Jiang, Daheng Wang, and Meng Jiang. Error-bounded graph anomaly loss for gnns. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1873–1882, 2020.

[84] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1269–1278, 2019.

[85] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S Yu. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 315–324, 2020.

[86] Shebuti Rayana and Leman Akoglu. Collective opinion spam detection: Bridging review networks and metadata. In *Proceedings of the 21th acm sigkdd international conference on knowledge discovery and data mining*, pages 985–994, 2015.

[87] Simon Klüttermann and Emmanuel Müller. Evaluating and comparing heterogeneous ensemble methods for unsupervised anomaly detection. pages 1–8, 06 2023.

[88] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[89] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[90] Kay Liu, Yingtong Dou, Yue Zhao, Xueying Ding, Xiyang Hu, Ruitong Zhang, Kaize Ding, Canyu Chen, Hao Peng, Kai Shu, George H. Chen, Zhihao Jia, and Philip S. Yu. Pygod: A python library for graph outlier detection. *arXiv preprint arXiv:2204.12095*, 2022.

[91] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

List of Figures

List of Tables

Appendix

Online resources

We utilized the following Python libraries for the implementation of our experimentation: Pytorch [88], PyTorch Geometric [89], PyGOD [90], and Scikit-Learn [91].

The source code for experimentation is available online at Github repository[1]. The repository contains the code for preparing training datasets, implementing GNN-DEAN, defining model parameters, and producing figure results.

It should be noted that the work utilizes ChatGPT to enhance the grammar of the text.

[1]https://github.com/anhntn15/god

Eidesstattliche Versicherung

(Affidavit)

Nguyen, Thi Ngoc Anh
Name, Vorname
(surname, first name)

230975
Matrikelnummer
(student ID number)

☐ Bachelorarbeit
(Bachelor's thesis)

■ Masterarbeit
(Master's thesis)

Titel
(Title)

Ensemble methods for outlier node detection on attributed static graphs

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund, 15.04.2024
Ort, Datum
(place, date)

Anh
Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund, 15.04.2024
Ort, Datum
(place, date)

Anh
Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**