BRAUDE

College of Engineering, karmiel

Software Engineering Department Braude

Academic College

Capstone Project Phase A – 61998

Spotting Suspicious Academic Citations Using Self-Learning Graph Transformers

24-1-R-16

Almog Madar, Mor Ben Haim

Almog.Madar@e.braude.ac.il

Mor.Ben.Haim@e.braude.ac.il

Supervised by
Prof. Zeev Volkovich

Advised by

Dr. Renata Avros

Book Repository
Spotting-Suspicious-Citations-Link

Table of Contents

Abstract

This book delves into a novel approach to identify potential citation manipulation within academic papers. This method utilizes perturbations of a deep embedding model, integrating Graph-Masked Autoencoders to merge textual information with evidence of graph connectivity. Consequently, it yields a more intricate model of citation distribution. By training a deep network with partial data and reconstructing masked connections, the approach capitalizes on the inherent characteristics of central connections amidst network perturbations. It demonstrates its ability to pinpoint trustworthy citations within the analyzed dataset through comprehensive quantitative evaluations. Additionally, it raises concerns regarding the reliability of specific references, which may be subject to manipulation.

Keywords: graph-masked autoencoders; manipulated citations; network perturbation.

1. Introduction

Manipulated citations involve the intentional inclusion of references in academic works to gain biased advantages rather than for their authentic academic merit. Rather than supporting the author's arguments or providing relevant background information, the primary aim is to artificially boost the citation count of the author, their work, a specific journal, and so forth. There are several key aspects associated with manipulated citations. They are commonly utilized to enhance researchers' perceived impact and prestige and artificially inflate journal metrics. A widespread practice in-volves promoting citation manipulation to increase a journal's citation impact factor, eventually assessing the journal's quality and prestige.

2. Background

2.1 The importance of detecting citation manipulation

The manipulation of citations in academic works is a practice that has far-reaching detrimental effects, undermining the very foundations of academic discourse: precision, impartiality, and scientific credibility. Researchers have acknowledged the uneven value of citations and have attempted to address this issue by differentiating and assigning weights based on the type of citation. However, most studies have focused solely on this specific approach, neglecting to consider the issue from a broader perspective.

Prabha [1] has highlighted the extent of the problem, revealing that more than two-thirds of references in a paper are deemed unnecessary. This finding provides further evidence of the widespread presence of dubious citations. The practice of citation manipulation poses a serious threat to the scientific community, eroding trust in research, distorting the research landscape by inflating citation counts for personal gain, and leading to the misallocation of resources.

To uphold the integrity of academic research, safeguard its credibility, and ensure its reliability, it is crucial to understand and address the issue of manipulated citations. This involves implementing measures such as establishing journal policies, providing training for reviewers, promoting transparency among researchers, and improving metrics. The issue of citation manipulation is a complex one [2-4], with many facets that need to be considered. It is not just about the number of citations, but also about the quality and relevance of those citations. By taking a comprehensive approach to tackling this issue, we can help to ensure that academic research remains a reliable and trustworthy source of knowledge.

2.2 Capturing Patterns Using Network-Based Approaches

In the past, manual inspection techniques and basic statistical analyses were used, but they have limitations in capturing complex patterns and subtle manipulations. Therefore, in recent years, new network-based approaches have emerged as promising methods for spatially identifying and understanding manipulations and citations. Graphs have a complex nature that allows for the identification of irregular structures and dependencies between relations. Undoubtedly, it has become a central player against conventional anomaly detection techniques.

By adopting the structure and relationships existing within the citation network, network-based methods could detect hidden and anomalous relations that indicate citation manipulation. The analysis is not focused on a single paper an individual but delves into the dynamics of the entire broad network, in order to generate a deep and comprehensive understanding of manipulation patterns. This approach is crucial for maintaining the integrity and reliability of academic research.

There are numerous research articles, such as [5,6,8] , that have focused on the use of deep learning methods to detect rumors and fake news within networks. The insights from these articles are fundamentally important in developing network-based approaches to detect manipulative citations in academic research. These studies highlight the potential of using advanced computational techniques to maintain the integrity of scholarly discourse [7,9].

2.3 Machine Learning

Machine learning is a subfield of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to perform tasks without explicit instructions. In other words, machine learning involves the creation of algorithms that can learn from and make decisions or predictions based on data.

2.4 Self-supervised learning

Self-supervised learning is a type of machine learning where the model learns to predict part of the input data from other parts of the input data. In the context of an autoencoder, self-supervision can be used to learner presentations of the input data in an unsupervised manner.

2.5 Neural Networks

A neural network, also known as an artificial neural network (ANN), is a subset of machine learning that is structured to mimic the human brain. It's a powerful tool in the field of artificial intelligence, enabling computers to learn from and make decisions based on data.

Neural networks consist of layers of nodes, each node designed to behave similarly to a neuron in the human brain. These layers include an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network.

2.6 Citation Network

A citation network, or citation graph, is a directed graph that depicts the citations within a collection of documents. Each node (or vertex) in the graph represents a document in the collection, and each directed edge from one document to another represents a citation. Citation networks allow us to explore the intellectual structures of research fields and provide useful information for knowledge organization and information retrieval.

2.7 BOW (Bag of Word)

The BOW model is a simplifying representation [Fig 1] used in natural language processing and information retrieval. In this model, a text (such as a sentence or an article) is represented as the bag of its words, disregarding grammar and even word order but keeping multiplicity.

The BOW model consists of two main components:

1. A vocabulary of known words.
2. A measure of the presence of these known words.

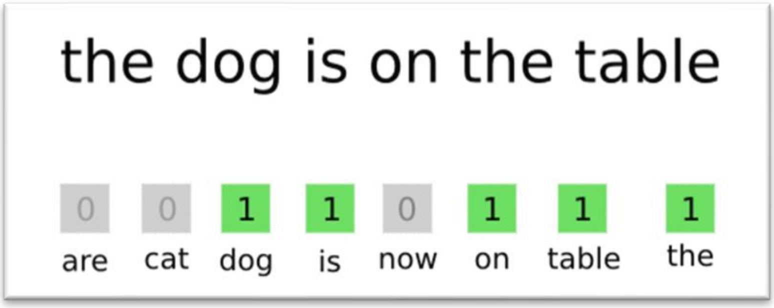


Fig 1:BOW representation

The term "bag" of words signifies that the model disregards any information about the order or structure of words in the document. The Bag of Words (BoW) model can be a valuable tool for text analysis in machine learning. The BoW model transforms text into a numerical representation that can be used as input for a machine learning algorithm. This process is often referred to as feature extraction or feature encoding.

2.8 Ego-Graph

An ego-graph, also known as an ego network, is a subgraph in graph theory that focuses on one node, known as the 'ego', and its immediate neighbors. These neighbors are often referred to as 'alters'. The ego-graph includes the ego, the alters, and all the edges among them.

2.9 Gophormer

Gophormer [13] is a novel model that has been proposed for tackling node classification tasks, a common problem in the realm of graph mining. Unlike traditional approaches that operate on full graphs, Gophormer innovatively implements transformers on ego-graphs. By focusing on ego-graphs, Gophormer can capture the local neighborhood information of each node, which can be particularly useful for node classification tasks.

2.10 Graphormer

Graphormer is an innovative model that has been proposed for tasks in the field of graph mining. It's a Graph Transformer model, modified to allow computations on graphs instead of text sequences by generating embeddings and features of interest during preprocessing and collation.

Graphormer [12] proposed three types of positional encodings to jointly capture the structural information of graphs, which are the centrality encoding, the spatial encoding, and the edge encoding. They encode the degree centralities, the shortest path distances, and the edge features respectively.

2.11 Masking mechanism

The masking mechanism in machine learning models, particularly in graph transformers, plays a crucial role in managing computational resources. By reducing the size of the input feature matrix, it significantly decreases memory consumption. This is particularly beneficial when dealing with large datasets. In our project, we leverage the power of graph transformers for our machine learning tasks. One challenge with graph transformers is their quadratic memory consumption with respect to the input length. To address this, we employ a masking mechanism.

2.12 Graph Masked Autoencoders (GMAEs)

Graph Masked Autoencoders (GMAEs) [10] are a type of self-supervised, transformer-based model designed for learning graph representations. The key features of GMAEs include the use of a masking mechanism and an asymmetric encoder-decoder design.

The masking mechanism in GMAEs involves taking partially masked graphs as input. The purpose of this is to reconstruct the features of the masked nodes, which is a form of self-supervised learning. This means that the model is trained to predict or fill in the missing parts of the input data, which can help it learn useful features and representations of the data.

The encoder and decoder in GMAEs are asymmetric. The encoder, which is responsible for transforming the input data into an encoded representation, is a deep transformer. This means it has multiple layers, allowing it to learn more complex patterns in the data. On the other hand, the decoder, which is responsible for reconstructing the input data from the encoded representation, is a shallow transformer. This means it has fewer layers, which can make it more efficient and faster to train.

The combination of the masking mechanism and the asymmetric design makes GMAEs a memory-efficient model compared to conventional transformers. This can be particularly beneficial when working with large datasets or complex tasks, as it can help to reduce computational requirements and improve performance.

2.13 Input Embedding

In the context of our model, embeddings play a crucial role. They are essentially a form of feature extraction, where high-dimensional data is transformed into a lower-dimensional space. This lower-dimensional space is designed to capture the essential characteristics of the original data.

For text data, we use the Bag of Words (BoW) method to create a word vector for each article. This transforms the text of each article into a numerical representation that can be processed by our model.

For graph data, we use Gophormer to create an ego-graph for each article. This involves transforming the graph structure of each article into a lower-dimensional embedding. The ego-graph captures the local neighborhood information of each node in the graph, which can be particularly useful for tasks such as node classification.

Finally, we use Graphormer positional encodings to capture the structural information of graphs. These encodings include centrality encoding, spatial encoding, and edge encoding. Each of these encodings transforms different aspects of the graph structure into a numerical form that can be processed by our model.

In summary, embeddings are a powerful tool for transforming complex, high-dimensional data into a form that can be effectively.

3. Mathematical Background

3.1 Asymmetric Encoder-Decoder Architecture (Autoencoder)

An autoencoder is a specific type of artificial neural network that is utilized for the purpose of learning efficient representations, or encodings, of input data. This is achieved through a process of dimensionality reduction, where the goal is to learn a compressed representation of the input data. Autoencoder are a type of neural network that operate in an unsupervised learning paradigm.

An asymmetric autoencoder is a type of autoencoder that has an unequal number of encoders and decoders[Fig 2].

The asymmetric structure reduces the number of parameters to be estimated and potential overfitting. It has been found that such autoencoders are more accurate compared to traditional symmetrically stacked autoencoders for classification accuracy and yield slightly better results on compression problems.

In some implementations, an asymmetric residual deep autoencoder is constructed to learn the features embedded in high-dimensional data. The asymmetric residual autoencoder uses a residual connection to enhance the feature extraction ability of the encoder with a deeper network, while a shallow CNN is adopted as the decoder.

The objective of an autoencoder is to minimize the difference, or error, between the original input and the reconstructed output, thereby learning an efficient and compact representation of the input data. Internally, it has a hidden layer h that describes a code used to represent the input.

The network may be viewed as consisting of two parts, an encoder function:

$$h = f(x)$$

And a decoder that produces a reconstruction:

$$r = g(h)$$

If an autoencoder succeeds in simply learning to set:

$$g(f(x)) = x$$

everywhere, then it is not especially useful. Instead, autoencoders are designed to be unable to learn to copy perfectly. They are restricted in ways that allow them to copy only approximately. And to copy only inputs that resemble the training data. Because the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data.

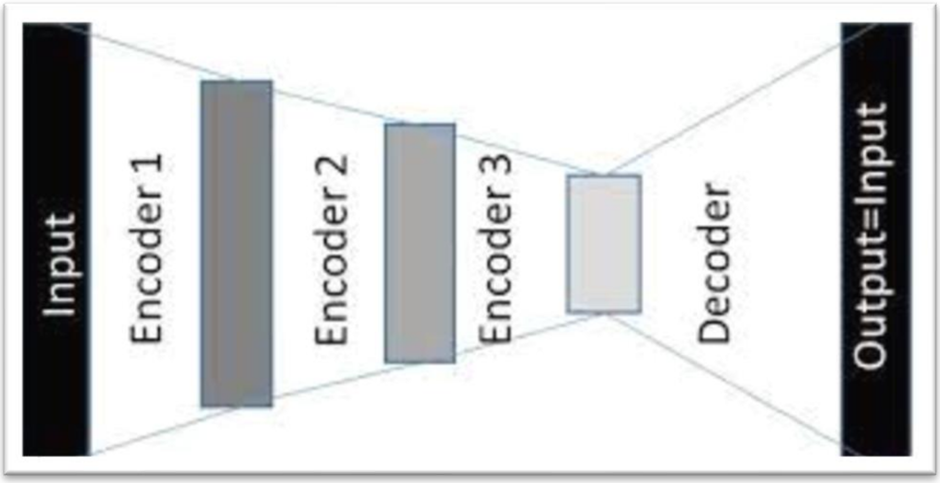


Fig 2: Asymmetric Autoencoder

3.2 Graph Transformers

Graph Transformers are a powerful type of neural network designed to excel with data structured as graphs. Unlike traditional transformers that work best with sequential data like text, Graph Transformers can leverage the inherent structure and features within a graph to learn informative representations.

3.2.1 Key Strengths of Graph Transformers

- **Attention to Graph Structure:** They go beyond just node features, incorporating the connections and relationships between nodes to understand the bigger picture. This makes them ideal for tasks where edge information is crucial, such as analyzing molecule bonds or knowledge graph relationships.
- **Capturing Complex Relationships**: The attention mechanism allows Graph Transformers to focus on relevant parts of the graph, similar to traditional transformers. This enables them to capture intricate, long-range dependencies within the data.
- **Generalization of Transformers**: Graph Transformers build upon the foundation of traditional transformers, but with key adaptations for graph data. They utilize Laplacian eigenvectors for position encoding, naturally aligning with the graph structure compared to the sinusoidal encodings

used in NLP. Additionally, batch normalization replaces layer normalization for faster training and improved generalization.

- **Edge Representation Advantage**: Graph Transformers can incorporate edge features into their architecture. This proves particularly valuable for tasks where the interactions between nodes hold significant information, like bond types in molecules or relationships in knowledge graphs.

3.2.2 Mathematical Foundations

Graph Transformers rely on several key mathematical concepts to operate on graph data. Here's a brief overview:

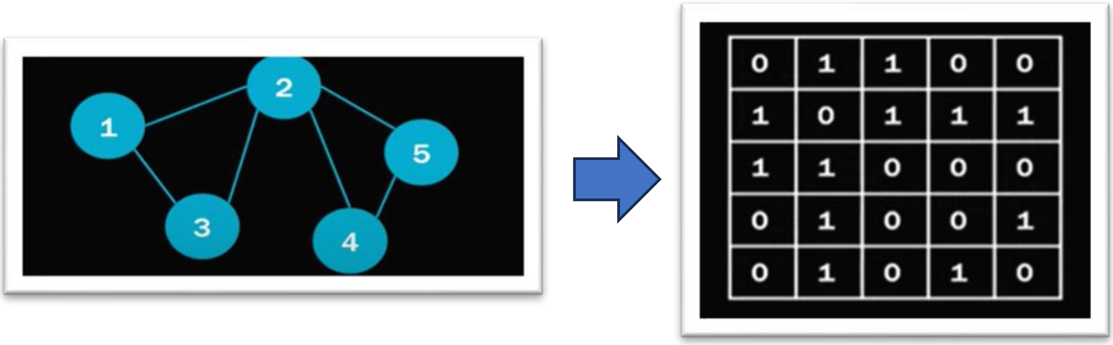
- **Adjacency Matrix (A):** This square matrix captures the connections within the graph. A value of 1 at position (i, j) indicates an edge between node i and node j, while 0 signifies no connection.
- **Degree Matrix (D):** This diagonal matrix holds the degree (number of connections) for each node on the diagonal.
- **Laplacian Matrix (L):** Defined as D - A, the Laplacian matrix captures the graph's structure. Decomposing this matrix into eigenvectors allows each node to select a subset for its positional encoding vector.

By effectively combining graph structure, node features, and attention mechanisms, Graph Transformers offer a promising approach to learning meaningful representations from graph data. This makes them a valuable tool for various tasks involving structured information.

3.3 Attention mechanism in Graph Transformers

In graph theory, a graph G is typically described as an ordered pair $G = (V, E)$, where V is the set of vertices (or nodes) and E is the set of edges.

Suppose there are n_V nodes and n_E edges in a graph. An adjacency matrix $A \in R^{n_V \times n_V}$ [Fig 3]is used to describe the connections among nodes. Each entry in A is either 1 or 0, representing whether there is an edge between the two nodes or not.



0	1	1	0	0
1	0	1	1	1
1	1	0	0	0
0	1	0	0	1
0	1	0	1	0

Fig 3: Adjacency matrix of graph

In addition to the adjacency matrix, some graphs also have auxiliary information such as node features [Fig 4] $X_V \in R^{n_V \times d_V}$ and edge features $X_E \in R^{n_E \times d_E}$ Here, d_V and d_E are the numbers of dimensions of node features and edge features respectively.

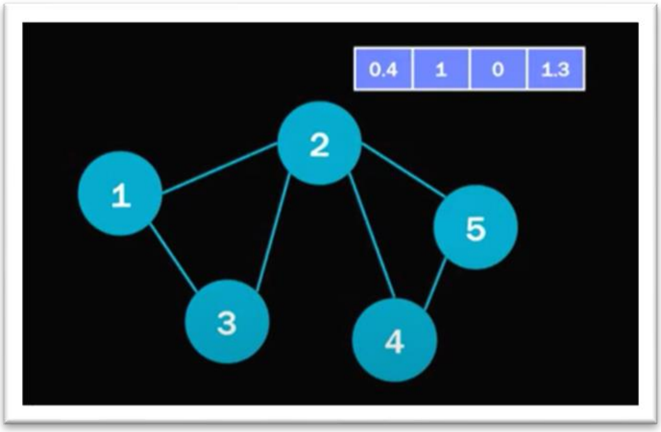


Fig 4: Node #2 vector features

Self-supervised learning of graph neural networks (**GNNs**) is a learning paradigm that aims to learn an accurate representation of the graphs in an unsupervised manner. The goal is to train a model f that takes the graph data (without labels) as input and outputs the embeddings of each node.

In this context, we assume that we only have node features and edge features as auxiliary information. Thus, the model f can be expressed as:

$$f(A, X_V, X_E) = H_o$$

Where $H_o \in R^{n_V \times d_o}$ contains the output embeddings of each node and do is the number of output dimensions.

A Transformer is a type of neural network architecture that is composed of several layers. Each layer in a Transformer contains two main components: a **self-attention module** and a **feed-forward network**.

Let's assume the input hidden representation matrix [Fig 5] is $H \in R^{n \times d}$, where n is the number of instances (or nodes in the case of graph data), and d is the number of hidden dimensions.

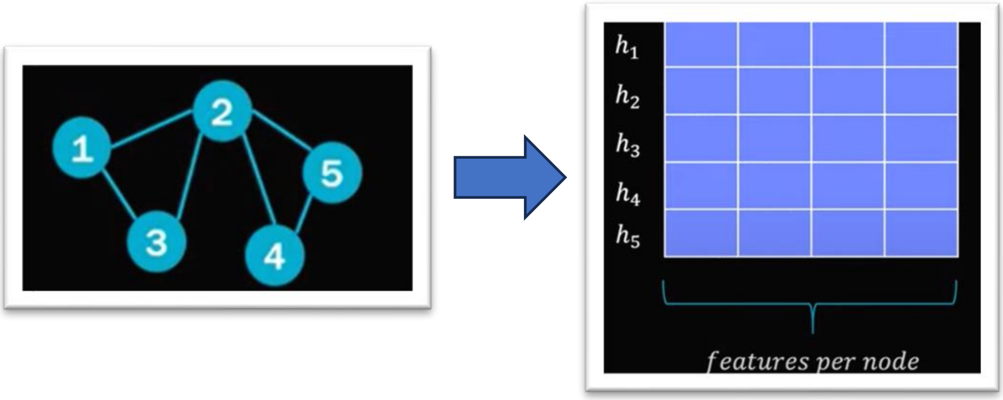


Fig 5: Input hidden representation matrix

The self-attention module contains three learnable matrices:

$$W_Q \in R^{d \times d_q} , W_K \in R^{d \times d_k}, \text{ and } W_V \in R^{d \times d_v}$$

Which stand for **Queries**, **Keys**, and **Values** respectively. We usually have $d_q = d_k$ due to the self-attention mechanism. The self-attention module first computes the three matrices as follows:

$$Q = HW_Q, K = HW_K, V = HW_V$$

Then, the dot product is applied to the queries with all keys to obtain the weights on the values. The final output matrix is computed by:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

The obtained output matrix is a matrix with size $n \times d_v$ [Fig 6] where each row represents the output representation of the corresponding instance.

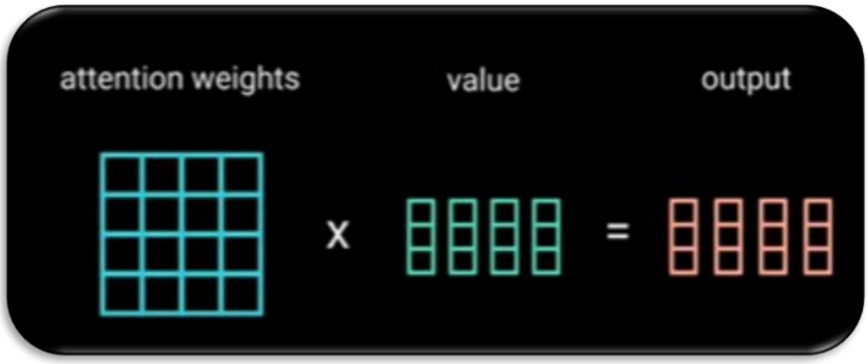


Fig 6: Obtained output matrix

Then, a 2-layer feed-forward network (FFN) is used to project the node representations x and increase model capacity. The FFN is defined as:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

where W_1 and W_2 are learnable weight matrices, and b_1 and b_2 are learnable biases.

3.4 Shared learnable mask token

"Shared learnable mask token" refers to models of artificial neural networks, particularly those that use a method called "Masked language modeling" or "Masked image modeling". During training, the model replaces inputs at certain locations with special characters called "MASK token" or "learnable mask embedding". The "MASK token" is a learned vector that serves as a placeholder for a missing node feature that needs to be predicted. When the model sees this token, it tries to "fill in" the representation of the missing information, according to the context of the surrounding information.

In the context of a game, the mask token, denoted as x_m, plays a crucial role in the process of learning from the game's data. This mask token is used to replace the features of certain nodes (in this case, the "masked" nodes) in the game's data structure.

3.4.1 Working process

- Initialization
  We start with an output embedding matrix X_e from the encoder.
  We then initialize a mask token
  $x_m \in R^{1 \times d_e}$ and use it to replace the masked nodes in X_e , resulting in a new matrix X'_e.
- Shared Mask Token
  The mask token $x_m \in R^{1 \times d_e}$ is shared across all masked nodes. This means that all masked nodes use the same mask token.
- Updating the Mask Token
  The mask token $x_m \in R^{1 \times d_e}$ is updated in each iteration through backpropagation, just like all other learnable parameters in the model.
- Reconstruction
  The output of the decoder is the reconstructed node features of the masked nodes. A typical reconstruction loss (like cross-entropy loss) is applied to these reconstructed features and the original features of the masked nodes.

In essence, the mask token $x_m \in R^{1 \times d_e}$ serves as a placeholder for the masked nodes during the learning process. It allows the model to learn from the rest of the data, and then uses what it has learned to try and reconstruct the features of the masked nodes. This process is crucial for the model to effectively learn from the game’s data and make accurate predictions or decisions based on that data.

4. Research Process

4.1 Model overview

This section outlines a method to spot unusual citation patterns in academic networks, suggesting potential manipulation. By removing nodes randomly and observing the network’s response, we can identify citations that don’t fit the expected pattern, revealing possible fraudulent activities.

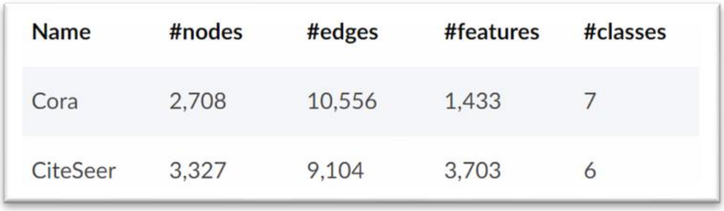
After obtaining node embeddings, we measure node similarity with metrics like cosine similarity. Treating the citation graph as undirected, we focus on paper interconnectivity. We introduce a similarity measure (S) and a threshold value (Tr) to refine link predictions. If the similarity score exceeds Tr, nodes are linked. otherwise, they’re not, allowing for a tailored analysis of network links.

In the context of the graph masking process, the fraction (Fr) represents the proportion of nodes randomly masked during each iteration of the training phase. Specifically, if (Fr) is set to 0.3, this indicates that 30% of the nodes are concealed from the model at each step, mirroring the distribution of the training and test datasets. This masking strategy is instrumental in enabling the model to learn from a subset of the data, thereby enhancing its robustness and predictive accuracy when encountering new, unseen information.

4.2 Datasets

Planetoid datasets are used for semi-supervised learning research on graphs. They include the Cora and CiteSeer repositories [Fig 7], which contain scientific publications. In these datasets, each node represents a document, and each edge represents a citation link. The classes in these datasets represent the topics of the documents. Cora has 7 classes, while CiteSeer has 6 classes.

- Cora dataset (accessible at Planetoid) .
- CiteSeer dataset (accessible at Planetoid) .



Name	#nodes	#edges	#features	#classes
Cora	2,708	10,556	1,433	7
CiteSeer	3,327	9,104	3,703	6

Fig 7: Cora and CiteSeer repositories

4.3 Model Architecture

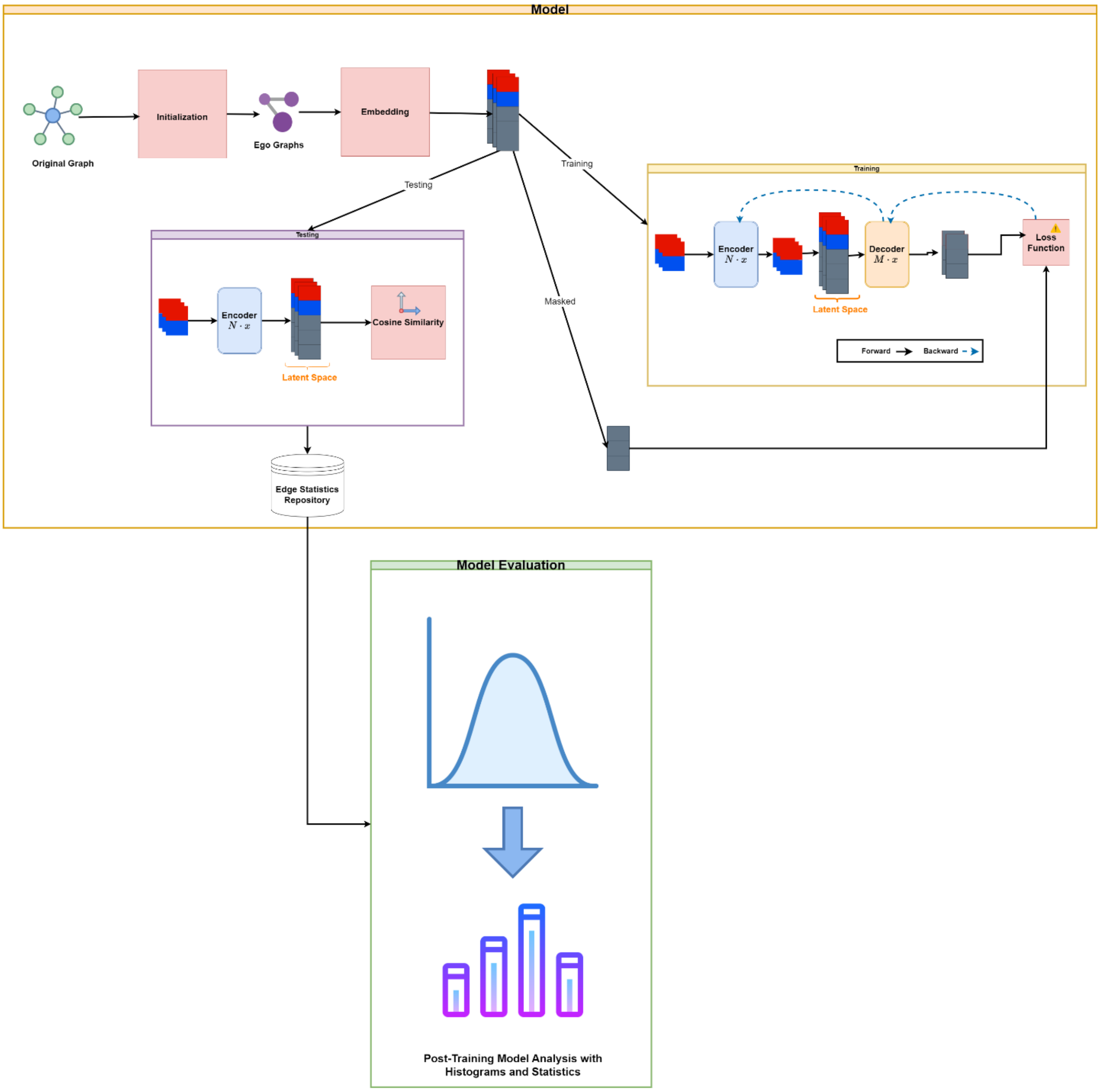


Fig 8: Model Architecture

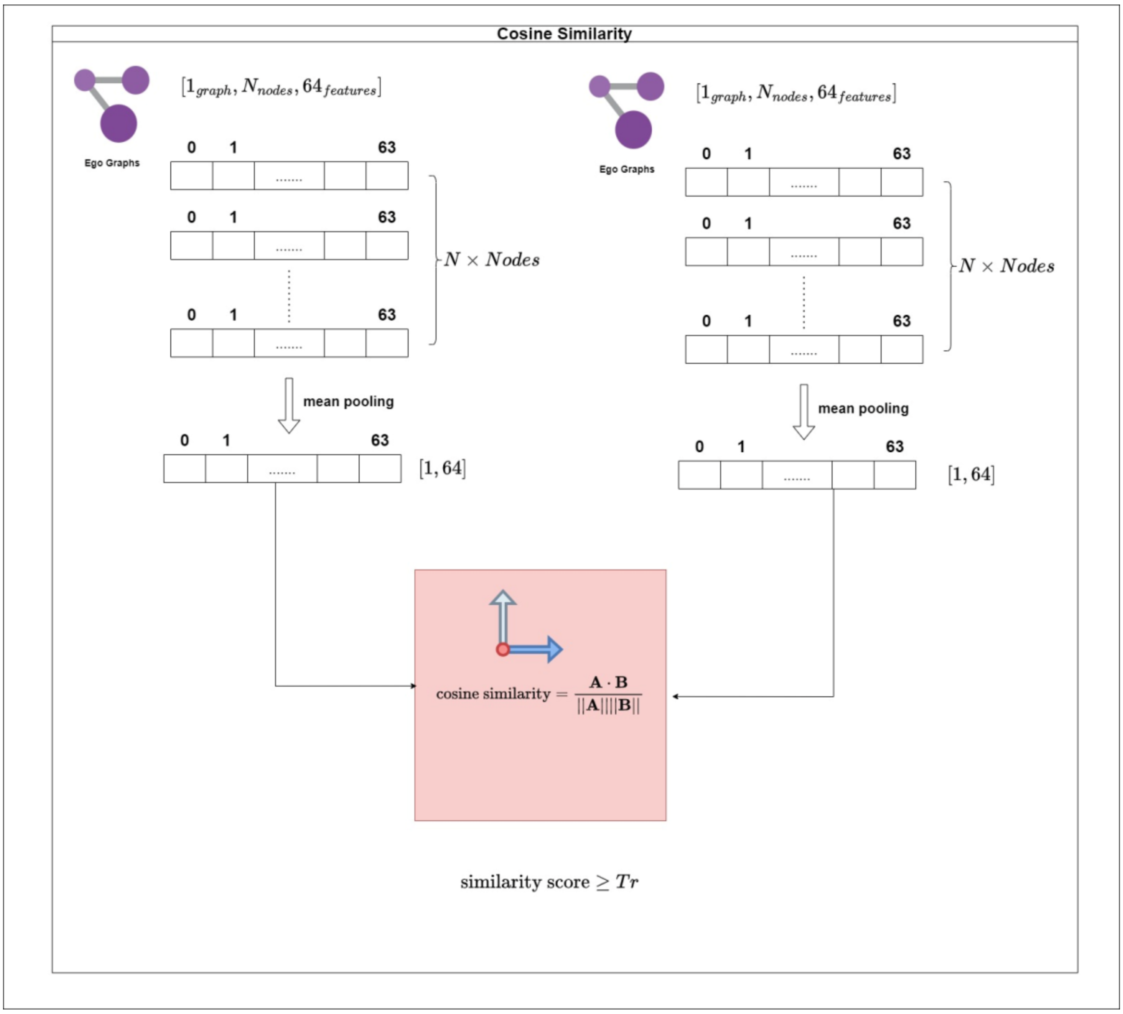

Cosine Similarity
Ego Graphs
$[1_{graph}, N_{nodes}, 64_{features}]$
0 1 63
....
$N \times Nodes$
mean pooling
$[1, 64]$
Ego Graphs
$[1_{graph}, N_{nodes}, 64_{features}]$
0 1 63
....
$N \times Nodes$
mean pooling
$[1, 64]$
cosine similarity $= \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| ||\mathbf{B}||}$
similarity score $\geq Tr$

Fig 9: Latent space Cosine Similarity

4.3.1 Initialization

During the initialization phase [Fig 8], we effectively perform an import of a Planetoid-type graph, which represents either Cora or CiteSeer as appropriate. Each paper within the dataset is represented by a bag-of-words feature vector, indicating the presence (1) or absence (0) of specific words in the document. Additionally, we construct a database for the purpose of tracking the removal of edges. This data structure, referred to as the GraphDataModule, encapsulates all the aforementioned information.

Using the Fr value, we split our dataset into training and testing sets. Here, we ensure that there are no corresponding edges between these sets (training_set and testing_set). For each group (training_set and testing_set), we utilize Gophermer to create an ego-graph for each node, incorporating its neighbors up to a depth of L2.

This process is crucial for establishing the foundation of our graph-based semi-supervised learning system. It enables efficient tracking and manipulation of graph edges, paving the way for subsequent stages of model training and analysis.

4.3.2 Embedding

After the initialization stage, we aim to generate a positional encoding that captures the relative position of each node within the graph [Fig 8]. To achieve this, we leverage the Graphomer behavior to generate three encodings:

1. Eigenvector Centrality Encoding: This encoding reflects the influence or importance of a node within the graph structure. Nodes with higher eigenvector centrality scores are considered more central and influential.
2. Spatial Encoding: This encoding captures the spatial relationship between nodes in the graph. Techniques like node embedding methods can be used to create this representation, where nearby nodes have similar encoding values compared to distant nodes in the graph.
3. Edge Encoding: This encoding represents the information associated with edges in the graph. It could capture features like edge weight, type, or directionality (if applicable).

Combining these encodings provides a richer representation of each node, incorporating its position, influence, and connection details within the graph structure. This enriched representation is crucial for subsequent stages of the graph-based semi-supervised learning process.

4.3.3 Training Stage

The primary goal of this stage [Fig 8] is to train the encoder. Here's how it works:

1. **Masking Removal:** We begin by removing the masked portion of the input data. This masking is often used during pre-training to focus the model on specific parts of the graph.
2. **Encoder with Learnable Tokens**: The unmasked data is then fed into an encoder that incorporates learnable tokens. These tokens serve as a representation for the entire graph structure, allowing the encoder to capture the overall context of the graph.
3. **Encoding Insertion and Decoding**: The resulting encoding from the encoder is inserted into a decoder. The decoder then attempts to reconstruct the masked portion of the input data.
4. **Prediction Comparison**: Finally, we compare the decoder's predicted masked portion with the actual target masked portion. During the prediction comparison step, we don't directly compare the predicted masked portion and the target masked portion. Instead, we calculate the Mean Squared Error (MSE) loss between them.

In essence, this stage trains the encoder to learn a representation of the graph structure by predicting the masked parts of the input data.

4.3.4 Testing Stage

After training the encoder, we leverage it to perform similarity analysis on unseen vectors representing ego-graphs from the testing set. The testing process [Fig 8]:

1. **Unseen Testing Set Vectors**: We introduce new, unseen vectors representing ego-graphs from the testing set. These vectors haven't been encountered by the encoder during training.
2. **Learnable Encoder Application**: These unseen testing set vectors are fed through the trained encoder. Recall that the encoder incorporates learnable tokens to capture the overall graph structure, allowing it to generalize to unseen data.
3. **Mean Pooling for Dimensionality Reduction**: To reduce the dimensionality of the encoded vectors, we perform mean pooling. This operation averages the values across all dimensions, resulting in a single value that summarizes the encoded representation.
4. **Cosine Similarity Calculation** [Fig 9]: Finally, we calculate the cosine similarity score between pairs of the mean-pooled encoded vectors. The cosine similarity measures the directional similarity between these vectors, effectively reflecting the level of similarity between the corresponding ego-graphs in the testing set.

Essentially, this process evaluates the encoder's ability to generalize to unseen data by analyzing the similarity between ego-graphs in the testing set based on their encoded representations and cosine similarity scores.

4.3.5 Model Evaluation

The graph is a visual representation of the distribution of edge recoveries in the citation network analysis, showing how often certain connections are reconstructed during the perturbation experiments. The distribution helps identify potentially manipulated citations by analyzing the stability of citation reconstructions under network perturbations. In the model analysis, we consider adding color segments based on different reconstruction levels.

4.4 Hyper Parameters (HP)

To train the model and achieve desirable results, we first need to define initial hyperparameters. These will be used for our initial training runs. We have chosen the values for **d, N_encoder_layers, N_decoder_layers, batch size, dropout rate, and number of heads** based on successful results from the

GAME [10]. The effectiveness of the remaining hyperparameters will be determined through the training runs.

4.4.1 Cora base HP

- d = 64 (Embedding dimension).
- N_encoder_layers = 4 (Number of encoder layers).
- N_decoder_layers = 2 (Number of decoder layers).
- L2 = 5 (Numbers of neighbors in ego-graphs).
- N_iter = 50. (Number of epochs in the training process).
- Fr= 30%. (The fraction of the omitted nodes).
- S - the cosine similarity.
- Tr - 0.9/0.95. (The link prediction threshold).
- Peak_lr = 1×10^{-4} $, End\_lr = 1 \times 10^{-9}$. (Learning rates).
- Batch size = 64.
- Dropout rate = 0.5.
- Num_of _heads = 8.
- N_iter = 1000. (The maximal number of iterations in GMAE training).

4.4.2 CiteSeer base HP

In comparison with the previous data, the following parameters are changed.

- N_encoder_layers = 8 (Number of encoder layers).

4.4.3 Loss Function formula for Mean Square Error (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2$$

Where:

- n is the number of observations.
- Y_i is the actual value of the observation.
- $\widehat{Y}_i$ is the predicted value of the observation.

The MSE measures the average of the squares of the errors, that is, the average squared difference between the estimated values and the actual value. It's a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.

4.5 Processing and Evaluation

The adapted approach to evaluating the reliability of citations involves the following steps:

1. Load a graph $G = (V, E)$, including additional information containing node features X_v (with dimension d_v).
2. Repeat N_iter times:
   a. Randomly mask a fraction Fr of nodes in the input graph.
   b. Feed the non-masked nodes into the encoder and obtain their embeddings.
   c. Use a shared learnable mask token to represent the embeddings of the masked nodes and concatenate it with the encoder output.
   d. Calculate the similarity score for all pairs of the masked nodes using the measure S .
   e. Reconstruct the network of the omitted masked nodes by identifying potential links with similarity scores that meet or surpass the threshold (Tr).
3. For each connection, count how many times it is rebuilt throughout the iterations.

5. Evaluation/Verification Plan

We'll demonstrate the unit test cases designed to guarantee the reliability of our model's core functionalities. By leveraging the unittest framework, we subject various components to rigorous testing, solidifying the model's overall robustness.

Test Case ID	Description	Function Tested	Expected Outcome
1.1.1	Test dataset retrieval	get_dataset	Function successfully loads the specified dataset. Returned data matches the expected format.
1.1.2	Test dataset handling (non-existent data)	get_dataset	Function throws an appropriate error when the data is not found.
1.2.1	Test edge creation	Edge class constructor	Creates an edge object with the provided attributes (source, target, weight, etc.).
1.2.2	Test edge attribute access	Edge class getter methods	Getter methods return the correct values set during edge creation.
1.3.1	Test edge dictionary construction (empty graph)	buildDictOfAllEdges	Function creates an empty dictionary for an empty graph.
1.3.2	Test edge dictionary construction (populated graph)	buildDictOfAllEdges	Function builds a dictionary with all edges and their attributes.

1.4.1	Test vector padding (shorter vector)	padVectors	Function pads a shorter vector with the specified padding value to reach the desired length.
1.4.2	Test vector padding (longer vector)	padVectors	Function truncates a longer vector to the desired length.
1.5.1	Test cosine similarity calculation (known vectors)	calc_CosineSimilarity	Function calculates the correct cosine similarity between two known vectors.
1.5.2	Test cosine similarity handling (zero vector)	calc_CosineSimilarity	Function handles zero vectors appropriately (e.g., returns specific value or throws an error).
1.6.1	Test GMAE vector creation (basic case)	createGmaeVectors	Function generates GMAE vectors for a simple graph structure.
1.6.2	Test GMAE vector creation (complex graph)	createGmaeVectors	Function generates GMAE vectors for a more complex graph structure with multiple hops.

6. Summary

This work highlights the negative impact of citation incentives in research and proposes a novel method for their detection. Network analysis using machine learning and neural networks will identify manipulative citation patterns. An asymmetric encoder-decoder architecture utilizing autoencoders is introduced for efficient data representation. The book outlines the data collection, model architecture, and evaluation plan.

7. Reference

[1] Prabha, C.G. Some aspects of citation behavior: A pilot study in business administration. J. Am. Soc. Inf. Sci. 1983, 34, 202–206.

[2] Resnik, D.B.; Gutierrez-Ford, C.; Peddada, S. Perceptions of Ethical Problems with Scientific Journal Peer Review: An Exploratory Study. Sci. Eng. Ethics 2008, 14, 305–310.

[3] Wilhite, A.; Fong, E. Coercive citation in academic publishing. Science 2012, 335, 542–543.

[4] Wren, J.D.; Georgescu, C. Detecting anomalous referencing patterns in PubMed papers suggestive of author-centric reference list manipulation. Scientometrics 2022, 127, 5753–5771.

[5] Dong, M.; Zheng, B.; Quoc Viet Hung, N.; Su, H.; Li, G. Multiple rumor source detection with graph convolutional networks. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, Beijing, China, 3–7 November 2019; pp. 569–578.

[6] Lu, Y.J.; Li, C.T. Graph-aware co-attention networks for explainable fake news detection on social media. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Virtual Conference, 5–10 July 2020; pp. 505–514.

[7] Bian, T.; Xiao, X.; Xu, T.; Zhao, P.; Huang, W.; Rong, Y.; Huang, J. Rumor detection on social media with bi-directional graph convolutional networks. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; Volume 34, pp. 549–556.

[8] Yu, S.; Xia, F.; Sun, Y.; Tang, T.; Yan, X.; Lee, I. Detecting outlier patterns with query-based artificially generated searching conditions. IEEE Trans. Comput. Soc. Syst. 2020, 8, 134–147.

[9] Liu, J.; Xia, F.; Feng, X.; Ren, J.; Liu, H. Deep Graph Learning for Anomalous Citation Detection. IEEE Trans. Neural Netw. Learn. Syst. 2022, 33, 2543–2557.

[10] Zhang, S.; Chen, H.; Yang, H.; Sun, X.; Yu, P.S.; Xu, G. Graph Masked Autoencoders with Transformers. arXiv 2022, arXiv:2202.08391.

[11] Dwivedi, V.P.; Bresson, X. A Generalization of Transformer Networks to Graphs. arXiv 2020, arXiv:2012.09699.

[12] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, “Do transformers really perform bad for graph representation?” arXiv preprint arXiv:2106.05234, 2021.

[13] J. Zhao, C. Li, Q. Wen, Y. Wang, Y. Liu, H. Sun, X. Xie, and Y. Ye, “Gophormer: Ego-graph transformer for node classification,” arXiv preprint arXiv:2110.13094, 2021.