Software Engineering Department Braude

Academic College

Capstone Project Phase B – 61999

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

1. General Description

- **Project Goal**: The project aims to identify potential citation manipulation within academic papers using a novel approach that integrates Graph-Masked Autoencoders (GMAEs) to merge textual information with graph connectivity.
- **Project Implementation**: Our implementation involves training a deep network with partial data, reconstructing masked connections, and analyzing citation stability under network perturbations to pinpoint trustworthy citations. We leverage advanced computational techniques, including graph transformers and self-supervised learning, to create a more intricate model of citation distribution. It focuses on capturing complex patterns and subtle manipulations within citation networks. By randomly removing nodes and observing the network's response, we can identify citations that do not fit the expected pattern, revealing potentially fraudulent activities through multiple independent experiments (reconstruction of masked connections). We generate embeddings for each node (ego graph) and measure the similarity between two citation nodes (link prediction) in the test network. The similarity is measured using cosine similarity, treating the citation graph as undirected. Thus, in the development phase, we have a similarity measure (S) and a threshold value (Tr) to refine link predictions. If the similarity score exceeds Tr, nodes are linked; otherwise, they are not, allowing for a tailored analysis of network links. In the context of the graph masking process during development stages, 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 during the training phase (including the connecting edges). These 30% of nodes with connecting edges will become the test set for link reconstruction. This masking strategy helps the model learn from a subset of the data, thereby enhancing its robustness and predictive accuracy when encountering new, unseen information.
- **User Audience:** The primary audience includes researchers, academic institutions, and journal reviewers who are interested in maintaining the integrity and reliability of academic research by detecting and addressing citation manipulation. Also, data scientists and machine learning engineers can leverage the methodologies and insights presented to enhance their own models for detecting anomalies and manipulations in various datasets, such as financial transactions, social media interactions, network communication and network security logs.

2. Solution Description

The solution is presented in two parts. The first part focuses on the algorithmic approach, while the second delves into the software design and implementation.

2.1 Code Algorithm

2.1.1 Load Graph Data

- Load a graph G = (V, E) along with node features X_v (with dimension d_v).
- Data will be loading to Graph Data Module Struct.

2.1.2 Iterate for N times

a) **Split Graph Nodes(V) into Training and Test Sets**:

- 70% training set and a 30% test set.
- Ensure that training and test edges are disjoint, meaning each edge's endpoints belong to the same set.
- All the tracking structures will be updating.

b) **Training Phase (Sub citation based on 70% nodes**):

- **Masking Removal:** We begin by removing randomly the masked portion of nodes the input graph
- **Encoder with Learnable Tokens:** The unmasked nodes are then feed 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.
- **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.
- **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.

c) Testing\Evaluation Phase (Sub citation based 30% nodes):

- **Unseen Testing Set Vectors:** We will introduce new, unseen vectors representing ego-graphs from the testing set. These vectors haven't been encountered by the encoder during training.
- **Learnable Encoder Application:** These unseen testing set vectors are feed through the trained encoder. Recall that the encoder incorporates learnable tokens to capture the overall graph structure, allowing it to generalize to unseen data**.**

- **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.
- **Cosine Similarity Calculation:** Finally, 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.
- **Reconstruct Small Network:** Reconstruct the network of the omitted masked nodes by identifying potential links with similarity scores that meet or exceed the threshold (Tr). Update the relevant numOfRecoveries parameter for each edge if necessary.

2.1.3 Save Edges Statistics Repository

create and save various result text files based on allUndirectedEdges dictionary.

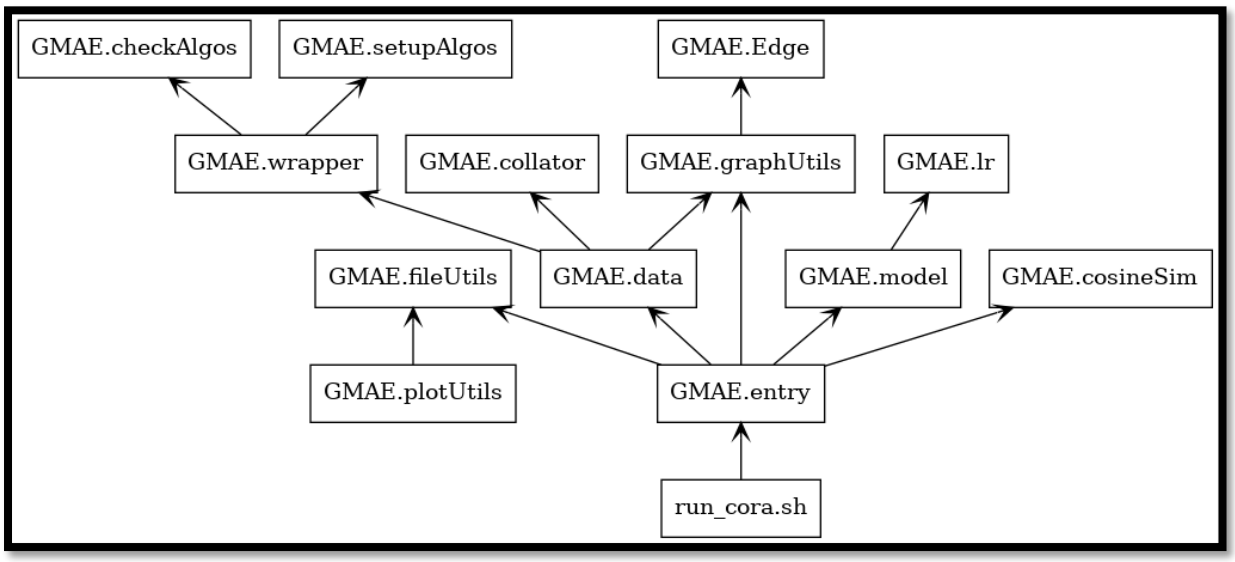
2.1.4 Analysis results with histogram and Statistics

In our analysis, we delve into the recovery metrics of our system, specifically focusing on successful recoveries. To visualize this crucial aspect, we employ a histogram—a powerful tool for understanding data distributions. To enhance interpretability, we apply a color gradient to the bars. Each color corresponds to a stability level.

- Green: High stability (minimal variance in recovery metrics).
- Red: Low stability (significant fluctuations, potential issues).

The analysis done by result files generated in section 2.1.3.

2.2 Structure of the software



2.2.1 Bash Script (run_cora.sh)

This bash script is designed to set up and run a machine learning experiment using specified or default parameters. The script checks if certain environment variables are set. If not, it assigns default values to them. Later, run Python Script (entry.py) with a set of command-line arguments that are derived from the variables defined earlier.

Arguments:

- **Num_workers**: Specifies the number of worker threads to use for data loading. More workers can speed up data loading but will use more CPU resources.
- **Seed:** Sets the random seed for reproducibility. Using the same seed ensures that the experiment can be repeated with the same results.
- **Mask_ratio:** Defines the ratio of data to be masked during training.
- **Dataset_name:** Specifies the name of the dataset to be used for the experiment. In this case, it is set to "Cora".
- **GPUs:** Indicates that the experiment should use 1 or more GPUs. This is important for leveraging GPU acceleration for training.
- **Accelerator:** Specifies the use of Distributed Data Parallel (DDP) for multi-GPU training. Even though only one GPU is specified here, this argument sets up the framework for distributed training.
- **Precision:** Enables mixed precision training with 16-bit floating point numbers. This can speed up training and reduce memory usage without significantly affecting model accuracy.
- **Arch:** The variable contains a set of hyperparameters related to the model architecture, such as learning rates, regularization parameters, batch size, and dimensions of various layers. It is passed as a single string of arguments.
- **N_encoder_layers:** Specifies the number of encoder layers in the model. Encoder layers are used in transformer-based model (GMAE).
- **N_decoder_layers:** Specifies the number of decoder layers in the model. Decoder layers are used in transformer-based model (GMAE).
- **Warmup_updates:** Sets the number of warm-up updates. During the warm-up phase, the learning rate is gradually increased to its peak value to stabilize training.
- **Tot_updates:** Specifies the total number of updates (training steps) to be performed during the experiment.
- **Default_root_dir:** Sets the root directory where the experiment results, including model checkpoints and logs, will be saved.
- **Reload_dataloaders_every_epoch:** Ensures that the data loaders are reloaded at the beginning of each epoch.
- **Test or Validate:** argument would trigger the testing\validation phase only, without training.

2.2.2 Main Machine Learning Pipeline (GMAE.entry)

This script (entry.py) serves as the entry point for the core machine learning or data processing pipeline. The script initializes the following key components based on the arguments provided by the bash script:

- **Model (GMAE.node):** Defines and initializes the machine learning model.
- **Data Module (GMAE.data):** Handles data loading and preprocessing.
- **Trainer (GMAE.node):** Manages the training process.

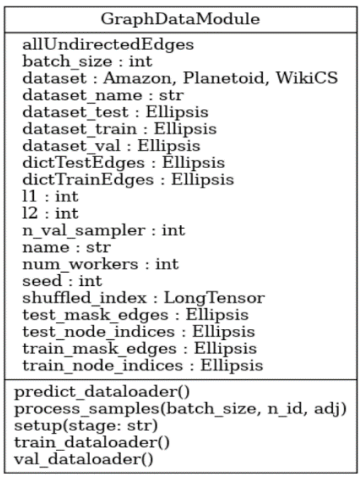
Then script executes 50 independent experiments, each following these steps:

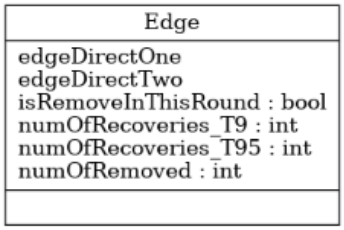
a) **Data Preparation:** The data module is configured for both the training and testing phases.
b) **Model Training:** The model undergoes training using the prepared data.
c) **Link Prediction Evaluation:** The trained model is evaluated by attempting to reconstruct connections (predicting links) within a sub-citation network composed of 30% of the nodes. Link predictions are generated using methods from the cosineSim module to identify potential edges between nodes.
d) **Edge Statistics Update:** Following the evaluation, the script updates the edges statistics repository (allUndirectedEdges) using methods from the cosineSim module.

Upon completion of all experiments, the script has finished its execution, signifying the software's operation is complete.

2.2.3 Graph Data Management and Sampling for Machine Learning (GMAE.data)

Handling graph datasets, preprocessing, and providing data loaders for training, validation, and prediction in a machine learning context. This module contains important class GraphDataModule.





GraphDataModule is designed to handle graph datasets efficiently, providing methods to create data loaders for training, validation, and prediction. It leverages PyTorch Geometric's NeighborSampler for sampling neighbors and processes the samples to create adjacency matrices and other necessary structures for graph-based learning tasks (ego graphs). The GraphDataModule class inherits from LightningDataModule (PyTorch Lightning framework). The ego graph is based on an imported dataset 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.

In an advanced stage of system optimization, auxiliary structures were added to facilitate tracking of reconstructions in all those independent experiments:

- allUndirectedEdges: A dictionary containing all edges in an undirected graph.
- DictTrainEdges\DictTestEdges: Dictionaries containing the edges for training and testing, respectively. These dictionaries change in each round.
- Training node indices \Test node indices : Lists of node indices for the training and testing sets in the Planetoid graph. These lists change in each round.
- Train mask edges \ Test mask edges : Masks for the training and testing edges in the Planetoid graph. These masks change in each round.

2.2.4 Machine Learning Model (GMAE.model)

Define a graph-based masked autoencoder model (GMAE) using PyTorch Lightning by handling the model architecture, training, and testing steps, as well as configuring optimizers and learning rate schedulers by using the GMAE.lr module.

2.2.5 Edge Restoration Using Cosine Similarity (GMAE.cosineSim)

Module focuses on vector operations, specifically padding vectors, calculating cosine similarity, generating embeddings (ego graphs), and evaluating edge restoration.

2.2.6 File Operations and Data Persistence (GMAE.fileUtils)

Saving and loading data using pickle and creating various text files based on given parameters. pickle is a module in Python used for serializing and deserializing Python object structures.

2.2.7 Graph Data Preparation and Edge Management Utilities (GMAE.graphUtils)

Building a dictionary of edges and preparing data for training and testing in graph-based machine learning tasks.

2.2.8 Custom Learning Rate Schedulers in PyTorch (GMAE.lr)

Module defines a custom learning rate scheduler PolynomialDecayLR, which adjusts the learning rate according to a polynomial decay schedule with an optional warmup period.

2.2.9 Batch Collation and Padding for Graph Neural Networks using PyTorch (GMAE.collactor)

Module handles padding of various tensor dimensions and collating them into a batch, specifically for use in graph neural networks using PyTorch.

2.2.10 Graph Positional Encoding with Floyd-Warshall Algorithm in PyTorch(GMAE.Wrapper)

The module is focused on preparing graph data for machine learning tasks. It defines a custom dataset class and a preprocessing function that computes various graph-related features, including shortest paths and node degrees, using the Floyd-Warshall algorithm. This processed data will serve the Graphormer algorithm in terms of positional encoding.

2.3 Research/Development Process

a) **Problem Identification**: We started by identifying the issue of suspicious academic citations, which can undermine the integrity of scholarly work.
b) **Literature Review**: We conducted a thorough review of existing methods for detecting citation manipulation, concentrating on graph-based and deep learning approaches. After this comprehensive analysis, the GMAE model was identified as best aligning with our requirements. Subsequently, we sought an existing implementation to serve as a foundation for building our model to predict connections between nodes. A suitable implementation was found on GitHub [Link].

c) **Model Design**: Developed a novel approach using Graph-Masked Autoencoders (GMAEs) and Graphormer architecture to integrate textual information with graph connectivity.
d) **Data Collection**: Gathered a dataset of academic citations, including both genuine and manipulated examples, to train and test our model. The planetoid dataset brought by Pytorch-Geometric [Link].
e) **Hyperparameter Selection**: Carefully chose hyperparameters such as embedding dimension, number of encoder and decoder layers, learning rate, batch size, and dropout rate. 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 [Link]. The effectiveness of the remaining hyperparameters will be determined through the training runs.
f) **Model Training:** Trained the model using perturbations of a deep embedding model to enhance its ability to detect anomalies in citation patterns.
g) **Evaluation**: Evaluated the model's performance through extensive experiments, comparing it against baseline methods to ensure its effectiveness.
h) **Unit Testing:** Throughout the development process, unit tests were implemented to ensure the correctness and reliability of individual components. This helped in identifying and fixing bugs early, ensuring the robustness of the overall system.
i) **Iteration**: Refined the model based on evaluation results, iterating on the design and training process to improve accuracy.

2.4 Tools Used

a) **Deep Learning Frameworks**: PyTorch and PyTorch Lightning for model development and training, simplifying the training process, and improving code readability.
b) **Torch Geometric**: For working with graph-structured data.
c) **Graph-Masked Autoencoders (GMAEs)**: For integrating textual and graph data.
d) **Graphormer Architecture**: To enhance the model's ability to analyze citation networks.
e) **Cython**: To optimize performance-critical parts of the code. Using when we calculate the short path base on Floyd Warshall Algorithm.
f) **TensorBoard**: For visualizing training metrics and model performance.
g) **Data Processing Tools**: Python libraries like Pandas and NumPy for data manipulation and preprocessing.
h) **Pickle**: For serializing and deserializing Python object structures.
i) **Visualization Tools**: Matplotlib visualizing data and model performance.

2.5 Client Interface

a) **Regular Updates**: Maintained regular communication with the supervising professor through meetings and progress reports.

b) **Feedback Loop:** Followed the professor's instructions to ensure the development of the right model for spotting manipulation of edges in the network and making the necessary reconstructions.
c) **Demonstrations**: Provided periodic demonstrations of the model's capabilities and performance to the professor.
d) **Documentation**: Delivered comprehensive documentation detailing the model's design, usage, and performance metrics. For instance, result txt files and training logs.

2.6 Challenges and Solutions

a) Library Updates
- **Challenge:** Replacing deprecated libraries with new, efficient ones.
- **Solution:** The research tries to incorporate updated libraries and frameworks to ensure compatibility and performance**.**

b) Data structure Selection for Edge Storage and Tracking
- **Challenge:** Choosing a data structure for storing reconstructed edges and tracking.
- **Solution:** A dictionary was used to compile all the network edges. In each experiment iteration, we know whether an edge belongs to the training set or the testing set. The data is updated based on the same database throughout the run.

c) Edge Masking Strategy
- **Challenge:** Edge masking - performing random masking in each iteration.
- **Solution**: A Boolean mask array is used to filter all edges. If an element in the Boolean array is true, the corresponding edge in the set of all edges is kept. If it is false, the edge is discarded.

d) Training and Testing Set Discrepancies
- **Challenge:** Ensuring the training set (70% nodes) and testing set (30% nodes) are representative and balanced.
- **Solution:** The study carefully partitioned the dataset into distinct groups to maintain consistency and reliability in model evaluation. Subsets were created for training and testing to ensure balanced representation.

e) Hyperparameter Optimization
- **Challenge:** Identifying optimal hyperparameters that enhance model training without causing overfitting.
- **Solution:** The study employed a linear decay learning rate scheduler and early stopping mechanisms to fine-tune the model effectively.

f) Checking Node Similarity
- **Challenge:** Determining the most effective method to measure similarity between nodes.
- **Solution**: The study utilized cosine similarity to assess the similarity between node vectors, ensuring an invariant measure to vector lengths.

g) Vector Similarity for Ego-Graphs
- **Challenge:** Deciding whether to flatten ego-graphs into single vectors for similarity checks.

- **Solution**: The approach focused on maintaining the structural integrity of ego-graphs, leveraging their immediate neighborhood for more accurate reconstruction. Specifically, mean pooling was applied to the ego-graphs, followed by similarity checks on the pooled vectors.

2.7 Results and conclusions

GMAE explores a variety of settings for its encoder layers, ranging from 1 to 30 while maintaining a constant of two layers for the decoder. Additionally, the mask ratio, determining the percentage of nodes subjected to masking, is adjusted between 0.7 and 0.8 with a step size of 0.1. The hidden dimensions are set at 64 for each layer, and each transformer layer incorporates eight attention heads. A linear decay learning rate scheduler is applied to enhance the training process, starting with a warm-up stage of 40,000 steps and gradually reducing the learning rate to a final value of $1 \times 10\text{-}9$ after a maximum of 400,000 training steps. The peak learning rate is defined as $1 \times 10\text{-}4$.

Our implementation incorporates the EarlyStopping Hooks callback from the PyTorch Lightning library to address a specific task to detect early signs of process stabilization. This callback is employed to halt training when a monitored metric ceases to improve. It is initialized with four parameters. The first parameter, 'metric', is set to 'train_loss'. The second parameter, 'mode', is set to 'min', signifying that training will conclude when the monitored metric stops decreasing. The third parameter, 'patience', is set to '500', indicating the number of training epochs with no improvement, after which training will be terminated. In this scenario, training will stop if the monitored metric shows no improvement for five hundred training epochs. The fourth parameter, 'check_on_train_epoch_end', is set to 'True'. When true, the callback assesses whether to stop training after each training epoch.

The current experiments adopt a dual focus, aiming to comprehend the scrutinized citation network structure and the method's effectiveness in identifying potentially irrelevant citations. To address this, each test is conducted twice, involving two versions of a given dataset. The initial version is the dataset in its original form, while the second version introduces noise by adding random connections, comprising 20% of the original connections in the source.

2.7.1 Cora dataset experiments results

Experiments are performed with the following set of parameters:

- d = 64 (Embedding dimension).
- N_encoder_layers = 4 (Number of encoder layers).
- N_decoder_layers = 2 (Number of decoder layers).
- L2 = 5/2000 (Numbers of neighbors in ego-graphs).
- N_iter = 50. (Number of experiments/rounds)
- 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 × 10-9. (Learning rate).
- Batch size = 64.
- Dropout rate = 0.5.
- Num_heads = 8.
- N_epochs = 1000. (The maximal number of epochs in GMAE training).

Cosine similarity functions as a metric for assessing the similarity between two vectors within a vector space by determining the angle's cosine. This yields a numerical representation indicating the degree of similarity. The scale of cosine similarity ranges from −1 to 1, where 1 signifies identical vectors, 0 implies no similarity, and −1 denotes entirely dissimilar vectors. The calculation involves dividing the dot product of the vectors by the product of their magnitudes or norms, ensuring that the similarity measure remains invariant to the lengths of the vectors, depending solely on their directions. The utility of cosine similarity extends across various domains, including natural language processing, information retrieval, and data mining. It provides a method for quantifying the similarity between vectors or documents based on their corresponding orientations within a multi-dimensional space.

Two bar charts analyze the distribution of scores obtained during the tests at two link prediction thresholds, Tr = 0.9 and 0.95, and two values of neighbors in ego-graphs, **L2 = 5 and L2 = 2000**. Opting for a second choice eliminates de facto limitations on the number of neighboring selections.

The data range is partitioned into four equal segments outlined by data quartiles, each assigned a distinct color for visual clarity: red, yellow, blue, and green. This color scheme highlights specific regions of interest, particularly the red zone at the bottom, which is expected to contain a higher proportion of low-confidence scores, and the green zone at the top, where high-confidence scores are anticipated.

The horizontal axis of each histogram represents the count of instances where a specific number of papers (shown on the vertical axis) were successfully recovered based on the chosen threshold. This allows us to analyze the distribution of recovered citations at different confidence levels and identify potential patterns within the data. The "red" category at the bottom of the graph is predicted to contain more suspected citations, while the "green" category at the top is expected to include consistently cited papers.

The charts portray the count of successfully recovered instances on the horizontal axis, with the associated unnormalized frequencies displayed on the vertical axis. These frequencies signify the number of notes successfully retrieved corresponding to each recovery count. Notably, the categories marked by colors, such as the lowest "red" category (anticipated to contain the most suspected citations) and the highest "green" category (indicative of the most consistent ones), are of primary interest.

1. **Case of L2 = 5**

The corresponding frequencies of the reconstruction edge numbers are given in Table 1.

Table 1. The recovering edge distributions for the CORA dataset for *L*2 = 5.

Values/*Tr*	0.90	0.95
0	785	1913
1	5993	6293
2	4044	3703
3	2463	2036
4	1370	1060
5	643	458
6	289	182
7	91	50
8	31	19
9	9	4
10	1	1

The quartile values are presented in the subsequent table (Table 2).

Table 2. The quartile values of the recovering edge distribution for the CORA dataset for L2 = 5.

Q/*Tr*	0.90	0.95
*Q*1	0.52	0.32
*Q*2	1.27	0.94
*Q*3	2.39	1.97

Upon comparison of the results presented in Figure 2 and Tables 1 and 2 with those detailed in [1], it becomes clear that the higher recovery threshold (0.95) contributes to the observation and that both distributions demonstrate statistically significant positive skewness. The distribution is skewed towards lower values, characterized by a long right tail with the mean lying to the right of the median.

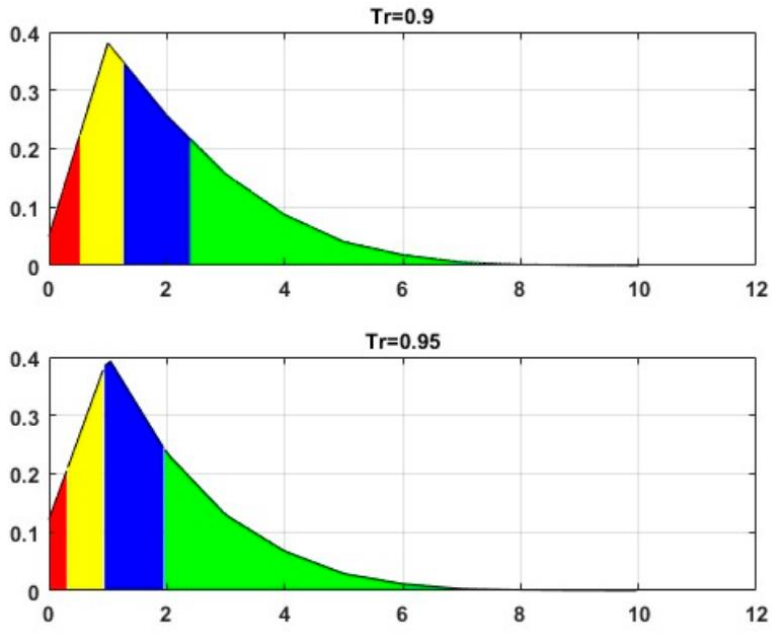


Figure 2. Distributions of edge recovering for the CORA dataset for L2 = 5

2. Case of $L2 = 2000$

As previously said, choosing $L2 = 2000$ discards the limitation on the number of nearest neighbors. The associated frequencies are provided in Table 3 and 4.

Table 3. The recovering edge distributions for the CORA dataset for L2 = 2000.

Values/*Tr*	0.90	0.95
0	170	822
1	6803	5334
2	4551	4060
3	3255	2987
4	2148	1950
5	1184	991
6	476	392
7	191	154
8	49	38
9	8	5
10	2	2

Table 4. The quartile values of the recovering edge distribution for the CORA dataset for L2 = 2000.

Q/*Tr*	0.90	0.95
*Q*1	0.67	0.63
*Q*2	1.54	1.54
*Q*3	2.80	2.78

Upon examination of the bar chart in Figures 2 and 3 and Tables 1 and 2 generated for different L2 values, a distinct similarity among them becomes evident. The distributions are not significantly different. This observation suggests the presence of a consistent underlying structure within the dataset that remains resilient to variations. Most data point the cluster toward the left side, with a right tail extending further. The overall trend remains unaltered, and this observation highlights the existence of a robust and unwavering underlying structure within the dataset that remains impervious to permutations.

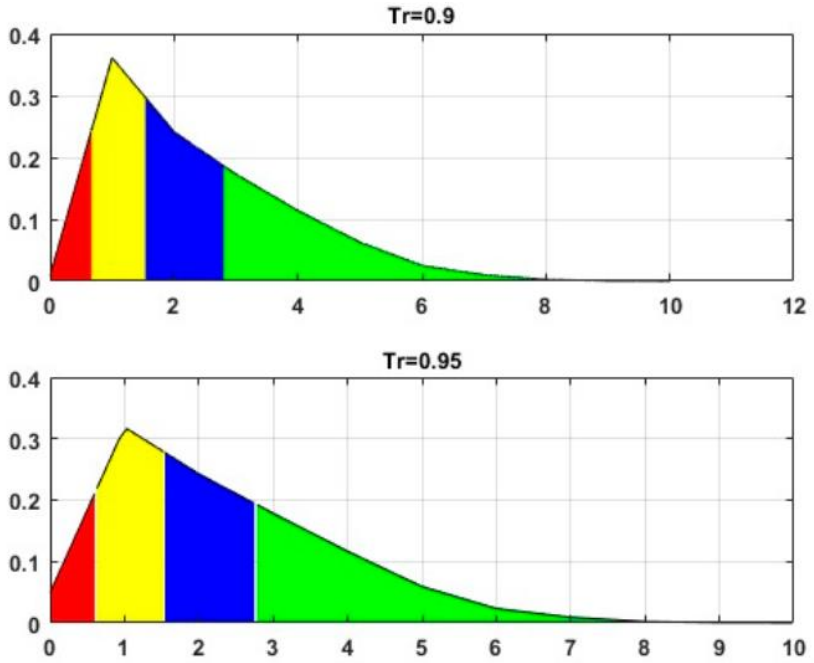


Figure 3. Distributions of edge recovering for the CORA dataset for L2 = 2000.

Given these observations, it becomes interesting to contemplate an experiment designed to uncover the robustness of the approach in identifying spurious citations and to gauge the system's response to adding artificial noise references.

3. Case of *L*2 = 5 for a Dataset Noised CORA Version

As a precautionary measure, we opt to examine a perturbed version of the CORA dataset for sanity checks. This involves introducing artificial randomness by adding 20% more edges to the dataset. The following Figure 4 exhibits histograms of the recovered edges.

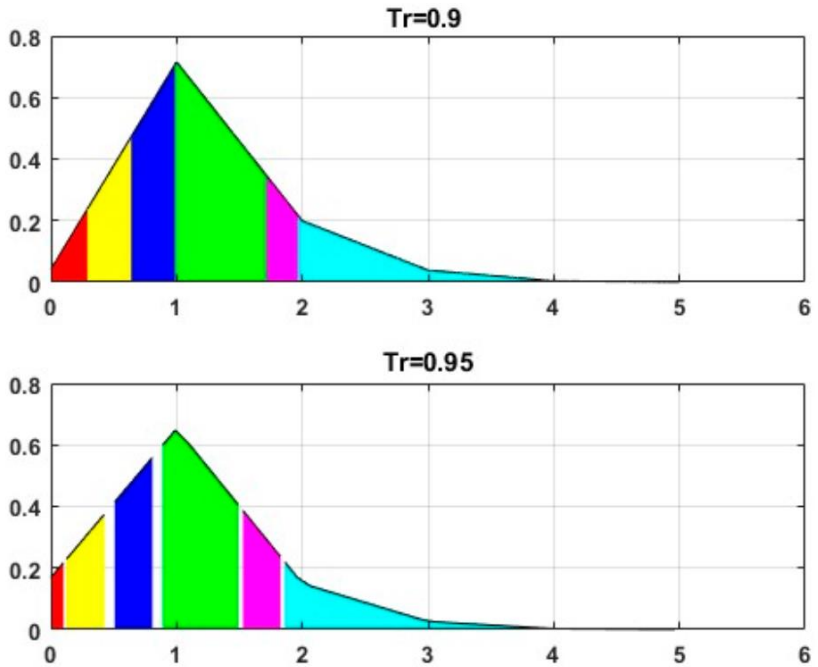


Figure 4. Distributions of edge recovering for the CORA disturbed dataset for *L*2 = 5.

The colors in this visualization correspond to ten successive deciles, dividing the data into ten equal frequency groups. The associated distributions and percentiles are given in the following Tables 5 and 6.

Table 5. The recovering edge distributions for the CORA disturbed dataset $L2 = 5$.

Values/*Tr*	0.9	0.95
0	804	3252
1	13,808	12,590
2	3844	2883
3	749	506
4	76	55
5	9	4

Table 6. The percentile values of the recovering edge distribution for the CORA disturbed dataset for $L2 = 5$.

Percental/*Tr*	0.9	0.95	Colar
*Q*1	0.29	0.12	red
*Q*2	0.64	0.51	yellow
*Q*3	0.99	0.89	blue
*P*90	1.72	1.53	green
*P*95	1.97	1.86	magenta
*P*100	5.00	5.00	cyan

Compared to the preceding experiments, the most recent evaluation reveals a significant decline (approximately 50%) in the central tendency of the variable depicting the quantities of the reconstructed edges. We noted a significant increase in the occurrence of edges reconstructed only once, which closely matched the number of introduced artificial citations. The size of the group reveals that it encompasses not only the added edges but also a significant number of existing true edges whose restoration is compromised by the current network noise. Furthermore, the overall value range of the variable exhibited a predictable decrease, aligning with our expectations. These observations are reinforced by a distinct and notable downward trend, indicating a stronger concentration of values towards the lower spectrum of the variable's range. Such a substantial shift strongly implies that the introduction of noise likely impeded the network's capacity to accurately encode and reconstruct edges, leading to a considerable reduction in the quantities of reconstructed edges. Consequently, the proposed methodology effectively captures the distortion within the underlying network structure.

2.7.2 CiteSeer dataset experiments results

In comparison with the previous data, the following parameters are changed in the Experiments:

- *N_encoder_layers* = 8 (Number of encoder layers).
- $L2 = 5$ (Numbers of neighbors in ego graphs).

The number of encoder layers is determined based on the experiments conducted in [2][4]. The outcomes of the experiments are presented in Figure 6 and Tables 7 and 8.

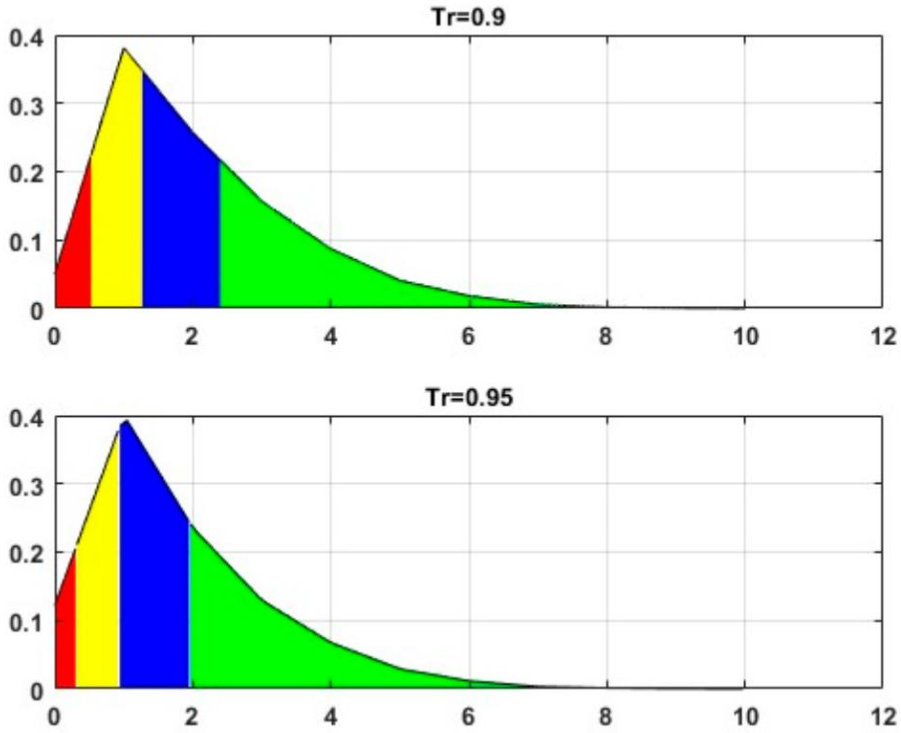


Figure 6. Distributions of edge recovering for the CiteSeer dataset for $L2 = 5$.

Table 7. The recovering edge distributions for the CiteSeer dataset for $L2 = 5$.

Values/*Tr*	0.90	0.95
0	19	133
1	8995	8935
2	3054	3019
3	549	533
4	90	87
5	15	15

Table 8. The quartile values of the recovering edge distribution for the CiteSeer dataset for $L2 = 5$.

Q/*Tr*	0.9	0.95
*Q*1	0.35	0.34
*Q*2	0.71	0.70
*Q*3	1.17	1.16

The results obtained from the CiteSeer dataset exhibit an inherent resemblance in its internal structure to one of the Cora datasets. Notably, the range of reconstructed edges is a bit broader, which could be connected to the CiteSeer dataset's denser configuration.

2.7.3 Conclusion

This paper discusses a new attitude to identifying illegitimate citations. The method is built upon the Generalization of Transformer Networks for Graphs and incorporates a masking mechanism to disrupt patterns in altered citation embeddings. Testing validates the approach's efficacy, with masking embeddings provided by the transformer method shining as a dependable tool for uncovering citation manipulation.

While detecting anomalies under regular citation patterns, the model has limitations with multi-disciplinary works and "sleeping beauties"—articles that went unnoticed initially but later experienced a surge in recognition. This phenomenon can arise from breakthrough discoveries or, simply, later appreciation, challenging this method's detection capabilities.

Around 75% of the total edges (citations) prove susceptible to the distortion procedure, failing to withstand it. The instability of these edges, with their heightened sensitivity to data modifications, sets them apart from the system core's reliable internal structure. Consequently, the associated citations may be deemed dubious and potentially manipulated. This underscores a nuanced dimension of the dataset's integrity and emphasizes the potential impact of specific edges on its structural stability.

The analysis, even though it explores datasets with distinct internal structures, yields sufficiently similar results. This unexpected finding points towards a possible universal inclination within the mutual citation system, suggesting the presence of shared characteristics that transcend the specificities of individual datasets. An intriguing observation is the
consistent revelation of a stable core within the citation network across both datasets. While the precise mechanism behind this core's formation remains unclear, it could be linked to the gradual accumulation of reliable links over time. Interestingly, even in datasets like these, which receive regular updates to incorporate newly published articles, as indicated
in [1], most edges showcase instability and a lack of relevance. This consistency across different datasets points toward a generalizable property regarding edge reliability, implying that a considerable portion of connections within citation datasets might be less trustworthy or more susceptible to manipulation.

Expanding on this observation, it is imperative to recognize that the positive skewness in the distribution of reconstruction scores signifies a prevailing tendency for data points to lean toward lower scores. With its pronounced right-skewed tail, this unimodal distribution indicates that a substantial portion of the data is concentrated on the left side. At the same
time, the mean is disproportionately influenced towards higher scores. Consequently, the prevailing pattern suggests that many references exhibit relatively modest reconstruction scores, prompting consideration of their potential suspicion or manipulation.

The consumption of Graph-Masked Autoencoders (GMAEs) results in a more refined model of citation distribution, capturing the intrinsic connections between papers using additional textual information. This approach distinguishes itself from [1] by uncovering characteristic right-skewed unimodal empirical distributions, indicating a closer alignment
with actual citation behavior.

One of the experiments delves into the model's readiness when confronted with an artificially disturbed citation graph, aiming to gauge the approach's trustworthiness. Essentially serving as a sanity check, this assessment validates the model's adeptness in flagging artificially introduced links as highly suspect. The attained findings underscore the model's proficiency in detecting anomalies, affirming its effectiveness and reliability.

The suggested method leverages a stable knowledge core within a graph to track the latest research developments in a specific field. Regularly updating the dataset with new articles and integrating them into the existing knowledge networks provides a dynamic overview of research trends and advancements. Link evaluation for a specific article can be achieved by applying the aforementioned procedure, followed by analyzing the links' position within the general recovery histogram.

2.8 Lessons learned

2.8.1 Evaluation of Current Practices

- **Effectiveness:** The methodologies employed were largely effective in meeting the project goals based on Research/Development Process , Challenges and Solutions and Client Interface. However, there were instances where the expected outcomes were not fully realized. When following a research direction and implementing it, one anticipates a certain expected result. However, this result may not always be achieved, leading to the discovery of a new direction that may ultimately be more accurate. Therefore, our internal iteration process and interface with the professor are of high importance for efficiency.
- **Efficiency:** While the project was completed within the allocated budget, the development and research process could have been expedited if we had access to computing resources provided by the college.

2.8.2 Identification of Strengths

- **Approach:** The collaborative approach adopted by the team significantly enhanced the quality of the final deliverable.
- **NLP knowledge:** The team's extensive knowledge and skills in transformer models and natural language processing (NLP) tasks played a crucial role in the project's success.

- **Rapid Learning of New Technologies:** The team's ability to quickly learn and implement new technologies, such as PyTorch Lightning, allowed us to focus more on the core model training and less on the surrounding configurations.
- **Agile methodologies:** The use of agile methodologies allowed for greater flexibility and adaptability throughout the project lifecycle. This included maintaining Backlog, ToDo, and Done lists of works.
- **Implementation of a Testing Plan:** Developing and executing a comprehensive testing plan ensured the correct implementation of the project.

2.8.3 Areas for Improvement

- **Initial Misunderstanding of Supervisor's Instructions:** In the early stages, there was a lack of understanding of the supervisor's instructions, leading to partial or inaccurate implementations. This required us to undertake more sprints in terms of the agile method to correct and refine our work.

- **Validation Layer Addition:** Adding a validation layer during the training process, in addition to the existing training and testing phases, could have accelerated the identification of the correct hyperparameters for the task.

2.9 Project Benchmarks

- Develop a machine learning model capable of identifying suspicious citations within academic articles.
- Ensure the model's accuracy and reliability in detecting citation anomalies.
- Publish the model and its findings in a reputable academic journal.

2.9.1 Achievement

- **Model Development and Implementation**: The machine learning model was successfully developed and implemented using GMAE.
- **Model Accuracy and Reliability:** Through rigorous testing and evaluation, the model demonstrated a high degree of accuracy in identifying suspicious citations. Its performance was validated using reconstruction links in the sub-citation network.
- **Publication:** As a testament to the project's success, the model and its findings were published in the academic journal [3]. This publication signifies that the model has been recognized by the academic community as a valuable contribution to the field and validates its ability to achieve the project's objectives.

3. User's Guide Operating Instructions

Below is an overview of the system operation and installation in terms of Python and AI frameworks, designed to guide users through the process.

3.1 System Requirements and Running Instructions

3.1.1 Run the project locally

To run the project locally, ensure your system meets the following requirements:

- Operating System: Linux
- Python Version: Python 3
- Disk Space: At least 8 GB of free space
- RAM: Minimum 16 GB
- GPU RAM: 15GB
- IDE Editor: Visual code, Spider, PyCharm.
- Git – Version Control System.

Running the Project Locally on a Linux Machine by README file [Link]. The instructions include a bash script that installs the project dependencies (frameworks).

3.1.2 Run project in Colab Notebook

You can run the project on Google Colab by clicking the link below:

https://colab.research.google.com/drive/1vKgnY6cUeADSVMuDVkZcsuLRD47gJLwA?usp=sharing

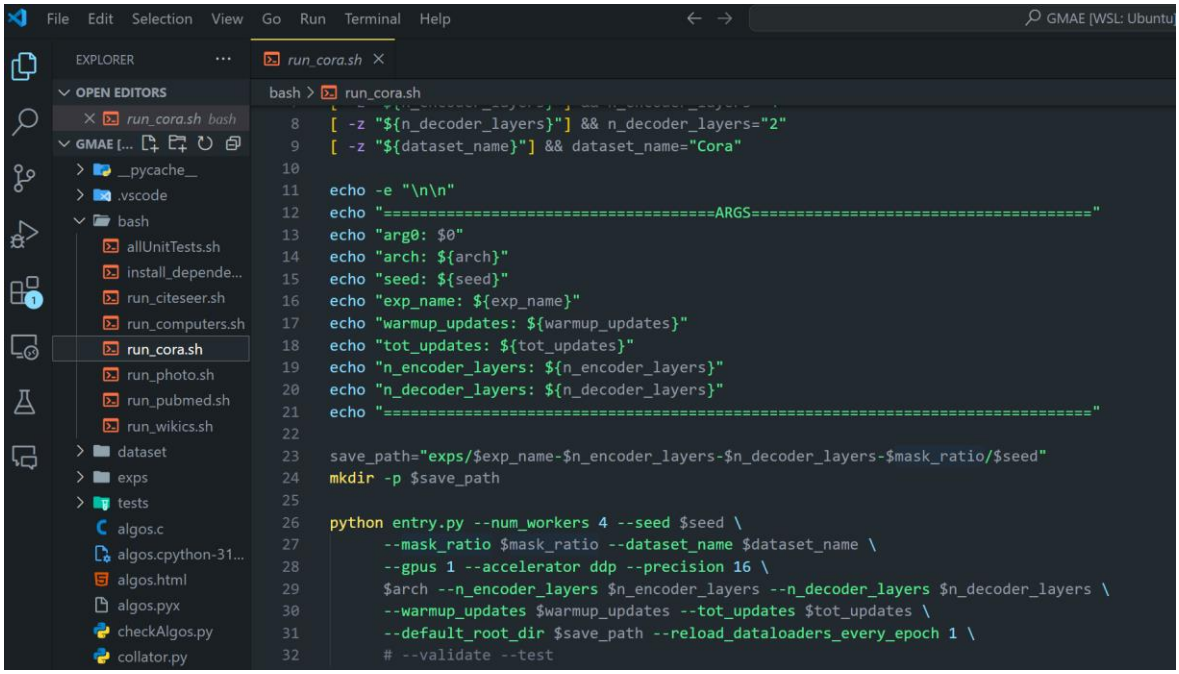
In the notebook all the project dependencies install made in the first scope. All results files will be store in folder and download automatically in the end of the training.

3.1.3 Model Bash Script

This bash script is designed to set up and run a machine learning experiment using specified or default parameters. The script checks if certain environment variables are set. if not, it assigns default values to them. Model bash files will be stored in the bash folder.

- All the explanation of the parameters (Arguments) in section 2.2.1.
- Before running the run_cora.sh script, edit the parameters as needed.
- Save your changes by pressing `Ctrl + S` before running the script.

Cora Bash script file:



4. Maintenance Guide

The project software structure is demonstrated in detail in section 2.2 . Running instructions located also at README file [Link]. By following this maintenance guide and the README file, you can ensure that the project remains robust, maintainable, and scalable.

4.1 Key Basic Components

- **Entry.py:** The main entry point of the application. It initializes and runs the model training and evaluation pipeline.
- **Model.py:** Defines the model architecture and training configurations.
- **Bash folder:** Contains shell scripts for running experiments and installing dependencies.

4.2 Running the project

- **Install Dependencies:** Run the bash/install_dependencies.sh script to install all necessary dependencies.
- **Run Experiments:** Use the provided shell scripts to run specific experiments. For example, to run the Cora experiment: run_cora.sh.

4.3 Testing

Unit tests are in the test directory. run all unit tests by execute bash/allUnitTests.sh.

4.4 Maintenance Tips

- **Keep Dependencies Updated:** Regularly update the dependencies listed in bash/install_dependencies.sh to ensure compatibility and security.

- **Code Documentation:** Ensure all functions and classes are well-documented. Use docstrings to describe the purpose and usage of each function and class.
- **Consistent Coding Style:** Follow a consistent coding style throughout the project.
- **Regular Testing:** Run unit tests regularly to catch any regressions or bugs early.

4.5 Dependency Issues

If you encounter issues with dependencies, ensure that all required packages are installed and compatible with your Python version.

4.6 Runtime Errors

Check the logs for detailed error messages. Use debugging tools like pdb in colab or launch.json in Visual code to trace and fix issues. To address runtime errors and performance issues, you can configure the `launch.json` file in .vscode directory to run entry.py with the parameters .

4.7 Performance Issues

Profile your code to identify bottlenecks. Optimize critical sections of the code to improve performance. Performance can be improved by adjusting parameters like num_workers, gpus, and accelerator.

5. Reference

1. *Detecting Pseudo-Manipulated Citations in Scientific Literature through.* **Avros, R., et al.** s.l. : Mathematics, 2023.

2. *A Generalization of Transformer Networks to Graphs.* **Dwivedi, V.P. and Bresson, X.** s.l. : arXiv, 2020. 2012.09699.

3. *Spotting Suspicious Academic Citations Using Self-Learning Graph Transformers.* **Avros, R., et al.** s.l. : Mathematics, 2024.

4. *Graph Masked Autoencoders with Transformers.* **Zhang, S., et al.** s.l. : arXiv, 2022. 2202.08391.