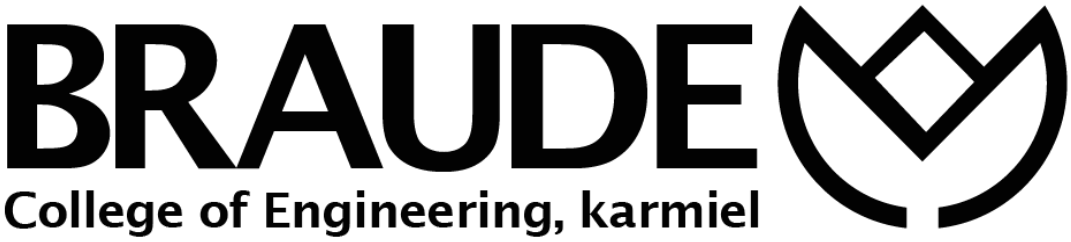


Software Engineering Department

Braude Academic College

Capstone Project Phase B

Detecting and Evaluating Anomalously Cited Papers Over Time using Anomaly Detection in Dynamic Graphs via Transformers

24-2-R-7

Israel Shushan
Israel.Shushan@e.braude.ac.il

Noah Soskha
Noah.Soskha@e.braude.ac.il

Supervised by

Prof. Zeev Volkovich

Dr. Renata Avros

Book Repository

Detection-of-Anomalous-Cited-Papers

TABLE OF CONTENTS

1. PROJECT MOTIVATION 4

2. PROBLEM STATEMENT 4

3. RESEARCH GOALS 5

3.1. Develop Anomaly Detection Framework: 5
3.2. Implement Temporal Citation Analysis: 5
3.3. Evaluate Effectiveness and Accuracy: 5
3.4. Contribute to Academic Integrity: 5

4. SOLUTION DESCRIPTION 5

4.1. Code Algorithm 5
Load Citation Graph Data 5
Contextual Node Encoding: 6
Node Encoding: 6
4.2. Negative Sampling and Second Edge Embedding 6
Pseudo-Anomalous Edge Generation: 6
Repeat Contextual Encoding for Pseudo-Anomalous Edges: 6
4.3. Anomaly Detection and Training 7
Discriminative Anomaly Detector: 7
Loss Function: 7
Optimization: 7
4.4. Testing Phase 7
Contextual Node Encoding for Test Edges: 7
Dynamic Graph Transformer: 7
Anomaly Scoring: 7
Output Anomaly Scores: 7
4.5. Save Results and Analyze 7
Save Anomaly Scores: 7
Analysis and Visualization: 7
4.6. Code architecture: 8
Main Machine Learning Pipeline: 8
0_prepare_data.py 8
AnomalyGeneration.py 9
1_train.py 9
settings.py 9
DynamicDatasetLoader.py 9
DynADModel.py 10
BaseModel.py 10
Component.py 10
utils.py 11
anomaly_tracking_utils.py 11

5. RESEARCH/DEVELOPMENT PROCESS 12

5.1. Literature Review 12
Shallow Learning-Based Methods 12
Deep Learning-Based Methods 12
5.2. Problem Identification 13
5.3. Model Design 13
5.4. Data Collection 14

5.5. Hyperparameter Tuning 14
5.6. Model Training 14
5.7. Evaluation 14
5.8. Iteration 15
5.9. Tools used 15
5.10. Challenges and Solutions 15
Challenges Faced: 15
5.11. Results and Conclusions 16
High-Energy Physics Theory Citation Network [17] dataset experiments results: 16
Experiment 1: 16
Results: 16
Experiment 2: 17
Experiment 3: 17
6. CONCLUSIONS 20
7. USERS GUIDE 21
7.1. Requirements 21
7.2. Usage 21
Prepare Data 21
Train Model 21
7.3. Setup In *Google Colab*: 21
Clone Repository: 21
Activate environment and install dependencies: 21
Prepare Data: 21
Train Data: 21
8. MAINTENANCE GUIDE 22
9. REFERENCES 22

1. Project Motivation

Academic citation networks play a critical role in understanding the flow of knowledge and influence between research papers. However, detecting anomalies within these networks, such as citation manipulation or irregular patterns, has become increasingly important due to its impact on research quality and academic integrity. Traditional methods of anomaly detection often fail to address the temporal dynamics and complexity of evolving citation networks.

This project seeks to fill this gap by leveraging advanced machine learning techniques, specifically the **Transformer-based Anomaly Detection in Dynamic Graphs (TADDY)** [1] framework, to detect and analyze anomalous citation behaviors over time. By integrating transformer-based models, we aim to provide a strong solution for identifying citation anomalies, uncovering suspicious citation patterns, and ensuring the reliability of academic metrics. Moreover, we aim to detect the papers that cite other papers in an anomalous way the most.

2. Problem Statement

In academic citation networks, irregular citation behaviors, such as sudden spikes or unexplained trends, can distort the integrity of academic metrics and impact the evaluation of research contributions.

Traditional anomaly detection techniques often lack the capability to account for both the temporal evolution and structural complexity inherent in dynamic citation networks. Moreover, they typically rely on static graph representations, which fail to capture how citation patterns change over time. This limitation hinders the ability to identify anomalous citation behaviors accurately and comprehensively.

The primary problem addressed in this project is: how to efficiently detect citation anomalies in dynamic academic networks, considering temporal changes, spatial relationships and scalability?

The goal of this project is to bridge this gap by developing a solution that uses TADDY [1], a transformer-based anomaly detection framework, to dynamically analyze and interpret citation patterns. With this approach, we aim to improve the detection of anomalous citations and provide information about their evolution over time. Through this approach, we seek to enhance the detection of anomalous citations and provide insights into their evolution over time, identifying the papers where these anomalous citations appear.

3. Research Goals

This project aims to address the challenge of detecting and analyzing anomalous citation behaviors in dynamic academic networks by leveraging advanced machine learning techniques. The following are the primary research goals:

3.1. Develop Anomaly Detection Framework:

Utilize the TADDY framework to identify citation anomalies by analyzing both spatial relationships and temporal dynamics within academic citation networks.
Enhance TADDY to output meaningful results, tracking utilities and anomaly scores.

3.2. Implement Temporal Citation Analysis:

Design and implement an analysis method that enables the visualization and quantification of anomalous citation trends over time.
Track the evolution of citation anomalies across multiple time windows to provide a comprehensive understanding of their progression.

3.3. Evaluate Effectiveness and Accuracy:

Assess the model's performance using key metrics such as ROC-AUC, standard deviation, and histogram analysis. Additionally, introduce a temporal anomaly evolution metric, specifically focusing on the top ten edges with the highest variation in anomaly scores over time. This metric provides a deeper understanding of how the most dynamically anomalous edges evolve, offering critical insights into their behavior within the network. Validate the framework on simulated citation networks and real-world datasets.

3.4. Contribute to Academic Integrity:

Enhance the reliability of academic metrics by providing tools for identifying citation anomalies that may affect research evaluation processes.
Contribute to the broader field of dynamic graph anomaly detection with insights and methodologies applicable beyond citation networks.

4. Solution Description

4.1. Code Algorithm

Load Citation Graph Data

- **Input Data:**

  A dynamic citation graph $G = (V, E)$ where:

  $\boldsymbol{V}$ - Nodes representing academic papers.

  $\boldsymbol{E}$ - Edges representing citation relationships between papers over time.

  Each node includes temporal and structural attributes to capture citation behavior and network relationships.

- **Graph Management:**
    - The graph data is split into temporal snapshots, representing citation activities across time.
    - Data is preprocessed and organized in the Graph Data Management Module for node and edge-level operations.

Contextual Node Encoding:

- **Diffusion Matrix Calculation:**

  For each node v at each timestamp t, a diffusion matrix is computed to represent the influence propagation in the graph.

- **Substructure Sampling:**

  For each target edge, the top k nodes with the highest diffusion values are selected to form the local substructure of the current edge.

Node Encoding:

Diffusion-Based Spatial Encoding: Computes the global structural role of nodes using Personalized PageRank [2] values derived from the diffusion matrix.

Distance-Based Spatial Encoding: Encodes local proximity relationships, focusing on the connections around the target edge.

Relative Temporal Encoding:

- Captures temporal dynamics by calculating the time difference between the target edge's occurrence and the current timestamp.
- The resulting encodings are summed together to create a single encoding vector for each node.

Edge Embedding: The contextual node encodings are passed through the Dynamic Graph Transformer, which processes spatial-temporal relationships and outputs edge embeddings.

4.2. Negative Sampling and Second Edge Embedding

Pseudo-Anomalous Edge Generation:

Since no labeled anomalies are present, negative sampling is performed by randomly sampling node pairs that do not exist as edges in the graph. These serve as pseudo-anomalous edges.

Repeat Contextual Encoding for Pseudo-Anomalous Edges:

The first stage (contextual node encoding and substructure sampling) is repeated for each pseudo-anomalous edge to produce its edge embedding.

4.3. Anomaly Detection and Training

Discriminative Anomaly Detector:

A fully connected neural network (Discriminative Anomaly Detector) is used to classify edges. It takes as input the edge embeddings (both normal and pseudo-anomalous) and computes an anomaly score for each edge.

Loss Function:

The model minimizes a loss function designed to distinguish between normal and pseudo-anomalous edges. This allows the model to learn effectively over multiple iterations.

Optimization:

The model is trained using backpropagation and gradient descent, iteratively updating the parameters of the transformer and the anomaly detector.

4.4. Testing Phase

Contextual Node Encoding for Test Edges:

For each edge in the tested timestamp, the substructure sampling and spatial-temporal encoding steps are performed, similar to the training stage.

Dynamic Graph Transformer:

Encoded information is passed through the Dynamic Graph Transformer to generate edge embeddings for the tested edges.

Anomaly Scoring:

The generated edge embeddings are passed through the trained Discriminative Anomaly Detector, which computes an anomaly score for each edge. The anomaly detector does not update its weights during this stage.

Output Anomaly Scores:

Anomaly scores range between **0 and 1**, where scores closer to one indicate a higher likelihood of the edge being anomalous.

4.5. Save Results and Analyze

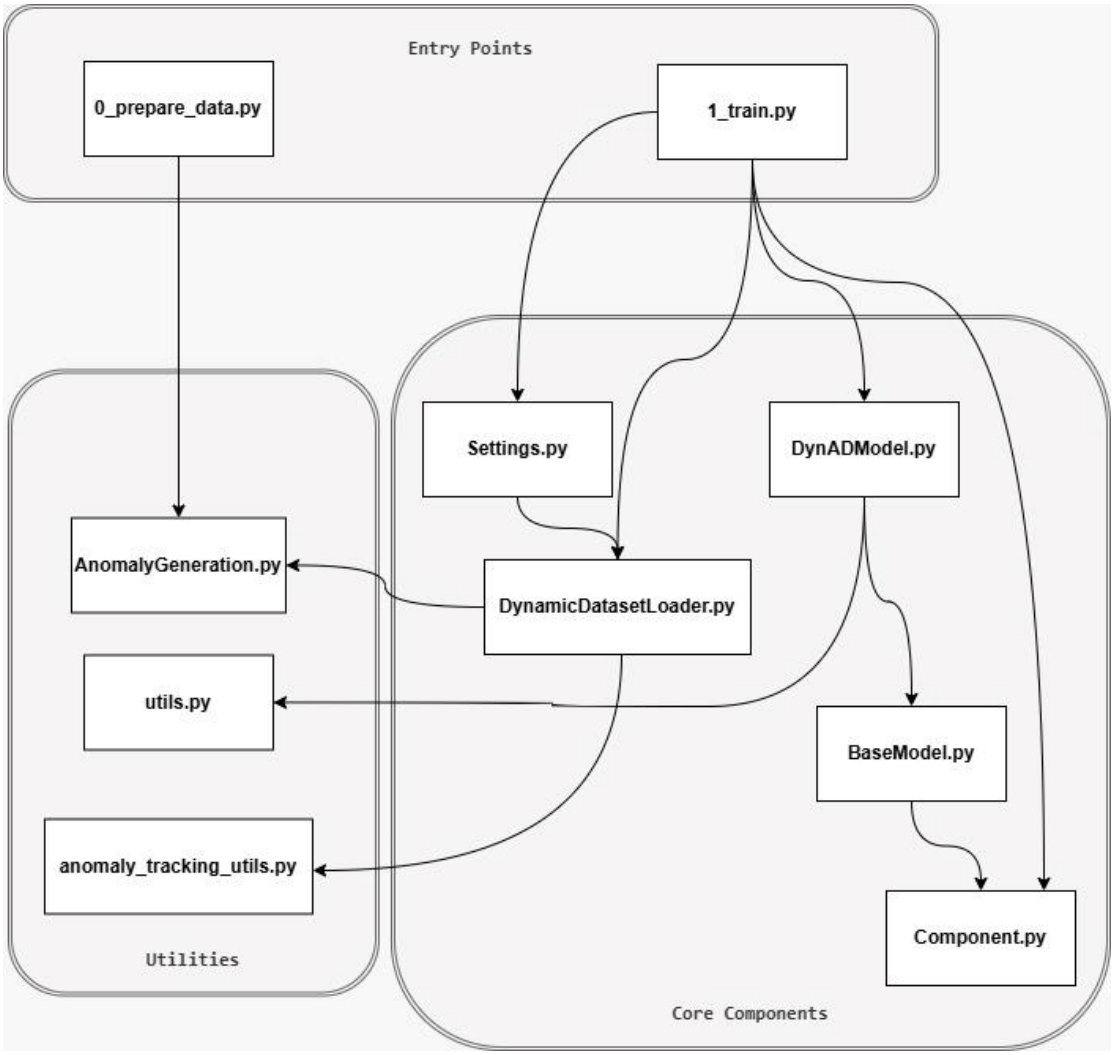
Save Anomaly Scores:

- Save the anomaly scores for all edges in a repository for further analysis.
- Include details such as edge IDs, timestamps, and anomaly scores.

Analysis and Visualization:

- Create histograms and other statistical visualizations to analyze anomaly score distributions.
- Highlight and determine the top ten papers that got the highest anomaly score.

4.6. Code architecture:



Main Machine Learning Pipeline:

0_prepare_data.py

The preprocessing framework comprises two primary components: the PreprocessDataset module and the generateDataset function. The PreprocessDataset module implements comprehensive data normalization protocols, encompassing node remapping strategies and edge list standardization, thereby transforming heterogeneous temporal edge sequences into a unified format optimized for model ingestion. The generateDataset function extends this pipeline through a bifurcated methodology: it first executes controlled anomaly injection utilizing stochastic perturbation techniques with configurable anomaly percentages (parameterized via anomaly_per), followed by temporal partitioning of the edge streams into discrete snapshots (determined by snap_size_dict). This framework preserves both topological fidelity and temporal causality through systematic edge

processing procedures, including self-loop elimination and duplicate edge handling, while maintaining explicit mappings between original and processed node identifiers for post-analysis traceability.

AnomalyGeneration.py

establishes an anomaly synthesis in dynamic graph structures, facilitating the construction of controlled experimental datasets essential for model validation protocols. The core component, anomaly_generation, implements a data partitioning methodology parameterized by train_per and anomaly_per, enabling to split the input graph into training and evaluation subsets while introducing synthetically generated anomalous edges.
The implementation includes two fundamental preprocessing functions for maintaining data integrity: (1) edgeList2Adj, which executes topological transformation from sparse edge list representations to dense adjacency matrices, facilitating efficient spectral computations and clustering operations; and (2) processEdges, which performs systematic edge sanitization through removal of self-referential cycles and redundant edge instances, ensuring topological consistency in the resultant graph structure.

1_train.py

train.py constitutes the primary experimental orchestration framework for the (TADDY) architecture, implementing a comprehensive end-to-end training pipeline. The framework encompasses three critical components: (1) temporal graph data ingestion utilizing the DynamicDatasetLoader class, which facilitates efficient processing of dynamic network structures through systematic snapshot generation and adjacency matrix normalization; (2) extensive hyperparameter configuration protocols for architectural optimization, including embedding dimensionality (embedding_dim), multi-head transformer specifications (num_attention_heads, num_hidden_layers), and learning rate scheduling (lr); and (3) DynADModel training integration with specialized anomaly detection objectives, incorporating performance metrics such as ROC-AUC for quantitative evaluation.

settings.py

implements a hierarchical configuration management framework that orchestrates the experimental pipeline's core operational components. The framework establishes a systematic methodology for dataset initialization and method execution through two primary functions: (1) dataset loading protocols that facilitate the ingestion and preprocessing of dynamic graph structures, and (2) method invocation mechanisms that coordinate the execution of transformer-based anomaly detection procedures. This modular architecture ensures clear separation between data handling and model execution phases, enabling systematic evaluation of experimental configurations across different datasets and methodological variations.

DynamicDatasetLoader.py

implements a framework for temporal graph data processing, using sophisticated data ingestion and transformation methodologies essential for dynamic graph analysis. The framework encompasses three primary components: (1) advanced adjacency matrix preprocessing, including row-wise and symmetric normalization protocols coupled with sparse tensor conversion for computational efficiency; (2) eigenvalue-based graph representation generation, parameterized by teleportation probability c, facilitating the

capture of higher-order structural patterns through systematic eigen-decomposition of normalized adjacency matrices; and (3) temporal snapshot management, enabling controlled partitioning of graph sequences into training and evaluation sets with configurable ratios.
The implementation incorporates specialized data transformation protocols, including sparse matrix operations for memory-efficient processing and systematic eigen-adjacency computation for enhanced spatial-temporal feature extraction. The framework maintains explicit control over snapshot generation through configurable window sizes and neighbor sampling parameters, ensuring robust temporal consistency in the processed graph sequences.

DynADModel.py

DynADModel.py implements a comprehensive transformer-based architecture for anomaly detection in dynamic graphs, incorporating sophisticated feature extraction and learning mechanisms. The framework encompasses four principal components:

Transformer-Based Feature Learning: The architecture uses a hierarchical transformer encoder for generating contextualized node and edge representations.

Adversarial Learning through Negative Sampling: The framework implements a systematic negative sampling strategy that generates controlled pseudo-anomalous edges by stochastic perturbation of existing graph structures.

Optimization Framework: The training methodology employs binary cross-entropy loss optimization with Adam optimizer, learning rate scheduling (lr) and weight decay regularization (weight_decay) for model convergence and generalization capabilities.

Performance Quantification: The model integrates evaluation protocols, including ROC-AUC computation and temporal anomaly tracking through the AnomalyTracker module, enabling detailed analysis of detection performance across multiple temporal snapshots.

BaseModel.py

implements a transformer-based architecture that orchestrates three primary computational stages: (1) edge-centric feature encoding through specialized EdgeEncoding modules that capture positional, hop-distance, and temporal relationships; (2) multi-layer transformer processing utilizing configurable attention mechanisms for deep feature extraction; and (3) strategic feature aggregation through BertPooler for downstream anomaly detection tasks. The architecture maintains explicit control over embedding propagation and attention mask generation, ensuring robust feature learning across dynamic graph structures.

Component.py

establishes the architectural primitives essential for transformer-based graph processing: (1) MyConfig, which implements a systematic configuration protocol for transformer hyperparameters, including attention head count, hidden dimensionality, and dropout rates; (2) EdgeEncoding, which realizes multi-dimensional embedding generation through parallel processing of initial positions (max_inti_pos_index), hop distances (max_hop_dis_index), and temporal features; and (3) TransformerEncoder, which

implements sequential attention-based processing through configurable layer stacking (num_hidden_layers) with specialized normalization and dropout mechanisms.

utils.py

The utility framework implements a suite of graph processing and embedding generation methodologies essential for dynamic graph analysis. The framework encompasses three primary computational components:

Weisfeiler-Lehman Graph Processing [3]: implementation includes compute_zero_WL for baseline feature generation, providing a controlled initialization state for subsequent embedding processes.

Temporal-Spatial Distance Computation: The compute_batch_hop function implements methodology for calculating multi-dimensional distance metrics across temporal graph snapshots. This component uses configurable window sizes and k-neighbor sampling, generating structured representations of node relationships through parallel processing of spatial proximity (hop distances) and temporal evolution patterns.

Embedding Generation Pipeline: The dicts_to_embeddings function realizes an embedding transformation protocol, converting multi-dimensional feature dictionaries into structured tensor representations. This process uses five parallel embedding streams: raw features, WL-based roles, hop distances, intimacy rankings, and temporal positions.

anomaly_tracking_utils.py

The AnomalyTracker implements a comprehensive framework for temporal anomaly analysis and performance quantification in dynamic graph structures, incorporating sophisticated tracking mechanisms and multi-dimensional visualization capabilities. The framework encompasses three primary analytical components:

Temporal Pattern Analysis: The system implements a methodology for tracking edge evolution through specialized data structures (edge_history, edge_scores, timestamp_scores), facilitating granular analysis of behavioral patterns across temporal snapshots. This includes variance analysis through configurable thresholds (persistence_threshold, high_variance_threshold) for identifying consistently anomalous patterns and significant behavioral transitions in the network topology.

Statistical Analysis and Visualization: The framework incorporates statistical analysis protocols through: (1) score variation analysis using standard deviation computations and kernel density estimation, (2) persistent anomaly identification based on configurable thresholds, and (3) temporal transition tracking with comprehensive state change logging. These analyses are complemented by automated visualization generation, including score variation plots, distribution histograms, and temporal evolution charts.

Edge Mapping and Data Management: The system implements edge mapping capabilities, maintaining bidirectional mappings between modified and original edge identifiers. This facilitates accurate tracking and reporting while preserving referential integrity throughout the analysis pipeline. Results are systematically archived through comprehensive CSV exports and visualization generation, enabling detailed post-hoc analysis of temporal patterns.

5. Research/Development Process

5.1. Literature Review

We reviewed existing methods for anomaly detection in dynamic graphs, focusing on shallow and deep learning-based approaches.

Shallow Learning-Based Methods

GOutlier [4]: Detects outliers using dynamic network partitions to maintain connectivity.
CAD [5]: Tracks changes in graph structure and edge weights to identify anomalies.
CM-Sketch [6]: Combines structural and historical data for edge anomaly detection.
StreamSpot [7]: Employs clustering with a similarity function for heterogeneous graphs.
SpotLight [8]: Uses randomized sketching to separate anomalous and normal instances.

These methods are limited by their inability to capture complex spatial and temporal relationships.

Deep Learning-Based Methods

NetWalk [9]: Generates embeddings with random walks and dynamic reservoirs.
AddGraph [10]: Combines GCNs for spatial features and GRU-attention for temporal patterns.
StrGNN [11]: Extracts subgraphs and uses stacked GCNs and GRUs for spatial-temporal learning.

TADDY Framework:
TADDY overcomes limitations by:

1. **Simultaneous Spatial-Temporal Modeling**: A transformer-based model captures both spatial and temporal dependencies.
2. **Comprehensive Node Encoding**: Combines diffusion, distance, and temporal encodings for richer graph representations.

These advancements make TADDY a suitable framework for anomaly detection in dynamic citation networks.

5.2. Problem Identification

This research addresses the fundamental problem of detecting irregular citation behaviors in dynamic academic graph structures, where traditional static analysis methods prove insufficient for capturing the temporal evolution of citation relationships.

The complexity of this challenge stems from multiple dimensions: (1) the inherent temporal dynamics of citation networks, where relationship patterns evolve and emerge over time; (2) the multi-scale nature of citation behaviors, ranging from individual paper-to-paper relationships to broader citation communities; and (3) the need to distinguish between natural citation evolution and potentially anomalous patterns. The identification of such anomalous patterns is crucial for maintaining the validity of bibliometric analysis.

5.3. Model Design

The proposed methodology implements a transformer-based architecture for dynamic graph anomaly detection through the TADDY framework. The architectural design encompasses three fundamental components for comprehensive feature extraction and representation learning:

Multi-dimensional Node Encoding: The framework implements a node representation strategy through parallel processing of three distinct embedding spaces: (a) diffusion-based embeddings capturing higher-order structural patterns through eigenvalue decomposition of normalized adjacency matrices (parameterized by teleportation probability c=0.15), (b) distance-based embeddings encoding topological relationships through hop distance computation (configured via neighbor sampling parameter k), and (c) temporal embeddings capturing evolutionary patterns across configurable window sizes.

Transformer-based Feature Integration: The architecture employs a hierarchical transformer encoder (implemented through num_hidden_layers transformer blocks with num_attention_heads attention heads) to process the multi-dimensional node representations. This mechanism facilitates the integration of spatial and temporal features through self-attention operations.

Edge Representation Generation: The framework implements systematic edge embedding generation through aggregation of transformed node features, incorporating both local structural properties and global temporal context. This process is optimized through binary cross-entropy loss computation and negative sampling strategies for anomaly detection.

5.4. Data Collection

Dynamic citation graph snapshots were prepared, where:

- Normal edges represent standard citation behaviors.
- Pseudo-anomalous edges were generated via negative sampling due to the lack of labeled anomalies.
- The datasets were preprocessed into temporal snapshots, with node and edge features encoded according to the algorithm's protocols.

5.5. Hyperparameter Tuning

Key parameters were carefully configured to optimize model performance, including:

- Embedding size for node and edge representations.
- Transformer layers and attention heads for capturing complex spatial-temporal relationships.
- Learning rate, batch size, and a threshold for anomaly score classification. Initial values were chosen based on related transformer-based models and iteratively refined during training.

5.6. Model Training

Training involved a multi-step process:

- Contextual node encoding generated embeddings for normal edges.
- Negative sampling produced pseudo-anomalous edges, which were similarly encoded.
- Both edge types were passed through the Dynamic Graph Transformer, with anomaly scores computed using the Discriminative Anomaly Detector. The model minimized a Binary Cross-Entropy Loss function to differentiate normal from pseudo-anomalous edges.

5.7. Evaluation

The primary evaluation metric employs ROC-AUC computation across multiple temporal granularities: (1) snapshot-wise AUC calculation for temporal performance analysis, and (2) aggregate AUC computation across the complete sequence for overall performance quantification. The framework uses both training phase monitoring through epoch-wise ROC-AUC tracking and detailed anomaly pattern analysis utilizing the AnomalyTracker system with configurable persistence thresholds (default: 0.8) and variance analysis (high_variance_threshold: 0.15).

Performance assessment is conducted across multiple benchmark datasets ('uci', 'digg', 'btc_alpha', 'btc_otc', 'year_1992', 'year_1993', 'five_year') with varying anomaly percentages (0.01, 0.05, 0.1) and training partition ratios. The evaluation pipeline generates comprehensive analytical outputs including temporal score distributions, transition analyses, and persistent anomaly identification through automated reporting protocols.

5.8. Iteration

Refinements were made based on evaluation results to:

Improve contextual encoding and sampling strategies.

Optimize transformer architecture and hyperparameters.

These enhancements strengthened the model's effectiveness in detecting anomalies in dynamic citation networks.

5.9. Tools used

The implementation architecture integrates five essential computational frameworks: (1) PyTorch and Hugging Face Transformers for neural network development and transformer-based components; (2) Anaconda and Git for environment management and version control; (3) NetworkX [12] and SciPy [13] for graph operations and sparse matrix computations; (4) NumPy [14] and Pandas for numerical processing and data manipulation; and (5) Matplotlib [15] and Seaborn [16] for analytical visualization.

5.10. Challenges and Solutions

Challenges Faced:

Cleaning The Dataset: The cit-hepth dataset architecture comprises three primary files, with the core structural information encoded in a CSV format. This CSV file implements a binary edge representation through two essential fields: Source_paper and Target_paper, which collectively define the citation relationships within the graph structure. Each field utilizes an INTEGER data type, where individual papers are assigned unique identifiers. However, the initial implementation employed exceptionally large integer values for these identifiers, leading to computational overhead during processing operations. To address this performance constraint, we implemented a node identifier remapping strategy, transforming the original large-scale integers into a compressed, sequential representation ranging from 1 to 27,770 (corresponding to the total node count in the dataset). This optimization significantly mitigated the computational bottleneck while preserving the structural integrity of the citation network.

Library dependencies: A technical challenge encountered during implementation was the management of legacy library dependencies and version compatibility issues within the Python ecosystem. To address these constraints and ensure long-term reproducibility of our experiment, we implemented a containerized solution utilizing the Anaconda environment management system. This decision was driven by several key considerations: First, rather than pursuing continuous library updates—which introduce potential instability and dependency conflicts—we established a stable, version-controlled environment that encapsulates all required dependencies. This approach not only mitigates the complexity of Python dependency management but also ensures experimental reproducibility across different computational environments and temporal contexts.

The adoption of Anaconda environment management provides several methodological advantages: (1) it establishes a deterministic execution environment that remains stable

over time, (2) it eliminates the need for continuous library updates and dependency reconciliation, and (3) it facilitates experimental reproducibility by other researchers through exact replication of the computational environment.

5.11. Results and Conclusions

High-Energy Physics Theory Citation Network [17] dataset experiments results:

Experiment 1:

Anomaly Percentage	(--anomaly_per):	0.1
Training Portion	(--train_per):	0.5
Number of Neighbors	(--neighbor_num):	8
Window Size	(--window_size):	3
Embedding Dimension	(--embedding_dim):	32
Number of Hidden Layers	(--num_hidden_layers):	4
Number of Attention Heads	(--num_attention_heads):	4
Maximum Epochs	(--max_epoch):	100
Learning Rate	(--lr):	0.001
Weight Decay	(--weight_decay):	5e-4
Seed	(--seed):	1

Results:

Loss: 0.1680 AUC: 0.9510

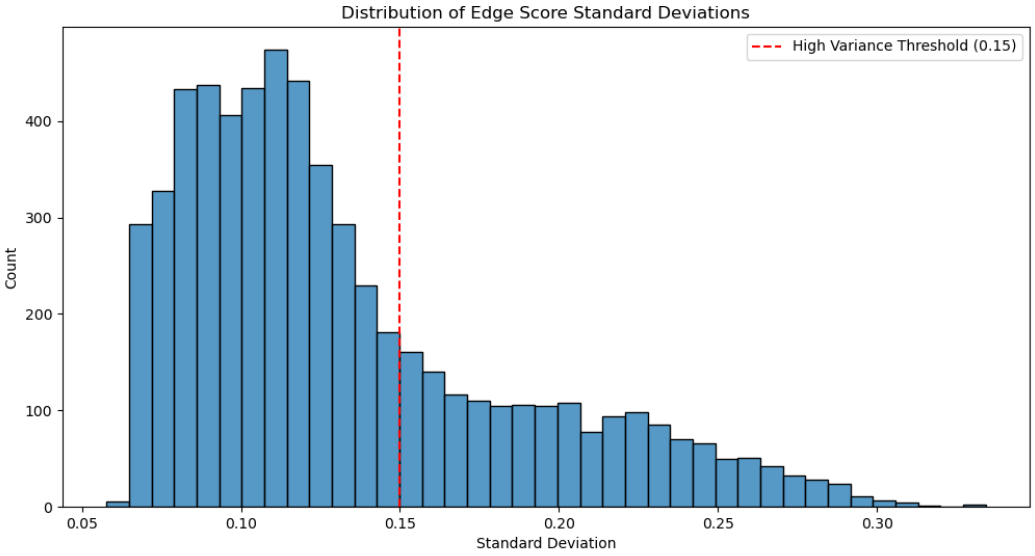


Figure 1 *- Edge score standard deviation distribution from Experiment 1, demonstrating right-skewed characteristics with 0.15 variance threshold. The distribution profile validates improved model discrimination between stable and temporally volatile citation patterns*

Experiment 2:

Dataset: High-Energy Physics Theory Citation Network [17]

Anomaly Percentage	(--anomaly_per):	0.1
Training Portion	(--train_per):	0.5
Number of Neighbors	(--neighbor_num):	10
Window Size	(--window_size):	2
Embedding Dimension	(--embedding_dim):	64
Number of Hidden Layers	(--num_hidden_layers):	8
Number of Attention Heads	(--num_attention_heads):	8
Maximum Epochs	(--max_epoch):	200
Learning Rate	(--lr):	0.001
Weight Decay	(--weight_decay):	5e-4
Seed	(--seed):	1

Results:

Loss: 0.1985 AUC: 0.9613

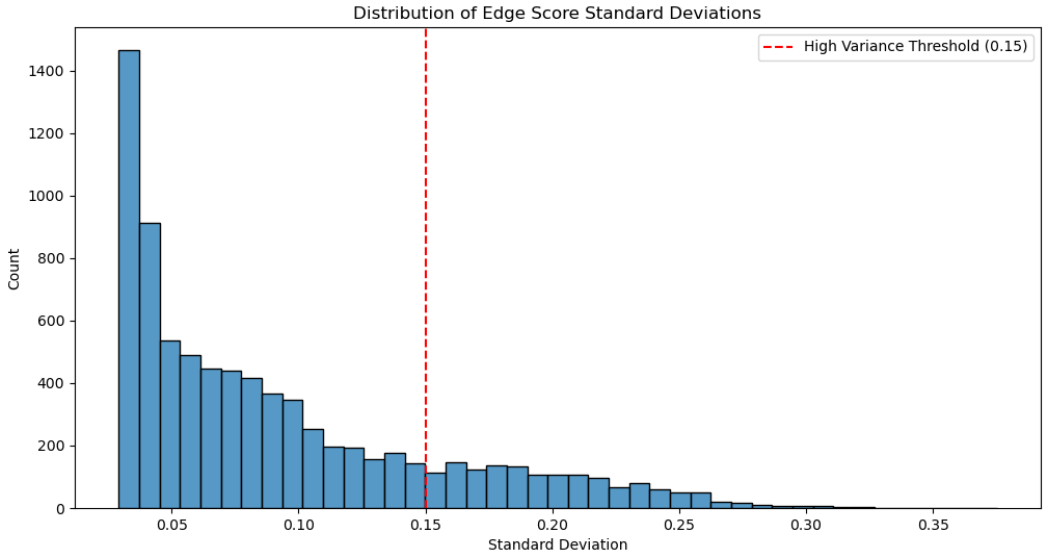


Figure 2 *- Edge score standard deviation distribution from Experiment 2, demonstrating exponential decay with a 0.15 variance threshold. The increased model complexity yields improved anomaly discrimination while maintaining distributional stability.*

Experiment 3:

Anomaly Percentage	(--anomaly_per):	0.1
Training Portion	(--train_per):	0.5
Number of Neighbors	(--neighbor_num):	5
Window Size	(--window_size):	2

Embedding Dimension	(--embedding_dim):	32
Number of Hidden Layers	(--num_hidden_layers):	2
Number of Attention Heads	(--num_attention_heads):	2
Maximum Epochs	(--max_epoch):	100
Learning Rate	(--lr):	0.0005
Weight Decay	(--weight_decay):	5e-4
Seed	(--seed):	1

Results:

Loss: 0.1340 AUC: 0.9579

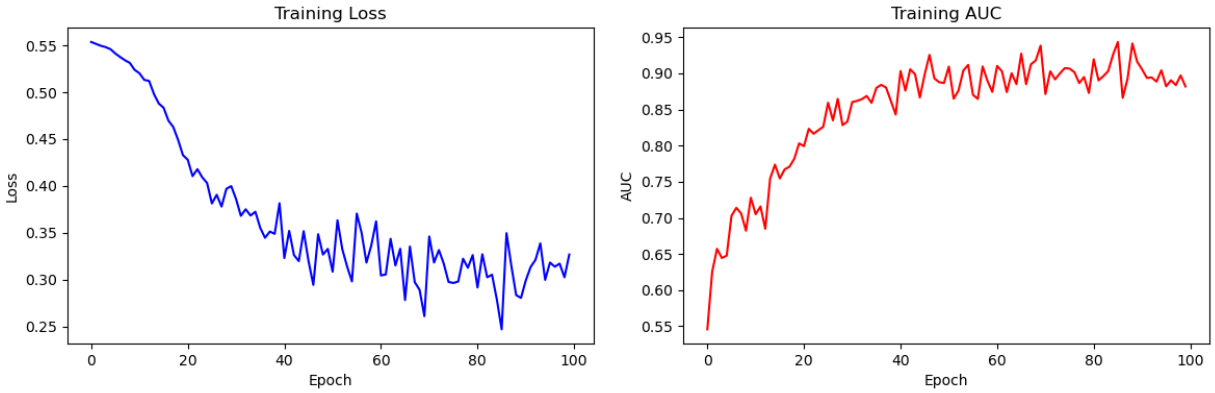


Figure 3 *- This figure shows the model's training progression over 100 epochs, displaying two key metrics: Loss (left) and AUC (right). The loss curve demonstrates a steady decrease, indicating effective model convergence, while the AUC curve shows improvement in the model's ability to distinguish between normal and anomalous citations.*

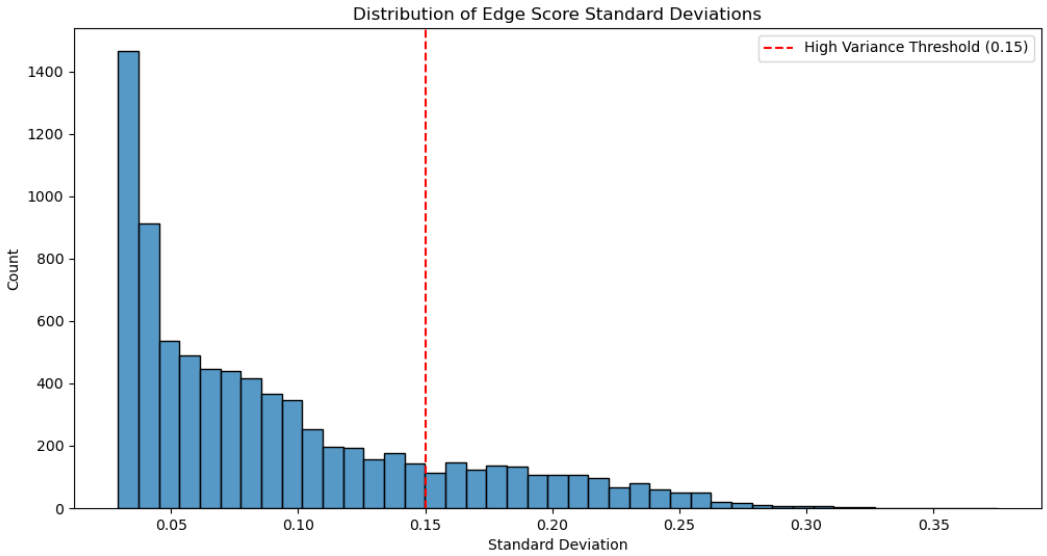


Figure 4 *- Histogram depicting the distribution of edge score standard deviations in the citation network, with a high-variance threshold at 0.15. The exponential decay pattern demonstrates model stability, with most edges maintaining consistent classifications while identifying a distinct subset exhibiting significant temporal variance, indicative of potential citation anomalies.*

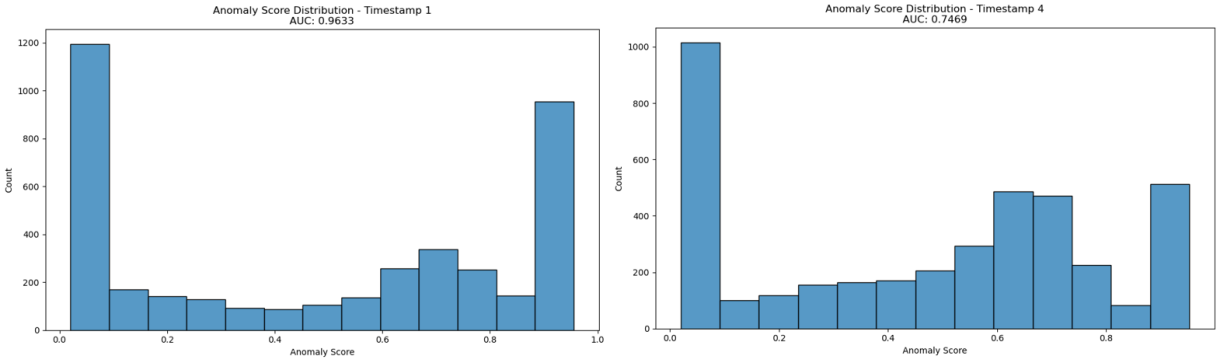


Figure 5 *- Comparative histograms of anomaly score distributions at timestamps 1 and 4 (AUC: 0.9533 and 0.7469 respectively). The distributions transition from a polarized state with peaks at extreme scores (timestamp 1) to a more balanced configuration (timestamp 4), suggesting model adaptation in classification behavior. The reduction in AUC coupled with more granular score distribution indicates refinement in the model's discrimination capabilities during the training process.*

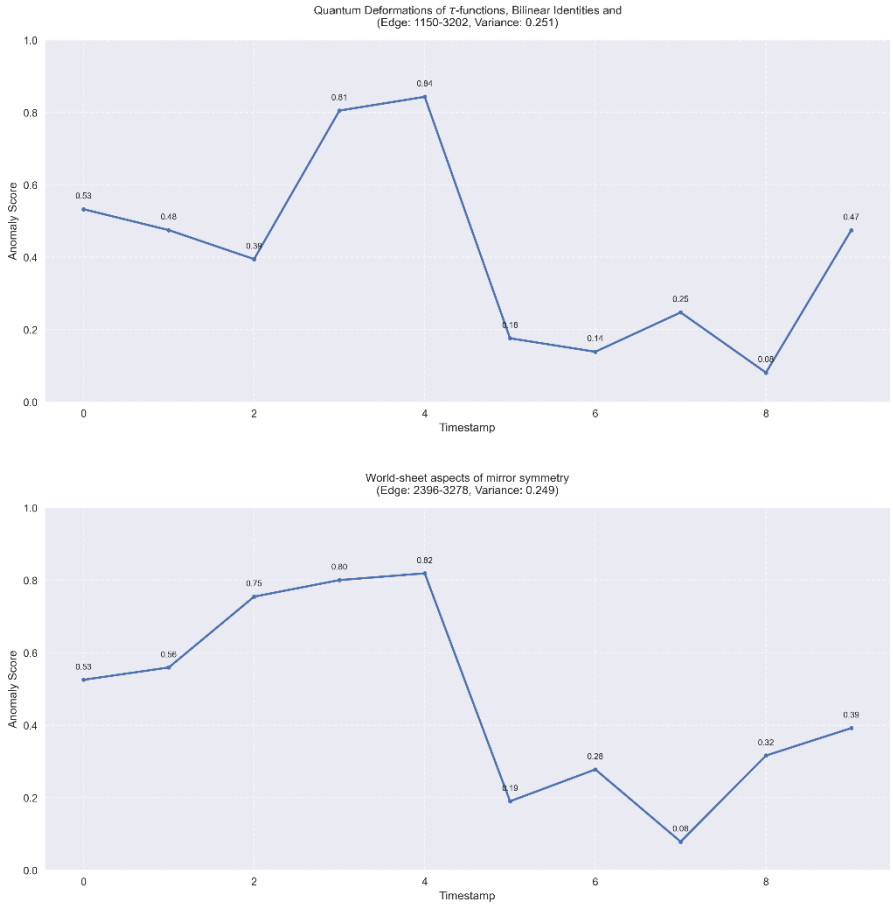


Figure 6 *- Temporal evolution of anomaly scores for two high-variance citation edges ("Quantum Deformations of τ-functions" and "World-sheet aspects of mirror symmetry"), demonstrating significant classification instability ($\sigma \approx 0.25$). The persistent score volatility, characterized by dramatic fluctuations between normal and anomalous states, provides strong evidence for the anomalous nature of these citations despite their variable classification scores*

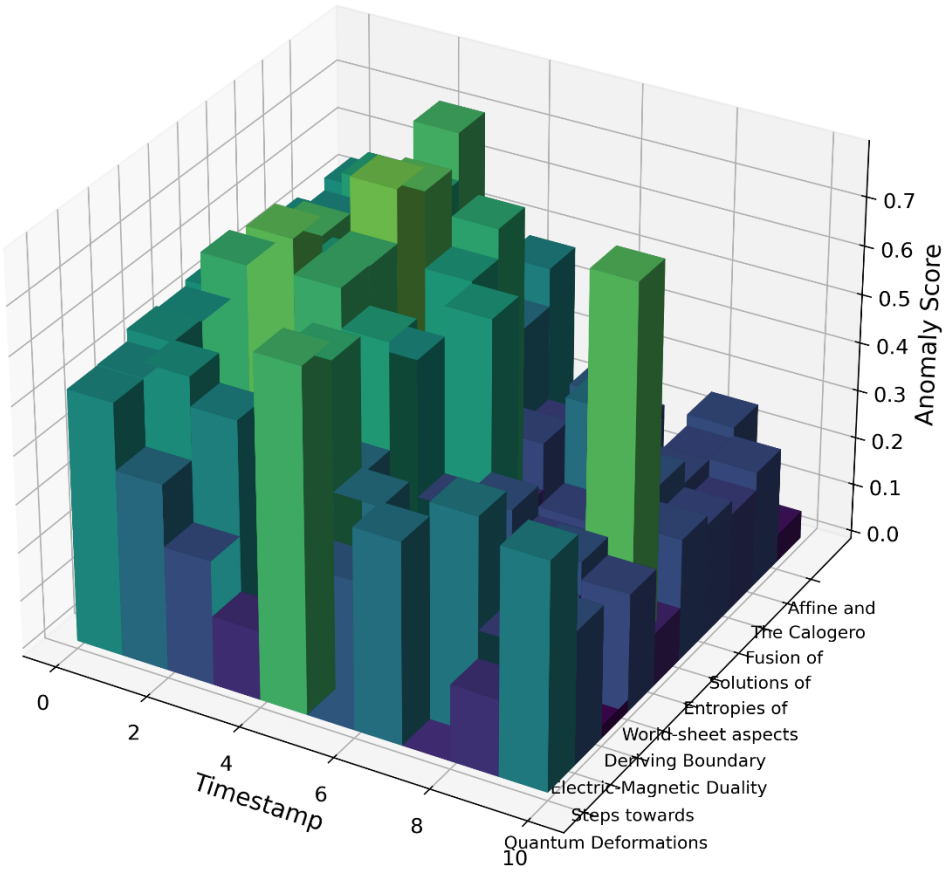


Figure 7 *- Three-dimensional visualization depicting temporal anomaly score distributions across the ten highest-variance papers in the citation network. The z-axis quantifies anomaly scores (0.0-0.8), x-axis represents temporal progression (timestamps 0-10), and y-axis enumerates paper titles. The heterogeneous height variations across timestamps demonstrate consistent classification instability patterns, characteristic of potentially anomalous citation relationships.*

6. Conclusions

This project explores the dynamic nature of anomalies in citation networks. Using a transformer-based model, it offers an advanced perspective on how citation patterns evolve across space and time. By incorporating pseudo-anomalous edges, the model enables effective training even without labeled anomalies, showcasing its practicality for real-world citation networks.

The findings indicate that most papers maintain stable citation patterns, reflecting their typical impact on scientific research. However, a subset of articles exhibits notable variations in citation behavior, signaling changes in their influence within the scientific community. These articles merit closer examination and thoughtful analysis.

7. Users Guide

7.1. Requirements

```
Python==3.8
PyTorch==1.7.1
Transformers==3.5.1
Scipy==1.5.2
Numpy==1.19.2
Networkx==2.5
Scikit-learn==0.23.2
To install requirements:
pip install -r requitements.txt
```

7.2. Usage

Prepare Data

```
python 0_prepare_data.py --dataset five_year --anomaly_per 0.1
```

Train Model

```
python 1_train.py --dataset five_year --anomaly_per 0.1
```

7.3. Setup In *Google Colab*:

Clone Repository:

```
!git clone https://github.com/kookmao/Detection-of-Anomalous-Cited-
Papers
Refresh after running this cell:
!pip install -q condacolab
import condacolab
condacolab.install()
```

Activate environment and install dependencies:

```
!conda create -n myenv python=3.8 -y
!conda init myenv
!conda activate myenv
```

Prepare Data:

```
!conda run -n myenv --live-stream python 0_prepare_data.py --dataset
five_year --anomaly_per 0.1
```

Train Data:

```
!conda run -n myenv --live-stream python 1_train.py --dataset five_year
--anomaly_per 0.1
```

8. Maintenance Guide

The implementation prioritizes experimental reproducibility over continuous library maintenance. While we acknowledge the ongoing advancements in library efficiency, particularly in transformer architectures and graph processing frameworks, our primary objective is to demonstrate result replicability. The Anaconda environment serves as a reference implementation, establishing a verified configuration for result validation. Future researchers may opt to upgrade library dependencies, particularly the transformer components and neural network frameworks, provided they maintain functional equivalence. We believe that this approach balances the dual requirements of result verification and technological advancement.

9. References

[1] Y. Liu, S. Pan, Y. G. Wang, F. Xiong, L. Wang, Q. Chen and V. C. Lee, "Anomaly Detection in Dynamic Graphs via Transformer"," *IEEE Transactions on Knowledge and Data Engineering,* 2023.

[2] L. Page, "The PageRank citation ranking: Bringing order to the web," 1999.

[3] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research,* 2011.

[4] C. C. Aggarwal, Y. Zhao and P. S. Yu, "Outlier detection in graph streams," in *2011 IEEE 27th International Conference on Data Engineering*, 2011.

[5] K. Sricharan and K. Das, " Localizing anomalous changes in time-evolving graphs," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014.

[6] G. Cormode, "Count-Min Sketch," 2009.

[7] E. Manzoor, S. M. Milajerdi and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016.

[8] D. Eswaran, C. Faloutsos, S. Guha and N. Mishra, "Spotlight: Detecting anomalies in streaming graphs," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[9] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018.

[10] L. Zheng, Z. Li, J. Li, Z. Li and J. Gao, "AddGraph: Anomaly Detection in Dynamic Graph Using Attention-based Temporal GCN," in *IJCAI*, 2019.

[11] L. Cai, Z. Chen, C. Luo, J. Gui, J. Ni, D. Li and H. Chen, "Structural temporal graph neural networks for anomaly detection in dynamic graphs," in *Proceedings of the 30th ACM international conference on Information & Knowledge Management*, 2021.

[12] A. Hagberg, P. J. Swart and D. A. Schult, "Exploring network structure, dynamics, and function using NetworkX," 2008.

[13] "SciPy: Fundamental algorithms for scientific computing in Python," [Online]. Available: https://github.com/scipy/scipy.

[14] "NumPy: Array programming with Python," [Online]. Available: https://github.com/numpy/numpy.

[15] [Online]. Available: https://github.com/matplotlib/matplotlib.

[16] [Online]. Available: https://github.com/mwaskom/seaborn.

[17] J. Leskovec, J. Kleinberg and C. Faloutsos, "Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2005.

[18] D. Guo, Z. Liu and R. Li, "RegraphGAN: A graph generative adversarial network model for dynamic network anomaly detection," *Neural Networks,* 2023.

[19] J. Gehrke, P. Ginsparg and J. M. Kleinberg, "Overview of the 2003 KDD Cup," *SIGKDD Explorations,* 2003.