Software Engineering Department Braude

Academic College

Capstone Project Phase B – 61999

Analyzing article's citations using - anomaly detection in dynamic graphs via transformer

24-2-R-2

Project Members:

Elroei Seadia, Dvir Bublil

Dvir.Bublil@e.braude.ac.il

Elroei.Seadia@e.braude.ac.il

Supervisor

Prof. Zeev Volkovich

Advisor

Dr. Renata Avros

Project's Git Repository

Table of Contents

Abstract:

This research deals with the struggle of identifying citations anomalies in academic research. Traditional citation evaluation and detection strategies frequently miss planned manipulations and sincere mistakes. To overcome these limitations, we propose a new technique using TADDY (Transformer-Based Anomaly Detection for Dynamic Graphs). After adapting TADDY to our model, it will treat citation networks as dynamic graphs and allowing analyze complex patterns over time to identify intricate citation behaviors. The research emphasizes the importance of citations anomaly detection, the limitations of existing strategies, and the way dynamic graphs and transformers provide an improved solution. In addition, our findings reveal that TADDY successfully detects anomalous citation patterns, providing a sturdy and efficient framework which can detect citation anomalies. By employing this method, we can trace articles, analyze their evolution over time, and gain valuable insights for the academic community.

The provided numerical experiments demonstrate high ability of the method to recognize anomaly detections in dynamic manner.

Keywords:

Manipulated Citations, Dynamic Graph, Anomaly Detection, Research Analysis, Citation Networks.

1. Introduction

For decades, citation analysis has played a key role in assessing research and its effects. For example, by counting how often a study is cited or referenced by others, and by examining the citations it receives and those it makes, researchers, institutions, and policymakers have gotten useful insights into the importance of scholarly contributions. This measure has helped to inform funding choices, recognize academic accomplishments, and spot new research areas. Going beyond simple counts, looking at citation patterns over time has offered more understanding of a research article's path how it shapes later work, and its overall influence on the field. But the growing problem of citation manipulation and the shortcomings of

old-school methods like counting and studying citations have created a need for smarter ways to do this. While current techniques give some useful information, they often miss the mark when it comes to spotting sneaky misconduct or keeping up with new trends. Some researchers are deliberately trying to make their work look more important by including fake citations or ones that aren't relevant. Others make honest mistakes when citing sources. These issues can make it hard to track and analyze papers and decide which are truly influential and which ones seem important because of manipulation or errors. There are existing ways to try to find unusual citation patterns that might indicate problems. These methods might look at sudden spikes in citations ("citation bursts"), how often papers are cited together ("co-citation patterns"), or how individual researchers cite sources. However, these approaches have limitations [1]. They might miss important new discoveries if they focus too much on bursts. They might not catch problems in single papers if they only look at connections between papers. And they can be influenced by how researchers in different fields typically cite sources or who they collaborate with. This means they might miss real problems, like attempts to manipulate citations, or mistake innocent mistakes for something more serious. Research is constantly evolving, and the way papers are cited is changing too. We need better tools to keep up with this complexity. Some researchers are looking at using machine learning to improve how we detect unusual citation patterns. This research explores a promising tool called Transformer-based Anomaly Detection for Dynamic Graphs (TADDY). We will use TADDY to analyze citation networks as constantly changing graphs in different timestamps. By doing this, it can learn complex patterns that help differentiate between real problems, like attempts to manipulate citations, and simple mistakes. By tracking the evolution of citations over time, we can construct a detailed trajectory for each article, revealing how its citation patterns change [2]. This enables us to identify articles that exhibit anomalous behavior, either consistently or over specific periods. Adapting TADDY on citation networks could lead to a more reliable and adaptable way to find unusual patterns, ultimately helping to ensure research is honest and transparent, and provide as a tool to track research behavior over time.

This research will delve into the background and related work on citation analysis, mathematical frameworks underpinning our approach, the working process from data collection to model evaluation, and the design and implementation of TADDY algorithms. Highlighting the effectiveness of TADDY in improving citation anomaly detection and as a result of that, promoting research analyze by tracking his path during analyzing his behavior in several timestamps.

1.1 Project Description

Research Goal: In our research, we aim to uncover and understand anomalies within citation networks by identifying unusual citation patterns (edges) and anomalous articles (nodes). By adapting the TADDY framework for anomaly detection in dynamic graphs, we focus on tracking these anomalies over time to uncover how articles evolve throughout the dataset's timeline. This approach provides unique insights into citation trends, highlights influential or unconventional works, and examines the dynamic development and impact of academic articles. By addressing these anomalies, our research not only supports academic integrity but also reveals overlooked patterns and contributes to a deeper understanding of citation behaviors.

User Audience: This research targets academics, researchers, and institutions invested in analyzing citation networks and ensuring the credibility of academic publications. Additionally, it serves as a resource for data scientists exploring anomaly detection in dynamic graphs across other domains, such as social networks or financial systems, demonstrating the versatility of these techniques.

2. Background and Related Work

2.1. The Limits of Existing Methods and a New Approach

Uncovering hidden patterns within the citations network is crucial for preserving research integrity and ensuring accurate analysis. Identifying the ones anomalies – unusual citation patterns that deviate from the norm – is crucial for upholding accept as proper with in instructional discourse.

Imagine a bustling city where streets represent research papers and intersections symbolize citations. New buildings (papers) are constantly under construction, and existing roads (citations) are built, rerouted, or even closed as new connections form. Traditional methods for anomaly detection, like counting the traffic on a single street, fail to capture the complexity of this dynamic urban landscape.

Current methods for detecting citation anomalies often rely on techniques that struggle to keep pace with the increasingly sophisticated tactics employed by those seeking to manipulate citations. In addition, traditional citation analysis methods, like counting citations and analyzing co-citations, are easy to bypass through strategic citation manipulation, as Fong and Wilhite's study found [5]. These methods often miss subtle patterns of citation manipulation and struggle to keep up with the changing tactics

researchers use to boost their citation numbers. They can't adjust to new manipulation strategies or catch the small details of unintended mistakes.

For example, one common approach relies on centrality measures like degree centrality (number of citations received) or betweenness centrality (importance of a publication in connecting others). While these metrics offer a basic understanding of citation patterns, they have significant limitations. A study [6] highlights how these measures can be easily manipulated by citation cartels artificially inflating citation counts. Additionally, they struggle to identify subtle anomalies, such as missing citations or unusual citation patterns within specific research communities.

Other processes consist of statistical analysis and key-word analysis. Statistical strategies, like studying citation frequency or co-quotation analysis (identifying papers regularly referred to collectively), can offer a simple expertise of citation styles. However, they warfare to seize the intricate relationships within quotation networks and are not adaptable to evolving tendencies. Analyzing the key phrases used inside citations can provide insights into the context of citations, however depending totally on keywords can be deceptive, as similar key phrases can seem in unrelated contexts. Additionally, this approach requires extensive information pre-processing and can be computationally expensive.

Recent advancements have explored using neural networks for anomaly detection in quotation networks. These networks are powerful tools, capable of gaining knowledge of complex, non-linear relationships inside statistics. However, neural networks also have barriers in this context. The "black box" hassle makes it tough to apprehend how they arrive at their anomaly detections, that could preclude consider and restriction the ability to refine and improve the exist version [7]. Neural networks require big amounts of fantastic training statistics to function effectively, and acquiring clean and complete citation statistics may be challenging, mainly for emerging research fields where facts might be scarce. Moreover, neural networks trained on historical records might warfare to evolve to new and evolving manipulation tactics employed by awful actors.

These limitations highlight the need for a more robust and adaptable approach to citation anomaly detection.

This is where our study comes in. We suggest a new approach using the Transformer-based Anomaly Detection for Dynamic Graphs (TADDY) framework. While we'll explain TADDY in more detail later, the main idea is this: TADDY views the citation network as a changing graph (over a timestamps) where each publication is a point, and citations are the links between them. By analyzing this dynamic graph, TADDY can uncover useful information that might indicate manipulation or errors. Our goal is to develop a more robust and adaptable approach for discerning truth from

anomalies within citation data, fostering a trustworthy and reliable academic ecosystem. This approach also provides the ability to analyze and track the evolution of research over the years.

2.2. Dynamic Graphs as Key to Citation Anomaly Detection

The traditional methods, which includes counting the wide variety of instances a paper is referred to and more, offer only a basic understanding and fail to capture the dynamic and evolving nature of networks (as shown in **[Fig. 1.]**. New publications appear each day, continuously converting the landscape of citations. Subtle anomalies, like a surprising increase in citations from a particular group of authors or lacking hyperlinks inside a studies network, can without difficulty be neglected.

This is in which dynamic graphs come into play. Dynamic graphs treat citations networks as dwelling entities that evolve and adapt over time. This approach allows researchers to:

- Uncover Hidden Knowledge Flows: Dynamic graphs move beyond static snapshots, reflecting the continuing float of citations and the development of recent connections between papers. They capture the continuous development of studies, where new findings generate fresh references and reshape the educational panorama.
- Highlight Subtle Anomalies: By closely monitoring the evolution of relationships in the community, dynamic graphs can monitor subtle adjustments in quotation styles that is probably missed via conventional analysis. For instance, an unexpected upward push in citations from a specific group of authors over a short duration may want to imply capacity manipulation.
- Stay Ahead of Manipulation: As research practices evolve, so do the techniques of folks that would possibly try and manipulate the system. The adaptable nature of dynamic graphs allows in figuring out new types of citation misconduct, acting like a vigilant observer who watches for suspicious patterns.

Using dynamic graphs, researchers can develop more effective and flexible methods for detecting citation anomalies. This leads to a more trustworthy academic environment, where the integrity of citations is maintained, and the true impact of research is accurately reflected. Dynamic graphs are crucial for ensuring that knowledge flows freely and that significant discoveries are properly recognized. In addition, provide us clearly image about the research and discover the ability to track ana analysis research according their evolving.

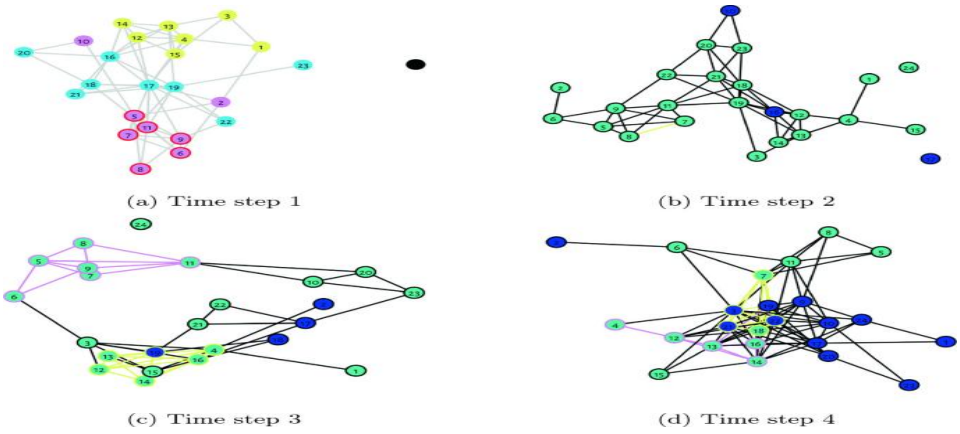

Fig. 1. An example of the evolution of a nature network with different time periods(a, b, c, d). That can be represent a dynamic graph.

2.3. Anomaly Detection in Dynamic Graphs

Detecting unusual connections in dynamic information networks is vital for researchers, especially since these networks are constantly evolving. Traditional methods, such as analyzing structural links or using deep learning techniques, can uncover hidden patterns but often have limitations. For example, approaches like GOutlier [8] and CAD focus on structural connectivity and changes in edge weights to detect anomalies, while deep learning methods like NetWalk [9] and AddGraph [10] use advanced models like Graph Convolutional Networks (GCNs) [11] and Gated Recurrent Units (GRUs) [12] to capture both spatial and temporal patterns.

The TADDY framework stands out by using a transformer network to simultaneously handle both spatial and temporal information, unlike other methods that separate these tasks. TADDY is also better in analyzing unlabeled data, such as social media posts without captions, through a unique encoding process that integrates both spatial and temporal features. This makes TADDY particularly effective in identifying subtle and significant anomalies in dynamic networks, offering a more comprehensive solution compared to existing approaches in the context of anomalies detection in dynamic graphs.

2.4. Transformers

Imagine a detective board filled with evidence: photos of suspects, maps of locations, and a messy timeline of events. In the more traditional approach, a detective would carefully look at each piece of such evidence one after another to put the story together. But TADDY is the detective, and this tool can scrutinize everything instantaneously through its transformer network. This "tool" is the transformer network, a powerful technique in neural network. It is an ultra-powered brain that is good at understanding relations and context between different pieces of information. In TADDY's case, the information is the ever-changing network of connections in a dynamic graph (especially the constantly changing nodes and edges and, as a result, their context and role). The transformer is designed in such a way as to enable a look at the structure of the network (who is connected to whom) and also how the structure of that network is changing over time (new connections appearing, old ones fading). Again, this joint analysis, being driven by the attention mechanism **[see, Fig.2]**, empowers TADDY to pull out important patterns or relationships in dynamic complex networks even if these patterns are very weak or changing very fast. That is why other approaches are more like detectives with individual tools, and TADDY's transformer network makes it the super sleuth. It can analyze the whole "crime scene" at once, which thereby leads to understanding the evolving information landscape. Be able to effectively analyze the dynamic network of citations and their interactions to extract relevant information where traditional methods of mitigation are falling short. The attention mechanism permits the transformer network to focus on certain nodes and edges in the dynamic graph by similarity calculation and weighting clues proportional to their relevance, which sets out the importance of that clue for comprehension of overall network architecture and changes.

- Attention mechanism:

$$\boldsymbol{H^{(l)} = attenion\left(H^{(l-1)}\right) = softmax\left(\frac{Q^{(l)}K^{(l)T}}{\sqrt{d_{emb}}}\right)V^{(l)}}$$

  The output hidden state of a layer is calculated by applying a SoftMax function to the dot product of the query and key matrices, scaled by the square root of the embedding dimension, and then multiplying the result by the value matrix.

$\boldsymbol{Q^{(l)} = H^{(l-1)} * W_Q^{(l)}}$ The query matrix equals the multiplying of the previous layer's hidden state with the tax learnable weight matrix.
$\boldsymbol{K^{(l)} = H^{(l-1)} * W_K^{(l)}}$ The key matrix equals the multiplying of the previous layer's hidden state with ta learnable weight matrix.
$\boldsymbol{V^{(l)} = H^{(l-1)} * W_V^{(l)}}$ The value matrix is equal to the multiplying of the previous layer's hidden state with the learnable weight matrix.

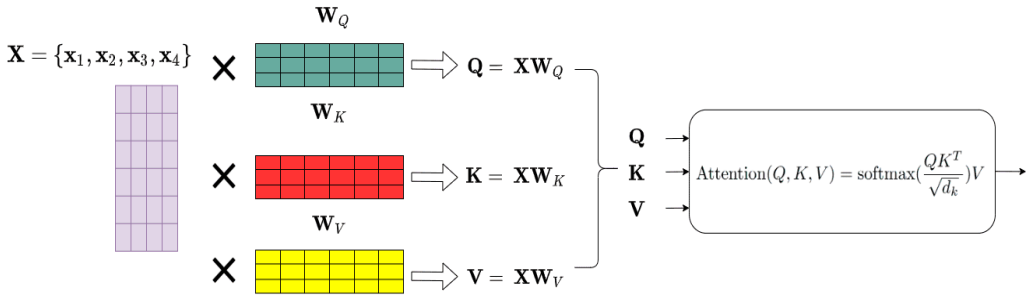

Fig. 2. Example of transformer uses 3 different representations: the Queries, Keys and Values of the embedding matrix. This can easily be done by multiplying our input $X \epsilon R^{N*d_{model}}$ with 3 different weight matrices $W_Q, W_k, W v \epsilon R^{N*d_{model}}$. In essence, it's just a matrix multiplication in the original node embeddings. The resulted dimension will be smaller: $d_k < d_{model}$

2.5. Snapshots

Snapshots refer to temporary captures of the dynamic graph at specific moments. Think of the dynamic graph as a constantly changing web of connections. Snapshots are like grabbing still frames of this ever-evolving network, capturing its state at specific points in time. These snapshots are crucial for TADDY's analysis, but unlike relying solely on individual pictures, TADDY's strength lies in its ability to analyze both the snapshots themselves and how the connections within them evolve over time.

2.6. Timestamps

Timestamps are like labels on those snapshots of the dynamic graph and determine the exact moment to capture the state of the network. Imagine the dynamic graph as a constantly changing network of connections, and snapshots as pictures capturing its state at specific moments. Timestamps act like labels on these pictures, telling you exactly when each snapshot was taken. This allows TADDY to not only analyze the connections within each snapshot but also track how they change over time. It's like comparing photos taken at different points to see how the network evolves.

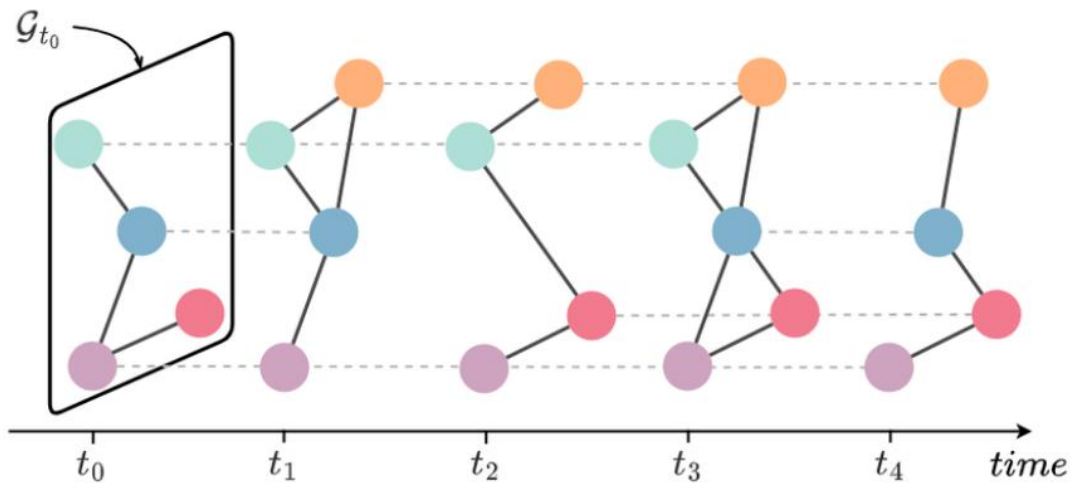

Fig. 3. A Discrete-Time Dynamic Graph defined over five timestamps (in each timestamp we took a snapshot of the evolving graph). The evolution of a Continuous-Time Dynamic Graph through the stream of events until the timestamp t_4

2.7. Binary adjacency matrix

Binary adjacency matrix acts like a map of connections, telling you who's connected to whom at a specific moment.

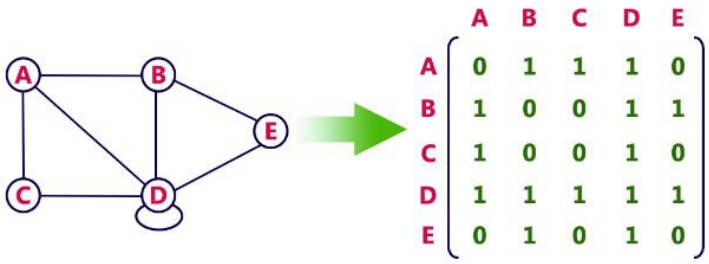

Fig. 4. An example of binary adjacency matrix for given graph, one indicating there is connection between two points, otherwise zero

2.8. Anomaly edge detection to ranking problem

The approach to detecting unusual connections and anomalies in dynamic networks is rooted in ranking dynamics. A network is conceptualized as a complex system where nodes are ranked based on their interaction levels and their context (structural and temporal). Edges represent the connections between these ranked entities. The model focuses on identifying disruptions in this ranking order. A sudden surge in interactions for a previously low-ranking node is considered an anomaly, potentially signaling a hidden connection or suspicious activity. By prioritizing these ranking shifts, we efficiently uncover unusual patterns that deviate from the network's normal behavior.

2.9. Anomaly score

An anomaly score is a numerical value that represents how much an element deviates from what is considered normal or expected within a dataset. The anomaly score helps identify patterns that deviate from the norm. It's like having a spotlight that shines on potentially suspicious connections, allowing to take a closer look and understand why they might be unusual.

2.10. Target Edge

Each edge in dynamic graphs is viewed as the center of the sampled substructure and is denoted as a target edge.

2.11. Target Nodes

The source node and destination node of the target edge denote as target nodes.

2.12. Contextual Nodes

Neighboring nodes of target node in the sampled substructure, are denote as contextual nodes.

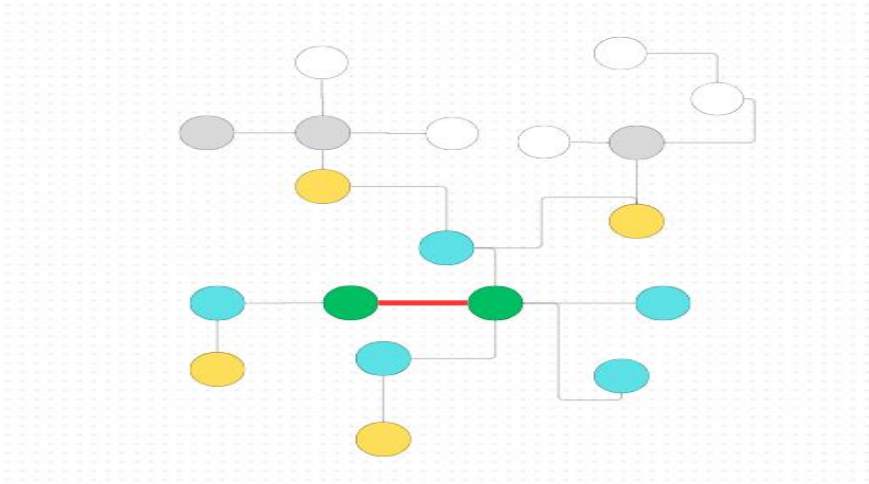

Fig. 5. Illustrate a graph where the target edge is highlighted with a red line, connecting two target nodes shown in green. Surrounding these are contextual nodes: blue for nodes at distance 1 from the target nodes, yellow for distance 2, and gray for distance 3 or more. Here, the contextual nodes are determined based on distance, but different methods can define contextual nodes differently, which will be described in more detail in the next sections.

2.13. Spatial Local Information

This refers to details about the immediate surroundings of a particular point

or element within a network. Imagine a map – spatial local information for a city might include the names of its closest neighboring towns, the types of businesses located nearby, or the presence of a park across the street.

2.14. Spatial Global Information

This focuses on the bigger picture within a network, considering the entire structure and how different elements relate to each other across space. Going back to the map analogy, spatial global information would be understanding the city's location within the country, its connection to major highways, or its position relative to other large cities.

2.15. Temporal Information

This refers to information related to time. In a network context, it could be when connections were formed, how they change over time, or the order in which events occur within the network. Imagine a dynamic traffic map – temporal information would tell you the current traffic flow, how it changes throughout the day, or how congestion builds and dissipates over time.

3. Mathematical Background

3.1 Notations

$\boldsymbol{A} \in \boldsymbol{R^{n_x n}}$ **–** The Binary adjacency matrix, each entry in the matrix corresponds to the connection between two specific nodes (i, j). 0 indicates that there is no direct connection between node i and node j. 1 indicates that there is direct connection between node i and node j.

$\boldsymbol{A^t}$ **-** The binary adjacency matrix at timestamp t.

$\boldsymbol{n^t}$ – The number of nodes at timestamp t.

$\boldsymbol{m^t}$ – The number of edges at timestamp t.

$\boldsymbol{G} = \{\boldsymbol{g^t}\}_{\boldsymbol{t=1}}^{\boldsymbol{T}}$ – A graph steam with a maximum timestamp of T.

$\boldsymbol{g^t} = (\boldsymbol{V^t}, \boldsymbol{\varepsilon^t})$ – The snapshot graph at timestamp t.

$\boldsymbol{V^t}$ – The node set at timestamp t.

$\boldsymbol{\varepsilon^t}$ – The edge set at timestamp t.

$\boldsymbol{v_i^t} \in \boldsymbol{V^t}$ – A node with index i at the timestamp t.

$\boldsymbol{\varepsilon_{i,j}^t} = (\boldsymbol{v_i^t}, \boldsymbol{v_j^t}) \in \boldsymbol{\varepsilon^t}$ **–** An edge between $\boldsymbol{v_i^t}$ and $\boldsymbol{v_j^t}$ at the timestamp t.

$\boldsymbol{f}(\cdot)$ – Anomaly score function.

$\boldsymbol{G_\tau^t} = \{\boldsymbol{g^{t-\tau+1}}, \dots, \boldsymbol{g^t}\}$ – The sequence of graphs with timestamp t as the end.

$\boldsymbol{S}(\boldsymbol{e_{tgt}^t})$ – The substructure node set of target edge $\boldsymbol{e_{tgt}^t}$.

$\boldsymbol{x_{diff}(v_j^i)}$ – The diffusion-based spatial encoding of node $\boldsymbol{v_j^i}$.

$\boldsymbol{x_{dist}(v_j^i)}$ – The distance-based spatial encoding of node $\boldsymbol{v_j^i}$.

$\boldsymbol{x_{temp}(v_j^i)}$ – The relative temporal encoding of node $\boldsymbol{v_j^i}$.

$\boldsymbol{x(v_j^i)}$ – The fused encoding of node $\boldsymbol{v_j^i}$.

$\boldsymbol{X(e_{tgt}^t)}$ – The encoding matrix of target edge $\boldsymbol{e_{tgt}^t}$.

$\boldsymbol{H^{(l)}}$ – The output embedding of the l-th layers of Transformer.

$\boldsymbol{Q^{(l)}}$ – The query matrix of the l-th layers of Transformer.

$\boldsymbol{K^{(l)}}$ – The key matrix of the l-th layers of Transformer.

$\boldsymbol{V^{(l)}}$ – The value matrix of the l-th layers of Transformer.

$\boldsymbol{z(e_{tgt}^t)}$ – The embedding of target edge $\boldsymbol{e_{tgt}^t}$.

$\boldsymbol{W_Q^{(l)}, W_K^{(l)}, W_V^{(l)}}$ – The learnable parameters of Transformer.

$\boldsymbol{e_{pos,i} \in \varepsilon^t}$ – The i-th positive edge from $\boldsymbol{\varepsilon^t}$.

$\boldsymbol{e_{neg,i} \in \varepsilon_n^t \sim P_n(\varepsilon^t)}$ – The i-th negative edge by negative sampling $\boldsymbol{P_n(\varepsilon^t)}$.

$\boldsymbol{w_S, b_S}$ – The learnable weights and parameters of the scoring module.

$\boldsymbol{k}$ – The number of contextual nodes.

$\boldsymbol{d_{enc}}$ – The dimension of encoding.

$\boldsymbol{d_{emb}}$ – The dimension of embedding.

$\boldsymbol{L}$ – The learnable parameters of Anomaly Detector.

$\boldsymbol{\tau}$ – The size of time window.

$\boldsymbol{n_s = \tau(k+2)}$-is the number of nodes in the substructure of node set $\boldsymbol{S(e_{tgt}^t)}$.

$\boldsymbol{e_{pos}, i}$ - The i -th positive (normal) edge.

$\boldsymbol{e_{neg}, i}$ – The i -th negative (pseudo-anomalous) edge.

$\boldsymbol{S \in R^{n_x n}}$ – graph diffusion matrix.

$\boldsymbol{T \epsilon R^{n_x n}}$ - is the generalized transition matrix derived from A.

$\boldsymbol{Q_k}$ - The weighting coefficient that determines the balance between global and local information.

Source: [13]

3.2. Graph Diffusion Matrix

The Graph Diffusion Matrix is essentially important in modeling the flow of information over a network. In our case, this helps capture structural and temporal dynamics within the graph, which is applied to anomaly detection. It is done by simulating the diffusion process using the Personalized PageRank (PPR) method over the matrix. Personalized PageRank (PPR) is a variant of the classic PageRank algorithm that considers the importance of nodes relative to some particular node or group

of nodes. PPR simulates a random walk on a graph in which, at each step, a person can either continue moving with probability α or return back to the starting node. Such an approach focuses the ranking in a local neighborhood of nodes and is hence quite useful where the relationship of a node to a specific set of nodes is more important than its global importance. The PPR formula reflects such a localized importance and is useful in identifying anomalies by pointing out deviations from the expected spreading pattern within the network.

There's a probability $\boldsymbol{\alpha}$, The formula for PPR is:

$S^{PPR} = \boldsymbol{\alpha}(\boldsymbol{I_n} - (\mathbf{1} - \boldsymbol{\alpha})\boldsymbol{D}^{-\frac{\mathbf{1}}{\mathbf{2}}}\boldsymbol{A}\boldsymbol{D}^{-\frac{\mathbf{1}}{\mathbf{2}}})^{-\mathbf{1}}$

$\boldsymbol{\alpha}$ is the teleport probability, controlling the likelihood of returning to the starting node.

$\mathbf{I_n}$ is the identity matrix.

$\boldsymbol{D}$ is the diagonal degree matrix of the graph.

3.3. Single-layer linear encoding learning function

The single-layer linear encoding learning function translates differences in node attributes into a linear encoded format. This is useful for many tasks (distances, connections, etc.). We'll be using this approach in three different contexts:

Diffusion-Based Spatial Encoding

$$\boldsymbol{x_{diff}}(\mathbf{v}_j^i) = \boldsymbol{linear}(\boldsymbol{rank}\left(\mathbf{S}_{e_{tgt}}^i\right)[\boldsymbol{idx}(\mathbf{v}_j^i)]) \in \boldsymbol{R}^{d_{enc}}$$

Distance-Based Spatial Encoding

$$\boldsymbol{x_{dist}}(\mathbf{v}_j^i) = \boldsymbol{linear}(\boldsymbol{min}(\boldsymbol{dist}(\mathbf{v}_j^i, \mathbf{v}_1^i), \boldsymbol{dist}(\mathbf{v}_j^i, \mathbf{v}_2^i))) \in \boldsymbol{R}^{d_{enc}}$$

Temporal Encoding

$$\boldsymbol{x_{temp}}(\mathbf{v}_j^i) = \boldsymbol{linear}(||\boldsymbol{t} - \boldsymbol{i}||) \in \boldsymbol{R}^{d_{enc}}$$

3.4. Encoding fusion

Encoding fusion, in general, refers to the process of combining information extracted from different encoding methods.

$$\boldsymbol{x}(\mathbf{v}_j^i) = \boldsymbol{x_{diff}}(\mathbf{v}_j^i) + \boldsymbol{x_{dist}}(\mathbf{v}_j^i) + \boldsymbol{x_{temp}}(\mathbf{v}_j^i) \in \boldsymbol{R}^{d_{enc}}$$

In our case, this fused encoding represents each node in the substructure with a comprehensive blend of spatial and temporal information, allowing for a more detailed and nuanced understanding of the node's role within the graph.

3.5. Encoding Matrix

For a given target edge e_{tgt} , the encoding of each node in its substructure node set is calculated and stacked into an encoding matrix $\boldsymbol{X}(\boldsymbol{e_{tgt}})$

$$\boldsymbol{X}(\boldsymbol{e}_{\boldsymbol{tgt}}^{\boldsymbol{t}}) = \oplus_{\mathbf{v}_j^i \in \boldsymbol{S}(e_{tgt}^t)} [\boldsymbol{x}(\mathbf{v}_j^i)]^{\boldsymbol{T}} \in \boldsymbol{R}^{(\tau(k+2)) \times \boldsymbol{d_{enc}}}$$

3.6. Scoring Module

A scoring module, typically a fully connected neural network layer with a sigmoid activation function, computes the anomaly scores for each edge embedding:

$$\boldsymbol{f}(\boldsymbol{e}) = \boldsymbol{Sigmoid}(\boldsymbol{z}(\boldsymbol{e})\boldsymbol{w_S} + \boldsymbol{b_S})$$

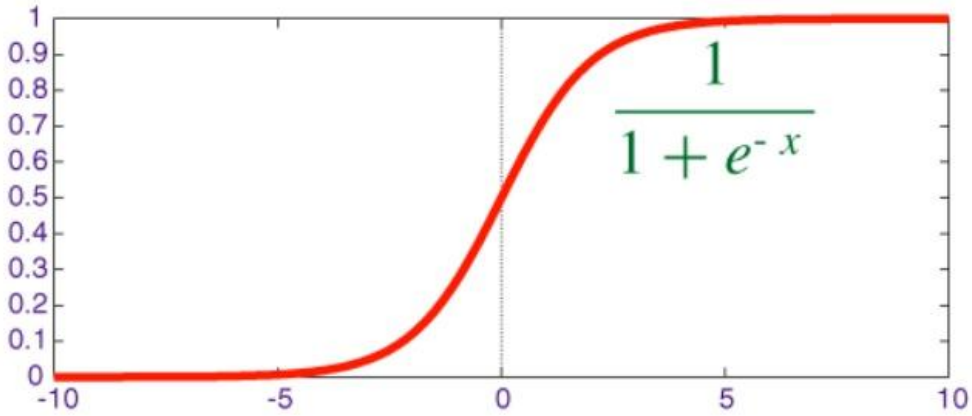

Fig. 6. The sigmoid activation function maps input values to a range between 0 and 1, making it useful for binary classification tasks by converting linear outputs into probabilities

3.7. Loss Function

A loss function is how the progress of a model is rated. Cross-entropy loss, used for classification problems, gives feedback when the model's guesses are off. By reducing this loss, the model get better at making accurate predictions. In our case the model is trained by using a binary cross-entropy loss function

$$\boldsymbol{L} = -\sum_{\boldsymbol{i=1}}^{\boldsymbol{m^t}} \mathbf{log}\,(\mathbf{1} - \boldsymbol{f}(\boldsymbol{e_{pos}}, \boldsymbol{i})) + \mathbf{log}\,(\boldsymbol{f}(\boldsymbol{e_{neg}}, \boldsymbol{i})),$$

which helps differentiate between normal and pseudo-anomalous edges.

3.8. Hyperparameters

- **Number of Contextual Nodes (**k**)**: Determines the number of neighboring nodes in edge-based substructure sampling. Larger k improves performance but increases computational cost. K =

5 balances efficiency and accuracy.

- **Time Window Size:** This hyperparameter $\boldsymbol{\tau}$ determines the length of the graph sequence the algorithm analyzes along the timeline. For instance, with $\boldsymbol{\tau}$ =3, we utilize 4 snapshots spanning t-3 to t (t,t-3,t-2,t-1). This time window enables capturing the dynamic evolution of node connections over time.
- **Encoding/Embedding Dimension (d):** Specifies the vector dimensionality for node encoding. Smaller d may miss information, while larger d can introduce noise. $d = 16$ is generally effective.
- **Number of Transformer Layers (L):** Controls the number of layers in the dynamic graph transformer. More layers enhance node interactions, but too many slow down the model. $L = 2$ is typically optimal.
- **Training Ratio:** Refers to the proportion of data used for training. Higher ratios improve performance but increase the risk of overfitting.

3.9. Edge-based substructure sampling

In the context of anomaly edge detection within dynamic graphs, capturing the spatial-temporal context of a target edge is crucial for effective identification. This involves pinpointing not only the target nodes directly connected to the potentially anomalous edge, but also the context nodes surrounding them at various points in time. Prior research suggests that anomalies often reside within local substructures of the graph, prompting our focus on these smaller, highly connected areas.

To efficiently select the most informative context nodes, particularly in large, intricate networks, we leverage a diffusion-based sampling technique. This method contrasts with simpler approaches like H-hop sampling, which selects neighbors based solely on their distance from the target nodes. Diffusion-based sampling, in contrast, grants a global perspective of the graph, allowing us to assess the relative importance of each potential context node in relation to the target edge. This enables a more nuanced understanding of the context surrounding the potential anomaly.

The dynamicity of the graph demands to be refined by one step further. We decompose the dynamic graph in discrete time slices and treat each slice as a static graph. On these static representations, the algorithm relies on a diffusion-based sampling technique to capture temporal evolution of context nodes surrounding the target edge. The process of substructure generation is such that the target edge's spatial-temporal context is well captured through informative nodes collected from different time slices.

This set of nodes will compose the final substructure for the target edge, and its context will be elaborated. This will facilitate the analysis of its potential anomalies**.**

Formally:

Based on the adjacency matrix, **S** is defined as the diffusion matrix graph by summing an infinite series $\boldsymbol{S} = \sum_{\boldsymbol{k=0}}^{\infty} \boldsymbol{Q_k T^k}$.

The value $\boldsymbol{S_{i,j}}$ reflects the relative importance of node **j** to node **I** . A higher value indicates a stronger connection between them, while a lower value indicates a weaker connection. Unlike the adjacency matrix, the values in S are continuous, not just 0 or 1, which helps in understanding the relative importance of neighbors and provides insights into the strength of their relationships.

To ensure the convergence of the infinite series, let's consider some stringent conditions, the sum of $\sum_{\boldsymbol{k=0}}^{\infty} \boldsymbol{Q_k}$ should approach 1, and Q_k should be [0,1]. Additionally, the eigenvalues of **T -** $\boldsymbol{\lambda_i}$ bounded [0,1].

By making specific adjustments to the definitions of **T** and **Q**, we can obtain different versions of graph diffusion that emphasize various types of information. For example, a higher eigenvalue λ will focus on the local neighborhood of the target node.

Given a diffusion matrix **S**, row $\boldsymbol{S_i}$ shows the connection of node i to all other nodes from a global perspective rather than a local one. Each cell $\boldsymbol{S_{i,j}}$ represents the degree of connectivity between **i** and **j** with a continuous value between 0 and 1, allowing for a more precise distinction between nodes compared to other approaches like h-hop method as mentioned before.

By leveraging this approach, the algorithm can select a fixed number of the most relevant and important context nodes for a given target node. For each target edge we can compute the context vector for this edge by summing the context vectors computed for the two endpoint nodes of this edge. Then sorting the context vector and choose the **K** highest nodes, indicating the most significant nodes, to form the set of context nodes $\boldsymbol{U}(\boldsymbol{e_{tgt}})$. The target nodes are not chosen when selecting the K context nodes but are added to the final sample set of the substructure. This results - $\boldsymbol{S}(\boldsymbol{e_{tgt}}) = \{\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{U}(\boldsymbol{e_{tgt}})\}$ is the final group of sampled nodes in the substructure for the target edge $\boldsymbol{e_{tgt}}$.

Moving from static graphs to dynamic ones, a mechanism is introduced to handle the evolving nature of connections over time. Given a target edge within a time period t, denote a sequence of graphs (snapshots) $G_\tau^t = \{g^{t-\tau+1}, \dots, g^t\}$ of length τ.

For each graph $g^i \in G_\tau^t$, we compute the diffusion matrix S^i and extract its

connectivity vector $s^{i}_{e^{t}_{tgt}}$, selecting the top k nodes and adding the target nodes. Finally, the result of multiple time series t is getting the substructure node set $s(e^{t}_{tgt}) = U^{t}_{i=t-\tau+1}(s^{i}(e^{t}_{tgt}))$.

3.10. Spatial-temporal Node Encoding

Spatial-temporal node encoding is a technique used to represent nodes in dynamic graphs by incorporating both spatial (structural) and temporal (time-based) information. This encoding is crucial for understanding the roles and interactions of nodes in evolving networks, such as social networks, e-commerce platforms, and cybersecurity systems.

Unlike static graphs, which have attributes and data similar to other entities (such as images) with raw features for each element, dynamic graphs often lack inherent attributes. This means it is challenging to find suitable natural data inputs for neural network models. Therefore, the critical question arises: how do we construct informative encodings as inputs for neural networks from dynamic graphs? While there are currently several solutions, they are not the most efficient or effective. For example, a commonly used method is node identity encoding, which works similarly to one-hot encoding in NLP, representing each node with a unique vector. However, this approach has limitations:

- Limited Information: This encoding cannot simultaneously capture spatial/structural and temporal information, representing only the node identity and struggling to reflect its structural role and temporal state in the graph.
- Unsuitable for Large, Dynamic Graphs: Nodes are frequently created and removed, making this method impractical.

The new approach is designed to build a new spatial-temporal node encoding for dynamic graphs. This encoding consists of three components, which are eventually combined into a single vector that serves as the node encoding input containing comprehensive information. These components are**:**

Diffusion-Based Spatial Encoding, Distance-Based Spatial Encoding, Relative Temporal Encoding

The first two encodings represent the structural role of each node from both a global and local perspective, respectively. The temporal encoding provides information about the timing of each element within the substructure. A crucial note: we will use a linear learning function instead of the COS/SIN functions used in the original Transformer, as these are more flexible for learning relationships between different time sequences or locations in the graph.

Now, let's delve into the three encodings:

Diffusion-Based Spatial Encoding

Diffusion provides a global perspective, capturing the global role of a node and understanding its structural function. This information is crucial for the neural network to comprehend the relative importance of each node in relation to the target edge, effectively understanding the strength of the connection between the target edge and each contextual node. To avoid identical encodings for nodes with the same diffusion values, we will use rank-based encoding. For each node in the substructure sampled at a single time point for the target edge, a ranked list based on their diffusion values is created, and each node's rank is used as the basis for its encoding. The diffusion-based spatial encoding is computed using a single-layer linear transformation function which is similar to the positional encoding in Transformers.

Distance-Based Spatial Encoding

The goal is to capture the local role of a node concerning the target edge. Unlike diffusion, which focuses on the global role, local information helps the neural network learn the immediate significance of a node relative to the target edge. For each node in the substructure at a single time point in the target edge's node group, its distance to the target edge serves as the basis for encoding. The distance to the target edge is defined as the minimum of the relative distances to the two nodes comprising the edge, with the distance to the target nodes themselves being zero.

Relative Temporal Encoding

The goal is to capture the temporal information of each node relative to the target edge in the substructure at each time point. Unlike the absolute time encoding in the original Transformer, we propose a relative encoding for dynamic graphs, comparing different time points of a node. The time difference is essential for identifying normal or anomalous target edges in a dynamic graph. For each node in the substructure of the target edge at the current timestamp, the source of the relative temporal encoding is defined as the difference between the time when the target edge occurred (T) and the current timestamp (I).

Fusion the Encodings

The final step is merging the three encodings to obtain a comprehensive and informative picture for each node in the substructure of the target edge's node group. The merged encoding serves as the next input stage for the Transformer. For computational efficiency, the Fusion combine the components using summation instead of concatenation, resulting in a simpler and more efficient calculation.

Ultimately, for each target edge, we compute the encoding of each node in its substructure's node group and assemble these encodings into a single encoding matrix. This matrix represents the "attributes" of the target edge and serves as the input to the Transformer.

3.11. Dynamic graph transformer

Analyzing dynamic graphs, where connections evolve over time, poses a challenge for neural networks. Capturing both spatial (network structure) and temporal (time-varying changes) information is crucial for accurate anomaly detection. Our proposed approach introduces a novel single transformer encoder architecture to effectively capture both spatial and temporal information simultaneously. This eliminates the need for complex hybrid systems or multiple separate models. The encoder takes as input the nodes embedding vectors for multiple time steps, obtained from the previous stage. It then undergoes two key processes:

- **Transformer Model:** This generates meaningful encoding vectors for each node based on the input, utilizing multiple attention/encoding layers. These layers enable nodes to "communicate" and exchange information, allowing each node to learn to focus on other relevant nodes in the context of the target edge. This captures complex relationships between nodes in both time and space.
  The last layer in the transformer module $\boldsymbol{H^{(l)}}$ acts like a final filter, refining its understanding of each node in the network. The resulting output, called the "output node embedding matrix" ($\mathbf{Z}$), is essentially a collection of "node profiles." Each row in this matrix represents a single node, with the corresponding embedding vector capturing its key characteristics and connections within the network.

- **Pooling Model:** The goal of this section is to transform the encoding matrix Z into a single vector for the target edge $\boldsymbol{z(e_{tgt}^t)}$ **.** This vector represents the overall significant features of the target edge, capturing the information learned from the connected nodes. We employ a pooling method previously used in [14], which involves simply averaging the enhanced encoding vectors of the relevant nodes for the target edge. Mathematically, this is represented as the following function:
  $\boldsymbol{z(e_{tgt}^t) = pooling(Z) = \sum_{k=1}^{n_s} \frac{(Z)_k}{n_s}}$
  This pooling method effectively combines the information from the connected nodes into a single vector, providing a comprehensive

representation of the target edge for anomaly detection, the single vector for given edge is $\boldsymbol{z}(\boldsymbol{e}_{\boldsymbol{tgt}}^{\boldsymbol{t}})$.

3.12. Discriminative Anomaly Detector

Finding anomalies in dynamic graphs without any pre-labeled examples can be extremely confusing. Here is how this is achieved in our model. First, pseudo-negative edges are created by randomly selecting two nodes from the training data that are not yet connected. If the pair of nodes selected already have a connection, they are discarded, and another pair is selected until an unconnected pair is found. For each of these pseudo-negative edges, the surrounding nodes are examined, and their spatial and temporal information is encoded. These encodings are then fed into a transformer model specifically designed for dynamic graphs, producing an overall encoding vector for each pseudo-negative edge. This vector is run through a neural network with a sigmoid function, resulting in an anomaly score ranging from 0 to 1, where 0 signifies normality and 1 indicates a highly anomalous edge. The loss function (binary cross-entropy loss) treats normal edges as 0 and pseudo-negative edges as 1. The network minimizes the difference between the desired and actual outputs during training to better distinguish normal from anomalous edges. In this manner, anomalies can be detected even without pre-labeled examples. Generating pseudo-negative edges exposes a variety of potential anomalies to the network.

4. Process

4.1. TADDY Overview

The TADDY framework for detecting anomalies in dynamic graphs works in a few clear steps using the methods that we explained and show in the previous sections. First, it looks at the local areas around each edge it wants to examine and picks out the most relevant nearby nodes based on how they connect within the graph. Then, it gathers detailed information about each node, including how it's connected to others and how it changes over time. This information is turned into a feature vector for each node. These vectors go into a transformer model that learns to understand patterns in both space (how nodes are connected) and time (how these connections change). The model then pools these learned patterns to create a representation for each edge. Finally, it scores each edge to see how likely it is to be an anomaly. To train the model, both real data and artificially created anomalies are used, so the model learns to tell the difference between normal and abnormal edges effectively.

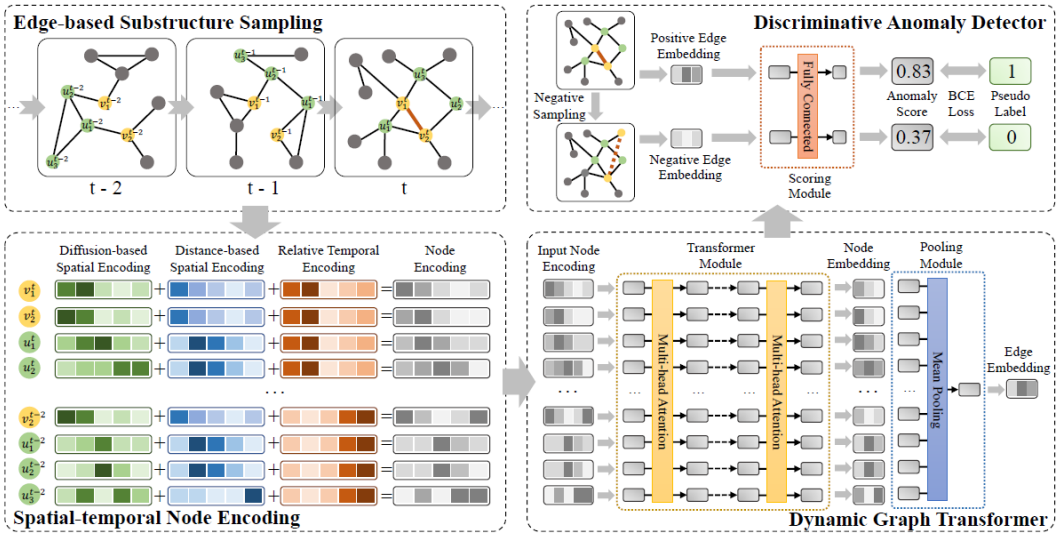

Fig. 7 . . The overall framework of TADDY. The framework is composed of four components: edge-based substructure sampling, spatial-temporal node encoding, dynamic graph transformer, and discriminative anomaly detector.

4.2 Code structure

This section elaborates on the proposed solution and its implementation, meticulously designed to achieve the research objectives, as illustrated in the following diagram.

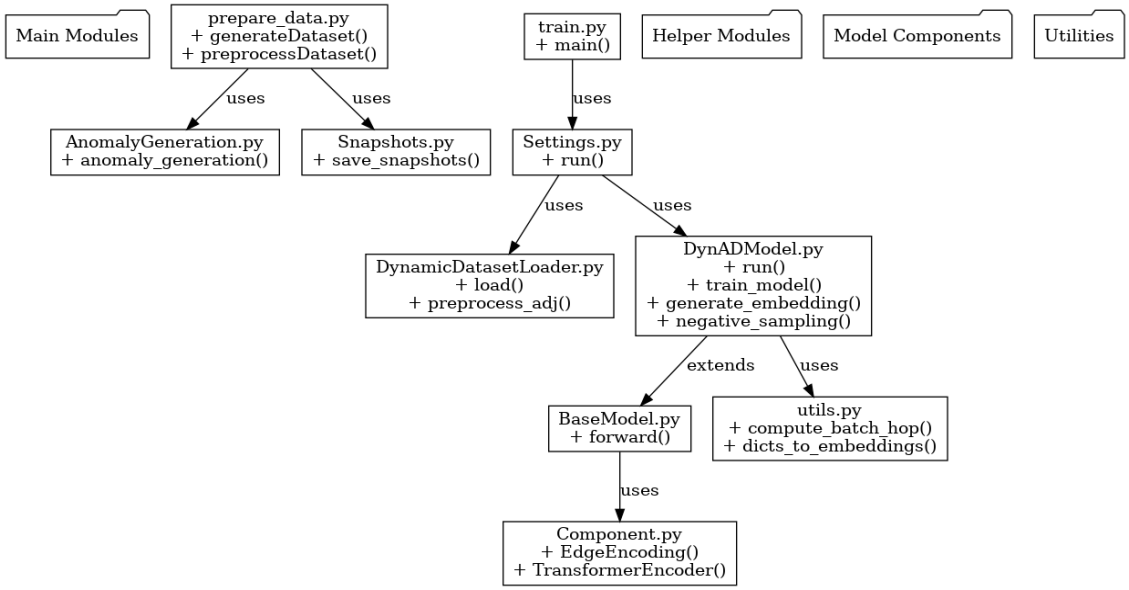

Fig. 8 . . Architecture of a software system. It illustrates the relationships and dependencies between main modules. Each module includes functions or methods, with arrows indicating usage or extension connections. encoding, dynamic graph transformer, and discriminative anomaly detector

4.2.1 Data preparation

The process begins with downloading and extracting relevant information from the PubMed dataset. Article names, publication dates, and keywords like MeSH (Medical Subject Headings) terms are used to categorize the articles, enabling the tracking of anomalous behavior during the execution of the model. The dataset is then adjusted into a format compatible with the TADDY framework, ensuring that it meets the requirements for dynamic graph analysis. This preprocessing phase is a one-time operation, with the resulting data stored in a file for subsequent use. Then preprocessing the raw data, introducing controlled anomalies, and splitting the dataset for training purposes (including snapshots generating).

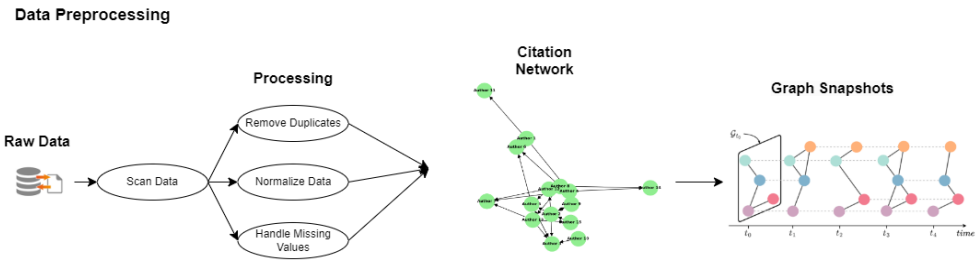

Fig. 9 The preprocessing of the dataset, handle it from the raw data to the suitable data input to TADDY model (original illustration, the last photo source already mention above).

The main steps of the data preparation process involve:

- **generateDataset (from prepare_data.py):**
  The purpose of this function is to load, clean, and preprocess the citation data from the specified raw dataset (PubMed). **Methods**:
  - **Loading the Data**: The citation data from PubMed is loaded from the file processed_pubmed.txt located in the data/raw/ directory. Only the first two columns (representing citation relations between articles) are kept and loaded as a NumPy array.
  - **Reordering and data representation:** The edges (citation relationships) are reordered to ensure the smaller article ID comes first. This prevents any issues with incorrect citation direction.
  - **Removing Self-Loops:** Self-loops are removed. These are edges where an article cites itself. Any such edges (where the source and target article IDs are the same) are filtered out.
  - **Removing Duplicates:** Duplicates are removed using np.unique. This ensures that each citation relationship between articles is unique, even if two articles cite each other multiple times.
  - **Re-indexing Vertices:** Each article ID is re-indexed to have a

consistent set of vertex IDs. This is done using np.unique to map each unique article ID to a new index.
    - **Saving the Processed Data:** The cleaned and processed data is saved in the data/interim/ directory for future use.
- **PreprocessDataset (from prepare_data.py):**
  After preprocessing the dataset, this function generates a synthetic dataset with anomalies. The goal is to add anomalies into the citation network and create training and test datasets. **Methods**:
    - **anomaly generation (from AnomalyGeneration.py):** Introduces synthetic anomalies into the citation network by identifying clusters of articles and generating edges between different clusters to simulate anomalous behavior.
    - **split_data (from AnomalyGeneration.py):** Splits the raw dataset into training and testing sets based on a specified percentage, ensuring compatibility with the model's requirements.
    - **save_snapshots (from Snapshots.py):** Generates and saves visual snapshots of the graph at different stages **[see, Fig.11]**, illustrating how the citation network and anomalies evolve over time.
    - **edgeList2Adj :** Converts the edge list into an adjacency matrix, representing citation relationships between articles.
    - **spectral_clustering:** Applies spectral clustering to the adjacency matrix to partition articles into clusters, identifying potential anomalies through connections between distinct clusters.
    - **generate_fake_anomalies:** Generates synthetic anomalous edges by selecting pairs of articles from different clusters and creating artificial connections.
    - **processEdges:** Refines generated anomalous edges by removing self-loops and duplicate edges, ensuring valid and realistic anomalies.
    - **generate_anomalous_test_set:** Creates a synthetic test set by replacing normal edges with generated anomalies and assigning labels (normal or anomalous).
    - **create_train_network :** Converts the training edges into a sparse adjacency matrix for efficient memory usage, representing citation relationships for model training.
    - **synthetic_test_set:** Returns the synthetic test set containing both normal and anomalous edges for model evaluation.

Citing Article ID	CITED ARTICLE ID	IMPORTANCE NUMBER	TIMESTAMP (EDGE)
6092	1234	1	157766402
7636	5678	1	157766403
14442	91011	1	157766404

Fig. 10 Illustration of Three Rows from the Raw Data That Was Generated from the PubMed Dataset, to Fit the Model Requirements.

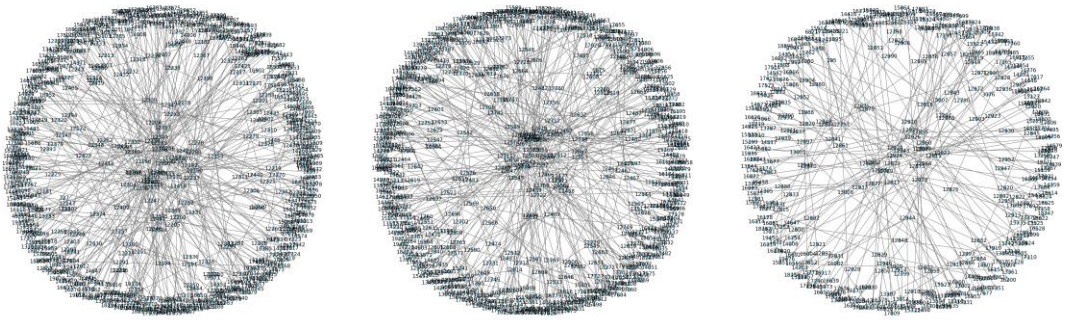

Fig. 11 Three distinct states of the citation graph within a single snapshot, showcasing its evolving structure.

4.2.2 Model Training

The training process focuses on detecting anomalies in dynamic graphs using a custom **PyTorch** framework. It begins by configuring datasets, model parameters, and other training settings through a set of command-line arguments. The **DynamicDatasetLoader.py** prepares the data, and the **DynADModel**.py defines the architecture. Finally, the Settings class orchestrates training and evaluation.

The main steps of the Model Training involve:

- **Dynamic Dataset Loader (DynamicDatasetLoader.py)**
  - **Load_hop_wl_batch:** Loads precomputed dictionaries containing hop distances, Weisfeiler-Lehman graph kernel features [17], and batched subgraph data.
  - **Normalize and normalize_adj:** Perform row and symmetric normalization of sparse matrices to improve convergence and stability in graph neural networks, ensuring balanced node connections.
  - **Preprocess_adj:** Symmetrically normalizes adjacency matrices, adds

self-loops, and converts them to PyTorch tensors.

- **Get_adjs:** Processes graph edges to compute adjacency matrices and eigenvalues for reuse.

- **Transformer-based Graph Encoding (Component.py)**
  - **EdgeEncoding**: This class generates embeddings for graph edges based on different features (such as initial positions, hop distances, and time distances). It uses embedding layers for these features, applies layer normalization, and adds dropout to regularize the model.
  - **TransformerEncoder**: Processes hidden states across multiple layers with self-attention mechanisms.
  - **TransformerLayer**: This class defines the core transformer layer that includes self-attention and optionally cross-attention mechanisms. It also processes the input through intermediate and output layers.
  - **MyConfig**: Defines model parameters like attention heads and dropout rates, etc.
- **Dynamic Anomaly Detection Model (DynADModel.py)**
  - **Initialization**: Configures learning rate, weight decay, and model parameters like snapshot embeddings and anomaly classification flayers.
  - **Forward**: Processes graph features, and the output sequences are aggregated over several snapshots. The model passes the aggregated output through the classification layer to generate the final predictions.
  - **Batch Processing**: Splits data into manageable batches for efficient training.
  - **Evaluate**: Computes AUC scores for anomaly detection.
  - **Generate_embedding**: Creates graph embeddings for nodes and edges using hop distance features. These embeddings are used to represent nodes and edges in the dynamic graph.
  - **Negative Sampling**: Generates negative samples (fake edges) by randomly perturbing the edges in the graph.
  - **Visualization**: Methods like plot_loss_curve and plot_roc_curve generate training loss and performance visualizations.
  - **Train_model**: Manages the training process, computes loss using binary cross-entropy, and updates weights.
  - **Anomalous Edge Detection**: Identifies and saves edges classified as anomalous based on a dynamic threshold.
- **Base Model (BaseModel.py)**

  Forward: Processes inputs through embeddings, transformer layers, and pooling to generate outputs.

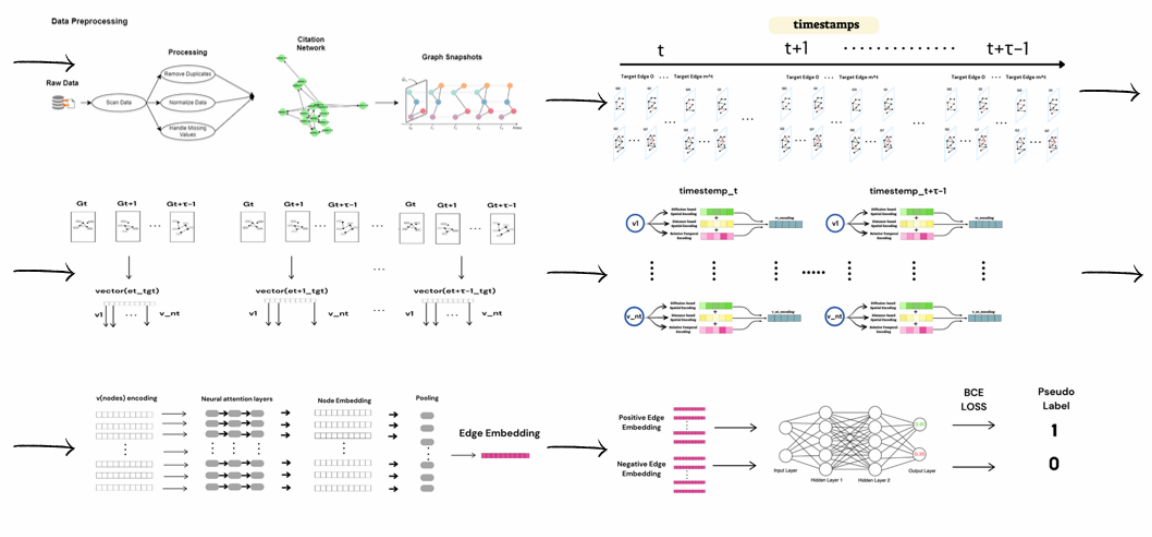

Fig. 12. Illustrates the entire process described above. It encompasses data preprocessing, dynamic graph generation, snapshot division, method extraction for temporal and structural node/edge information, insertion into neural network-based transformers, pseudo-negative edge generation, model training (including loss function application), and finally, the output of predictions (anomaly edges).

5. Challenges And Solutions

- **Challenge**: The dataset was incomplete and lacked critical information for analysis.
  Solution: To address this, multiple APIs were utilized to fetch missing pieces of data. This iterative process ensured that the dataset was enriched and completed. The finalized dataset was then saved as a raw file for further use.
- **Challenge**: Determining the optimal hyperparameters for the model to improve performance.
  Solution: Systematic tuning of hyperparameters was performed through grid search and experimentation. This ensured the model achieved its best possible performance.
- **Challenge**: Managing outdated or incompatible libraries in the codebase.
  Solution: Libraries were updated or adjusted to ensure compatibility with newer versions. Additionally, changes were documented, and a revised requirements.txt file was created to streamline future installations.
- **Challenge**: Identifying anomalous articles from the anomalous edges detected by the algorithm.
  Solution: A comprehensive logic was developed to analyze multiple conditions for each article. By examining the evolving relationships of

articles, the logic distinguished the true anomalous articles, providing deeper insights into the dataset.

- **Challenge**: Creating visual representations of snapshots and citation networks.
  Solution: After researching and exploring various libraries, a suitable toolset was identified and customized to meet the dataset's needs. The data was adjusted for compatibility with these libraries, enabling effective and insightful visualizations.

6. Testing And Evaluation

Below are the tests subjects of our model.

Test ID	DESCRIPTION	FUNCTION TESTED	EXPECTED RESULT	PASSED TESTING
1.1	Test dataset loading	PreprocessDataset	Successfully loads the dataset and creates the interim file.	✓
1.2	Test handling when the raw file is not found	PreprocessDataset	Raises an exception stating that the raw file was not found.	✓
1.3	Structure verification for generating dataset	GenerateDataset	Successfully verifies the structure of edges, anomalies, and training data.	✓
2.1	Test dataset loading functionality	Dynamic DatasetLoader.load	Verifies that the dataset is loaded correctly with attributes snap_train and snap_test.	✓
2.2	Row normalization of a sparse matrix	DynamicDatasetLoader.normalize	Successfully normalizes rows in the sparse matrix.	✓
2.3	Symmetrical normalization of adjacency matrix	DynamicDatasetLoader.normalize_adj	Returns a correctly normalized adjacency matrix.	✓
2.4	Conversion of sparse matrix to PyTorch sparse tensor	DynamicDatasetLoader.sparse_mx_to_torch_sparse_tensor	Converts sparse matrix to PyTorch sparse tensor with correct size and values.	✓

Test ID	DESCRIPTION	FUNCTION TESTED	EXPECTED RESULT	PASSED TESTING
3.1	Evaluation of model using AUC for multiple snapshots	DynADModel.evaluate	Returns correct AUC for individual snapshots and combined data, with verified anomalous edge saving.	✓
3.2	Saving edge embeddings for snapshots	DynADModel.save_edge_embeddings	Saves edge embeddings for all snapshots and verifies correctness.	✓
3.3	Model initialization verification	DynADModel constructor	Verifies that model configuration is correctly set (e.g., hidden_size, attention_heads).	✓
3.4	Training hyperparameters verification	DynADModel training parameters	Ensures correct setup for hyperparameters like max_epoch, lr, and weight_decay.	✓
3.5	Negative sampling to generate edges	DynADModel.negative_sampling	Successfully generates negative samples that are not part of the original positive edges.	✓

The evaluation of the TADDY framework emphasizes its capacity to identify unusual connections within dynamic graphs. The primary evaluation metric employed is the ROC-AUC **[see, Fig.14]** score, which serves as an effective measure of the framework's ability to differentiate between normal and anomalous edges. Verification of TADDY's performance is conducted by assessing its accuracy in identifying the anomalies that were intentionally introduced. This step not only quantifies the efficacy of the method but also aligns its outcomes with the insights provided by domain experts.

7. Results

All results presented in this section were obtained using the following initial parameter settings:

- **Dataset**: The dataset used is *PubMed*, a medical research dataset primarily focused on diabetes-related research.
- **Anomaly Percentage**: The anomaly percentage was set to **0.01**, meaning that 1% of the data was designated as anomalous.
- **Training Percentage**: The percentage of data used for training the model was set to **70%** to fit the size of the dataset.
- **Neighbor Number**: The number of neighbors considered for each node in the graph was set to **5**.

- **Window Size**: The window size for the sliding window mechanism used during graph construction was set to **4**.
- **Embedding Dimension**: The dimensionality of the node embeddings was set to **64**.
- **Number of Hidden Layers**: The model architecture used **6 hidden layers**.
- **Number of Attention Heads**: The number of attention heads used in the attention mechanism was set to **2**.
- **Maximum Epochs**: The model was trained for a maximum of **120 epochs .**
- **Learning Rate**: The learning rate for the optimizer was set to **0.00005**.
- **Weight Decay**: The weight decay used in the regularization of the model was set to **5e-4**.
- **Seed**: The random seed for initialization was set to **1** to ensure reproducibility of results.

Dataset Information

The dataset used in this project was downloaded on 15.11.24 and spans the period from 1945 to 1977. The key statistics of the dataset are as follows:

- **Nodes (papers)**: 19,717
- **Edges (citations)**: 44,338

This dataset primarily focuses on diabetes-related medical research, and it is commonly used as a benchmark in machine learning and graph-related studies. It has been widely utilized for tasks such as node classification, link prediction, and graph representation learning. It is crucial to note the dataset's asymmetry, with a significantly higher number of papers published in 1977 followed by 1976 and then 1975.

Now, let us see the research findings:

The loss plot demonstrates a substantial reduction in training loss over 120 epochs, indicating effective learning by the model. This suggests that the model successfully captures the complex behavior of the citation network, efficiently learning intricate features of its underlying structure and dynamic relationships.

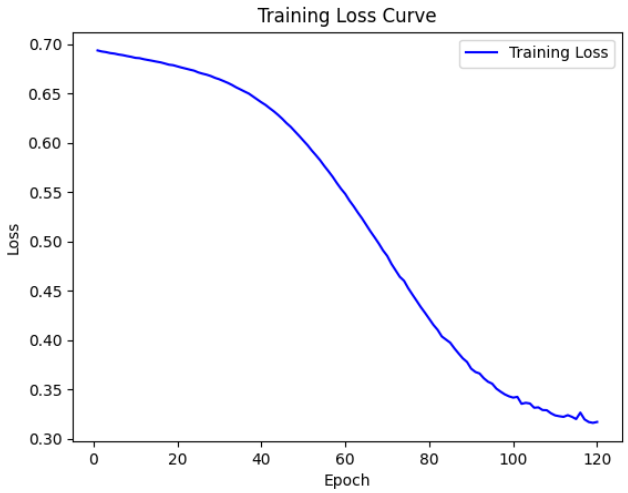

Fig. 13. The training loss curve over 120 epochs.

The AUC plot demonstrates high predictive performance, with the model achieving an AUC score of approximately 0.75. This indicates a strong ability to distinguish between anomalous and normal edges in the citation network. The high AUC suggests that the model can reliably identify anomalous edges with high probability, enabling effective extraction of these potentially interesting or unusual research findings.

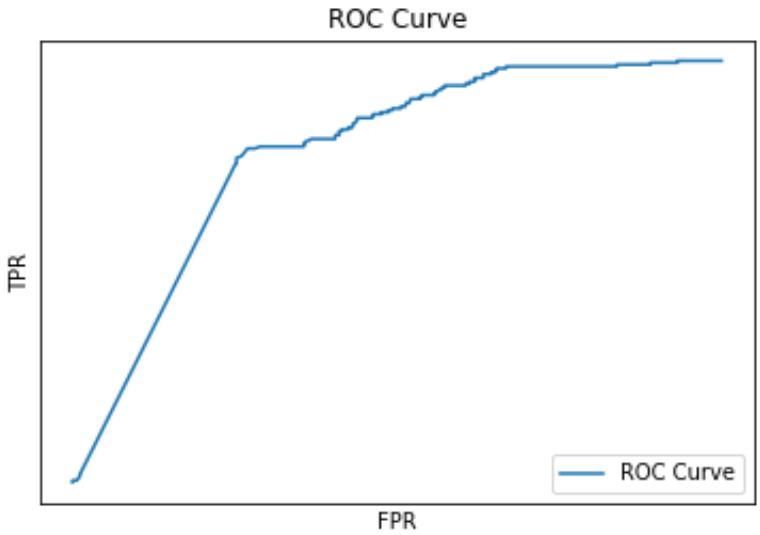

Fig. 14. ROC curve shows the model's performance in correctly identifying positives (TPR) versus the rate of false positives (FPR). A curve closer to the top-left corner indicates higher accuracy.

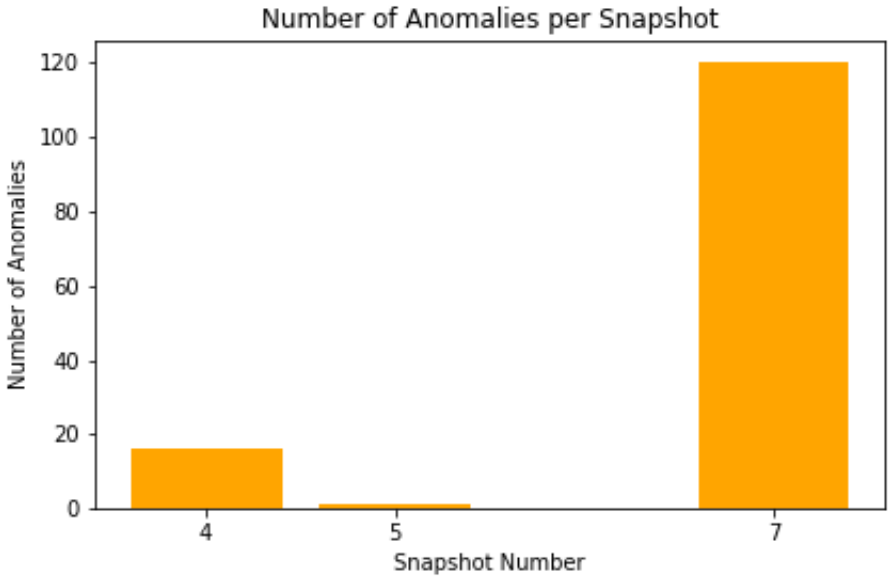

Fig. 15. Illustrates the distribution of anomalous edges detected by the model across different snapshots. Notably, snapshot number 7 exhibits the highest concentration of anomalies, which can be attributed to the significantly larger number of articles published in that particular year, as reflected in the dataset.

In line with the research objectives, the detected anomalous edges are stored for subsequent analysis. This enables researchers to track and investigate these anomalous articles, potentially uncovering novel research directions or identifying areas of significant change or disruption within the citation network. Now let's delve into some of the key findings from the analysis:

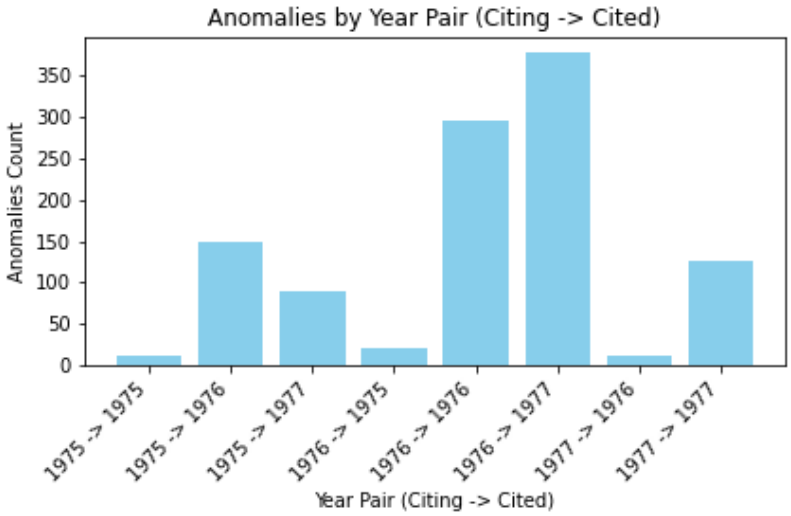

Fig. 16. This bar chart shows the distribution of anomalies based on year pairs, representing the citation relationships (Citing -> Cited). A notable concentration of anomalies occurs in the 1977 -> 1977 pair, highlighting a specific trend in the data (we can explain this by the amount of research from this year against the others).

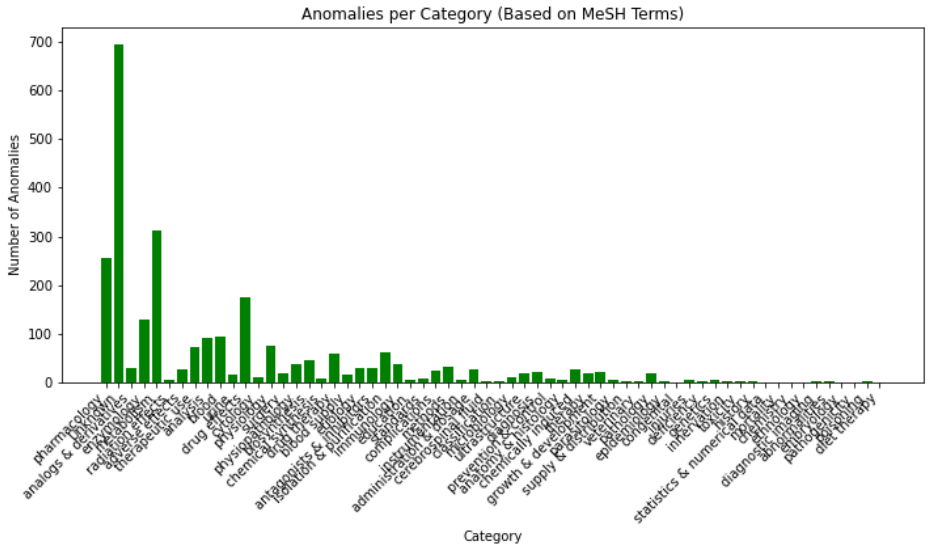

Fig. 17. Illustrates the distribution of anomalies across various categories based on MeSH terms. As we can see a lot of articles MeSH terms was unknown because of lake in the dataset.

A user-friendly GUI was developed to facilitate the tracking of anomalous articles identified by the model. The interface allows users to select an anomalous article from a list, where each entry includes a unique identifier and its name. Once selected, a time series chart is dynamically generated below, displaying the anomaly status (0 = Normal, 1 = Anomalous) of the article over time. This visual representation helps users to understand the temporal evolution of the article's anomaly status, revealing when and how anomalies occurred.

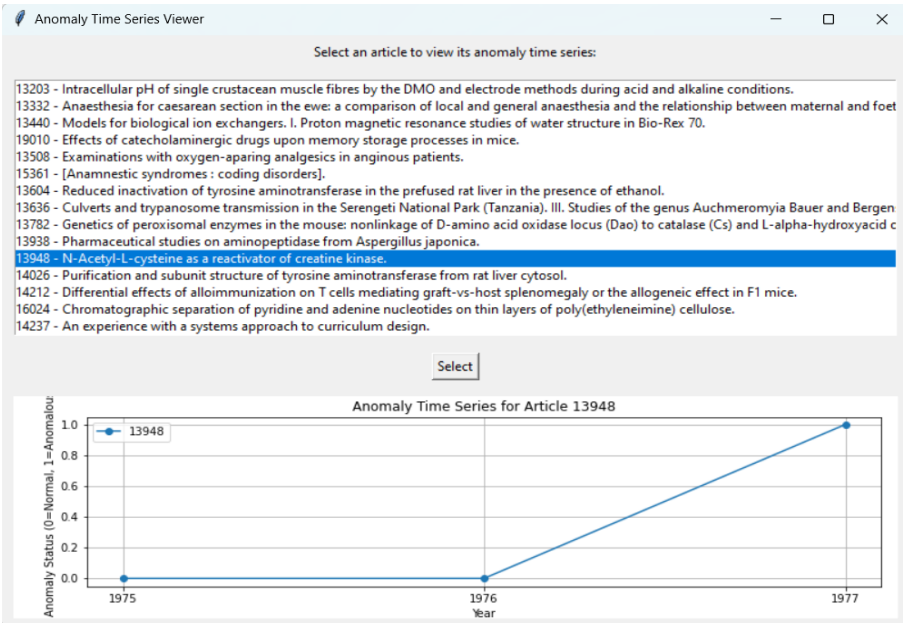

Fig. 18. Illustrates selection of anomalous article. We can see during 1975 – 1976 he was normal and detect as anomalous in 1977

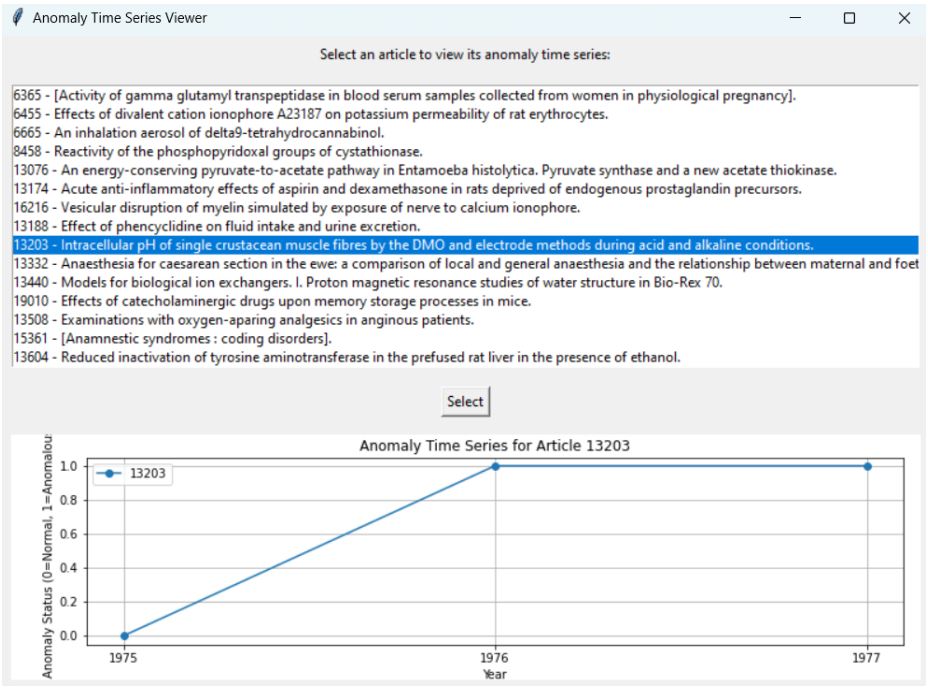

Fig. 19. Illustrates selection of anomalous article. We can see during 1975 he was normal and detect as anomalous in 1976 – 1977.

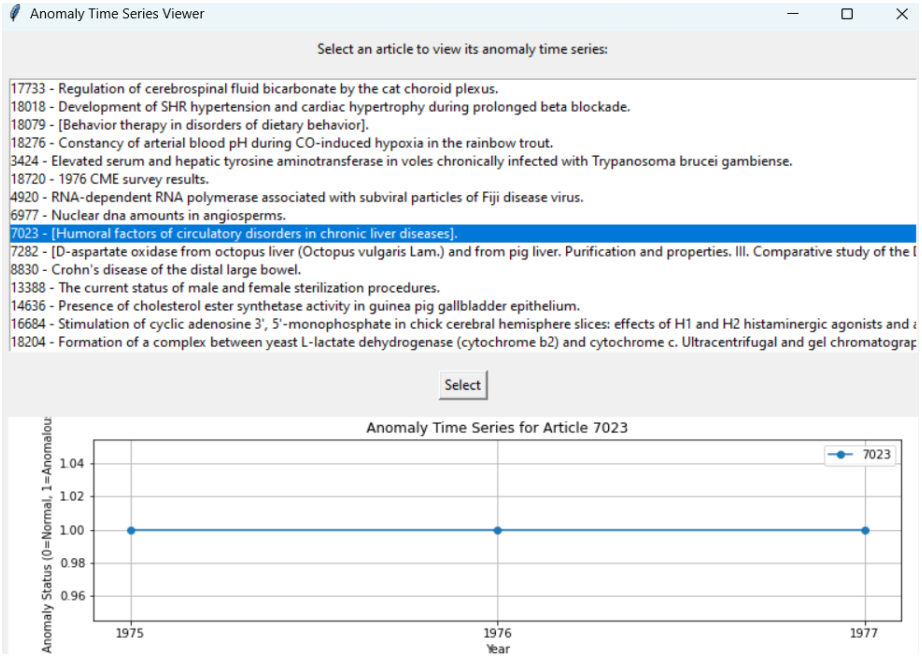

Fig. 20. Illustrates selection of anomalous article. We can see he was detect as anomalous in all the dataset years 1975-1977.

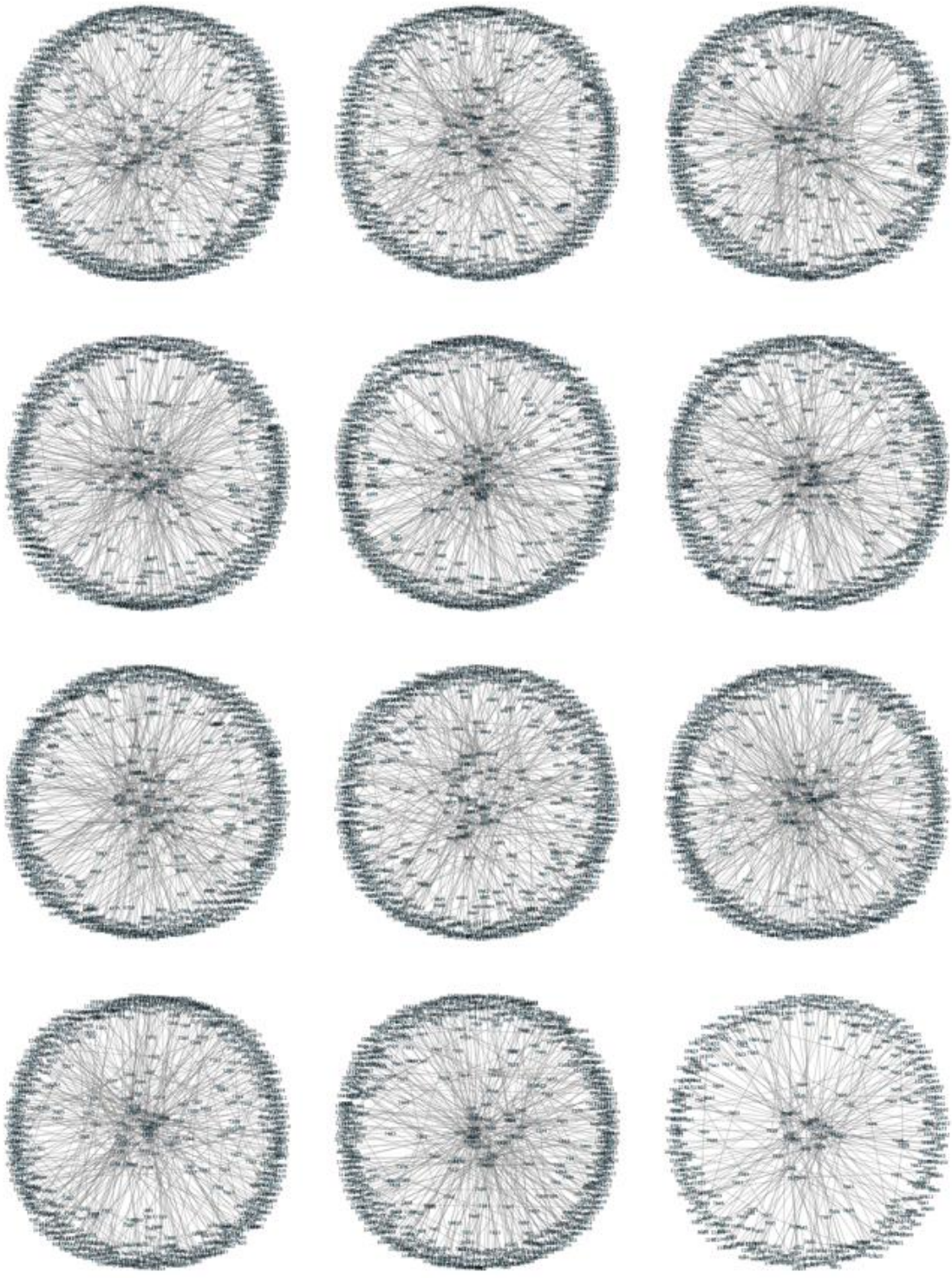

Fig. 21. This image presents a series of network graphs, each depicting a snapshot of a dynamic system over time. The graphs illustrate the evolution of the network's structure, including nodes (articles) and edges (citations) that represent entities and their interactions. From the results we can understand that centered nodes are most mentioned and, in the edges, usually the less used articles.

The left image provides a comprehensive view of the citation network within a given snapshot, illustrating the intricate web of connections between articles. The right image offers a refined perspective, where the model's learning process has been applied. In this image, anomalous edges, identified as deviations from expected patterns, are visually marked in red, enabling researchers to readily identify and investigate these potentially significant connections.

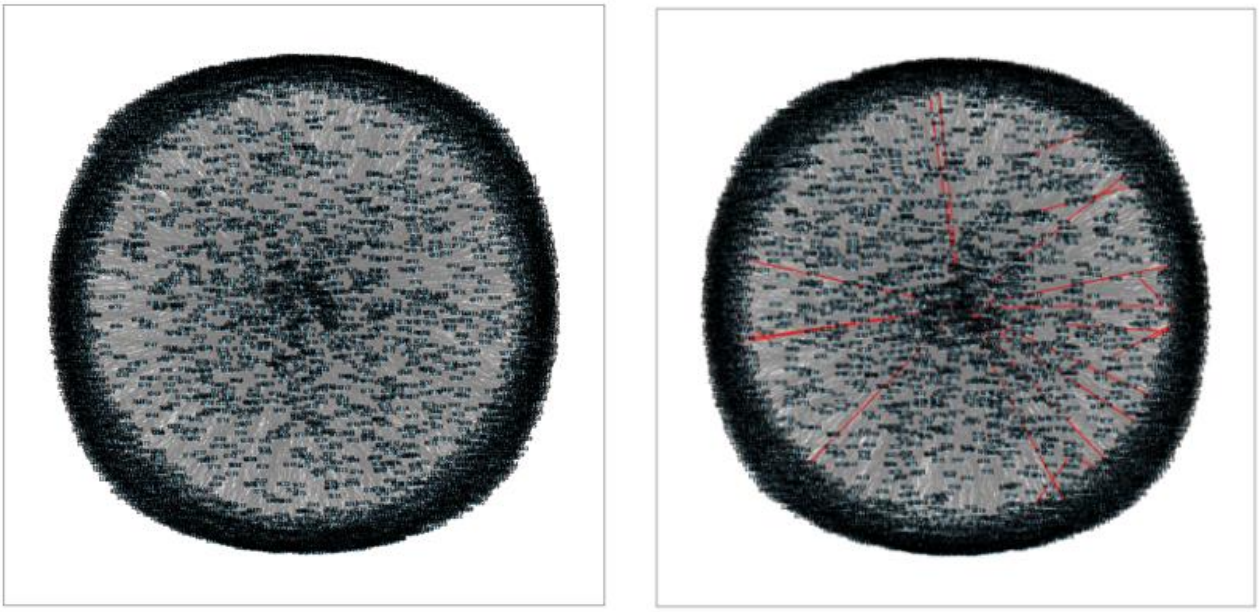

Fig. 22. Show the snapshots graph structure built from citations and articles, the red edges in the right shows the same snapshot with the anomalous edges.

8. Tools Used

- **Deep Learning Frameworks : PyTorch** , Simplified model development and training with cleaner, modular code.
- **Graph Processing: Torch Geometric**, Used to handle graph-structured data efficiently for citation network analysis.
- **Visualization Tools: Matplotlib** - Visualized snapshots, citation networks, and anomalies for analysis and presentation. **Image** – For saving and showing plots.
- **Data Processing: NumPy**: **Processed** and prepared data for model training and evaluation. **Pickle:** Streamlined storage and retrieval of serialized data.
- **Additional Utilities: APIs for Data Enrichment**: Filled gaps in the dataset by fetching missing metadata to ensure completeness. **NetworkX**: Assisted in constructing and visualizing the citation networks for anomaly detection.

- **Transformers**: **Hugging Face Transformers:** Utilized for integrating text-based data with graph models, enriching anomaly detection capabilities.
- **Interactive Elements**: **ipywidgets**: Developed interactive notebooks for exploring citation networks and visualizing anomaly evolution dynamically.

9. User's Guide Operating Instructions

The following guide provides a comprehensive overview of the system's operation and installation. Running the project in different system descriptions may lead to different results or even prevent it from working at all.

9.1 Run the project locally

- **System Requirements:**
  Operating System: Windows 11
  Python Version: Python 3
  Disk Space: 1TB free space
  RAM: 32 GB
  GPU: Nvidia RTX 3050 6 GB ram
  CPU: 13th Gen Intel(R) Core(TM) i7-13650HX 2.60 GHz
  IDE Editor: Spyder.
  Git – Version Control System.

Important Note:
The project folder also contains a subfolder called Dataset_Preprocessing. This folder includes several files with a README that explains how to use them. **However, you don't need to use these files** because we have already downloaded the dataset locally. All algorithms in this folder are designed to handle the download and preprocessing work automatically, which helps speed up the model. The raw data is already downloaded and ready for insertion into the model.

Steps to Run the Project: Project's Git Repository

1. Clone the Git repository and download the project folder.
2. Install the dependencies by running the **requirements.txt** file in your Python environment (Spider).

3. Run the **prepare_data.py** script to prepare the data, start preprocessing, and generate anomalies & construct the graph before running the model.
4. Run the **train.py** script to start the model training (including evaluation).
5. Run the **create_and_save_snapshots.py** script to save visual graphs of the snapshots into the snapshots/combined/ folder.
6. Run the **visual.py** script to extract anomalous articles from the anomalous edges and generate visual PNG files in the results folder.
7. Run the **visualArticles.py** script to open a GUI **[see, Fig.19]** for tracking anomalous articles found during the dataset timeline. You can select articles from the list and plot their behavior (anomalous or normal) over time.

You can also view some of the plots in the results section in the results folder.

9.2 Run the project in Colab Notebook

Important Note:

To run the project in Google Colab, it is needed to ensure that the system has sufficient CPU power to run the model effectively. The minimum required CPU/GPU RAM is about 100 GB. If you choose a different option, the code may not work as expected or yield different results.

There is a step-by-step guide within the Colab notebook, which will help you through the process. To run the project, simply open the notebook and follow the instructions inside.

The code in the Colab we show the last results. You can explore them without running the project.

To run the project in Colab Press HERE.

10. Developer Guide

This program operates based on the requirements outlined in the libraries file. As technology constantly evolves, many of the libraries utilized within this project are subject to change.

To maintain the project's functionality and longevity:

- **Customize your IDE or cloud environment:** Ensure your development environment aligns with the project's specific library and dependency requirements.
- **Keep the code updated:** Regularly review and update the code to incorporate the latest library versions and adapt to evolving technologies.

By adhering to these guidelines, you can ensure the ongoing viability and success of this project.

11 References

[1] Nur Aiza, W. S., Shuib, L., Idris, N., & Normadhi, N. B. A. (2023). Features, techniques and evaluation in predicting articles' citations: A review from years 2010–2023 Scientometrics. Akadémiai Kiadó, Budapest, Hungary.

[2] Bai, X., Xia, F., Lee, I., Zhang, J., & Ning, Z. (2016). Identifying anomalous citations for objective evaluation of scholarly article impact. *PLOS ONE, 11*(9), e0162364.

[3] Fong EA, Wilhite AW (2017) Authorship and citation manipulation in academic research. PLoS ONE 12(12): e0187394. https://doi.org/10.1371/journal.pone.0187394

[4] Kojaku, S., Livan, G., & Masuda, N. (2021). Detecting anomalous citation groups in journal networks. Scientific Reports, 11(1), 14524.

[5] Zhang, G., Ding, Y., & Milojević, S. (2012). Citation content analysis (CCA): A framework for syntactic and semantic analysis of citation content. arXiv:1211.6321 [cs.DL]

[6] Joshi, P.B., Pandey, M. (2024). Deception Through Manipulated Citations and References as a Growing Problem in Scientific Publishing. In: Joshi, P.B., Churi, P.P., Pandey, M. (eds) Scientific Publishing Ecosystem. Springer, Singapore. https://doi.org/10.1007/978-981-97-4060-4_17

[7] Petch, J., Di, S., & Nelson, W. (2022). Opening the black box: The promise and limitations of explainable machine learning in cardiology. *Journal Name*.

[8] *C. C. Aggarwal, Y. Zhao, and S. Y. Philip, "Outlier detection in graph streams," in ICDE. IEEE, 2011, pp. 399–409.*

[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 SIGKDD, 2018, pp. 2672–2681.

[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, pp. 4419–4425.

[11] Zhang, S., Tong, H., Xu, J. *et al.* Graph convolutional networks: a comprehensive review. *Comput Soc Netw* **6**, 11 (2019). https://doi.org/10.1186/s40649-019-0069-y

[12] Dey, R., & Salem, F. M. (2017). Gate-variants of gated recurrent unit (GRU) neural networks. Circuits, Systems, and Neural Networks (CSANN) LAB, Michigan State University.

[13] Y. Liu et al., "Anomaly Detection in Dynamic Graphs via Transformer," in IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 12, pp. 12081-12094, 1 Dec. 2023, doi: 10.1109/TKDE.2021.3124061.

[14] Y. Liu, Z. Li, S. Pan, C. Gong, C. Zhou, and G. Karypis, "Anomaly detection on attributed networks via contrastive self-supervised learning," IEEE TNNLS, 2021.

[15] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in SIGKDD, 2014, pp. 701–710.

[16] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in SIGKDD, 2016, pp. 855–864.

[17] Schulz, T. H., Horváth, T., Welke, P., and Wrobel, S., "A generalized Weisfeiler-Lehman graph kernel," arXiv preprint arXiv:2101.08104, 2021.