Software Engineering Department

Braude College of Engineering

Capstone Project Phase A

25-2-R-1

Lorentzian Anomaly Attention (LAA):

A Self-Attentive Approach to Citation Networks

Adar Budomski

Shoval Ben Shushan

Supervisor: Dr. Dvora Toledano Kitai

https://github.com/adar688/Lorentzian-Anomaly-Attention.git

Abstract **3**
1. Introduction **4**
2. Theoretical background **5**
2.1 Graph Networks 5
2.1.1 Static Networks 6
2.1.2 Dynamic Networks 6
2.1.3 Citation Networks 8
2.2 Graph Embedding 8
2.2.1 Embedding in Static Networks: 8
2.2.1.1 Matrix factorization-based methods: 8
2.2.1.2 Random walk-based methods: 9
2.2.1.3 Machine learning and deep learning-based embedding methods: 10
2.2.2 Embedding in Dynamic Networks: 10
2.2.2.1 Snapshot-based Embedding Methods: 10
2.2.2.2 Deep Learning and Self-Attention–Based Methods: 11
2.2.3 Hyperbolic Space and Dynamic Networks 11
2.2.4 Embedding in Hyperbolic Space 12
2.2.4.1 Hyperbolic Temporal Graph Networks (HTGN) 13
2.2.4.2 Dynamic Hyperbolic Graph Attention Network (DHGAT) 13
2.2.4.3 DynHAT – Dynamic Hyperbolic Attention Network 14
3. Preliminaries **14**
3.1 The Lorentz Model of Hyperbolic Space 14
3.1.2 The Exponential Map 16
3.1.3 The Logarithmic Map 17
3.2 The DynHAT Model in Lorentz Space 18
3.2.1 Preprocessing Stage 18
3.2.2 A. Structural Information Representation Layer 18
3.2.2.1 Projection into Lorentzian Space via Exponential Mapping: 18
3.2.2.2 Linear Transformation in Lorentzian Space: 19
3.2.2.3 Attention Score Computation: 19
3.2.2.4 Hyperbolic Aggregation of Neighbor Features: 20
3.2.3 B. Temporal Information Representation Layer 20
3.2.3.1 Temporal Input Preparation: 20
3.2.3.2 Projection into Tangent Space: 20
3.2.3.3 Temporal Attention with Masking: 21
3.2.3.4 Projection Back to Lorentzian Space: 21
3.2.4 C. DynHAT Model Optimization 21
3.3 Anomaly detection 22
3.3.1 Isolation Forest 23
3.3.2 Local Outlier Factor 23
4. Approach **24**
4.1 Proposed Model: Anomaly Detection in Dynamic Academic Citation Networks 24
4.1.1 Stage 0: Preprocessing and Input Representation 24

4.1.2 Stage 1: Lorentzian Structural Representation via Hyperbolic Self-Attention. 25
4.1.3 Stage 2: Temporal Representation via Self-Attention.....27
4.1.4 Stage 3: Anomaly Detection Based on Temporal Node Behavior..... 28
4.1.5 Stage 4: Validation Based on Noise Injection via Fake Nodes.....33
5. Tools and Technological Framework..... 37
5.1 Python..... 37
5.2 Visual Studio Code.....37
5.3 Scikit-Learn..... 37
5.4 NetworkX.....38
5.5 Google Colab..... 38
6. Challenges..... 38
6.1 Challenges Encountered During Phase A.....38
6.1.1 Familiarization with Hyperbolic Geometry..... 38
6.1.2 Deepening Understanding of Advanced Deep Learning Models..... 39
6.1.3 Developing Academic Writing Skills in English.....39
6.2 Anticipated Challenges in Phase B..... 39
6.2.1 Identifying a Suitable Dataset.....39
6.2.2 Selecting an Appropriate Development and Execution Environment.....39
6.2.3 Integration of Model Components..... 39
7. Expected Outcomes.....40
References.....40

Abstract

In an era of growing research output, dynamic citation networks serve as a key tool for analyzing the evolution and influence of scientific publications over time. This project presents a model for anomaly detection in such networks, where nodes represent papers and edges denote time-varying citation relationships.
The model integrates hierarchical structural encoding via embedding in Lorentzian space with temporal analysis using a Self-Attention mechanism capturing both the local structure and its development over time. Anomaly scores are assigned to each node using Isolation Forest and Local Outlier Factor, based on deviations from expected structural and temporal patterns.
To assess the model's effectiveness and robustness an iterative controlled noise-injection process is introduced. In each iteration, artificial nodes with simulated citation behaviors are injected into the network, and the model's ability to accurately identify them as anomalies is evaluated.
This proposed approach aims to achieve high detection accuracy and robustness under noisy conditions and to uncover irregular citation patterns, such as sudden influence or atypical behavior, thereby contributing to a deeper understanding of dynamics in academic research.

1. Introduction

In the rapidly evolving landscape of academic research, citation networks serve as a central tool for analyzing the development, dissemination, and impact of scientific knowledge. In these networks, nodes represent academic papers and directed edges represent a citation between them. As new papers are published and cited, citation networks evolve dynamically over time, reflecting the shifting patterns of scholarly influence [7].

Detecting anomalies within these dynamic networks is essential for identifying atypical citation behaviors, such as breakthrough publications, short-term viral trends, or even potential instances of citation manipulation. Anomalies may signify emerging research fronts, irregular citation patterns, or other phenomena of interest to scientometric analysis [7].

The current project focuses on the detection of anomalies in dynamic academic citation networks. The core objective we aim to solve is the identification of papers whose citation behavior deviates significantly from the expected structural and temporal patterns. Such deviations may indicate significant discoveries, temporary influences, or other unusual dissemination patterns.

To this end, we propose a model that analyzes both the structural properties of the network at discrete time intervals and the temporal evolution of nodes across time. At each time step, the citation graph is encoded into a low-dimensional representation that preserves hierarchy in Lorentzian space, a variant of hyperbolic geometry suitable for modeling hierarchical and scale-free networks [4],[5]. Subsequently, a temporal encoding module based on a Self-Attention mechanism is applied to capture changes in node behavior over time [9],[13].

In the final stage, anomaly detection is performed based on the integrated structural and temporal embeddings of each node. For this purpose, two unsupervised statistical methods are used: Isolation Forest (IF) and Local Outlier Factor (LOF) [1]. These methods assign anomaly scores to each node at each time step, considering its deviation from the typical behavior in the network. Statistical analysis of the resulting anomaly score vectors (e.g., computing the mean and standard deviation per node) allows for the identification of entities with significant abnormality over time.

To assess the reliability and robustness of the proposed model, a perturbation-based validation procedure is introduced.

In this framework, synthetic noise is injected into the network by adding fake nodes that simulate papers with artificial citation patterns. These injected nodes are assigned simulated static features, as well as random connections to real papers simulating a typical citation behavior that violates the natural structure and dynamics of the citation graph.

The injection process is carried out iteratively. In each iteration, a new portion of fake nodes is injected and the updated network is processed through the entire model pipeline, including structural encoding, temporal encoding, and anomaly detection. Anomaly scores are then calculated for both real and synthetic nodes.

The model's performance is evaluated by measuring its ability to correctly identify the injected nodes (true positives) while minimizing false positives among real nodes. This evaluation provides insight into the model's stability and accuracy even under non-ideal conditions and highlights its robustness to noise and irregular input data.

Ultimately, the model is expected to identify diverse temporal anomaly patterns in citation networks and to generate detailed anomaly profiles for each paper, including statistical summaries such as the mean and standard deviation of anomaly scores. The controlled noise injection strategy will enable assessment of the model's accuracy and robustness, with a target detection rate exceeding 85% and a low false positive rate.

The remainder of this document is organized as follows. Section 2 presents the theoretical background, including an overview of graph-based representations, embedding methods, and the use of hyperbolic geometry in network analysis [6][10]. Section 3 introduces the Lorentzian model and details the architecture of the proposed DynHAT-based anomaly detection model [5]. Section 4 describes the complete anomaly detection approach, including data processing, structural and temporal encoding, anomaly detection, and the validation procedure. Section 5 outlines the tools and technologies that will be used. Section 6 discusses key challenges and limitations faced in phase A and those expected ahead in phase B. Finally, Section 7 summarizes the expected outcomes of the project.

2. Theoretical background

2.1 Graph Networks

Graph Networks are computational models based on mathematical graphs, structures consisting of nodes and edges that define relationships or interactions between entities. Nodes represent entities from various domains, while edges capture the relationships among them.

Graph networks are widely employed in tasks such as node classification, link prediction, and other network-based learning problems. They can be broadly categorized into two categories: static networks, where the structure remains unchanged over time, and dynamic networks, where nodes and edges may appear, disappear, or change as the network evolves.

Examples of Graph Networks include social networks (e.g., friendship connections on Facebook or follow relationships on Twitter), biological networks (e.g., protein interactions or gene regulation), transportation networks (e.g., where routes and travel times vary throughout the day), communication networks (such as data traffic between servers or mobile devices) and citation networks (where each node is a scientific paper and edges represent citations).

Graph networks can also be classified based on their structural properties. These include directed vs. undirected graphs (depending on whether connections have a specific direction), weighted graphs (where edges are assigned numerical values), and attributed graphs (where nodes or edges contain additional feature information).

By leveraging graph networks, it is possible to analyze not only individual entities, but also the intricate relationships among them. This makes them particularly suitable for studying structured and networked data. Moreover, graph networks facilitate the identification of complex structural patterns that may be difficult to detect using traditional data representation.

2.1.1 Static Networks

Static networks represent relationships at a fixed point in time, operating under the assumption that the structure of nodes and edges remains unchanged throughout the analysis. They provide a simplified representation of systems where temporal variations are either negligible or not the primary focus of the study. Examples of static networks include transportation maps with fixed routes or molecular structures in chemistry.

These networks facilitate the application of classical graph theory and machine learning techniques without the need to address the complexity introduced by temporal dynamics. While static networks are generally easier to model and analyze, they may fail to capture the evolving nature of many real-world systems, potentially overlooking critical patterns in time-dependent data. Consequently, in domains where relationships frequently change, dynamic network models are often more suitable.

2.1.2 Dynamic Networks

Dynamic networks are networks in which both the nodes and their connections evolve over time. These changes may include the addition or removal of nodes, the strengthening or weakening of connections, or the emergence and dissolution of structural patterns within the network [10].

Unlike static networks, which capture relationships at a fixed point in time, dynamic networks require specialized methods to analyze and understand changes over time. The analysis of such networks involves tracking structural changes, identifying trends, and predicting the evolution of future connections [5].

Analyzing dynamic networks presents significant challenges due to the continuous evolution of their structure. As nodes and edges are added, removed, or modified over time, maintaining an accurate representation of the network becomes a complex task. Furthermore, different networks evolve at varying rates—some undergoing gradual transformations, while others experience rapid and unpredictable changes. This variability necessitates the development of flexible and robust modeling techniques that can adapt to diverse temporal dynamics. Another fundamental challenge is the high dimensionality and complexity of large-scale networks, where intricate hierarchical relationships must be efficiently captured without losing essential structural properties.

To address these challenges, advanced models have been developed to represent dynamic information effectively while preserving the fundamental characteristics of the network over time. Once these challenges are managed, dynamic network analysis can be applied to various tasks, including link prediction and pattern recognition. By examining the evolving structure of a network, it becomes possible to forecast future connections between nodes, a capability particularly valuable in citation prediction, social network analysis, and

recommendation systems. Additionally, dynamic network analysis facilitates the identification of structural patterns and deviations from expected behavior, enabling the detection of emerging research trends, evolving collaboration patterns, and anomalies such as unusual citation behaviors.

By leveraging sophisticated modeling approaches, it becomes feasible to address the inherent complexities of dynamic network analysis while unlocking meaningful insights that support predictive and exploratory tasks within evolving networked systems.

In this project, we focus on academic citation networks, where each node represents a scientific publication, and the edges a variety of systems such as citations between papers. Citation networks are inherently dynamic, as new publications are continually added and new links are formed when papers cite existing works. Thus, the structure of the network changes and evolves over time.

Furthermore, any network, whether static or dynamic, can be represented computationally using feature vectors. These vectors are used to represent the nodes, with each node described by a numerical vector that encoding various properties. This representation enables the application of machine learning techniques, statistical analysis, and data mining to the network.

The features in these vectors can generally be categorized into four main types:

- Structural features: For example, the number of incoming (in-degree) and outgoing (out-degree) citations, the centrality of a paper (i.e., its importance in the overall network structure), and more.

- Textual features: For example, the topic or field of the paper, as well as embeddings representation of its title, abstract, or full text.

- Temporal features: For example, the publication date, the rate at which the paper accumulates citations over time, or the number of citations received in a specific time period.

- Relational features: For example, the proximity to important nodes in the network, or characteristics of the papers it cites or that cite it.

With rich feature vectors, it is possible to train predictive models capable of forecasting which papers are likely to cite each other in the future, a key task in citation network analysis. Additionally, such models can identify emerging research trends by analyzing how feature values change over time (e.g., a sharp rise in citations or increased connectivity to new communities), and detect anomalies or unusual patterns, such as papers that receive an unusually high number of citations in a short period, or citations from unexpected sources.

2.1.3 Citation Networks

Citation networks provide a fundamental framework for analyzing the organization and progression of scientific knowledge. At their core, these networks depict the relationships between academic publications, with individual papers as nodes and citations forming the edges that connect them.

One of the remarkable characteristics of citation networks is their inherently hierarchical and uneven nature. Certain publications, often recognized as landmark contributions within their respective fields, accumulate a significant number of citations and act as central hubs within the network. In contrast, other papers occupy more peripheral positions, attracting limited attention.

Beyond the simplistic quantification of citations, citation networks uncover complex patterns in the dissemination and evolution of ideas. They facilitate the identification of interconnected clusters of papers, shedding light on the formation of scientific communities and their developmental trajectories over time. By tracing citation paths, researchers can discern the emergence of groundbreaking concepts and explore how distinct research domains intersect and influence one another.

Additionally, citation networks play a critical role in evaluating academic impact. While raw citation counts offer a basic measure of influence, network-oriented approaches provide a deeper and more refined understanding by considering the origin and context of citations. Advanced computational models leverage these nuances to rank papers, authors, and journals, offering insights that surpass traditional bibliometric measures [7].

2.2 Graph Embedding

Graph embeddings encompass a family of techniques that map nodes, edges, or entire graphs into low-dimensional vector spaces while preserving their structural and semantic properties. The primary objective is to convert complex and often sparse graph data into continuous representations that facilitate downstream machine learning tasks such as link prediction, node classification, clustering, and visualization. Effective graph embeddings aim to capture key topological patterns, including node proximity, community structure, and higher-order relationships, thereby enabling efficient and scalable analysis of large and complex networks.

2.2.1 Embedding in Static Networks:

2.2.1.1 Matrix factorization-based methods:

Matrix factorization-based embedding methods learn low-dimensional node representations by decomposing a predefined similarity or adjacency matrix. These methods aim to preserve structural relationships, such as proximity or connectivity, by capturing latent patterns embedded in the graph's global topology. An example of this is HOPE (High-Order

Proximity Preserved Embedding) [10], a method designed to preserve high-order proximity information in directed graphs, particularly in cases where relationships are asymmetric. While traditional embedding approaches often focus on local, symmetric structures, HOPE addresses this limitation by considering long-range dependencies and directional influences between nodes. It achieves this by constructing a similarity matrix that encodes high-order proximity and then applying a generalized singular value decomposition (SVD) to learn node embeddings that preserve these complex relationships. The resulting low-dimensional representations reflect not only direct connections but also indirect, transitive links across the network. This makes HOPE especially effective in domains like citation or social networks, where influence and information flow are inherently directional and extend beyond immediate neighbors.

2.2.1.2 Random walk-based methods:

Random walk-based embedding methods generate node representations by simulating stochastic walks over the graph to capture local and global structural contexts. These methods encode node similarity based on co-occurrence patterns observed in the sampled sequences, effectively preserving both community structure and functional roles. One such method is DeepWalk [2], an unsupervised learning technique that transforms nodes in a graph into low-dimensional vector embeddings. By simulating truncated random walks, it generates node sequences that resemble textual data, allowing language modeling techniques to be applied to learn structural features. This approach encodes both proximity-based relationships and broader topological patterns, offering a comprehensive representation of the graph's structure.

Additionally, the method Node2vec [10] extends the DeepWalk approach by introducing a more flexible random walk mechanism, enabling the exploration of diverse structural roles within a graph. By adjusting two key parameters, it balances local neighborhood exploration with broader structural traversal, allowing it to encode both community affiliations and functional similarities. The method produces compact vector representations of nodes that effectively retain their topological context, supporting various analytical tasks including node classification, link prediction, and clustering. LINE [10] is an embedding method designed to efficiently represent large-scale information networks by preserving both first-order and second-order proximity among nodes. The first-order component captures the immediate relationships between directly connected nodes, while the second-order aspect reflects the similarity of nodes based on their shared neighbors. This dual preservation allows LINE to encode both local and broader structural characteristics of the network. Its architecture is particularly well-suited for scalability, making it effective for embedding very large graphs, and its representations support tasks such as link prediction and clustering in complex networked data.

However, while such static embedding methods are computationally practical and useful for modeling simple static networks, they are often unsuitable for representing the evolving structure found in complex, real-world networks, particularly those exhibiting dynamic behaviors or hierarchical organization, as seen in citation graphs.

2.2.1.3 Machine learning and deep learning-based embedding methods:

Machine learning and deep learning-based embedding methods aim to project entities (such as words, documents, or graph nodes) into continuous vector spaces, capturing their semantic or structural characteristics. These representations facilitate efficient computation and improve performance across various downstream tasks such as classification, prediction, and clustering. An example of this is GAT (Graph Attention Networks) [9], a neural architecture designed to learn low-dimensional vector representations (embeddings) for nodes in a graph. Unlike traditional methods that rely on fixed aggregation rules, GAT utilizes a masked self-attention mechanism that relies on fixed aggregation rules, GAT uses a self-attention mechanism that allows each node to focus on its neighbors with varying degrees of importance. This approach enables the model to capture both fine-grained local patterns and broader structural properties of the graph, without requiring prior knowledge of the graph's global topology. GATs are particularly effective in inductive settings, where the model needs to generalize to unseen nodes or graphs, and have demonstrated strong performance across multiple benchmark datasets.

Additionally, GCN (Graph Convolutional Networks) [12] is a foundational model in the family of Graph Neural Networks, designed for representation learning over graph-structured data. The core idea behind GCNs is to iteratively aggregate and transform information from a node's local neighborhood to generate embeddings that capture both feature and structural information. Each layer applies a spectral convolution operation, effectively propagating information across connected nodes while preserving the graph's topology. This allows the model to capture higher-order dependencies in the graph as depth increases. GCNs have become a benchmark approach due to their simplicity and strong performance in semi-supervised tasks such as node classification and link prediction across various domains including citation networks, social graphs, and biological systems.

2.2.2 Embedding in Dynamic Networks:

2.2.2.1 Snapshot-based Embedding Methods:

Snapshot-based embedding techniques model dynamic networks by discretizing time into sequential intervals, where each interval corresponds to a static graph snapshot. Within each snapshot, traditional embedding methods, such as matrix factorization or random walks, are applied independently or with temporal regularization to ensure smooth transitions between time steps. These methods provide a basic temporal modeling framework and are capable of capturing short-term dynamics. Such snapshot-based methods, while more temporally aware, often struggle to capture the continuous and hierarchical evolution characteristic of complex real-world networks, such as citation graphs.

One notable example is DynamicTriad [10], a temporal network embedding method that captures both the structural and evolutionary characteristics of dynamic graphs. Unlike static or snapshot-based models, DynamicTriad explicitly models the triadic closure process, a common phenomenon in real-world networks where open triads (i.e., two nodes connected to a common neighbor) tend to evolve into closed triangles. By simulating this process, the model learns node embeddings that reflect not only the current structural

configuration but also its potential evolution over time. This results in representations that are temporally smooth and structurally informative, making the method suitable for dynamic link prediction and related tasks.

2.2.2.2 Deep Learning and Self-Attention–Based Methods:

Recent advances in dynamic network embedding increasingly rely on deep learning architectures, particularly models incorporating self-attention mechanisms. Approaches such as temporal graph attention networks are designed to capture both the evolving network structure and the dependencies between nodes over time. Through the use of self-attention across temporal and structural dimensions, these models aggregate node representations in a context-aware and flexible manner, dynamically adjusting the importance of neighboring nodes and historical states. This process generates high-quality, temporally-aware embeddings that improve performance in tasks such as node classification and link prediction in dynamic graphs.

One such model, TGAT (Temporal Graph Attention Network) [11], is specifically tailored for representation learning in continuous-time dynamic graphs. It utilizes a temporal self-attention mechanism to aggregate information from a node's temporal neighborhood, effectively capturing both structural and temporal relationships. By incorporating time encoding functions, TGAT encodes relative temporal information, enabling the attention mechanism to prioritize neighbor nodes based on both their topological closeness and temporal significance. This approach allows TGAT to handle asynchronous interactions and generate context-aware embeddings, making it highly suitable for tasks such as temporal node classification and dynamic link prediction.

Another influential approach is EvolveGCN [5], a dynamic graph embedding technique that integrates graph convolutional networks (GCNs) with recurrent neural networks (RNNs) to capture temporal changes in evolving graphs. Instead of directly updating node embeddings over time, EvolveGCN employs an RNN to dynamically adjust the parameters of the GCN. This innovative design enhances the model’s ability to adapt its structure-learning capabilities in response to changes within the graph.

2.2.3 Hyperbolic Space and Dynamic Networks

Hyperbolic space is a powerful mathematical framework for representing complex structures that exhibit rapid expansion. Unlike Euclidean space, where areas and volumes grow polynomially with distance, hyperbolic space exhibits exponential growth in these properties [2]. This characteristic makes hyperbolic space particularly well-suited for representing hierarchical data, such as tree-like structures and networks with scale-free properties [6], where the number of connections follows a power-law distribution.

One of the key advantages of hyperbolic geometry is its ability to naturally embed hierarchical structures. In such embeddings, highly connected (or central) nodes are positioned near the center of the space, while peripheral nodes, which have fewer connections, are distributed toward the outer regions. This property enables efficient representation and organization of large, complex networks, making hyperbolic space

highly relevant for applications in network analysis, recommender systems, and natural language processing [5].

The exponential growth property of hyperbolic space provides an ideal mathematical foundation for representing dynamic networks, particularly those with hierarchical structures [2]. Since hyperbolic embeddings naturally encode hierarchical relationships, they are highly effective for capturing the evolving nature of dynamic networks. Central nodes, which play a key role in the network's evolution, are positioned closer to the center, while less influential nodes are distributed toward the periphery, the structural integrity of the network even as it evolves [4].

Hyperbolic space is, therefore, a highly effective framework for dynamic network representation, offering compact embeddings, improved hierarchical organization, and the ability to capture rapid network evolution efficiently.

Hyperbolic representations provide significant advantages for the analysis of dynamic networks by offering a mathematically efficient way to model their hierarchical and evolving nature. One of the key benefits is the ability to generate compact embeddings, where large-scale networks can be represented in a low-dimensional space without compromising their hierarchical structure. This property is particularly useful for preserving essential relationships while reducing computational complexity.

Additionally, the hierarchical organization inherent in hyperbolic geometry naturally aligns with the multi-level structure of real-world networks, where central nodes maintain a dominant role while peripheral nodes are positioned accordingly. This feature allows for a more accurate and meaningful representation of dynamic systems. Another crucial advantage is scalability and efficiency—hyperbolic models can effectively capture rapid network evolution while maintaining computational feasibility, making them particularly well-suited for large and complex dynamic systems.

Moreover, hyperbolic embeddings enhance link prediction, as their structural organization facilitates the accurate forecasting of future connections. This capability is particularly valuable in applications such as citation prediction and anomaly detection, where identifying emerging links and deviations from expected patterns is essential.

In conclusion, hyperbolic space serves as a robust theoretical framework for dynamic network analysis, offering a well-suited mathematical foundation for modeling complex and evolving systems. By leveraging hyperbolic embeddings, researchers can develop more effective models to analyze network dynamics, predict future structural changes, and detect anomalies in large-scale networks.

2.2.4 Embedding in Hyperbolic Space

Several models have been proposed to exploit the properties of hyperbolic space for dynamic graphs, combining geometric priors with deep learning architectures to capture both structural and temporal dependencies. Below, we review prominent methods developed for this purpose, including HTGN, DHGAT, and DynHAT.

2.2.4.1 Hyperbolic Temporal Graph Networks (HTGN)

HTGN [5] is a model designed to embed dynamic networks in hyperbolic space, while preserving both the hierarchical structure and temporal dynamics of the network. The model is composed of three main components:

- Hyperbolic Graph Neural Network (HGNN): This component generates node embeddings based on structural connections within each snapshot of the dynamic graph. All computations are performed directly in hyperbolic space, allowing the preservation of hierarchical relationships.

- Hyperbolic Gated Recurrent Unit (HGRU): This component captures the temporal node representations, consistency between embeddings at successive time steps. The objective is to ensure that temporal changes reflect gradual and meaningful developments in the network.

- Hyperbolic Temporal Attention (HTA): This mechanism enables the model to selectively focus on significant moments in a node's temporal trajectory, rather than treating all historical states uniformly. As a result, the model can better capture key transitions and improve the quality of the learned representations.

Additionally, HTGN incorporates a mechanism called Hyperbolic Temporal Consistency (HTC), which promotes smooth and coherent transitions in node embeddings over time, ensuring they reflect realistic and interpretable dynamics in the evolving network.

2.2.4.2 Dynamic Hyperbolic Graph Attention Network (DHGAT)

DHGAT [5] extends the concept of hyperbolic embedding to dynamic graphs by integrating self-attention mechanisms across both structural and the temporal dimensions. The model aims to learn node representations that are simultaneously structurally hierarchical and temporally aware. DHGAT consists of two main components:

- Hyperbolic Structural Self-Attention (HSSA): This module calculates the influence of neighboring nodes at each time step using hyperbolic distance as the basis for attention weighting, thereby preserving the local hierarchical structure of each snapshot of the dynamic graph.

- Hyperbolic Temporal Self-Attention (HTSA): This component models the influence of past time steps on the current representation of each node, emphasizing temporal continuity. Like the structural attention mechanism, the temporal attention mechanism operates directly in hyperbolic space, ensuring consistent structural dynamics over time.

A unique feature of DHGAT is that all computations are performed directly in hyperbolic space, eliminating the need for projection to Euclidean space. To enable this, the model employs Einstein gyro midpoints, a mathematical technique used for aggregating vectors in hyperbolic space with minimal distortion.

2.2.4.3 DynHAT – Dynamic Hyperbolic Attention Network

DynHAT [13] is another model designed to embed dynamic networks in hyperbolic space, designed to effectively capture both structural hierarchies and temporal dynamics through mechanisms. The model consists of two main components:

- Hyperbolic Structural Attention (HSA): This component operates within each network snapshot to compute node embeddings using attention mechanisms based on hyperbolic distances, thereby preserving the inherent hierarchical structure of the graph.

- Euclidean Temporal Attention (ETA): In contrast to the structural module, the temporal attention mechanism operates in Euclidean space. This design choice improves computational efficiency while allowing the model to adaptively focus on the most relevant past time steps for each node.

The core advantage of DynHAT lies in its hybrid use of geometric spaces: hyperbolic geometry for structural modeling and Euclidean space for temporal dynamics. This combination enables the model to balance representational accuracy and computational efficiency.

Compared to HTGN, which incorporates complex components such as HGRU and HTC, DynHAT utilizes a simpler yet flexible self-attention framework. Furthermore, unlike DHGAT, which performs all computations entirely in hyperbolic space, leading to increased computational overhead, DynHAT improves scalability by offloading temporal computations to Euclidean space. An additional benefit of DynHAT is its ability to dynamically identify the most influential time points in a node's history, resulting in superior performance in downstream tasks such as future link prediction, trend forecasting, and dynamic anomaly detection.

3. Preliminaries

3.1 The Lorentz Model of Hyperbolic Space

There are several geometric models for representing hyperbolic space. While these models are mathematically equivalent, they differ significantly in terms of computational and practical aspects. Among the most common used are the Poincaré disk model, which represents hyperbolic space as an open disk embedded in $\mathbb{R}^n$; the Klein model, in which

geodesics are represented as straight lines; and the Lorentz (or hyperboloid) model, which represents points as lying on a hyperboloid embedded in $\mathbb{R}^{n+1}$.

The Lorentz model offers a particularly efficient and flexible approach for representing hierarchical structures and temporal or evolving graphs. Like other hyperbolic models, it operates in a space of constant negative curvature, parameterized by K. A larger value of K corresponds to a flatter curvature, while a smaller value results in sharper negative curvature. This allows direct control over the curvature of the embedding space, enabling the model to adapt to various data types, ranging from relatively flat structures to deeply complex hierarchical patterns.

Geometrically, Lorentzian space is defined as the set of points in a higher-dimensional space $\mathbb{R}^{n+1}$ that lie on a surface known as a hyperboloid. This surface is characterized by a constraint based on the Lorentzian inner product, which will be formally introduced later.

One of the primary advantages of the Lorentz model is its computational simplicity. The distance between two points can be calculated using a straightforward formula, unlike models such as the Poincaré disk, where distance computations involve more complex formulas and require projection corrections when points approach or exceed the boundary of the disks.

Another key benefit lies in the ability to perform optimization using standard techniques such as Stochastic Gradient Descent (SGD), without the need for complex optimization procedures. Furthermore, it is possible to compute the centroid (center of mass) of a set of points directly using a closed-form expression, while other hyperbolic models typically require an iterative process for the same task.

As a result, the Lorentz model serves as a powerful tool, both mathematically and computationally, especially when working with dynamic graphs and deep hierarchical structures in complex data.

The Lorentz model defines a hyperbolic space as a Riemannian manifold of the form:

$$L_K^n = (L^n, g_x^K)$$

where L^n is the set of points in $\mathbb{R}^{n+1}$ satisfying the constraint:

$$L^n = \{x \in \mathbb{R}^{n+1} | \langle x, x \rangle_L = \frac{1}{K}\}$$

This constraint defines a hyperboloid, a surface embedded in a negatively curved space, which serves as the geometric foundation of Lorentzian space.

The Riemannian metric g_x^K governs how distances and angles are measured in the space, and is defined by a diagonal matrix of the form:

$$g_x^K = diag(-1, 1, \ldots, 1, 1)$$

A key characteristic of this metric is the presence of a single negative entry in this matrix. This fundamental asymmetry among the axes reflects the directional imbalance and is a necessary condition for inducing constant negative curvature for modeling hyperbolic geometry.

Under this formulation, the Lorentzian inner product between two vectors $u, v \in \mathbb{R}^{n+1}$ is defined as:

$$\langle u, v \rangle_L = \; - \; u_{n+1} v_{n+1} + \sum_{i=1}^{n} u_i v_i$$

This inner product serves as the foundation for defining norms, angles, and distances in Lorentzian space, in a manner consistent with hyperbolic geometry. Unlike the Euclidean inner product, it includes a single negative component, reflecting the intrinsically curved nature of the space.

The distance between two points $x, y \in L_K^n$ in Lorentzian space is defined as:

$$d_L(x, y) = \sqrt{K} arcosh \left(\frac{-\langle x, y \rangle_L}{K} \right)$$

Here, $\langle x, y \rangle_L$ denotes the Lorentzian inner product, and K>0 controls the magnitude of the negative curvature.

The tangent space $T_x L_K^n$ at a point $x \in L_K^n$ is an n-dimensional Euclidean vector space that serves as a local linear approximation of the curved space in the vicinity of the point x. The tangent space is defined as:

$$T_x L_K^n = \{ v \in \mathbb{R}^{n+1} : \; \langle x, v \rangle_L = 0 \}$$

This space consists of all vectors orthogonal (in the Lorentzian sense) to the point x (typically equal to zero) and enables standard linear operations such as addition and scalar multiplication.

3.1.2 The Exponential Map

The exponential map is a function that projects a vector from the tangent space $T_x L_K^n$ back onto the hyperboloid, such that the result is a valid point on the curved surface.

It is applied when a computation (e.g., a linear transformation) is performed in the local Euclidean space, and one wishes to return the result to the hyperbolic space.

The exponential map is defined as:

$$exp_x^K(v) = cosh\left(\frac{1}{\sqrt{K}}\|v\|_L\right)x + \sqrt{K}sinh\left(\frac{1}{\sqrt{K}}\|v\|_L\right)\frac{v}{\|v\|_L}$$

where:

- $x \in L_K^n$ is the base point on the hyperboloid
- $v \in T_x L_K^n$ is a tangent vector
- $\|v\|_L = \sqrt{\langle v, v\rangle_L}$ is the Lorentzian norm
- K controls curvature

This mapping is useful when embeddings or representations are learned in Euclidean space $\mathbb{R}^n$ and must be transferred to the Lorentzian manifold.

3.1.3 The Logarithmic Map

The logarithmic map is the inverse of the exponential map, that maps a point $p \in L_K^n$ back to a tangent vector in $T_x L_K^n$e enabling computations to be carried out in the linearized space.

The logarithmic map is defined as:

$$log_x^K(p) = \frac{d_L(x,p)}{\|p + \frac{1}{K}\langle x,p\rangle_L x\|_L}(p + \frac{1}{K}\langle x,p\rangle_L x)$$

where:

- $x \in L_K^n$ is the base point on the hyperboloid
- $p \in L_K^n$ is the target point
- $\langle x,p\rangle_L$ is the Lorentzian inner product
- $\|v\|_L = \sqrt{\langle v,v\rangle_L}$ is the Lorentzian norm
- $d_L(x,p)$ is the Lorentzian distance between x and p.

This transformation flattens the space around x, making it suitable for operations like attention, averaging, or other machine learning algorithms that are typically designed for Euclidean spaces.

3.2 The DynHAT Model in Lorentz Space

In conventional models for dynamic graphs (such as EvolveGCN), it is customary to separate static and dynamic components of the network.
The proposed model builds upon this approach, relying on DynHAT framework, by structuring the learning process into two two distinct layers:

- Structural Information representation Layer
- Temporal Information representation Layer

Rather than attempting to simultaneously learn all aspects of the graph, each layer specializes in capturing a distinct type of information. The Structural layer focuses on encoding node interactions within individual time snapshots, whereas the Temporal layer models the evolution of these interactions across time. This modular architecture enables a more accurate and enriched modeling of both the network's topological structure and its temporal dynamics.

3.2.1 Preprocessing Stage

The model begins with preliminary processing, wherein the input dynamic graph $G = (V, E)$ is partitioned into a sequence of discrete time windows $\left(G_t = \left(V_t, E_t\right)\right)_{t=1}^{T}$, where each time window $1 \leq t \leq T$ corresponds to a static snapshot. Within each snapshot t,the graph G_t is embedded in Lorentzian hyperbolic space to exploit its hierarchical and curvature-aware properties.

The model architecture consists of the following two primary components:

3.2.2 A. Structural Information Representation Layer

This layer focuses on learning node representations that capture the structure of the graph at a given timestamp utilizing a self-attention mechanism adapted to Lorentzian geometry. The process begins with node features in Euclidean space and generates embeddings that encode the local structural context of each node in hyperbolic space. Let v_{it} denote the feature vector of node i at timestamp t.

3.2.2.1 Projection into Lorentzian Space via Exponential Mapping:

The first step aims to map the node's Euclidean feature vector v_{it} to Lorentzian space with curvature K using the exponential map:

$$p_{it} = exp_x^K\left(v_{it}\right)$$

This projection enables the representation of hierarchical or long-range relationships more effectively.

3.2.2.2 Linear Transformation in Lorentzian Space:

To enrich the expressiveness of node embeddings while preserving the geometric structure of Lorentz space, each node vector p_{it} , originally obtained via exponential mapping from Euclidean space, is further transformed into a more advanced latent representation h_{it}.

This transformation follows a three-step process:

- **Projection to Tangent Space:** Since linear operations cannot be performed directly in curved Lorentzian space, the embedding p_{it} is first mapped back to the tangent space at the origin using using logarithmic map $log_x^K(p_{it})$. This projection enables Euclidean-based manipulations while maintaining consistency with the underlying manifold.
- **Linear transformation in Tangent Space**: Within the tangent space, a standard linear transformation is applied by multiplying $log_x^K(p_{it})$ with a weight matrix W and adding a bias term b (optional). This step corresponds to classical feedforward learning in Euclidean geometry:

$$h_{it} = log_x^K(p_{it}) \cdot W + b$$

- **Projection Back to Lorentzian Space:** The resulting vector h_{it} is then mapped back to Lorentzian space using exponential mapping:

$$p'_{it} = exp_x^K(h_{it})$$

This procedure ensures that transformations preserve the curvature-aware structure of the space. The resulting vector p'_{it} serves as the refined representation of node i at time t, suitable for downstream processing such as attention-based aggregation.

This step yields curvature-aware transformed embeddings within Lorentzian space.

3.2.2.3 Attention Score Computation:

To determine the relative importance of neighboring nodes in the embedding process,, an attention mechanism is applied. Specifically, for each node i , attention coefficients α_{ij} are computed using a concatenation of the transformed embeddings of node i and each of its neighbors $j \in N_i$, followed by a LeakyReLU activation and a softmax normalization:

$$\alpha_{ij} = softmax\left(LeakyReLU\left(\left[Wp_{it} || Wp_{jt}\right]\right)\right)$$

where || denotes vector concatenation.

3.2.2.4 Hyperbolic Aggregation of Neighbor Features:

The updated embedding of node i at time t, denoted p'_{it}, is obtained by aggregating the features of its neighbors in a geometry-aware manner. Each neighbor's embedding p_{jt} is first projected into the tangent space at p_{it} via the logarithmic map $log_{p_{it}}\left(p_{jt}\right)$. A weighted average is then computed using the attention coefficients α_{ij}. The result is mapped back into Lorentz space using the exponential map:

$$p'_{it} = \sum_{j \in N_i} \alpha_{ij} \bullet \ log_{p_{it}}\left(p_{jt}\right)$$

This aggregation mechanism enables each node to update its representation based on the influence of its neighbors, while preserving the geometric properties of Lorentzian space.

The outcome is the set of refined node embeddings p'_{it}, that encodes both local structure and hyperbolic geometry at each timestamp.

3.2.3 B. Temporal Information Representation Layer

This layer is responsible for capturing temporal relationships and modeling the evolution of node representations over time. While the Structural Layer focuses on the static graph structure at individual timestamps, the Temporal Layer models the progression of nodes across time, enriching the representation with dynamic context.

The module proceeds through the following steps:

3.2.3.1 Temporal Input Preparation:

At the beginning of the learning process, the structural vectors $p_t = p'_{vt}$ obtained from the structural representation layer at each timestamp t serve as inputs to the temporal layer. To explicitly incorporate temporal positional information for each node, a dedicated positional embedding vector indicating the absolute time position pos_t is added to each structural vector:

$$q_t = p_t + pos_t$$

Each vector q_{vt} represents node v at time t, incorporating both its local structural context and its absolute position in the temporal sequence.

3.2.3.2 Projection into Tangent Space:

Each vector q_{vt} is then mapped into the tangent (Euclidean) space using the logarithmic map $log_x(q_{vt})$. This projection facilitates subsequent operations using Euclidean-based transformations.

3.2.3.3 Temporal Attention with Masking:

To model the temporal dependencies and historical evolution of each node v over time, we employ a self-attention mechanism inspired by the Transformer architecture. For each node v and at each timestamp t, its representation q_{tv} is used as a query to attend over its past representations (prior to t), thus enabling the model to learn how the local neighborhood of v evolves over time.

The attention mechanism is calculated using the the scaled dot-product attention formula:

$$q'_i = Attention(Q, K, V) = softmax(\frac{(qW_q)(qW_k)^t}{\sqrt{T}} + Mask)(qW_v)$$

where:

- Q, K, V are the Query, Key, and Value vectors obtained through linear transformations of the input q using the learnable weight matrices W_q, W_k, W_v, respectively.
- The Mask ensures that each node at time t cannot access information from future timestamps $> t$, s by assigning negative infinity values $-\infty$ to those positions, effectively setting their attention weight to zero.

3.2.3.4 Projection Back to Lorentzian Space:

After the attention computation, the resulting temporal representations, are mapped back to the Lorentz space using the exponential map, ensuring that the final temporal representations remain compatible with the underlying hyperbolic geometry.

The output of this layer is a set of final embedding vectors q'_t one for each node at wech timestamp t, incorporating both the structural context and its temporal dynamics in a manner consistent with Lorentzian space.

3.2.4 C. DynHAT Model Optimization

The DynHAT model is optimized through a supervised learning process using the link prediction task, aiming to infer the likelihood of future or missing connections between nodes in a dynamic network Training is guided by a binary cross-entropy loss function. This loss encourages the model to assign high probabilities to true (observed) links and low

probabilities to false (non-existent) links, thereby distinguishing between actual and spurious relationships over time.

The objective function is defined as follows:

$$L = \sum_{t=1}^{T} \sum_{v \in V} \Big(\sum_{m \in N_t(v)} - log\big(p\big(q_m^t, q_n^t\big)\big) - \sum_{m' \in P_t(v))} log(1 - p(\big(q_m^t, q_n^t\big))\Big)$$

where, T is the total number of time steps, V is the set of all nodes in the network, $N_t(v)$ denotes the set of positive neighbors (i.e., true links) of node v at time t, and $P_t(v)$ denotes the set of negative samples for node v at time t, i.e., randomly selected nodes not connected to v, and q_m^t, q_n^t are the embeddings of nodes m and n at time t, as produced by the Temporal Information Representation Layer.

The link probability function $p(u, v)$ between nodes u and v is computed using the Fermi-Dirac decoder, which maps the Lorentzian distances $d_L(u, v)$ into probabilities:

$$p(u, v) = \frac{1}{1 + e^{d_L(u,v) - 2}}$$

This probabilistic formulation ensures that node pairs closer in Lorentz space are assigned to higher link probabilities, reflecting stronger structural and temporal affinity, while distant pairs are deemed unlikely to be connected.

During training, for each node and at each time step, the model calculates Lorentzian distances between relevant node pairs, converts them into link probabilities, and compares them with the ground truth. This comparison enables the model to update its parameters to minimize the overall loss function, thereby improving its accuracy and reliability in predicting future links across the evolving network.

While the original DynHAT model includes a supervised training module for link prediction, this component is not implemented in the current project. Instead, we propose a custom anomaly detection module designed to identify irregular patterns in dynamic citation networks, based on the temporal behavior of node embeddings. This adjustment aligns with the project's goal of detecting anomalies rather than predicting future links.

3.3 Anomaly detection

Anomaly detection is a fundamental task in machine learning and data analysis, aimed at identifying patterns, behaviors, or events that deviate from the expected or normative functioning of a system. In the context of citation networks, anomalies may manifest in papers exhibiting irregular citation patterns, such as sudden spikes in citation counts, atypical growth trajectories, unexpectedly low impact, or links between conceptually unrelated fields.

By embedding nodes in the Lorentzian hyperbolic space, it becomes possible to analyze both the structural position of each node within the network hierarchy and its temporal

evolution. This geometric framework enables a more nuanced approach to anomaly detection, enabling the identification of deviations not only in structural patterns, but also in terms of temporal irregularities. For example, a paper may appear structurally consistent yet demonstrate anomalous behavior over time when compared to its own citation history or to similar publications.

Such a representation supports the detection of suspicious behaviors (e.g., excessive self-citation or anomalous cross-domain references), as well as the identification of underrecognized yet potentially significant scientific contributions. In doing so, the model enhances the ability to monitor, interpret, and map the dynamics processes underlying complex citation networks.

Common methods for anomaly detection in networks include:

- Local Outlier Factor (LOF): Identifies nodes with significantly lower local density relative to their neighbors.
- Isolation Forest (iForest): Detects anomalies by isolating observations through random partitioning of the feature space.
- Graph Neural Networks (GNNs): Learn structural representations of nodes and detect anomalies as deviations from the learned embedding space.

3.3.1 Isolation Forest

Isolation Forest is a tree-based anomaly detection method that operates on the principle of isolating data instances. Unlike traditional approaches that attempt to model the normal behavior of the data, iForest directly targets anomalies by measuring how easily individual data points can be separated ("isolated") from the rest.

The core idea is that anomalous points tend to be both rare and distant from the majority of the data, making them easier to isolate with fewer splits. In practice, the algorithm constructs an ensemble of binary trees (Isolation Trees), where data points are recursively partitioned using randomly selected features and split values. For each instance, the depth at which it becomes a leaf node, referred to as the *isolation depth,* is recorded. Points with shorter average path lengths are considered more likely to be anomalies, as they are more easily isolated.

Isolation Forest does not assume any specific distribution of the data, making it especially well-suited for large, high-dimensional datasets. It is also computationally efficient. The algorithm excels in detecting point anomalies and is widely used in domains such as fraud detection, network monitoring, predictive maintenance, and fault detection in complex systems.

3.3.2 Local Outlier Factor

The *Local Outlier Factor* (LOF) [1] is a density-based anomaly detection method that quantifies the degree of abnormality of a data point by comparing its local density to that of its neighboring points. The underlying intuition behind LOF is that anomalous instances often reside in regions of significantly lower density compared to their surrounding points.

LOF assigns an outlier score for each instance based on the ratio between its local density to the average local density of its k-nearest neighbors. This local density is estimated through the concept of *reachability distance*, which accounts for both the distance to neighboring points and their own local density. Instances that exhibit substantially lower density than their neighbors are assigned higher LOF scores, indicating a higher likelihood of being anomalous.

Unlike global anomaly detection methods, LOF focuses on local data characteristics, allowing it to detect anomalies that would not stand out in a global context but are distinctive within their immediate neighborhood. This makes LOF particularly effective for datasets with heterogeneous density distributions or complex spatial structures.

4. Approach

4.1 Proposed Model: Anomaly Detection in Dynamic Academic Citation Networks

The proposed model is designed to detect anomalous patterns in dynamic academic citation networks by capturing both the hierarchical structure and temporal evolution of the network. The approach combines dynamic node features with geometric graph embeddings in Lorentzian hyperbolic space and a temporal self-attention mechanism, enabling the identification of nodes whose citation behavior deviates significantly over time.

4.1.1 Stage 0: Preprocessing and Input Representation

The input to the model consists of:

- A a sequence of temporal graph snapshots $G_1,\ G_2,\ \ldots, G_T$, where each graph $G_t = \left(V_t, E_t\right)$ represents the state of the citation network at time step t. The nodes $v_i \in V_t$ corresponds to academic papers, and directed edges $(u, v) \in E_t$ indicate that paper u cites paper v at time t.
- A set of time-dependent feature vectors $\left\{v_{it}\right\}_{i \in V}$ for each node $v_i \in V_t$, capturing both static and dynamic attributes. Static components may include intrinsics such as publication year, venue, or topic classification which do not change over time. Dynamic components reflect temporally evolving aspects of each paper, such as citation count. Together, these feature vectors provide a comprehensive representation of each node's state at each time step $t,$ supporting the model's ability to capture both structural and semantic changes over time.

The core model consists of three main stages: structural encoding, temporal encoding, and anomaly detection phase.

4.1.2 Stage 1: Lorentzian Structural Representation via Hyperbolic Self-Attention

This stage implements the Structural Information Representation Layer described in Section 3.2.2.A of the DynHAT model. Its goal is to generate geometry-aware node embeddings that capture the local structure of the citation graph at each time step, using attention mechanisms adapted to Lorentzian hyperbolic space.

The process begins with the dynamic feature vectors $v_{it} \in \mathbb{R}^d$, which represent each node's attributes at time t and may include both static and dynamic components. These vectors are mapped into Lorentzian space using the exponential map, allowing for hierarchical and non-Euclidean relationships to be encoded.

To enhance representation expressiveness, the embeddings p_{it} are first projected into the tangent space using the logarithmic map. Within this Euclidean space, a standard linear transformation is applied via a learnable weight matrix. The resulting latent vectors are then projected back into Lorentzian space through the exponential map, yielding intermediate embeddings p'_{it}.

Next, an attention mechanism is applied to model the relative importance of each node's neighbors. Attention scores are computed based on the concatenation of the transformed embeddings of the node and its neighbors, passed through a LeakyReLU activation and normalized via softmax.

Finally, a hyperbolic aggregation step is performed. Each neighbor's embedding is projected to the tangent space centered at the target node, where a weighted average, based on attention scores, is computed. The aggregated vector is then mapped back into Lorentzian space, resulting in the final structure-aware embedding p''_{it} for each node at time t.

Algorithm 1: Lorentzian Structural Representation with Attention

Input:

- A sequence of temporal graph snapshots $\{G_t = (V_t, E_t)\}$, where V_t is the set of nodes at time t.
- A dynamic feature vector $v_{it} \in \mathbb{R}^d$ for each node $i \in V_t$, representing both static and dynamic node attributes at time t.
- Learnable parameters: Weight matrix W for linear transformation in tangent space, attention vector a for computing attention scores.
- Lorentzian exponential and logarithmic maps: exp_x^K, log_x^K defined at at a reference point x in tangent space

Output:

- A refined Lorentzian structural embedding $\{p''_{it}\}$ for each node $i \in V_t$, encoding local geometry-aware structural context.

Steps:

For each time step $t = 1, 2, \ldots, T$:

For each node $i \in V_t$:

Step 1: Project the Euclidean feature vector to Lorentzian space:

$$p_{it} = exp_0^K(v_{it})$$

Step 2: Linear transformation in tangent space

$$p'_{it} = exp_0^K\left(log_0^K(p_{it}) \cdot W\right)$$

For each node $i \in V_t$:

Step 3: Compute attention score α_{ij} for each neighbor $j \in N_i$:

Initialize empty list $e_i = [\,]$

for each neighbor $j \in N_i$:

$concat_{ij} = \left[Wp'_{it} || Wp'_{jt}\right]$ # Concatenation of transformed embeddings

$e_{ij} = LeakyReLU\left(a^T \bullet concat_{ij}\right)$ # Raw attention score

Append e_{ij} to e_i

Normalize attention scores using softmax:

for each neighbor $j \in N_i$:

$$\alpha_{ij} = softmax(e\_ij\ over\ all\ j \in N\_i)$$

Step 4: Perform hyperbolic aggregation using attention scores

$sum\_vector = 0$

for each neighbor $j \in N_i$:

$$p''_{it} = exp_{p'_{it}}^K\left(\sum_{j \in N_i} \alpha_{ij} \bullet \left(log_{p'_{it}}^K\left(p'_{jt}\right)\right)\right)$$

Return:

$\{p''_{it}\}$ for all $i \in V_t$ as the final structure-aware representations at time t.

4.1.3 Stage 2: Temporal Representation via Self-Attention

In this stage, we focus on modeling the dynamic behavior of each node i over time, while preserving the structural information captured in the previous stage. Specifically, for each node and time step t, we extract its Lorentzian structural embedding p''_{it}, obtained from Stage 1, and transform it into a temporally contextualized representation.

To facilitate temporal modeling, each embedding p''_{it} is first projected into the tangent (Euclidean) space via the logarithmic map log_0^K. A learnable positional encoding is then added to indicate the absolute position in the temporal sequence, resulting in a temporal token:

$$q_{it} = p''_{it} + pos_{it}$$

These temporal tokens $\{q_{it}\}_{t=1}^{T}$ form a sequence that is passed into a temporal self-attention module (Transformer encoder). This mechanism attends over the entire timeline for each node, allowing the model to capture long-range temporal dependencies and characterize the evolution of node i throughout the observed time window.

The output of the Transformer is a set of temporally enriched vectors in Euclidean space, which are then mapped back to the Lorentzian manifold via the exponential map

$$q''_{it} = exp_0^K\left(Transformer\left(\{q_{it}\}_{t=1}^{T}\right)[t]\right)$$

The resulting embeddings q''_{it} constitute the final dynamic representations, capturing both structural and temporal contexts in a way that is consistent with the underlying Lorentzian geometry.

Algorithm 2: Temporal Representation via Self-Attention

Input:

- A sequence of structural embeddings $\{p_{it}\}_{t=1}^{T}$ from Stage 1, for each node $i \in V$ and each time step $t = 1, 2, ..., T$.
- A positional encoding function $PosEnc(t) \in \mathbb{R}^d$
- A temporal self-attention module (Transformer encoder)

Output:

- A temporally enriched representation $\{q''_{it}\}$ for each node $i \in V_t$.

Steps:

For each node $i \in V$:

Initialize empty sequence $S_i = [\,]$

For each time step $t = 1, 2, \ldots, T$:

If node $i \in V_t$:

 # Step 1: Project structural embedding to tangent space

$$z_{it} = log_0^K\left(p''_{it}\right)$$

 # Step 2: Add positional encoding

$$q_{it} = z_{it} + PosEnc(t)$$

Else

 # If node is absent, use padding vector

$$q_{it} = zero\_vector$$

Append q_{it} *to* S_i

 # Step 3: Apply temporal self-attention

$$Z_i = TransformerEncoder(S_i)$$

For each time step $t = 1, 2, \ldots, T$:

 # Step 4: Map back to Lorentzian manifold

$$q''_{it} = exp_0^K\left(Z_i[t]\right)$$

Return:

$\left\{q''_{it}\right\}$ for all $i \in V$ and $t = 1,\ldots, T$

4.1.4 Stage 3: Anomaly Detection Based on Temporal Node Behavior

In the final stage of the model, anomalies are detected based on the temporal dynamics of each node's behavior in the evolving academic citation network. The temporally

contextualized node embeddings q''_{it}, obtained from the temporal self-attention module in Stage 2, serve as the input to unsupervised anomaly detection algorithms.

We employ two complementary models, Isolation Forest (IF) and Local Outlier Factor (LOF), to assess the consistency, reliability, and interpretability of anomaly detection outcomes.

For every time step $t \in \{1, ..., T\}$, both models are trained independently on the temporal embeddings $\{q''_{it}\}_{i \in V_t}$, and assign a time-specific anomaly score vectors per node i :

$$AS_i^{IF} = (AS_{i1}^{IF}, AS_{i2}^{IF}, ..., AS_{iT}^{IF})$$

$$AS_i^{LOF} = (AS_{i1}^{LOF}, AS_{i2}^{LOF}, ..., AS_{iT}^{LOF})$$

To characterize the anomaly profile of each academic paper (node), two statistical measures can be computed for each score vector AS_i: the mean anomaly score and the standard deviation.

- The Mean anomaly score μ_i

$$\mu_i = \frac{1}{T} \sum_{t=1}^{T} AS_i^t$$

This score captures the average degree of deviation exhibited by the node across all time steps. A high Mean indicates persistent abnormal behavior (e.g., consistently sparse connectivity or structural isolation), while a low Mean suggests alignment with expected citation dynamics.

- The Standard Deviation Anomaly Score σ_i

$$\sigma_i = \sqrt{\frac{1}{T} \sum_{t=1}^{T} \left(\mu_i - AS_i^t\right)^2}$$

This measure quantifies the variability of a node's anomaly level over time. High variance signals temporal volatility, such as sudden bursts of citations or brief deviations, while low variance reflects stability in anomalous or normal behavior.

Together, these metrics provide a nuanced temporal profile of each paper's role in the evolving citation network:

A low mean and low variance imply that the paper consistently conforms to expected citation patterns and exhibits no irregular behavior. A high mean coupled with low variance suggests persistent anomalousness, potentially indicating a unique structural position or sustained deviation. Conversely, a low mean with high variance denotes largely normal behavior interrupted by brief anomalies, while a high mean and high variance point to volatile dynamics, where the paper repeatedly shifts between conformity and deviation.

By applying both IF and LOF independently, we obtain two views on anomaly structure:

- IF detects global isolation based on feature distribution and recursively partitions the embedding space.
- LOF identifies local density deviations by comparing a node's neighborhood structure to its surroundings.

This dual evaluation enables cross-validation of the model's anomaly predictions. Consistent findings across both methods reinforce the robustness of the detection pipeline, while discrepancies reveal sensitivity to local versus global anomaly definitions.

Consequently, this stage not only enhances detection accuracy but also deepens our understanding of anomalous citation behavior in evolving academic ecosystems.

Algorithm 3: Anomaly Detection Based on Temporal Node Behavior

Input:

- Temporal embeddings $q''_{it} \in \mathbb{R}^d$ for each node $vi \in V$ at each time step $t = \{1, ..., T\}$.
- Two unsupervised anomaly detection models: Isolation Forest (IF) and Local Outlier Factor (LOF).

Output:

- For each node i: two temporal anomaly score vectors: AS_i^{IF}, AS_i^{LOF}
- Statistical profiles (mean and standard deviation): σ_i^{IF} , μ_i^{IF} , σ_i^{LOF} ,μ_i^{LOF}:

Steps:

1. For each time step $t = \{1, ..., T\}$:
   1.1. Collect temporal embeddings $\{q''_{it}\}_{i \in V_t}$.
   1.2. Train Isolation Forest model IF_t on $\{q''_{it}\}_{i \in V_t}$.
   1.3. Train Local Outlier Factor model LOF_t on $\{q''_{it}\}_{i \in V_t}$.
   1.4 For each node $i \in V_t$:
   1.4.1 Compute anomaly score $AS_{it}^{IF} = IF_t(q''_{it})$
   1.4.2 Compute anomaly score $AS_{it}^{LOF} = LOF_t(q''_{it})$.
2. For each node $i \in V$:

2.1. Construct temporal anomaly vectors:

$$AS_i^{IF} = (AS_{i1}^{IF}, AS_{i2}^{IF}, ..., AS_{iT}^{IF})$$

$$AS_i^{LOF} = (AS_{i1}^{LOF}, AS_{i2}^{LOF}, ..., AS_{iT}^{LOF})$$

2.2. Compute statistical measures:

$$\mu_i^{IF} = mean\left(AS_i^{IF}\right)$$

$$\sigma_i^{IF} = std\left(AS_i^{IF}\right)$$

$$\mu_i^{LOF} = mean\left(AS_i^{LOF}\right)$$

$$\sigma_i^{LOF} = std\left(AS_i^{LOF}\right)$$

3. Return for each node i:

3.1. Anomaly score vectors AS_i^{IF} and AS_i^{LOF}.

3.2. Mean and standard deviation: $\left(\mu_i^{IF}, \sigma_i^{IF}\right), \left(\mu_i^{LOF}, \sigma_i^{LOF}\right)$.

Stage 0: preprocessing

Input: Dynamic graph $G = (V, E)$

Split the dynamic graph into time-based snapshots $\{G_t = (V_t, E_t)\}_{t=1}^{T}$

$G_1 = (V_1, E_1)$ $G_2 = (V_2, E_2)$ ... $G_T = (V_T, E_T)$

Find v_{it} , Feature vector of node i at V_t

Stage 1: Lorenzian Structural Representation

A. Structural Information Representation Layer

Projection into Lorentzian Space via exponential mapping

$$p_{it} = exp_0^K(v_{it})$$

Linear Transformation in tangent space

$$p'_{it} = exp_0^K\left(log_0^K(p_{it}) \cdot W\right)$$

Calculate attention scores

$$\alpha_{ij} = \underset{j \in N_i}{softmax}\left(LeakyReLU\left(\left[Wp_{it} \| Wp_{jt}\right]\right)\right)$$

Perform hyperbolic aggregation of the attention score α_{ij}

$$p''_{it} = exp_{p'_{it}}^K\left(\sum_{j \in N_i} \alpha_{ij} \cdot \left(log_{p'_{it}}^K(p'_{jt})\right)\right)$$

The output of this stage:
Set of vectors $\{p''_{it}\}$ that capturing the structure of node i at time step t

Stage 2: Temporal Representation

B. Temporal Information representation Layer

Projecting into the tangent space and then adding a time-specific positional embedding to each structural vector.

$$q_{it} = log_0^K(p''_{it}) + P_t$$

Apply temporal self-Attention with Mask

$$q'_{it} = softmax\left(\frac{(qW_q)(qW_k)^T}{\sqrt{F}} + Mask\right)(qW_v)$$

Mapping back to Lorenzian manifold

$$q''_{it} = exp_0^K(q'_{it})$$

The output of this stage:
the final embedding q''_{it} of node i at time t, combining temporal and structural information

Stage 3: Train Anomaly Detection

Anomaly Detection via Isolation Forest (IF)

For each time t, $1 \le t \le T$ train Anomaly Detection on $\{q''_{it}\}$, $i \in V_t$

For each node i at time t compute anomaly score AS_{it}^{IF} reflecting its local deviation from the neighborhood density.

For each node $i \in V$ construct temporal anomaly vector

$$AS_i^{IF} = (AS_{i1}^{IF}, AS_{i2}^{IF}, .., AS_{iT}^{IF})$$

For each vector AS_i compute the two statistical measures

$$\mu_i = mean(AS_i^{IF})$$
$$\sigma_i = std(AS_i^{IF})$$

Analyze the behavioral pattern of each node i via the measures μ_i and σ_i

Type	Std	Mean
Normal	Low	Low
Constant anomaly	How	High
Temporary anomaly	High	Low
Irregular	High	High

Anomaly Detection via Local Outlier Factor (LOF)

For each time t, $1 \le t \le T$ train Anomaly Detection on $\{q''_{it}\}$, $i \in V_t$

For each node i at time t compute anomaly score AS_{it}^{LOF} reflecting its local deviation from the neighborhood density.

For each node $i \in V$ construct temporal anomaly vector

$$AS_i^{LOF} = (AS_{i1}^{LOF}, AS_{i2}^{LOF}, ..., AS_{iT}^{LOF})$$

For each vector AS_i compute the two statistical measures

$$\mu_i = mean(AS_i^{LOF})$$
$$\sigma_i = std(AS_i^{LOF})$$

Analyze the behavioral pattern of each node i via the measures μ_i and σ_i

Interpretation	Std	Mean
Locally normal	Low	Low
Persistently isolated	Low	High
Occasionally anomalous	High	Low
Volatile, disruptive behavior	High	High

Anomaly patterns identified using LOF are compared with those obtained from Isolation Forest, in order to assess consistency and strengthen the reliability of the anomaly detection process using Isolation Forest.

Figure 1 - The proposed model

4.1.5 Stage 4: Validation Based on Noise Injection via Fake Nodes

To evaluate the sensitivity, reliability, and robustness of the proposed anomaly detection model, we introduce an additional validation step involving iterative controlled noise injection into the original citation network. This step aims to assess the model's ability to detect anomalies even when the network contains fake nodes that do not correspond to the natural dynamic and hierarchical structure of the academic citation network.

In this process, fake nodes are randomly injected into the original citation network simulating artificial academic papers that do not exhibit typical citation behavior. For each fake node, we generate artificial static features. After the nodes are inserted into the network, they are then randomly connected to existing nodes, simulating interactions that do not reflect natural citation behavior.

The injection process is performed iteratively across multiple steps. In each iteration i, a predefined number k of fake nodes is randomly generated and inserted into the citation graph at a randomly selected time step t. These artificial nodes are assigned static feature vectors and temporal interactions and are then embedded using the same structural and temporal encoding mechanisms as the real nodes.

The key point is that in each iteration, the newly noisy network (with the injected f*ake* nodes) passes through the entire pipeline of the model. This includes all stages such as the temporal encoding, anomaly detection (using methods like Isolation Forest or LOF), and statistical analysis. The resulting anomaly scores are then computed for both the fake and real nodes. This allows for a comprehensive evaluation of how well the model can handle the noise and whether it can correctly distinguish between authentic citation behaviors and artificial disruptions.

This iterative noise injection process strengthens the validation of the model by exposing it to various artificial disruptions over several rounds, ensuring that the model's performance is consistently evaluated under changing conditions. The iteration process also helps gauge the model's adaptability to varying types of noise in the citation network.

To evaluate the model's performance under controlled noise injection, we define the following validation metrics:

- True Positive Rate (TPR): The proportion of fake nodes correctly detected as anomalies.
- False Positive Rate (FPR): The extent to which real nodes are mistakenly identified as anomalies due to the noise injection.

The goal of this validation step is to assess the model's ability to distinguish between authentic and artificial behaviors. It also tests the model's robustness to structural and temporal noise, ensuring that the model remains stable even when injected with false or unnatural information. Finally, it validates the reliability of the statistical metrics used by the model.

Algorithm 4: Validation Based on Noise Injection via Fake Nodes:

Input:

- Temporal citation network snapshots $\{G_1, G_2, G_3, \ldots, G_T\}$
- Structural and temporal encoding modules (from Stage 1 & 2)
- Anomaly detection algorithm: Isolation Forest /LOF (from Stage 3)
- N: Number of validation iterations, k percent of fake nodes per iteration.

Output:

- Anomaly scores for fake and real nodes in each iteration.
- Validation metrics: True Positive Rate (TPR), False Positive Rate (FPR).

Steps:

1. For each iteration $i = 1 \ to \ N$:

1.1. Compute total number of real nodes: $n = |V|$

1.2. Generate $k\%$ of n as fake nodes $\{f_1, \ldots, f_k\}$.

1.2.1. Assign random or structured static feature vectors x_{f_i} to each fake node

1.2.2. Inject random temporal interactions over time (e.g., citing or being cited by real nodes in G_{t_i})

- Add edges from/to fake nodes at random time steps
- Edges may simulate citations to or from real nodes

1.3. Augment the entire temporal network with fake nodes: Inject the fake nodes for all $t \in \{1, \ldots, T\}$ let $G'_t = G_{t_i} \cup (fake\ nodes\ and\ their\ edges\ at\ time\ t)$.

1.4. Apply full model pipeline to modified temporal network $\{G'_1, G'_2, \ldots, G'_T\}$

1.4.1. Compute structural embeddings for all nodes (real and fake) using Algorithm 1.

1.4.2. Compute temporal embeddings via self-attention (Algorithm 2).

1.4.3. Perform anomaly detection (Algorithm 3).

1.4.4. Compute anomaly scores AS_v^t for all nodes

1.5. Extract anomaly scores:

1.5.1. For each fake nodes f_j: collect temporal anomaly scores $\{AS_{f_j}^{t_i}\}$ across t_i

1.5.2. For each real node v, collect: $\{AS_v^{t_i}\}$

1.6. Determine anomaly labels:

- A fake node is correctly flagged if its anomaly score exceeds a threshold at any t

- A real node is falsely flagged if it exceeds threshold at any t

1.7. Compute validation metrics:

1.7.1. **True Positive Rate (TPR)** = (# of fake nodes correctly flagged) / total # of fake nodes

1.7.2. **False Positive Rate (FPR)**= (# of real nodes falsely flagged) / total # of real nodes

2. After N iterations:

2.1. Aggregate TPR and FPR statistics across all N iterations

3. Return:

3.1. Anomaly scores for real and fake nodes

3.2. Final TPR and FPR metrics

3.3. Robustness assessment under $k\%$ global noise injection

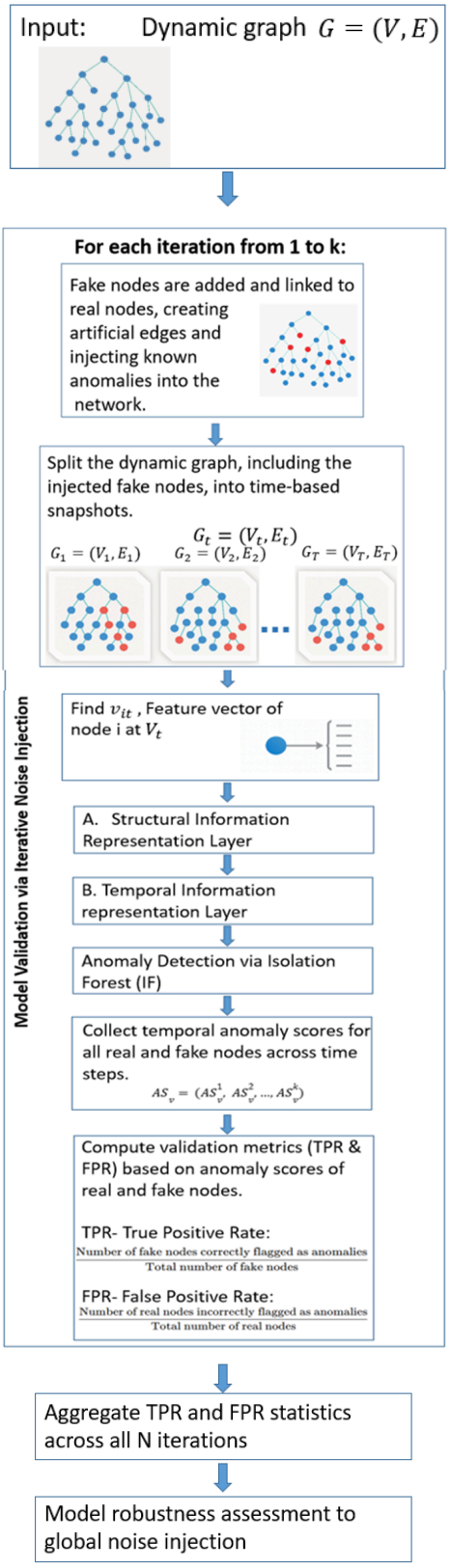

Figure 2 - The Validation process

5. Tools and Technological Framework

The successful implementation of the proposed research-oriented anomaly detection model relies not only on its theoretical design, but also on the careful selection of appropriate computational tools and development environments. This chapter outlines the technological stack adopted for the project, with a focus on open-source, scalable, and research-friendly solutions. Each tool is discussed in terms of usability, integration capabilities, performance, and suitability for both exploratory data analysis and scalable modeling. Together, these technologies provide a cohesive infrastructure for developing, testing, and validating the proposed anomaly detection model.

5.1 Python

Python is selected as the primary programming language due to its numerous advantages in the fields of data science, graph analysis, and machine learning research. As an open-source, dynamically typed, and highly readable language, Python enables rapid prototyping and efficient implementation of complex algorithmic pipelines. Its broad and active community has contributed to a rich ecosystem of scientific libraries, making it particularly well-suited for academic research. The ease of integration between its various modules enables the seamless construction of end-to-end workflows, from data preprocessing to model evaluation.

5.2 Visual Studio Code

Visual Studio Code (VS Code) was chosen as the integrated development environment (IDE) to support code authoring, project management, and version control. It is a lightweight yet powerful, feature-rich code editor that offers extensive support for Python development, including syntax highlighting, automatic error correction, virtual environment management, and direct script execution. Its extensibility through a wide range of plugins, such as Git integration, Jupyter notebook support, and AI-assisted coding tools, makes it ideal for research-oriented development. Thanks to its seamless compatibility with common tools research development, VS Code offers a convenient, accessible, and robust solution for both exploratory coding and long-term modular system design.

5.3 Scikit-Learn

Scikit-Learn is one of the most widely used and reliable machine learning libraries in the Python ecosystem, and it serves as the foundation for the anomaly detection stage of this project. It provides efficient, accessible, and trustworthy implementations of classical supervised and unsupervised learning algorithms, including anomaly detection models such as Isolation Forest and Local Outlier Factor (LOF). In addition, it provides essential components for data normalization, train-test splitting, statistical evaluation metrics, and pipeline management, enabling structured workflows and reproducible experimentation. Its intuitive API and consistent design make it accessible to users with varying levels of programming experience, while supporting a wide range of experimental configurations, modularity, and interpretable results suitable for academic research.

5.4 NetworkX

NetworkX is a widely-used Python library for the creation, manipulation, and analysis of complex networks. It can serve as the primary tool for representing temporal citation networks. NetworkX provides intuitive data structures for directed and undirected graphs and offers a broad set of built-in functions for calculating structural properties such as node degree, clustering coefficients, and connectivity. These capabilities make it particularly well-suited for handling the evolving nature of citation graphs and for performing both exploratory and algorithmic graph analysis in research settings.

5.5 Google Colab

Google Colaboratory (Colab) was chosen as the primary platform for the execution of the model. Colab is a cloud-based environment that supports interactive execution of Python code through a Jupyter-style interface. The choice of Colab is based on its combination of accessibility, user-friendliness, and access to high-performance computational resources, most notably, GPU acceleration. This feature is particularly important in the context of the model, which includes a temporal encoding stage based on a self-attention mechanism, potentially requiring significant computational resources when dealing with long time series or citation networks with many nodes.

Moreover, Colab offers seamless integration with widely used scientific and machine learning libraries, such as scikit-learn, PyTorch, and networkX, thereby facilitating an efficient, reproducible and well-organized research workflow. Altogether, Colab serves as an ideal environment for academic research that demands both computational power and transparency.

6. Challenges

6.1 Challenges Encountered During Phase A

The theoretical development of the project and the composition of its accompanying documentation presented several significant challenges. Addressing these challenges required in-depth engagement with unfamiliar areas of knowledge, as well as the development of advanced academic and research competencies.

6.1.1 Familiarization with Hyperbolic Geometry

The task of embedding hierarchical structures of citation networks into Lorentzian hyperbolic space necessitated delving into a mathematical domain non-Euclidean geometry, an area that was previously unfamiliar. Independent study of foundational concepts, such as geodesic curves, hyperbolic metrics, and geometric mappings, was essential for establishing a foundational understanding of the model’s structural component.

6.1.2 Deepening Understanding of Advanced Deep Learning Models

Designing the temporal encoding module required in-depth familiarity with advanced deep learning architectures, particularly the Self-Attention mechanisms and Transformer models. This involved reviewing recent research literature and critically evaluating the relevance and adaptation of such techniques to dynamic graph sequences.

6.1.3 Developing Academic Writing Skills in English

Composing the project's theoretical documentation in formal academic English posed a considerable challenge. This process included translating technical content into a professional language and coherent narrative, adhering to proper grammatical structures, and integrating domain-specific terminology aligned with academic standards.

6.2 Anticipated Challenges in Phase B

The shift from the theoretical design of the model to its practical implementation is expected to introduce a new set of technological and methodological challenges. These may directly impact the efficiency, validity, and empirical evaluation of the proposed approach.

6.2.1 Identifying a Suitable Dataset

A primary challenge in the implementation stage is locating and selecting an appropriate dataset that satisfies the project's core requirements: a dynamic citation network with well-defined temporal information, static node attributes (such as article topic, authorship, or publication venue), and an inherent hierarchical structure that facilitates meaningful analysis of anomalous behaviors over time. Additionally, the dataset must be publicly available for academic use and of sufficient scale to support statistically significant conclusions.

6.2.2 Selecting an Appropriate Development and Execution Environment

Given that the model involves computationally intensive operations such as hyperbolic space embedding and Transformer-based temporal encoding, a suitable execution environment must support GPU acceleration and sufficient memory capacity. The challenge lies not only in choosing between available platforms but also in balancing performance, cost-efficiency, availability, and ease of use throughout the development lifecycle.

6.2.3 Integration of Model Components

An additional challenge concerns the seamless integration of all the model’s core components into a unified system. Specifically, structural embeddings derived from hyperbolic space must be correctly aligned with the temporal encoder, and the resulting dynamic node representations must be efficiently fed into the anomaly detection algorithms. Given the complexity of each individual stage, combining them into a coherent and functioning pipeline may require extensive experimentation, debugging, and adaptation to the limitations of available libraries and frameworks.

7. Expected Outcomes

The proposed model is expected to demonstrate the ability to effectively detect temporal anomalies in dynamic academic citation networks through an integration of hierarchical representations based on hyperbolic geometry in Lorentzian space, temporal encoding using a Self-Attention mechanism, and the application of complementary anomaly detection algorithms such as Isolation Forest and Local Outlier Factor.

At the content level, the model is expected to generate, for each paper in the citation network, an anomaly profile comprising the mean and standard deviation of anomaly scores over time. The analysis of these profiles is anticipated to reveal meaningful patterns of abnormal citation behavior, such as temporary surges in attention followed by decline, or renewed interest in previously overlooked topics. These temporal dynamics will be explored through graphical analysis of the model's outputs. As such, the model is expected to advance the understanding of scientific activity and support the extraction of qualitative insights from the network's structural and temporal properties.

At the quantitative level, the model aims to achieve a True Positive Rate (TPR) of at least 85% in identifying anomalous nodes, while maintaining a low False Positive Rate (FPR). These metrics will be computed as part of a validation phase that involves controlled noise injection into the network by introducing artificial nodes that exhibit abnormal citation behavior. Evaluating the model's response to these injected nodes will allow for a comprehensive assessment of its sensitivity, precision, and robustness under complex and dynamic conditions.

References

1. Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). *LOF: Identifying density-based local outliers*. *ACM SIGMOD Record, 29*(2), 93–104. https://doi.org/10.1145/335191.335388

2. Chamberlain, B. P., Clough, J. R., & Deisenroth, M. P. (2017). *Neural embeddings of graphs in hyperbolic space*. arXiv preprint arXiv:1705.10359. https://arxiv.org/abs/1705.10359
3. Duan, D., Zha, D., Liu, Z., & Chen, Y. (2024). *Dynamic graph embedding via self-attention in the Lorentz space*. In *Proceedings of the 2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (pp. 199–204). IEEE. https://doi.org/10.1109/CSCWD61410.2024.10580502
4. Law, M. T., Liao, R., Snell, J., & Zemel, R. S. (2019). *Lorentzian distance learning for hyperbolic representations*. In *Proceedings of the 36th International Conference on Machine Learning* (Vol. 97, pp. 3672–3681). PMLR. https://proceedings.mlr.press/v97/law19a.html
5. Li, H., Jiang, H., Ye, D., Wang, Q., Du, L., Zeng, Y., Yuan, L., Wang, Y., & Chen, C. (2024). *DHGAT: Hyperbolic representation learning on dynamic graphs via attention*

networks. Neurocomputing, 568, 127038. https://doi.org/10.1016/j.neucom.2023.127038

6. Peng, W., Varanka, T., Mostafa, A., Shi, H., & Zhao, G. (2022). *Hyperbolic deep neural networks: A survey*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(12), 10023–10049. https://doi.org/10.1109/TPAMI.2021.3136921
7. Radicchi, F., Fortunato, S., & Vespignani, A. (2012). *Citation networks*. In A. Scharnhorst, K. Börner, & P. van den Besselaar (Eds.), *Models of science dynamics: Encounters between complexity theory and information sciences* (pp. 233–255). Springer. https://doi.org/10.1007/978-3-642-23068-4_7
8. Sinha, A., Zeng, S., Yamada, M., & Zhao, H. (2024). *Learning structured representations with hyperbolic embeddings*. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS 2024)*. https://github.com/uiuctml/HypStructure
9. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). *Graph attention networks*. In *International Conference on Learning Representations (ICLR 2018)*. https://arxiv.org/abs/1710.10903
10. Wang, L., Huang, C., Ma, W., Liu, R., & Vosoughi, S. (2021). *Hyperbolic node embedding for temporal networks*. Data Mining and Knowledge Discovery, 35, 1906–1940. https://doi.org/10.1007/s10618-021-00774-4
11. Xu, D., Ruan, C., Korpeoglu, E., Kumar, S., & Achan, K. (2020). *Inductive representation learning on temporal graphs*. In *Proceedings of the International Conference on Learning Representations (ICLR 2020)*. arXiv:2002.07962. https://arxiv.org/abs/2002.07962
12. Zhang, X., Zhou, J., Lin, Y., Sun, L., & Liu, Z. (2023). *Graph convolutional networks in language and vision: A survey*. Knowledge-Based Systems, 259, 110103. https://doi.org/10.1016/j.knosys.2022.110103
13. Zhang, Z., Zhang, H., & Xia, F. (2023). *Dynamic network embedding in hyperbolic space via self-attention*. Neurocomputing, 540, 126309. https://doi.org/10.1016/j.neucom.2023.126309