

`Software Engineering Department  
Braude College

Capstone Project Phase A – 61998

**Spectral Filtering-Based Anomaly Detection**

**25-2-R-11**

Students:

|  |  |  |
| --- | --- | --- |
| Itay Mohabati | 206525008 | [itaymohabati@gmail.com](mailto:itaymohabati@gmail.com) |
| Shai Yosef | 209247600 | shai44553@gmail.com |

Supervisors:

Dr. Renata Avros

Prof. Zeev Volkovich

# Table Of Contents

[Table Of Contents 2](#_Toc199417587)

[Abstract 3](#_Toc199417588)

[1. Introduction 3](#_Toc199417589)

[2. Problem Definition and Related Works 4](#_Toc199417590)

[2.1. Problem Definition 4](#_Toc199417591)

[2.2 Related Works 5](#_Toc199417592)

[**2.2.1.** **Spectral Filtering Methods: Prior Works and Limitations** 7](#_Toc199417593)

[3. Mathematical Background 8](#_Toc199417594)

[3.1 Notations and Definitions 8](#_Toc199417595)

[**3.2.1 GCN Algorithm** 10](#_Toc199417596)

[3.3. GraphSAGE 11](#_Toc199417597)

[**3.3.1. GraphSAGE Algorithm** 12](#_Toc199417598)

[3.4. K-means Clustering 13](#_Toc199417599)

[**3.4.1. K-means Algorithm** 13](#_Toc199417600)

[3.5. SpecF – Spectral Filtering for Anomaly Detection 14](#_Toc199417601)

[**3.5.1 Signal Generation** 15](#_Toc199417602)

[**3.5.2 SpecF Algorithm** 16](#_Toc199417603)

[4. Project Overview 18](#_Toc199417604)

[4.1. Workflow 18](#_Toc199417605)

[4.2. Pseudocode 21](#_Toc199417606)

[5. Expected Achievements 24](#_Toc199417607)

[6. Research / Engineering Process 24](#_Toc199417608)

[6.1 The Process 24](#_Toc199417609)

[6.2 Diagrams 26](#_Toc199417610)

[**6.2.1 Sequence Diagram** 26](#_Toc199417611)

[**6.2.2 Class Diagram** 27](#_Toc199417612)

[7. Evaluation and Testing Plan 27](#_Toc199417613)

[7.1 Evaluation 28](#_Toc199417614)

[7.2 Testing Plan 28](#_Toc199417615)

[8. Summary 30](#_Toc199417616)

[9. References 31](#_Toc199417617)

# **Abstract**

This project presents a novel framework for anomaly detection in community structures of complex networks using a spectral filtering-based approach. The methodology integrates graph embedding techniques, such as GraphSAGE and GCN, to generate informative node embeddings from structural and attribute information. These embeddings are clustered to identify communities, serving as the foundation for further anomaly detection.

The core of the framework is the SpecF algorithm, a spectral graph theory-based method designed to detect anomalous nodes within communities. SpecF applies the Graph Fourier Transform and a low-pass spectral filter to assess the smoothness of node signals within their respective communities. Deviations from this smoothness indicate potential anomalies, capturing irregular behavior that traditional clustering methods may overlook.

By combining embedding-driven community detection with spectral anomaly scoring, this approach addresses the challenges of scalability, structural complexity, and contextual relevance in large graphs. The resulting system is particularly suited for domains such as fraud detection, social network analysis, and biological systems, where anomalies often manifest subtly within structured communities.

# **1. Introduction**

Understanding community structures in large-scale networks is critical for uncovering the functional and organizational principles of complex systems. Community detection methods have proven effective in identifying modular organization within networks such as social graphs, biological interaction maps, and communication networks. However, these methods often overlook subtle irregularities that may signal anomalous behavior within otherwise well-defined communities.

Detecting anomalous nodes whose behavior or attributes diverge significantly from their peers is essential in domains like cybersecurity, fraud detection, epidemiology, and infrastructure monitoring. Traditional statistical techniques often struggle to distinguish meaningful irregularities from background noise, especially when working with attributed graphs or evolving network structures.

To address these challenges, this study integrates modern graph representation learning with spectral filtering-based anomaly detection. Node embeddings, generated using scalable algorithms like GraphSAGE and GCN, encode both topological and feature information. These embeddings serve as the basis for clustering, revealing latent community structures. The SpecF algorithm is then applied to identify anomalies by leveraging graph signal processing principles, particularly the graph Fourier transform and low-pass spectral filtering, to assess signal deviations within each community.

This framework provides a robust, unsupervised pipeline for identifying anomalies in networks with community structure. By capturing both global organization and localized deviations, it enables a deeper understanding of network behavior and highlights critical outliers that may signify emerging threats or hidden trends.

# **2. Problem Definition and Related Works**

## **2.1. Problem Definition**

In many real-world networks such as social platforms, communication systems, biological networks, and financial infrastructures nodes naturally form communities that reflect shared functions or interactions. However, within these communities, certain nodes may deviate from the norm, exhibiting unusual behavior or characteristics. Detecting such anomalies is critical, as they often signal fraud, errors, or emerging phenomena.

The problem addressed in this work is the detection of anomalous nodes within attributed graphs that contain community structure. Given a graph with nodes, edges, and node-level features, the goal is to identify outlier nodes whose signal patterns differ significantly from others in their respective communities.

To solve this, we first generate node embeddings using algorithms like GraphSAGE or GCN, capturing both structural and feature-based information. These embeddings are then clustered to reveal latent communities. Finally, the SpecF algorithm grounded on the spectral graph theory is applied to measure how much a node's signal deviates from the expected smoothness within its community, enabling precise anomaly detection.

This task presents several key challenges:

1. **Scalability**: Large graphs with high-dimensional features demand efficient and scalable embedding and filtering techniques.
2. **Context-Aware Detection**: Identifying anomalies relative to their own communities, rather than globally, requires a method that accounts for local structure and feature distribution.
3. **Signal Smoothness Measurement**: Quantifying how a node disrupts community-level signal smoothness in spectral space is non-trivial and requires robust filtering methods.

By combining embedding-based community detection with spectral filtering, the proposed approach addresses these challenges and enables effective, context-sensitive anomaly identification in complex networks.

## **2.2 Related Works**

Understanding and detecting anomalies in complex attributed networks requires integrating insights from multiple research domains. This section reviews key developments in four interrelated areas: community detection, graph-based anomaly detection, graph signal processing, and graph embeddings. While each area contributes essential tools, existing approaches often fall short in identifying subtle, context-dependent anomalies that occur within localized graph communities.

Girvan and Newman (2002) introduced a modularity-based framework for community detection by iteratively removing edges with high betweenness centrality, revealing latent community structures. Building on this, Blondel et al. (2008) proposed the Louvain method, a scalable approach that greedily optimizes modularity through a hierarchical clustering process, making it suitable for large-scale networks. These methods successfully identify structural partitions in graphs but are not designed to uncover nodes whose behavior deviates within otherwise similar communities.

Traditional graph anomaly detection techniques have primarily focused on global structural outliers. Akoglu et al. (2015) surveyed methods that largely overlook local context. Early approaches such as CODA and GOutRank began to address this limitation by evaluating anomalies in relation to their immediate neighborhoods or communities, enabling the detection of semantically deviant nodes. However, these techniques often rely on static graph representations and hand-crafted features, limiting their ability to adapt to dynamic or evolving networks.

Graph Signal Processing (GSP) extends classical signal analysis to irregular graph domains by utilizing tools such as the Graph Fourier Transform (GFT) and spectral filtering. Sandryhaila and Moura (2013) demonstrated how GSP can be used to detect irregular signals in sensor networks by removing high-frequency components. Egilmez and Ortega (2016) expanded this idea to localize collective anomalies in spatially structured data. Despite its strengths in signal denoising and fault detection, most GSP methods apply global filtering and do not consider community structure, limiting their sensitivity to localized deviations.

Prior to the widespread adoption of deep learning, early graph embedding algorithms focused on learning low-dimensional node representations that encode both topological proximity and attribute similarity. Techniques such as DeepWalk (Perozzi et al., 2014) and node2vec (Grover and Leskovec, 2016) used random walks to generate sequences of nodes and trained Skip-Gram models to learn embeddings that reflect network structure. TADW (Yang et al., 2015) and HSCA (Cao et al., 2016) extended this idea by integrating node attributes into the embedding process, enabling more semantically informed representations for tasks such as clustering and anomaly detection. While these methods improved over purely structural embeddings, they do not inherently model community-specific behavior and often require additional post-processing to identify anomalies.

These studies contribute foundational methods for community detection, anomaly scoring, signal filtering, and node representation in graphs. However, they do not fully resolve the problem of detecting anomalous nodes within attributed graphs that exhibit strong and dynamic community structure an issue this work seeks to address through community-aware filtering techniques.

### **2.2.1. Spectral Filtering Methods: Prior Works and Limitations**

Spectral filtering techniques play a crucial role in graph signal processing (GSP), providing a means to examine the smoothness and frequency properties of signals distributed over graph nodes. By applying eigen-decomposition to the graph Laplacian, these methods convert graph signals into the spectral domain, allowing for the separation of low-frequency components—indicative of structural regularity—from high-frequency components, which often signal potential anomalies.

**GraphWave** (Donnat et al., 2018) introduced spectral graph wavelets to encode structural roles of nodes based on local diffusion patterns. Although GraphWave effectively captures local topology and structural similarity, it is not designed for anomaly detection and does not incorporate any notion of community structure, which is essential for detecting context-specific irregularities.

Other spectral methods based on **heat kernel diffusion** or **residual energy minimization** (commonly used in physics-inspired anomaly detection) assess how smoothly node-level information propagates through the graph. While effective in identifying global irregularities, these methods are typically unaware of community partitions, making them less sensitive to localized anomalies that manifest only within modular structures.

More recently, **Pyramid Graph Neural Networks (PyGNN)** (Geng et al., 2023) combine graph sampling with spectral filtering to learn disentangled representations at multiple scales. PyGNN exemplifies the recent shift towards multi-scale spectral analysis, yet it remains a supervised framework aimed at representation quality rather than anomaly localisation and does not explicitly incorporate community structure. Despite these advances, existing spectral approaches generally **lack three key capabilities** that are critical for community-aware anomaly detection:

1. **Community Awareness**: Most models either ignore community boundaries or treat them implicitly, making them ill-equipped to detect intra-community outliers.
2. **Unsupervised Anomaly Detection**: Many spectral models require labeled data or task-specific supervision, limiting their applicability in unsupervised settings.
3. **Joint structural + community encoding** – Prior methods build their Laplacian solely from edge connectivity, whereas **SpecF** constructs an expanded Laplacian that also encodes community membership, enabling precise, context-aware filtering.

To address these limitations, **SpecF** applies a low-pass spectral filter tailored to the community structure of the graph. By comparing raw node signals to their smoothed counterparts within each community, SpecF identifies nodes whose behavior significantly deviates from localized norms. This allows the detection of **subtle, context-sensitive anomalies** that global or community-agnostic models often overlook.

# **3. Mathematical Background**

## **3.1 Notations and Definitions**

**Feature Matrix:**

In the context of attributed graphs, each node is associated with a feature vector where is the number of features per node. The feature matrix is given by:

Here, is the total number of nodes in the graph, and each row of corresponds to the feature vector of a node.

**Adjacency Matrix:**

The adjacency matrix of a graph encodes the pairwise relationships between nodes as follows:

For unweighted graphs, if there is an edge between node and node ; otherwise, .

For weighted graphs, represent the strength of the connection between nodes, such as correlation, distance, or some domain-specific metric.

**Node Embedding**

A **node embedding** is a way to represent each node in a graph as a **low-dimensional vector** that captures its **structural role** and possibly its **feature attributes**. The goal is to map each node ​ to a vector (number of nodes), in such a way that **similar nodes have similar embeddings.**

**Signal Vector**

A **graph signal** is defined as a function that assigns a real-valued scalar to each node. It is represented as a vector: for all nodes in the graph.

**Community**

A **community** in a graph is a group of nodes more densely connected internally than with the rest of the graph. The set of all communities is denoted as:

**Graph Fourier Transform (GFT)**

GFT transforms a signal defined on the **nodes of a graph** into the **spectral** **(frequency) domain**. This enables the analysis of how the signal varies with respect to the graph's structure such as how smooth or abrupt the signal is across neighboring nodes.

**3.2. GCN**

Graph Convolutional Networks (GCN) are used in our approach to generate node embeddings and cluster nodes into meaningful community representations. The core idea is to iteratively refine each node’s representation by aggregating feature information from its neighbors and applying a transformation. Each GCN layer performs a localized graph convolution, where a node combines its own features with those of its neighbors, followed by a non-linear activation. This process enables the model to effectively capture both the structural relationships in the graph and the intrinsic feature information of nodes.

### **3.2.1 GCN Algorithm**

**Inputs:**

* Feature matrix: , where **N** is the number of nodes and **F** is the number of features per node, each row is the feature vector of node .
* Adjacency Matrix: representing connections between nodes.

**Preprocessing the Graph Structure (Structure Layer)**

* Add self-loops to the adjacency matrix: ,
* Compute the degree Matrix:
* Normalized the adjacency matrix:

This normalization ensures that neighbor features are averaged properly, preventing high-degree nodes from dominating.

**Initial Feature Representation**

Set the initial node representation to the feature matrix:

**Graph Convolution Layer**

This is the core of the GCN - it updates node representations by aggregating from neighbors.

For each layer compute the next representation:

* are the node features at layer (start with )
* is the learnable weight matrix
* σ is a non-linear activation function(e.g.)

**Explanation:**

**Multiply of :** This step averages each node's features with those of its immediate neighbors. In a graph, each node is influenced by its neighbors. This multiplication allows each node to gather and blend information from its neighbors, effectively capturing local structure and patterns. For a node *i*, the new feature vector is the weighted average of its own features and those of its neighbors (thanks to normalization via ).

**Multiply by :** Applying a Linear Transformation: Once the node has aggregated information from its neighborhood, it passes this combined data through a learnable weight matrix This weight matrix acts like a filter, deciding **which features to emphasize** or **combine**, and is trained during learning to best represent the graph structure. This enables the network to learn new feature combinations.

**Then applying σ:** After the linear transformation, a non-linear function like **ReLU** is applied to the output. Without non-linearity, stacking multiple GCN layers would still behave like a single linear transformation (the model would remain shallow in what it can learn). The activation function allows the network to model **complex, non-linear patterns** in the graph.

**Final Embedding Output**

After L layers, the final output is:

* d is the size of the final embedding vector.
* Each row represents the final embedding of node .

## **3.3. GraphSAGE**

**G**raph **S**ample and **A**ggregate (GraphSAGE) is an inductive method that generates embeddings by sampling and aggregating features from a node’s neighborhood. Unlike GCN, which require access to the entire graph during training, GraphSAGE can generalize to unseen nodes and support dynamic graphs, making it more scalable and flexible.

The core idea behind GraphSAGE is to represent each node by repeatedly sampling a fixed number of neighbors and applying a learned aggregation function (e.g., mean, or pooling) to combine their feature information. This process is applied layer by layer, allowing each node to gather context from increasingly larger neighborhoods (multi-hop aggregation). The learned aggregator functions are shared across all nodes, enabling the model to effectively generate embeddings for new nodes even after the model has been trained.

### **3.3.1. GraphSAGE Algorithm**

**Inputs:**

* A graph where is the set of nodes and is the set of edges.
* Feature matrix X ∈ ℝⁿˣᵈ, where each node has node features
* Number of layers: determines the number of hops (neighborhood layers) from which to aggregate information. In other words, the depth of the neighborhood aggregation.
* Sampling size per layer : Specifies how many neighbors to sample for each node at each layer.
* Aggregation function to combine features from sampled neighbors: mean or max pooling.

**For each node :**

1. Initialization: the initial feature vector of node
2. Neighborhood Sampling: At each layer , a fixed-size set of neighbors

is randomly sampled.

1. Aggregation: Aggregate the representations of the sampled neighbors using a function such as:

* **Mean aggregator**:

average of neighbor features. This is a simple, parameter-free method that captures general information from the local neighborhood.

* **Max pooling**:

Each neighbor's vector is passed through a linear transformation and ReLU (It’s an **activation function** used in neural networks to introduce **non-linearity** into the model.),  
then an element-wise max is computed across all transformed neighbors.  
Useful for highlighting strong or dominant features in the neighborhood.

1. Combination: Combine the aggregated neighbor vector with the node’s current representation:

Where is a learnable weight matrix and is an activation function like ReLU.

1. **Final Embedding**: After layers, the final embedding ​encodes both structural and attribute information from the node’s local neighborhood.

## **3.4. K-means Clustering**

K-means is a classical unsupervised learning algorithm used to partition data into a predefined number of K clusters by minimizing the within-cluster variance. In graph analysis, K-means is commonly applied to node embeddings produced by models such as GCN or GraphSAGE, grouping nodes based on feature similarity in the embedding space. Although K-means is not inherently aware of graph structure, it remains a popular choice for the final clustering step due to its simplicity, efficiency, and scalability. The algorithm works by iteratively updating cluster centroids and assigning each node to the nearest centroid based on Euclidean distance, continuing this process until convergence.

### **3.4.1. K-means Algorithm**

**Inputs:**

* A set of node embeddings , where .
* The number of desired clusters

Initialization:

* Randomly initialize centroids:

Assignment Step:

* For each node embedding , assign it to the cluster with the nearest centroid using Euclidean distance:

Update Step:

* For each cluster , recompute the centroid as the mean of all points assigned to that cluster:

**Convergence:**

* Repeat the assignment and update steps until cluster assignments no longer change or a maximum number of iterations is reached.

**Final Output:**

* A cluster label for each node:
* A set of final cluster centroids:

## **3.5. SpecF – Spectral Filtering** **for Anomaly Detection**

Spectral Filtering for Anomaly Detection (SpecF) is an unsupervised anomaly detection algorithm designed for attributed graphs with community structure. It identifies nodes whose signal values deviate sharply from the expected norms within their own communities, making it well-suited for detecting context-aware, localized anomalies.

Grounded in graph signal processing (GSP) and spectral graph theory, SpecF operates in the spectral domain using the Graph Fourier Transform and a low-pass spectral filter. The key idea is to smooth the signal over the graph in a way that respects community boundaries. Nodes that undergo large changes in signal value after filtering are flagged as anomalies, as they break the smooth, expected patterns of behavior within their community.

### **3.5.1 Signal Generation**

Before applying the SpecF anomaly detection method, it is essential to generate the signal vector over the graph that reflects both normal behavior and potential anomalies. This is achieved through a two-stage process: first creating a baseline signal consistent within communities, then injecting anomalies to simulate unusual behavior.

**Normal Signal Generation**

To construct a coherent base signal, we define a metagraph , where each node represents a community in the original graph , and the edge weights correspond to the number of inter-community edges between and . For each community node ​, we define a community-level signal:

where is the set of neighbors of node ​ in the graph.

This community signal is then propagated throughout the graph using the following algorithm:

* **Input:** Graph , community signal vector , partition
* **Output:** Node-level signal vector

1. Mark all nodes in
2. Initialize auxiliary signal vector as zero
3. For each community
   * Identify node with the highest degree
   * Set ​, unmark
4. Create a queue of all unmarked nodes ordered by index
5. While :
   * Let be the first node in
   * For each ​:

* If is marked, unmark it and add to
* Remove ​ from

1. Normalize for each node :

The resulting vector reflects typical behavior expected within communities.

**Anomalous Signal Generation**

To simulate anomalies, we perturb the signal vector to obtain an anomalous signal . A specified percentage of nodes are randomly selected, and their values are increased relative to the maximum in their respective communities.

The Algorithm:

* **Input:** Graph , normal signal , anomaly rate , anomaly intensity
* **Output:** Anomalous signal vector

1. Set
2. Randomly select a subset such that
3. For each :

The vector serves as the input signal for the SpecF algorithm, capturing both standard and anomalous behaviors in a community-aware manner.

### **3.5.2 SpecF Algorithm**

**Inputs:**

* A graph with nodes and edges.
* A **signal vector** , where ​ is the value at node ​.
* A **community partition** , grouping the nodes into communities.

**Expanded Laplacian Construction**

Define an expanded adjacency matrix to encode both graph and community structure:

Then compute the **degree matrix** , where .  
Finally, Define the **Laplacian matrix**:

**Graph Fourier Transform (GFT)**

Compute the eigen-decomposition of :

Then, apply the Graph Fourier Transform (GFT) to the signal :   
where represents the signal in the **spectral domain**, decomposing it into low- and high-frequency components.

**Low-Pass Spectral Filtering**

Define the **cutoff frequency** ​, where is the number of communities.

Define the **ideal low-pass filter**:

Let be the filter matrix.

Then apply the filter:

Here, is the filtered (smoothed) version of the original signal.

**Anomaly Scoring and Detection**

To quantify deviations introduced by the filtering process, compute the **absolute deviation** for each node as follows:

where measures the magnitude of change in the signal at node , and a large value of may indicate an anomalous behavior. Let the deviation vector be defined as:

For each community , a community-specific threshold is computed to account for local variation:

This threshold captures both the average deviation and the standard deviation within the community, normalized by the maximum observed deviation in ​. A node is classified as anomalous if:

This approach ensures that anomaly detection is contextualized within each node’s community, allowing for more localized and accurate identification.

**Output:**

The resulting output is the set of nodes whose signal deviations significantly exceed those of their community peers:

# **4. Project Overview**

## **4.1. Workflow**

The project workflow is designed to detect anomalous behavior within communities in large-scale attributed graphs. It proceeds through a series of structured stages, each building upon the outputs of the previous step to incrementally refine the representation of the network and identify irregularities.

The process starts by constructing or obtaining a graph where nodes correspond to articles, and edges represent citation links. Each node is associated with a feature vector based on the article’s bag-of-words representation. The raw graph undergoes preprocessing to ensure consistency, which may include steps like normalization and noise removal.

Once the graph is prepared, the next step involves **generating node embeddings** using graph representation learning techniques. Two primary models are used: **Graph Convolutional Networks (GCN)** and **GraphSAGE**. These models integrate both the local structure of the graph and the attribute information of the nodes to produce compact, information-rich embeddings. These embeddings map each node to a vector in a lower-dimensional space, preserving structural and semantic proximity.

After embedding, the resulting node vectors are clustered using the **K-means** algorithm. Unlike traditional community detection methods that rely purely on topological properties, clustering in the embedding space enables the identification of communities based on both connectivity and attribute similarity. The number of clusters is predefined, and each node is assigned to one of the resulting communities based on its proximity to cluster centroids.

Before anomaly detection, **Signal Generation** is performed to assign a scalar signal to each node. These signals reflect expected behaviors within each community, such as aggregated feature values or other meaningful indicators. To test the anomaly detection system, an optional **Signal Injection** step introduces synthetic anomalies by perturbing the signal values of selected nodes, simulating unusual or unexpected behaviors within the graph.

Following signal generation and injection, the core of the anomaly detection mechanism is applied. The **SpecF (Spectral Filtering)** algorithm analyzes a signal over the graph, which may represent behavioral data, feature scores, or any scalar node attribute by transforming it into the spectral domain using the **Graph Fourier Transform (GFT)**. A **low-pass filter** is then applied, suppressing high-frequency components that represent irregularities. The filtered signal is compared to the original, and nodes with high deviation are flagged as potential anomalies.

To enhance sensitivity to context, the filtering and scoring are performed **within each community** using a modified Laplacian that encodes both structural and community membership information. Each community is assigned a **dynamic anomaly threshold** based on its internal deviation statistics, allowing detection to be tailored to the unique characteristics of each group.

The output of the workflow is a set of node-level anomaly scores and binary labels indicating whether each node is considered anomalous. These results can then be used for further analysis, such as visualizing abnormal patterns, correlating anomalies with known incidents, or informing downstream classification tasks.

A diagram of a diagram

AI-generated content may be incorrect.In summary, this modular and unsupervised pipeline integrates **graph embedding, clustering, and spectral filtering** to deliver robust and context-aware anomaly detection in networks with community structure.

## **4.2.** **Pseudocode**

**Input:**

A graph , where:

* : Nodes representing articles
* : Edges representing citations between articles
* : Feature matrix where each row is a bag-of-words vector for an article

**Output:** A set of anomalous nodes , identified based on intra-community signal deviations

**Step 1. Data Preparation**

**Objective:** Prepare and preprocess the citation graph and article features

**Input:** Raw citation graph

**Output:** Cleaned graph with structured citation edges and textual features

1. Represent each node as an article
2. Define edges : a directed or undirected edge between two articles indicates a citation
3. Construct a bag-of-words feature vector for each article based on its textual content
4. Normalize the feature matrix (e.g., using TF-IDF or row-wise L2 normalization)

**Step 2. Embedding Generation**

**Objective:** Generate low-dimensional embeddings that encode structural and textual features  
**Input:** Graph

**Output:** Embedding matrix

a. Choose a graph embedding model:

* Use **GCN** to perform convolution over graph structure
* Or use **GraphSAGE** to inductively aggregate from neighborhood features

b. For each node compute embedding

c. Assemble embeddings into matrix

**Step 3. Clustering**

**Objective:** Identify communities by clustering nodes in the embedding space

**Input:** Embeddings , number of clusters *k*

**Output:** Cluster assignments

* Select number of clusters
* Apply **K-means** clustering on embedding matrix
* Assign each node to cluster
* Output the cluster label vector

**Step 4: Signal Generation**

**Objective:** Generate a node-level signal vector consistent with community structure.

**Input:** Graph , community partition

**Output:** Normal signal vector

​Construct a smooth signal where nodes within the same community have similar values.

* Use a community-aware propagation and normalization strategy to assign signal values.

**Step 5: Anomalous Signal Injection**

**Objective:** Inject anomalies into the signal by perturbing selected nodes.

**Input:** Normal signal community partition anomaly rate anomaly intensity

**Output:** Anomalous signal vector

* Randomly select AN% of nodes as anomaly candidates.
* Increase their signal values based on the maximum in their respective communities to form

**Step 6. SpecF-Based Anomaly Detection**

**Objective:** Detect nodes that deviate from expected signal smoothness within their communities  
**Input:**

* Graph
* Node signal vector
* A set of community labels

**Output:** Set of potentially anomalous nodes

a. For each community :

1. Construct the expanded adjacency matrix ​, where:

1. Compute degree matrix , and Laplacian
2. Perform eigen-decomposition:
3. Apply the Graph Fourier Transform:
4. Define the low-pass filter based on the number of communities ( k ):
5. Apply the filter in the spectral domain:
6. Apply the inverse Graph Fourier Transform to obtain the filtered signal:
7. Compute the node-wise absolute deviation:
8. Define the deviation vector:
9. For each community , compute the anomaly threshold:
10. Mark node as anomalous if:

**Step 7. Output Results**

**Objective:** Present and utilize detected anomalies for interpretation and further analysis.

**Input:** Set of anomalous nodes

**Output:** Visualization or report of anomalies

* Return anomaly labels for each node
* Optionally visualize anomalies using graph layouts or community cluster coloring
* Use anomaly insights to support domain-specific tasks (e.g., detect fraudulent articles, uncover unusual citation behavior, etc.)

# **5. Expected Achievements**

This research aims to design a scalable and unsupervised framework for detecting anomalous behavior within graph-based communities by combining graph representation learning, clustering, and spectral filtering techniques. The methodology integrates GCN and GraphSAGE approaches for embedding graph nodes, a K-means clustering for discovering latent community structure, and the SpecF algorithm for identifying context-specific anomalies based on deviations in signal smoothness. The expected outcome is an effective and interpretable system for detecting irregular patterns in attributed graphs. The proposed approach is anticipated to enhance the detection of suitable anomalies possible overlooking the known in the field methods by providing greater insight into localized disruptions in graph structure across various real-world applications.

# **6. Research / Engineering Process**

## **6.1 The Process**

The research process was divided into two main phases: **Phase A: Research and Learning** and **Phase B: Implementation and Evaluation**.

**Phase A: Research And Learning**

This phase focused on investigating relevant algorithms for graph representation, clustering, and anomaly detection. The objective was to design a pipeline capable of identifying community-based anomalies in large-scale attributed graphs using scalable and interpretable techniques.

The research began with a comprehensive review of **graph neural networks** (GNNs) and their role in capturing both topological and feature-based information from graph-structured data. Foundational studies on **Graph Convolutional Networks (GCN)** were explored to understand how graph Laplacian-based operations enable localized filtering and smooth feature propagation. In parallel, **GraphSAGE** was studied to assess its inductive capabilities and neighborhood sampling strategies, making it more suitable for large or dynamic graphs where unseen nodes may appear at inference time.

To evaluate community structure in the graph, clustering techniques were compared. K-means was selected for its simplicity, scalability, and compatibility with the learned embeddings from GCN and GraphSAGE. It provided a straightforward way to group nodes with similar representations into communities, which formed the basis for community-aware anomaly detection.

For the anomaly detection component, the **SpecF algorithm** was studied in depth. The focus was placed on understanding spectral graph theory, graph Fourier transforms, and signal smoothness in the graph domain. The original SpecF formulation was analyzed, particularly its method of constructing an expanded adjacency matrix that encodes both topological and community-based relationships using fixed weights. The thresholding mechanism used to determine anomalies, based on normalized deviation from the smoothed signal, was also examined and validated for use in our pipeline.

**Phase B: Implementation and Evaluation**

In the implementation phase, the selected components were integrated into a complete anomaly detection framework. The pipeline begins by ingesting a graph with article nodes and citation edges, along with node-level features derived from a bag-of-words representation of article content. These inputs are then passed to either a **GCN** or **GraphSAGE** model, depending on the setting, to generate low-dimensional embeddings that encode both structural and contextual information.

Next, the node embeddings are clustered using the **K-means**, assigning each node to a specific community. These communities serve as the basis for the expanded adjacency matrix required by the SpecF algorithm. The matrix is constructed using fixed weights: 5 for connected same-community nodes, 3 for connected cross-community nodes, 1 for unconnected but same-community nodes, and 0 otherwise.

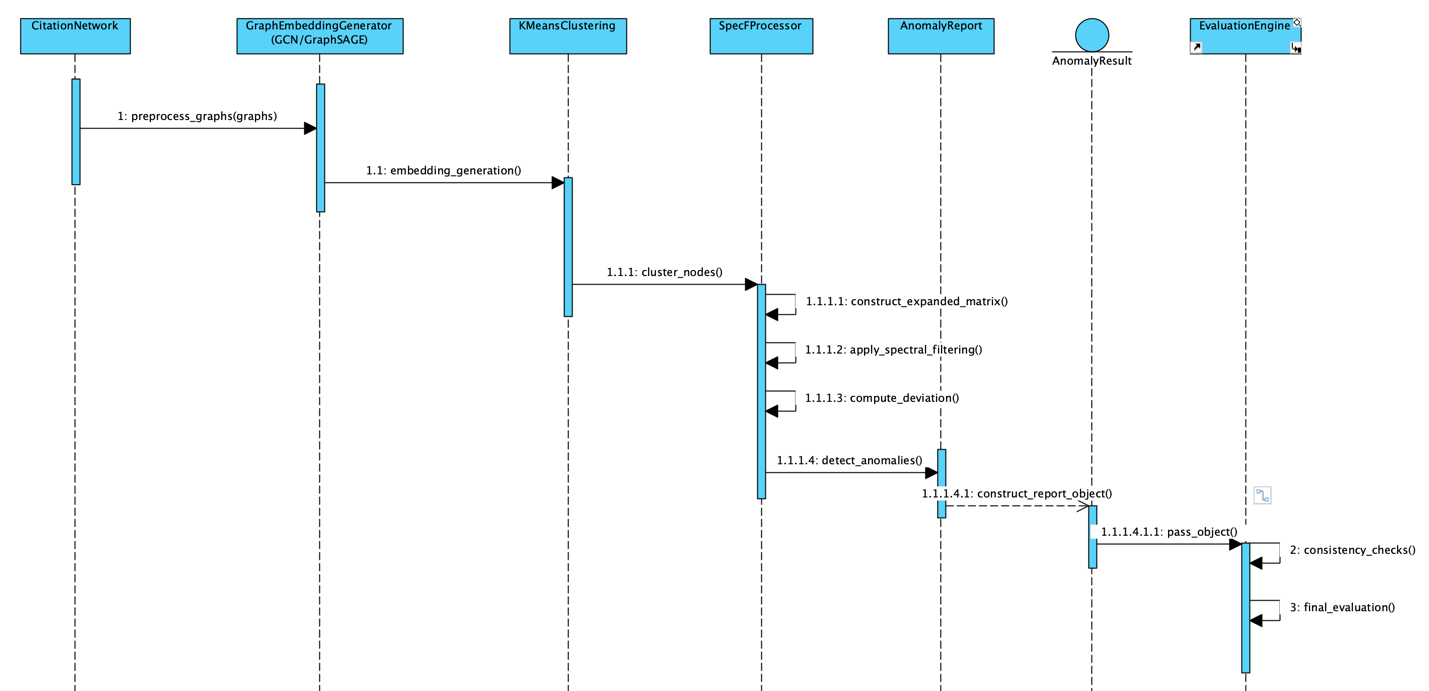
A scalar signal (e.g., a selected embedding dimension) is then analyzed within each community. The **Graph Fourier Transform** is applied to project the signal into the spectral domain. A low-pass filter is used to smooth the signal, and the difference between the original and the smoothed signal is used to compute node-wise deviations. A community-specific threshold is calculated based on the normalized combination of mean and standard deviation of deviations, as described in the original SpecF method. Nodes with deviations exceeding this threshold are flagged as anomalies.

The final output is a set of node-level anomaly labels, which can be used for further interpretation, visualization, or validation. The framework is designed to be modular and adaptable, supporting the substitution of components like clustering algorithms or signal definitions as needed.

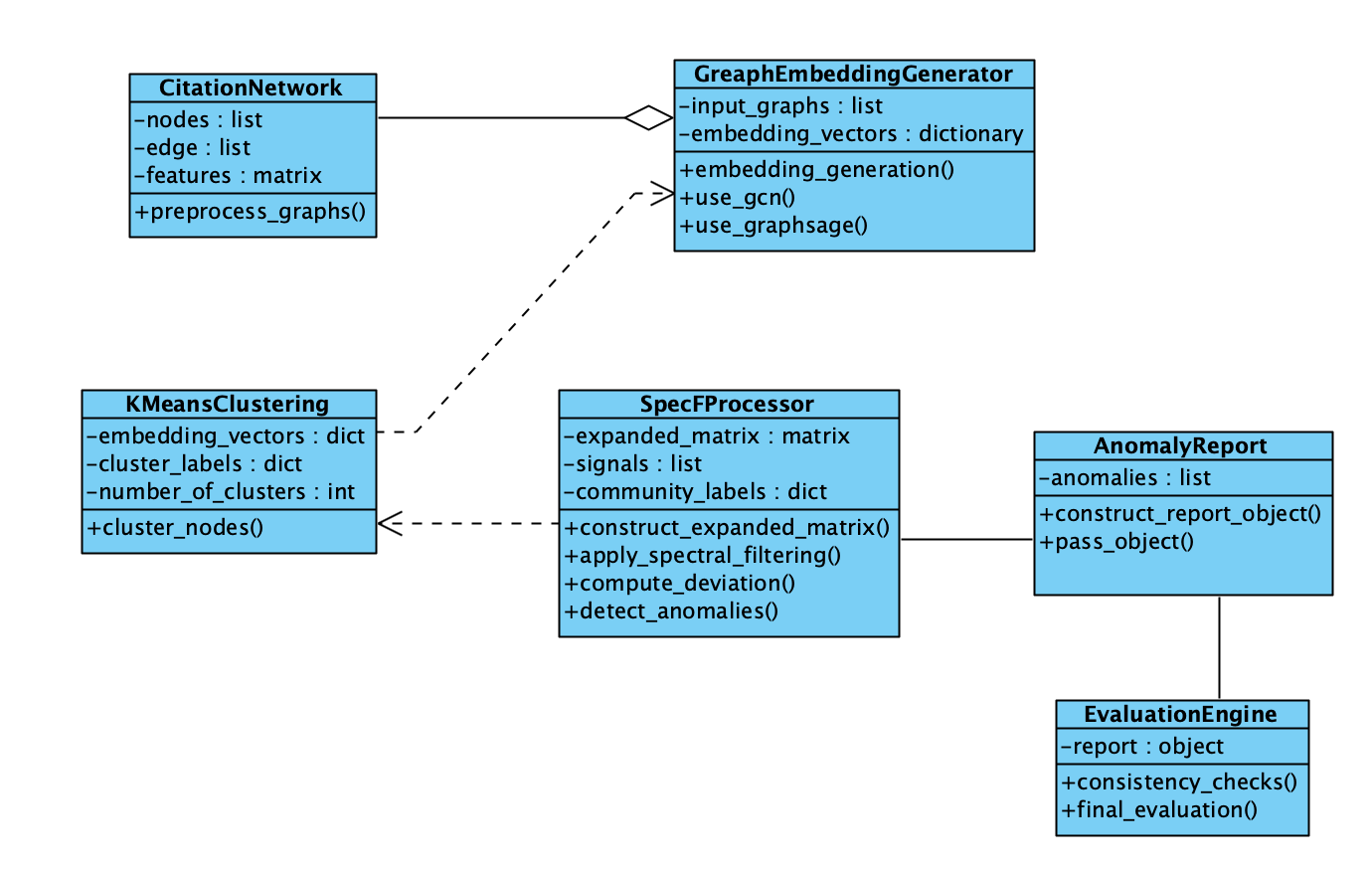
This process results in a flexible and interpretable anomaly detection pipeline that leverages the strengths of graph neural networks for representation learning, unsupervised clustering for community discovery, and spectral filtering for localized anomaly detection. It is expected to perform well in real-world scenarios where community context plays a critical role in identifying meaningful irregularities in graph data.

## **6.2 Diagrams**

### **6.2.1 Sequence Diagram**



### **6.2.2 Class Diagram**



# **7. Evaluation and Testing Plan**

In Part B of this research, we focus on implementing a modular anomaly detection framework that combines GraphSAGE, GCN, K-means clustering, and the SpecF algorithm. The objective is to detect anomalous nodes within community structures of large attributed graphs by analyzing deviations in scalar node signals.

The process begins by generating node embeddings using either **Graph Convolutional Networks (GCNs)** or **GraphSAGE**, both of which are designed to incorporate structural information and node features into dense vector representations. These embeddings are then clustered using the K-means algorithm to define community boundaries. Within each community, the SpecF algorithm is applied to detect nodes that significantly deviate from expected signal patterns, as measured through spectral filtering of the node signal.

## **7.1 Evaluation**

After the full implementation of the framework, we will evaluate its effectiveness by running it on real or synthetic graph datasets and analyzing the results. We will test both GCN and GraphSAGE as embedding methods and compare how each affects the clustering and anomaly detection results.

The evaluation will focus on ensuring that the system correctly detects meaningful anomalies within each community and that the overall structure of the graph is preserved through the embedding and clustering process. We will also test the consistency of the results across different datasets and configurations to confirm the correctness and stability of the implementation.

In addition, we will measure basic performance metrics such as runtime and memory usage to ensure that the approach is scalable and suitable for large graphs.

## **7.2 Testing Plan**

The testing plan is designed to verify the correctness, consistency, and integration of all components involved in the anomaly detection pipeline. Each component from data preprocessing to final evaluation is tested independently and as part of the full pipeline. The test cases are derived directly from the flow and structure represented in the class and sequence diagrams.

|  |  |  |  |
| --- | --- | --- | --- |
| ****Index**** | ****Expected Result**** | ****Test Description**** | ****Component**** |
| 1 | Clean graph with valid structure | Ensure that CitationNetwork removes duplicates and self-citations and produces the correct node-feature structure | CitationNetwork |
| 2 | Valid embedding vectors | Verify that the embeddings generated using either use\_gcn() or use\_graphsage() are correctly shaped and non-empty | GraphEmbeddingGenerator |
| 3 | Embeddings match input size | Ensure the number of embeddings equals the number of nodes provided in CitationNetwork | GraphEmbeddingGenerator |
| 4 | Nodes clustered into expected number of groups | Validate that KMeans assigns a cluster label to each node and respects the defined number\_of\_clusters | KMeansClustering |
| 5 | Expanded matrix constructed correctly | Check that the construct\_expanded\_matrix() method in SpecFProcessor builds a matrix using the fixed weights defined for edge and community relationships | SpecFProcessor |
| 6 | Signal is properly filtered | Ensure that apply\_spectral\_filtering() performs Fourier Transform and filtering without error and returns smoothed signals | SpecFProcessor |
| 7 | Deviations are computed accurately | Validate that compute\_deviation() returns a list of absolute differences between raw and filtered signals | SpecFProcessor |
| 8 | Anomalies are detected using the correct threshold logic | Confirm that detect\_anomalies() uses the formula: (mean + 2 \* std) / max to flag outliers | SpecFProcessor |
| 9 | Anomaly report is constructed correctly | Ensure that the report object is created and contains a list of detected anomalous nodes | AnomalyReport |
| 10 | Object is passed between components correctly | Check that the anomaly report is successfully passed from AnomalyReport to EvaluationEngine | AnomalyReport / EvaluationEngine |
| 11 | Consistency checks run correctly | Validate that consistency\_checks() assesses the validity and coherence of the anomaly report | EvaluationEngine |
| 12 | Final evaluation outputs result | Ensure that final\_evaluation() produces interpretive or metric-based results based on input report | EvaluationEngine |

# **8. Summary**

This research presents a novel framework for detecting anomalies in attributed graphs by combining graph embedding, clustering, and spectral filtering techniques. The system is built around core components including Graph Convolutional Networks (GCN), GraphSAGE, K-means clustering, and the SpecF algorithm.

The process begins with generating node embeddings using either GCN or GraphSAGE to capture structural and feature-based relationships in the graph. These embeddings are then clustered using K-means to uncover underlying community structures. Within each detected community, the SpecF algorithm applies spectral filtering to identify nodes whose signal patterns deviate significantly from the expected smoothness, indicating potential anomalies.

This integrated pipeline provides an unsupervised, modular approach to community-aware anomaly detection. It is designed to be scalable and interpretable, making it suitable for real-world applications such as citation networks, where subtle structural irregularities may signal emerging or unusual behaviors. The resulting system offers a valuable tool for analyzing graph-based data in a more context-sensitive and effective manner.

# **9. References**

1. Akoglu, L., Tong, H., & Koutra, D. (2014). [Graph based anomaly detection and description: A survey. Data Mining and Knowledge Discovery](https://arxiv.org/abs/1404.4679), 29(3), 626–688.
2. Blondel, V. D., Guillaume, J.L., Lambiotte, R., & Lefebvre, E. (2008). [Fast unfolding of communities in large networks](https://arxiv.org/abs/0803.0476). Journal of Statistical Mechanics: Theory and Experiment, 2008(10), P10008.
3. Cao, S., Lu, W., & Xu, Q. (2016). [Deep neural networks for learning graph representations](https://cdn.aaai.org/ojs/10179/10179-13-13707-1-2-20201228.pdf). In Proceedings of the AAAI Conference on Artificial Intelligence.
4. Donnat, C., Zitnik, M., Hallac, D., & Leskovec, J. (2018). [GraphWave: Node embedding via diffusion wavelets](https://www-cs.stanford.edu/~jure/pubs/graphwave-kdd18.pdf). In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 196–205).
5. Geng, H., Chen, C., He, Y., Zhang, S., Li, Y., Huai, M., Wang, H., & Yu, P. S. (2023). [Pyramid Graph Neural Network: A graph sampling and filtering approach for multi-scale disentangled representations.](https://dl.acm.org/doi/10.1145/3580305.3599478) In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 1729–1739).
6. Girvan, M., & Newman, M. E. J. (2002). [Community structure in social and biological networks](https://arxiv.org/abs/cond-mat/0112110). Proceedings of the National Academy of Sciences, 99(12), 7821–7826.
7. Grover, A., & Leskovec, J. (2016). [node2vec: Scalable feature learning for networks](https://arxiv.org/abs/1607.00653). In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 855–864).
8. Hamilton, W., Ying, R., & Leskovec, J. (2017). [Inductive representation learning on large graphs](https://arxiv.org/abs/1706.02216). In Advances in Neural Information Processing Systems, 30.
9. Kipf, T. N., & Welling, M. (2017). [Semi-supervised classification with graph convolutional networks](https://arxiv.org/abs/1609.02907). International Conference on Learning Representations (ICLR). <https://arxiv.org/abs/1609.02907>
10. Perozzi, A., Al-Rfou, R., & Skiena, S. (2014). [DeepWalk: Online learning of social representations](https://arxiv.org/abs/1403.6652). In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 701–710). <https://doi.org/10.1145/2623330.2623732>
11. Sandryhaila, A., & Moura, J. M. F. (2014). [Discrete signal processing on graphs: Frequency analysis](https://ieeexplore.ieee.org/document/6808520). IEEE Transactions on Signal Processing, 62(12), 3042–3054.

**Appendix A: Tools Used**

1. ChatGPT
2. Microsoft Word

**Appendix B: AI prompts used**

1. How should we pipeline the process of detecting communities to than provide it to the SpecF algorithm?.
2. What are node embeddings?.
3. What differences can we expect in the outputs of GraphSAGE and GCN in the entire pipeline process?
4. What clustering algorithms should we consider to produce communities from graph embeddings