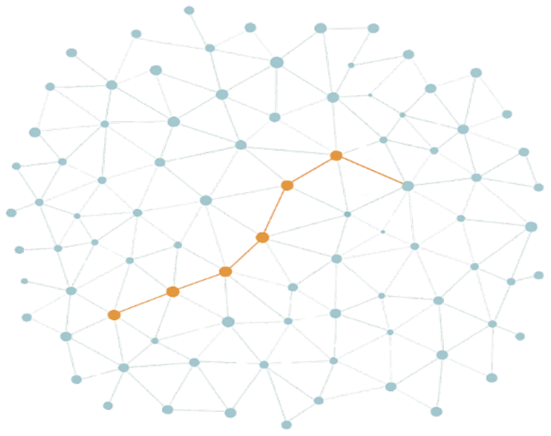
Multi-Hop Feature Quality Estimation for Unsupervised Graph Representation Learning



Capstone Project Phase A – 61998

25-2-R-10

Submitters:

Gad Azriel

Lidor Kupershmid

Supervisors:

Prof. Zeev Volkovich

Dr. Renata Avros

2025

Table of Contents

1. Abstract

This capstone project addresses the challenge of learning robust node representations from graph-structured data with noisy features, a common issue in real-world scenarios such as social networks, biomedical graphs, and financial systems.
We compare two approaches based on the recently introduced **Multi-Hop Feature Quality Estimation (MQE)** framework, which models node feature propagation through a conditional Gaussian distribution to estimate the quality of multi-hop features.
The first approach applies the original MQE method, focusing on its ability to learn noise-resilient node embeddings in a transductive setting. The second approach extends MQE by integrating **GraphSAGE**, a scalable and inductive Graph Neural Network architecture, to enable generalisation to unseen nodes and improved scalability.
This hybrid design is especially suited to large, noisy, and dynamic graphs, such as those found in social platforms or recommendation systems, where robustness and inductive capability are essential.

We experimentally evaluate both approaches across multiple datasets and noise conditions, analysing their performance in terms of robustness, inductive learning capability, and computational efficiency. Our findings reveal the trade-offs between model complexity, scalability, and noise resistance, and offer insights into how combining probabilistic modelling with neighbourhood sampling can improve real-world graph representation learning.
Keywords: Unsupervised Graph Representation Learning, MQE, GraphSAGE, Noise Robustness, Inductive Learning, Graph Neural Networks, Conditional Gaussian Modelling

2. Introduction

Unsupervised Graph Representation Learning (**UGRL**) is now an essential field in machine learning, making it possible to learn meaningful vector representations of graph-structured data without manual annotation. Such learned representations are commonly used in node classification, clustering, and graph-level prediction tasks. Most methods in UGRL are founded upon Graph Neural Networks (GNNs), which have shown excellent proficiency in capturing structural and relational knowledge.
Nonetheless, there are two significant issues with the current UGRL methods: they assume clean, complete, and reliable features for the nodes. Most real-world graphs present noisy, missing, or erroneous attributes. Individual profiles in social media can be missing or even intentionally wrong, citation networks may have OCR noise, and biological networks may have noise in measurements. Such noise in features is spread throughout the graph and may ruin the quality of the embedded nodes, particularly in the more modern GNN architectures.
Existing UGRL methods like contrastive learning or autoencoding often treat noisy features as equally trustworthy, causing the models to overfit or misclassify during unsupervised training. Highlighting the need for frameworks that explicitly model and mitigate feature noise during representation learning.

The **Multi-Hop Feature Quality Estimation** framework has been proposed to solve this, presenting an error-tolerant method for learning representations. The framework models the distribution over multi-hop propagated features with a conditionally Gaussian model, which enables it to value and discount low-quality information while learning. This leads to noise-robust node representations that are resistant to noisy input. Although robust, MQE is, by design, transductive and not scalable to large graphs or dynamic settings with unseen nodes. To address this, we plan to assess and augment **MQE** with **GraphSAGE**, an inductively learning-capable neighborhood-sampling GNN ideally suited for large graph applications.
In this study, we introduce and compare two methods:

1. MQE-only: The baseline method focusing solely on probabilistic multi-hop feature quality modeling.
2. MQE + GraphSAGE: A novel methodology that augments MQE with GraphSAGE to support scalable and inductive learning of representations for nodes.

We intend to assess both algorithms' robustness, effectiveness, and generality through large-scale experimentation on several different data sets with varying types of noise. The ensuing analysis can provide insights into constructing noise-robust and scalable graph learning models that support real-world applications like recommendation systems, fraud identification, analyzing social networks, and medical data modeling.

3. Related Work

Unsupervised Graph Representation Learning (UGRL) aims to learn expressive node embeddings without relying on labeled data. Early works such as **DeepWalk** [1] and **node2vec** [2] introduced random walk-based methods to capture graph structure, applying techniques like Skip-Gram to learn representations from co-occurrence statistics. While effective at preserving topological relationships, these methods could not incorporate node features and could not generalize to unseen nodes.
To address these limitations, **Graph Neural Networks** (GNNs) emerged as a robust framework for relational learning. Models such as **Graph Convolutional Networks** (GCNs), **Graph Attention Networks** (GATs), and **GraphSAGE** [3] extend deep learning to graph-structured data by enabling nodes to iteratively aggregate features from their neighborhoods. GraphSAGE introduced inductive learning by sampling node neighborhoods and learning aggregation functions, making it suitable for dynamic and large-scale graphs.
More recently, self-supervised learning methods have gained attention for their ability to learn robust embeddings through augmentation or reconstruction. Contrastive methods like **DGI** [4], **GRACE** [5], and **COSTA** [6] maximize mutual information between different views of the same graph to capture meaningful representations. However, these methods often assume clean input features and may struggle in real-world graphs that contain feature noise, missing values, or adversarial perturbations.
To address the limitations of noise sensitivity, generative approaches such as **MGAE** [7] and **GraphMAE** [8] have focused on reconstructing masked node features. These models are more tolerant of missing data but often incur higher computational costs and lack a principled mechanism for estimating or mitigating the quality of input features.
Multi-Hop Feature Quality Estimation (MQE) [9] introduced a novel unsupervised framework that models the distribution of multi-hop propagated features as a conditional Gaussian to tackle noisy feature learning directly. By estimating the mean and variance of each propagation step, MQE quantifies feature reliability and suppresses noise, leading to more robust embeddings. However, MQE operates in a purely transductive setting and cannot generalize to unseen nodes or dynamic environments.
Our work addresses this limitation by evaluating and extending MQE in two directions. First, we analyze the baseline MQE approach in isolation to assess its noise robustness. Second, we propose a hybrid model that integrates GraphSAGE into MQE, combining probabilistic quality estimation with inductive neighborhood sampling. This integration is designed to achieve robust, scalable, and generalizable graph representation learning under noise, an area that, to our knowledge, remains underexplored in the literature.

4. Background

Graph-based data is central to many real-world applications, from social networks and recommendation systems to biological networks and citation graphs. Learning effective representations from these structures is crucial for enabling predictive and analytical tasks.

This section outlines the fundamental concepts that underpin this project, including graph representation learning, the basics of Graph Neural Networks (GNNs), the challenge of noisy features, and the importance of inductive learning.

4.1. Graph Representation Learning

Graph representation learning is the domain of machine learning that deals with learning graph-structured data and transforming it into informative vector representations useful for various downstream tasks like node classification, link prediction, clustering, and graph-level analysis. Graphs naturally represent the connections and interactions among real-world systems like social networks, biological systems, citation graphs, and recommendation systems, where entities are represented as nodes and interactions as edges. Conventional machine learning techniques usually cannot straightforwardly handle graph data because of its non-regular structure and interdependencies. This issue is resolved in Graph Representation Learning, as it encodes the topological structure and node and edge attributes into lower-dimensional embeddings.

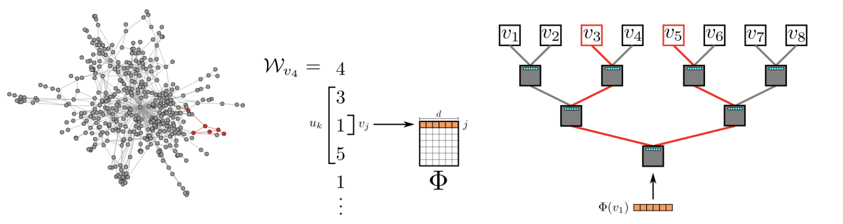
These embeddings aim to preserve the graph's global and local structure while making the data more compatible with standard machine learning models.

Early approaches such as **DeepWalk** and **node2vec** utilized random walks to discover node neighborhoods and learn node representations from patterns of co-occurrences, inspired by natural language models. More recently, Graph Neural Networks (GNNs), which pool neighboring node context in a learnable process, have become the new standard, allowing the models to encode rich, contextual node relations in the graph.

Broadly speaking, Graph Representation Learning offers the tools necessary to learn and analyze complex relational data in an efficient, scalable, and task-awarely.

4.1.1 DeepWalk

DeepWalk, introduced by Perozzi et al. (2014), is one of the earliest unsupervised methods for learning latent representations of nodes in a graph. It operates by simulating truncated random walks over the graph to generate sequences of nodes, which are then treated analogously to sentences in a language corpus. These node sequences are used to train a Skip-Gram model, originally developed for word embeddings in natural language processing, to predict a node's neighbors within a fixed-size context window. By doing so, DeepWalk effectively captures local neighborhood proximity and global structural patterns, such as community memberships and hubs. This method has the advantage of being scalable and parallelizable, making it suitable for large graphs. However, it is inherently transductive, meaning embeddings can only be learned for nodes seen during training. Furthermore, DeepWalk relies solely on the graph's topology and does not integrate node features, which limits its applicability in feature-rich domains such as social networks or biological systems.



(a) Random walk generation. (b) Representation mapping. (c) Hierarchical Softmax.

Fig. 1. *Illustration of the DeepWalk architecture. The process includes (a) random walk generation on the input graph, (b) representation learning through context window sampling, and (c) training with hierarchical softmax to compute node co-occurrence probabilities efficiently.*

4.1.2 Node2vec

Node2vec, proposed by Grover and Leskovec (2016), extends DeepWalk by introducing a biased random walk strategy that enables more nuanced exploration of graph neighborhoods. It introduces two hyperparameters, ppp and qqq, which control the likelihood of revisiting a node (depth-first search behavior) versus exploring new nodes (breadth-first search behavior). This flexibility allows node2vec to capture homophily (nodes with similar features tend to connect) and structural equivalence (nodes with similar roles in different parts of the graph), which are crucial for many real-world graph analytics tasks.

Like DeepWalk, node2vec also uses the Skip-Gram model to learn embeddings, but the modified walk strategy leads to more expressive and task-adaptable representations. The embeddings produced are helpful for a variety of downstream tasks such as node classification, link prediction, and community detection. However, node2vec shares some limitations with DeepWalk: it is still a transductive method, does not utilize node features, and does not generalize well to dynamic graphs or previously unseen nodes. Additionally, tuning the random walk parameters requires task-specific experimentation and can introduce computational overhead.

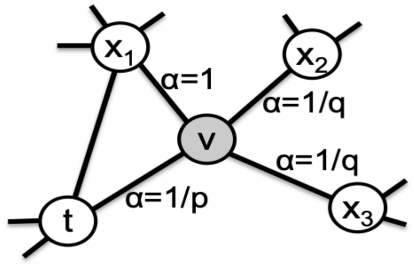


Fig. 2. *Node2Vec biased random walk guided by return parameter pp and in-out parameter qq, balancing BFS and DFS.*

4.2. Graph Neural Networks (GNNs)

Graph Neural Networks, or GNNs, extend deep learning to graph-structured data by enabling each node to learn from its neighbors through an iterative aggregation process. Unlike traditional data formats like grids or sequences, graphs represent entities as nodes and their relationships as edges, which makes the direct application of standard neural networks ineffective. GNNs address this by introducing a mechanism where, in each layer, a node updates its representation based on its features and the features of its neighboring nodes. At the heart of this process are two key functions: an aggregation function that gathers information from a node's neighbors, and an update function that combines this information with the node's current state. This allows the model to learn how each node should evolve based on its local context. By stacking multiple layers, GNNs enable nodes to capture information from multi-hop neighborhoods, leading to more expressive and informative embeddings. Different GNN variants implement these functions in distinct ways. For instance, GCN uses normalized averaging of neighbor features, GAT applies attention mechanisms to weigh neighbors differently, and GraphSAGE uses learnable aggregation methods like mean or LSTM to support inductive learning. Together, these models make GNNs suitable for a wide range of tasks on graphs, including node classification, link prediction, and recommendation.

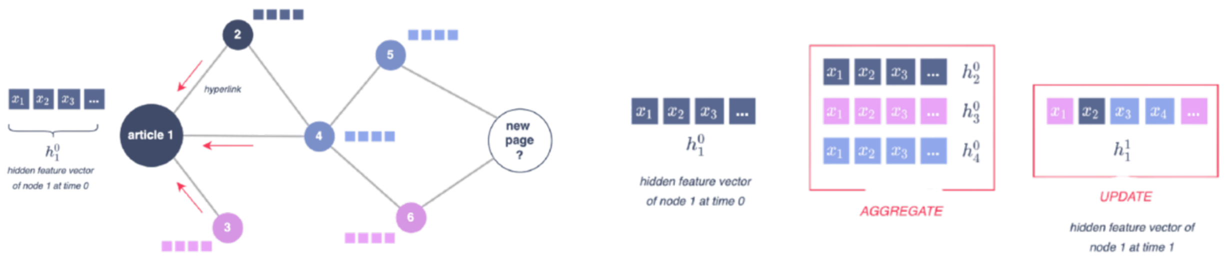


Fig. 3. *GNN message passing: a node updates its embedding by aggregating neighbor features (Link).*

4.3. The Challenge of Noisy Features

One of the significant challenges in using Graph Neural Networks (GNNs) on real-world data is handling noisy, missing, or unreliable node features. The vast majority of common GNN architectures assume that node attributes are clean and reliable.

This assumption, however, does not always hold in most real-world situations. For example, in social networks, users sometimes deliberately or inadvertently give false data, not fill in all the details, or use inconsistent data formats such as listing non-existent locations, leaving bios blank, or including emojis and slang in profile descriptions. In citation networks, the text of academic papers might contain OCR errors or inconsistent terminology, while in biological networks, experimental noise might result in inaccurate gene expression measurements. In recommendation systems, user-item interaction data might be sparse or skewed due to external influences like seasonal patterns or the activity of bots.

Because GNNs depend heavily on aggregating features from a node's local neighborhood, any amount of noise can spread through the network layers and weaken the quality of the learned node representations. This issue is even more pronounced in deeper GNN architectures, where noisy information is propagated and amplified over multiple hops, often leading to over-smoothing or feature contamination. As a result, the model may struggle to differentiate between nodes or learn functional structural patterns, ultimately reducing performance in tasks like node classification, link prediction, or community detection.

To address this, recent research has focused on developing noise-resilient GNN frameworks that can assess the quality of input features, reduce the impact of unreliable data, and place greater emphasis on more trustworthy signals. These efforts aim to make graph-based learning more robust and applicable in real-world settings where data imperfections are inevitable.

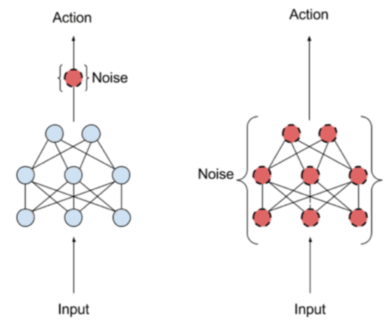


Fig. 4. *Noise added at input (left) versus amplified across GNN layers (right). (Link)*

4.4. Feature Propagation in GNNs

Feature propagation is at the heart of Graph Neural Networks (GNNs) work. It allows each node in the graph to gradually improve its internal representation by drawing information from its neighbors. In each layer of a GNN, a node receives feature vectors from the nodes directly connected to it. These incoming features are aggregated using operations like averaging, summing, or attention mechanisms, then merged with the node's current features. As more layers are stacked, information travels beyond immediate neighbors, capturing a broader view of the graph's structure.

Formally, the updated representation of a node at a given layer is calculated using a combination of its neighbors' features from the previous layer. The general form of this propagation rule is expressed in the following equation: $x_i^{(l)} = \sum_{v_i \in N(v_i)} a_{ij} \cdot x_j^{(l-1)}$. In this equation, the weights typically come from a normalized adjacency matrix, and the neighbor set defines which nodes contribute to the update. The process starts with the initial input features and is repeated for a set number of layers.

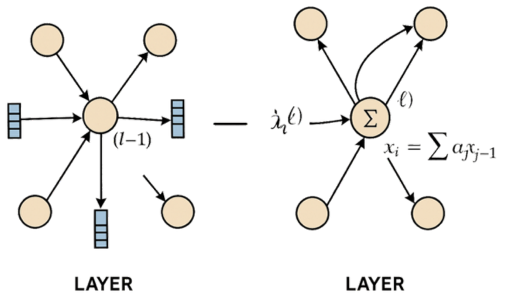


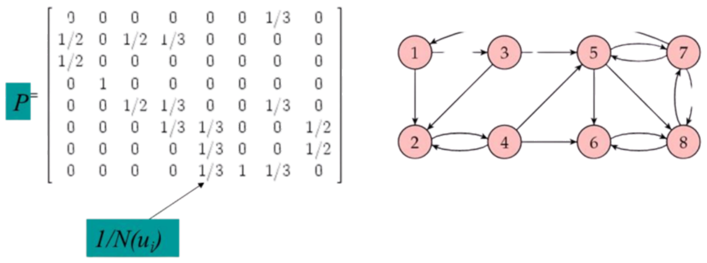
Fig. 5. *Feature propagation in a Graph Neural Network (GNN): each node updates its representation by aggregating features from its neighbors. (Link)*

4.5. Normalized Adjacency Matrix

In Graph Neural Networks, the adjacency matrix A defines how nodes are connected. Each entry A_{ij} is 1 if there's an edge between node i and node j, and zero if there isn't. While this raw form captures the graph's structure, using it directly during propagation can cause problems. Nodes with very few or very many connections might dominate or be overshadowed, leading to numerical instability and skewed feature aggregation. GNNs typically use a normalized version of the adjacency matrix to overcome this. This adjusts how connected each node is so that information from neighboring nodes is weighted more fairly during message passing.

The most commonly used version is the symmetric normalized adjacency matrix, defined as: $\hat{A} = D^{-\frac{1}{2}} \cdot (A + I) \cdot D^{-\frac{1}{2}}$ In this equation, D is a diagonal matrix where each entry represents the degree (i.e., number of connections) of a node, and I is the identity matrix, which adds self-loops allowing each node to include its features in the update process.

This normalization makes feature propagation behave like a smooth filter, similar to how Laplacians work in graph theory. It stabilizes learning, especially in deeper GNNs, and aligns well with concepts from graph signal processing. It helps reduce noisy, high-frequency signals and keeps the focus on smoother, more meaningful patterns across the graph.



$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 \\ 1/2 & 0 & 1/2 & 1/3 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 1 & 1/3 & 0 \end{bmatrix}$$

Fig. 6. *Symmetric normalized adjacency matrix $\hat{A} = D^{-\frac{1}{2}} \cdot (A + I) \cdot D^{-\frac{1}{2}}$ with a graph example. Shows how self-loops are added and edge weights adjusted based on node degrees to balance feature propagation. (Link)*

4.6. Gaussian Feature Estimation

Gaussian feature estimation brings a probabilistic approach to modeling node features by treating them as uncertain values, rather than fixed and equally reliable. Each feature is represented as a sample from a Gaussian distribution, which helps the model handle noise, missing data, or inconsistencies typical in real-world graphs like social networks or biological systems.

In this setup, a node's features are defined by a mean (expected value) and a variance (confidence level), often assumed to be independent across dimensions for simplicity. $x \sim N(\mu, \Sigma)$. Here, x is the feature vector, μ is the learned mean, and Σ reflects uncertainty. Features with high variance are treated as less reliable and have less influence during learning, while more certain features play a larger role. This allows the model to focus on trustworthy information, improving robustness and stability in noisy environments.

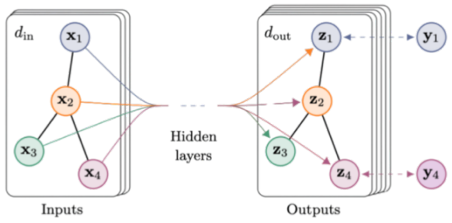


Fig. 7. *Gaussian feature estimation: node features are modeled as distributions with learned means and variances, helping the model reduce the impact of uncertain inputs in noisy graphs. (Link)*

4.7. Graph Convolutional Networks (GCNs)

Graph Convolutional Networks (GCNs), introduced by Kipf and Welling (2017), represent a seminal advancement in graph representation learning by generalizing convolutional neural networks to non-Euclidean domains such as graphs. Unlike earlier embedding methods based solely on random walks, GCNs learn node representations by recursively aggregating and transforming feature information from local neighborhoods. The core mechanism involves a spectral or spatial convolution operation, where each node updates its representation as a weighted combination of its features and those of its neighbors, typically using a normalized adjacency matrix. This approach allows GCNs to incorporate graph structure and node attributes in a learnable, end-to-end manner. While highly effective for semi-supervised learning tasks on static graphs, traditional GCNs operate in a transductive setting, meaning they cannot generalize to nodes or graphs unseen during training. Furthermore, GCNs may suffer from over-smoothing when stacking many layers, leading to indistinguishable node embeddings. Despite these limitations, GCNs laid the foundation for numerous subsequent architectures and remain a cornerstone in designing deep graph learning models.

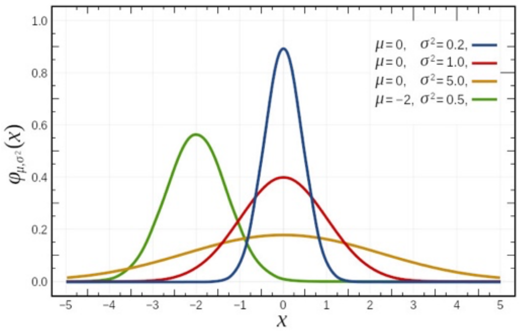


Fig. 8. *A multi-layer GCN where each node successively aggregates and transforms its neighborhood features via the normalized adjacency matrix (Link)*

4.8. Graph Attention Networks (GATs)

Graph Attention Networks (GATs), introduced by Veličković et al. (2018), extend the graph convolutional paradigm by incorporating an attention mechanism into the message-passing framework. Unlike Graph Convolutional Networks (GCNs), which apply uniform or pre-defined weights when aggregating neighborhood information, GATs dynamically compute attention coefficients for each neighboring node, allowing the model to assign varying levels of importance to different neighbors. This is achieved through self-attention operations, where the relevance of a neighbor's features is learned based on their similarity to the target node in the latent space. The attention mechanism improves the model's capacity to focus on informative or semantically aligned neighbors, making it particularly effective in graphs with heterogeneous or noisy structures. Moreover, GATs support inductive learning and can handle variable neighborhood sizes without requiring costly matrix operations. GATs significantly enhance the expressiveness and adaptability of graph neural networks by enabling fine-grained, data-dependent weighting of neighborhood contributions.

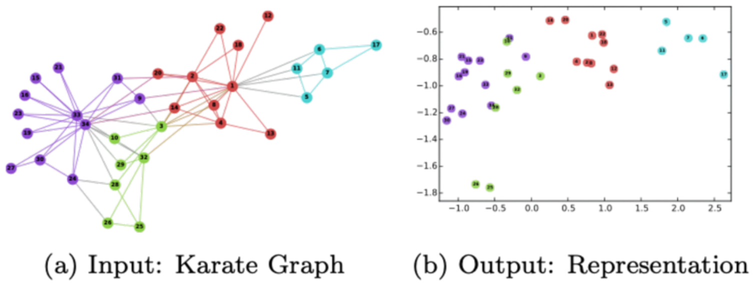


Fig. 9. *GAT architecture: Left – attention coefficients computed between node pairs. Right – multi-head attention lets each node weight neighbors differently before aggregation. (Link)*

4.9. Skip-Gram: Foundation for Node Embedding in Graphs

The Skip-Gram model, initially developed for word embedding in natural language processing (Mikolov et al., 2013), is a foundational technique for early graph representation learning methods such as DeepWalk and node2vec. In graphs, Skip-Gram is applied by treating nodes as words and random walks as sentences, effectively translating the structural context of nodes into a sequence-based format. Given its embedding, the model is trained to maximize the probability of a node's neighborhood (context), thereby capturing local structural and co-occurrence patterns. This unsupervised objective allows the learned embeddings to reflect topological proximity and shared community membership within the graph. Despite its success in preserving structural features, the Skip-Gram-based approach does not inherently incorporate node attributes. It cannot generalize to unseen nodes, which limits its applicability in dynamic or feature-rich graphs. Nonetheless, it laid the groundwork for subsequent advances in graph neural networks and contrastive learning techniques by highlighting the value of context-driven embedding generation.

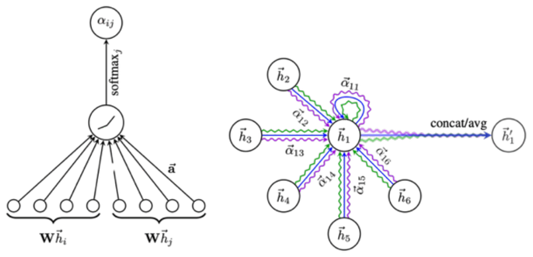


Fig. 10. *Skip-Gram–based embedding: nodes from the Karate graph (left) are embedded into a 2D space (right) using random walks and neighborhood prediction, capturing community structure and topological proximity. (Link)*

4.10. Meta-Representation Z

Meta-representation Z is a shared latent variable assigned to each node in a graph, designed to capture the underlying structure and semantics of the node's local and global context. It is learned jointly with a neural network model that estimates the statistical parameters—mean (μ) and standard deviation (σ)—across multiple hops or layers in the graph. This allows the model to account for uncertainty and variability in the data, making Meta-representation Z a robust and noise-aware final representation for each node. It aims to enable more accurate and generalizable tasks like classification, clustering, and link prediction within graph-based systems.

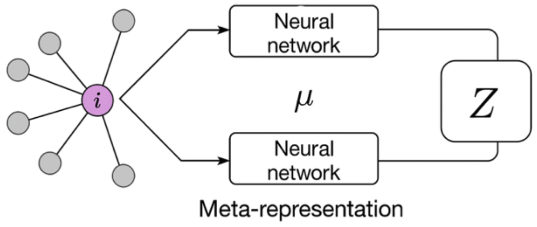


Fig. 11. *Meta-representation Z is learned per node by estimating mean (μ) and variance through neural networks, capturing uncertainty and structure from the node's neighborhood.*

4.11. Loss Function: Reconstruction Likelihood

The core learning objective of MQE is guided by a reconstruction likelihood loss function, which aims to accurately reconstruct the propagated node features while accounting for their estimated uncertainty. Specifically, for each propagation depth l, the model seeks to maximize the likelihood that the observed feature X(l) was drawn from the predicted Gaussian distribution N ((l),(l)). This probabilistic formulation ensures that the model not only predicts the central tendency of the features but also captures the confidence of its estimation through the standard deviation. To avoid trivial solutions such as the inflation of standard deviation values (l) to minimize the loss without meaningful learning a regularization term is included. This term penalizes excessively large variances and encourages the model to produce well-calibrated uncertainty estimates. The combined objective thus balances reconstruction accuracy with principled uncertainty quantification, reinforcing MQE's robustness in handling noisy or unreliable feature inputs.

4.12. Noise Intensity Estimation

An additional advantage of the MQE framework lies in its ability to estimate the noise intensity associated with individual nodes. This is achieved through the standard deviation (0) derived from the initial (unpropagated) feature distribution. Since (0) it captures the model's confidence in the raw node features, it is a direct indicator of feature noise. Nodes with higher values (0) are interpreted as having greater uncertainty or inconsistency in their input features, allowing MQE to localize and quantify noise at the node level. This property is particularly valuable for diagnostic analysis, enabling practitioners to identify unreliable regions within the graph. Moreover, the node-level noise scores can support auxiliary applications such as data cleaning, selective feature dropout, or the design of noise-aware training regimes that weight nodes differently based on their inferred reliability.

4.13. GraphSAGE: Inductive Graph Representation Learning

GraphSAGE (Graph Sample and Aggregate) is a pioneering framework designed to enable inductive learning on graph-structured data, addressing the limitations of earlier transductive models that required full access to the graph during training. Introduced by Hamilton et al. (2017), GraphSAGE generates node embeddings by sampling a fixed-size set of neighbors and aggregating their features through a learnable function. This strategy allows the model to generalize to previously unseen nodes or entirely new graphs at inference time, making it highly scalable and practical for large or dynamic graph scenarios. Unlike traditional methods such as DeepWalk or GCNs, which depend on static graph structures, GraphSAGE learns transferable functions rather than embedding lookups. By employing aggregation functions like mean, LSTM, or pooling, GraphSAGE preserves local structure while remaining computationally efficient. Its inductive capability makes it especially suitable for real-world applications such as recommender systems, web-scale social networks, and biological data, where graphs are often incomplete or evolve over time.

5. Project Overview

5.1. MQE Standalone Approach

The engineering design of the Multi-Hop Feature Quality Estimation (MQE) algorithm is systematically organized into three core components: Graph Structure Augmentation, Augmented Multi-Hop Feature Propagation, and Feature Quality Estimation. Each stage contributes to a robust and noise-resilient representation learning framework, enabling effective operation in real-world graphs with incomplete or noisy data.

5.1.1. Initialization and Input Preprocessing

The MQE framework begins with a graph $G = (V, E)$ defined by its adjacency matrix $A \in R^{N \times N}$ and node feature matrix $A \in R^{N \times F}$, which are the number of nodes and features per node. Additionally, a parameter L is defined to specify the number of propagation layers (i.e., hops). This stage ensures that all required structural and feature-based components are provided for downstream processing.

5.1.2. Graph Structure Augmentation

Before any feature propagation occurs, the adjacency matrix A is normalized to ensure numerical stability and to reduce bias from node degrees. Specifically, symmetric normalization is applied, $\hat{A} = D^{-\frac{1}{2}} \cdot A \cdot D^{-\frac{1}{2}}$ where D is the diagonal degree matrix of A. This normalization step standardizes feature aggregation during propagation. Following normalization, a **k-Nearest Neighbors** (kNN) graph augmentation is performed. The kNN algorithm constructs an enhanced graph structure by identifying the k closest nodes to each node, based on either Euclidean distance in feature space or other graph-based metrics. The resulting adjacency matrix A^* represents the augmented graph, capturing both structural and feature-proximity-based relationships. This matrix is later used for feature propagation.

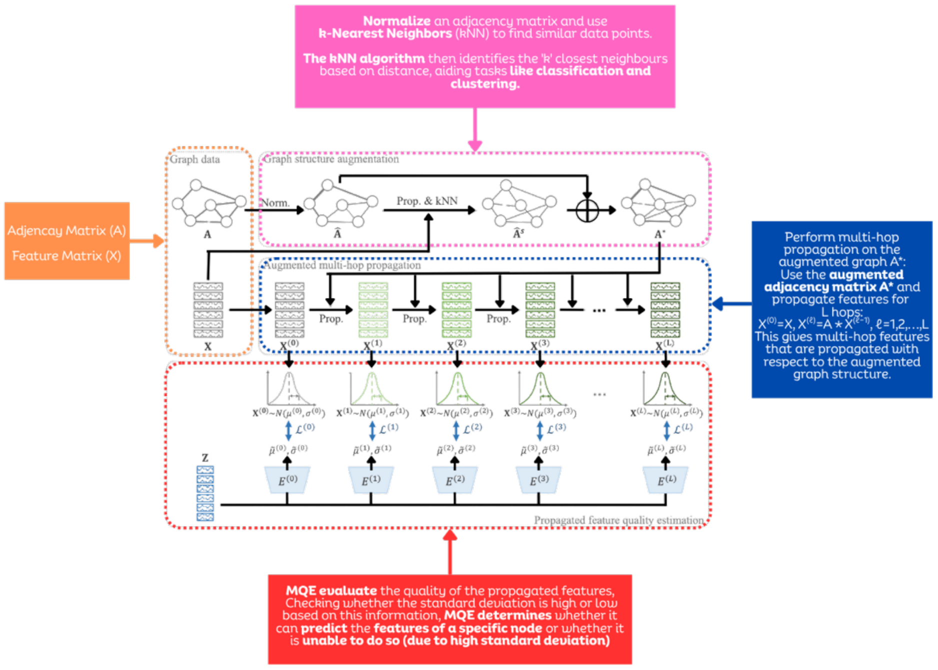
5.1.3. Augmented Multi-Hop Feature Propagation

Once A^* is computed, the node features are propagated through multiple layers to capture high-order neighborhood information.

The propagation is formulated recursively for each hop l as: $X^{(0)} = X, X^{(l)} = A^* \cdot X^{(l-1)}\ for\ l = 1,2,\dots,L$ This results in a sequence of propagated feature matrices $\{X^{(0)}, X^{(1)}, \dots, X^{(L)}\}$, each representing node features aggregated from an lll-hop neighborhood. Using the augmented adjacency matrix A^* ensures that the propagation respects both the original and the learned graph topology.

5.1.4. Feature Quality Estimation via Conditional Gaussian Modeling

The final and most novel component of MQE is the statistical estimation of feature reliability across propagation layers. Instead of treating all propagated features equally, MQE models each feature matrix $X^{(l)}$ as a sample from a **Gaussian distribution**: $X^{(l)} \sim N\left(\mu^{(l)}, \sigma^{(l)}\right)$. Here, $\mu^{(l)} \in R^{(N \times F)}$ and $\sigma^{(l)} \in R^{(N \times F)}$ denote the mean and standard deviation of the propagated features at layer l, both of which are learned under the supervision of a meta-representation Z. This meta-representation helps the model assess the consistency of features over different hops. Features with high variance $\left(\sigma^{(l)}\right)$ are considered noisy or unreliable and may be downweighted or discarded. Conversely, features with low variance are considered trustworthy and are retained for downstream tasks. This enables MQE to dynamically suppress noise and enhance robustness in the presence of corrupt, adversarial, or missing input features. By integrating graph augmentation, deep propagation, and variance-aware filtering, the MQE engineering process offers a scalable and noise-resilient approach to unsupervised graph representation learning.


Normalize an adjacency matrix and use **k-Nearest Neighbors** (kNN) to find similar data points.

The kNN algorithm then identifies the 'k' closest neighbours based on distance, aiding tasks **like classification and clustering.**

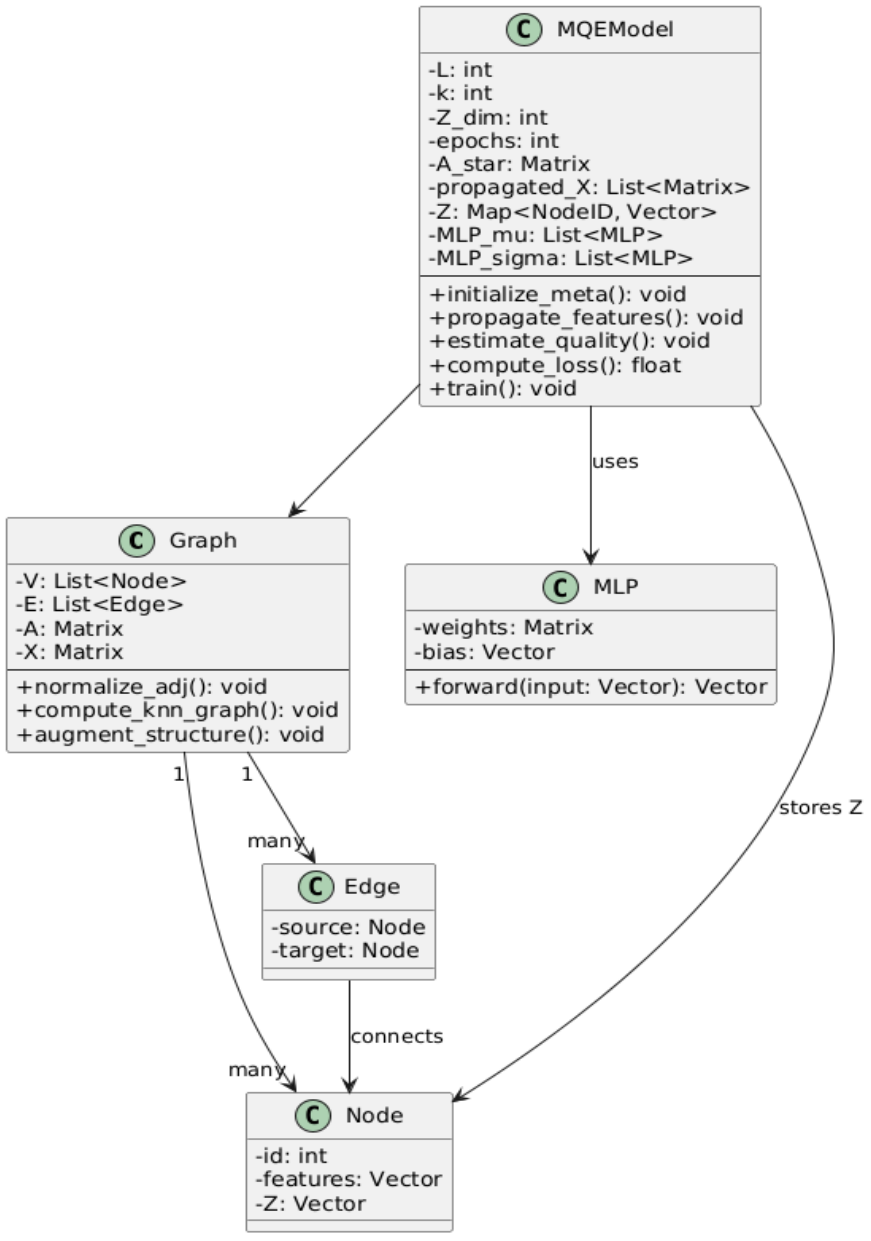
Perform multi-hop propagation on the augmented graph A*: Use the **augmented adjacency matrix A*** and propagate features for L hops: $X^{(0)}=X, X^{(\ell)}=A*X^{(\ell-1)}, \ell=1,2,\ldots,L$ This gives multi-hop features that are propagated with respect to the augmented graph structure.

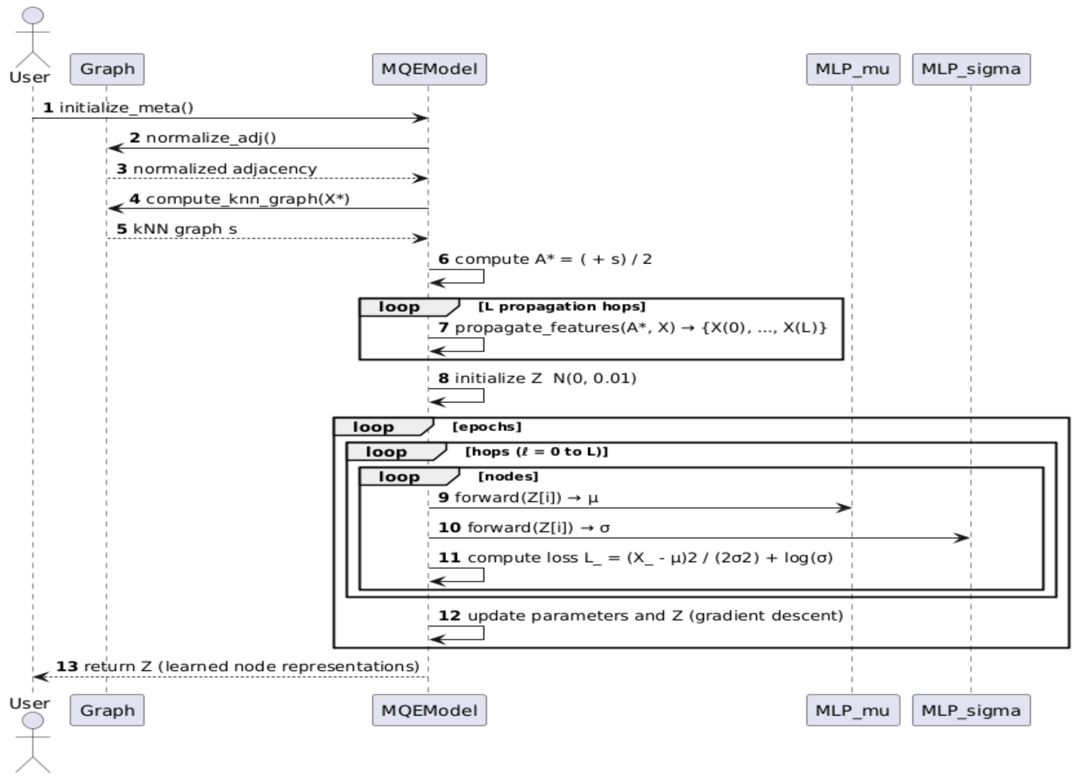
MQE evaluate the quality of the propagated features, Checking whether the standard deviation is high or low based on this information, **MQE determines** whether it can **predict** the **features of a specific node** or whether it is **unable to do so (due to high standard deviation)**

5.1.5. Pseudocode

```
Input:
    ■ G = (V, E): Input graph with nodes V and edges E
    ■ X: Node feature matrix
    ■ L: Number of propagation hops
    ■ k: Number of neighbors for kNN graph
    ■ Z_dim: Dimension of meta-representation Z
    ■ epochs: Number of training iterations

Output ⇒ Z: Final node representations

Step 1: Graph Structure Augmentation
    ■ Compute the normalized adjacency matrix $\hat{A}$ from $A$
    ■ Perform multi-hop propagation $\hat{A}$ to get features $\{X^0, X^1, \ldots, X^L\}$
    ■ Compute the summed feature vector $X^* = \sum_{l=0}^{L} X^l$
    ■ Construct a kNN graph using cosine similarity on $X^*$
    ■ Compute normalized $As \rightarrow \hat{A}s$
    ■ Compute the augmented adjacency matrix $A^* = \frac{(\hat{A} + \hat{A}_s)}{2}$

Step 2: Augmented Multi-Hop Feature Propagation
        Using $A^*$, compute multi-hop propagated features: $\{\hat{X}^{(0)}, \hat{X}^{(1)}, \ldots, \hat{X}^{(L)}\}$
Step 3: Initialize Meta Representations
        For each node i:
            Initialize $Z_i \sim N(0, 0.01)$
            Where $Z_i \in R^{Z_{dim}}$
Step 4: Train Estimators via Quality Estimation
```

```
For each hop ℓ from 0 to L:
    Define two MLPs:
        $E_{\mu}^{l}(Z_i) \rightarrow Estimate\ mean\ \mu_i^{(l)}$
        $E_{\sigma}^{l}(Z_i) \rightarrow Estimate\ std\ \sigma_i^{(l)}$

For epoch in 1 to epochs:
    For each ℓ in 0 to L:
        For each node i:
            Compute reconstruction loss:
                $L_i^{(l)} = \hat{X}_i^{(l)} - \frac{\mu_i^{(l)2}}{2\cdot\sigma_i^{(l)2}} + log\ log\ (\ \sigma_i^{(l)})$
    Total Loss = sum over all $L_i^{(l)}$
    Update model parameters and $Z$ use gradient descent

Return ⇒ Learned meta-representations $Z$ for all nodes
```




User
Graph
MQEModel
MLP_mu
MLP_sigma
1 initialize_meta()
2 normalize_adj()
3 normalized adjacency
4 compute_knn_graph(X*)
5 kNN graph s
6 compute A* = (+ s) / 2
loop [L propagation hops]
7 propagate_features(A*, X) → {X(0), ..., X(L)}
8 initialize Z N(0, 0.01)
loop [epochs]
loop [hops (ℓ = 0 to L)]
loop [nodes]
9 forward(Z[i]) → μ
10 forward(Z[i]) → σ
11 compute loss L_ = (X_ - μ)2 / (2σ2) + log(σ)
12 update parameters and Z (gradient descent)
13 return Z (learned node representations)

5.2. GraphSAGE-Augmented MQE Approach

The second design of the Multi-Hop Feature Quality Estimation (MQE) framework uses GraphSAGE-based propagation as a central part of the architecture. Breaking away from the initial matrix-based propagation in traditional MQE, this design makes use of learned neighborhood aggregation using GraphSAGE to promote generalization, facilitate inductive learning, and add more robustness to noise. In contrast to the isolated MQE, where feature propagation has to be recomputed on the entire graph when a new node is incorporated, the design facilitates more dynamic and scalable inference, allowing the model to compute embeddings of unseen nodes without reprocessing the graph. The overall pipeline is organized into four primary components: Input Initialization, Graph Structure Augmentation, GraphSAGE-Based Multi-Hop Feature Propagation, and Feature Quality Estimation through Conditional Gaussian Modeling.

5.2.1. Initialization and Input Configuration

The process begins with a graph $G = (V, E)$, defined A by its adjacency matrix and node feature matrix $X \in R^{N \times F}$, which is the number of nodes and features per node. A parameter L is specified to control the number of propagation layers (or hops). This setup ensures the necessary topological and feature-based context for downstream processing.

5.2.2. Graph Structure Augmentation

As in the standalone MQE, the graph structure is enriched to capture both structural and semantic similarities. The adjacency matrix A is symmetrically normalized $\hat{A} = D^{-\frac{1}{2}} \cdot A \cdot D^{-\frac{1}{2}}$ to stabilize feature scaling during propagation. In parallel, a k-nearest neighbors (kNN) graph is constructed based on feature-space similarity, producing an augmented neighborhood structure. However, unlike the original MQE, this matrix is not directly used for linear propagation, but rather to inform the sampling process in GraphSAGE.

5.2.3. GraphSAGE-Based Multi-Hop Feature Propagation

The core propagation mechanism is replaced with GraphSAGE, which computes node representations by aggregating features from a sampled set of neighbors at each hop. Let ${h_v}^{(0)} = x_v$ be the initial features of the node v. For each propagation layer $l \in \{1,2,\ldots,L\}$, node embeddings are updated as:

- $h_{N_{(v)}}^{(l)} = AGGREFATE^{(l)}(\{h_u^{(l-1)} \mid u \in N(v)\})$
- $h_v^{(l)} = \sigma \cdot \left(W^{(l)} \cdot CONCAT\left({h_v}^{(l-1)}, {h_{N_{(v)}}}^{(l)}\right)\right)$

Where $N_{(v)}$ it denotes the neighbors of node , AGGREGATE is a learnable function (e.g., mean, LSTM, or pooling), and it is a trainable weight matrices. This recursive propagation yields a sequence of feature embeddings $\{h^{(0)}, h^{(1)}, \ldots, h^{(L)}\}$ that encodes increasingly high-order neighborhood information.

5.2.4. Feature Quality Estimation via Conditional Gaussian Modeling

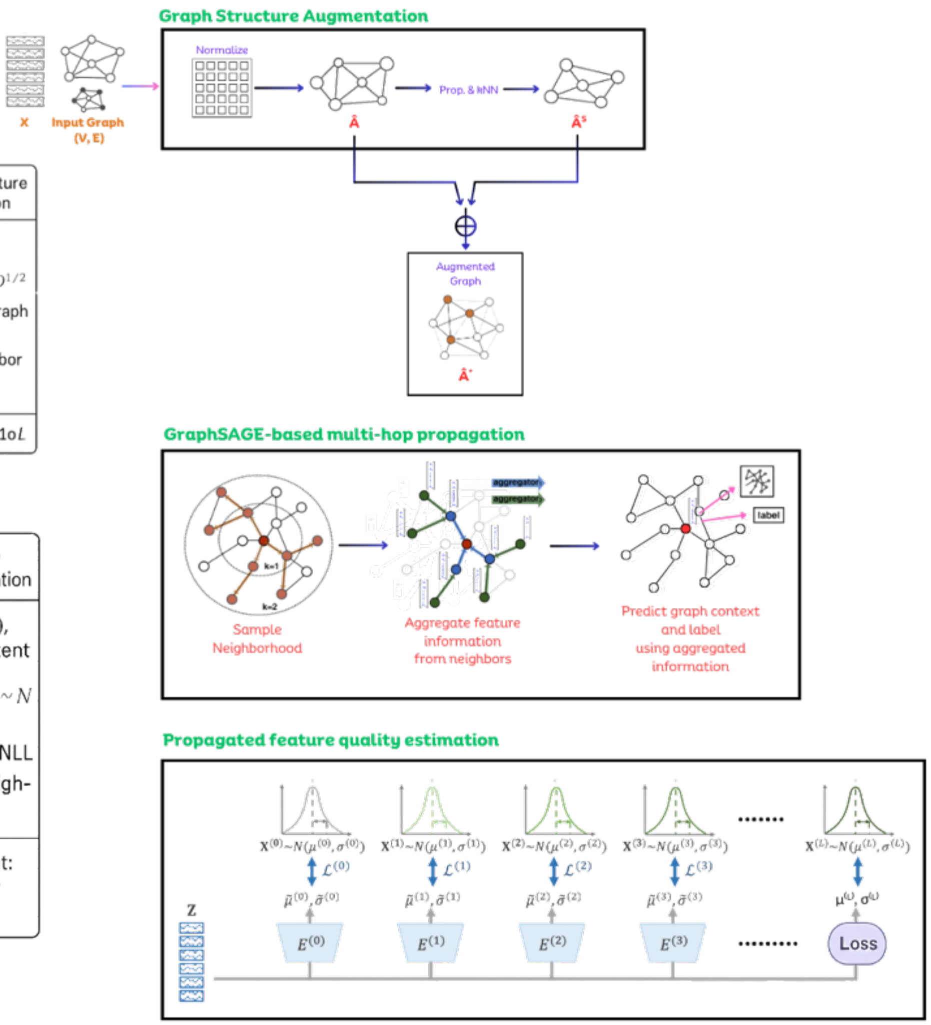
Following each propagation layer, the model estimates the statistical reliability of the propagated features. Each layer's output $h^{(l)}$ is modeled as a sample from a Gaussian distribution $h^{(l)} \sim N(\mu_1, \sigma_1)$. Where $\mu_1, \sigma_1 \in R^{N\times F}$ are learned representations of the mean and standard deviation of the embeddings at layer l. These parameters are learned under the supervision of a shared latent meta-representation Z, which captures common structural and semantic patterns across hops.

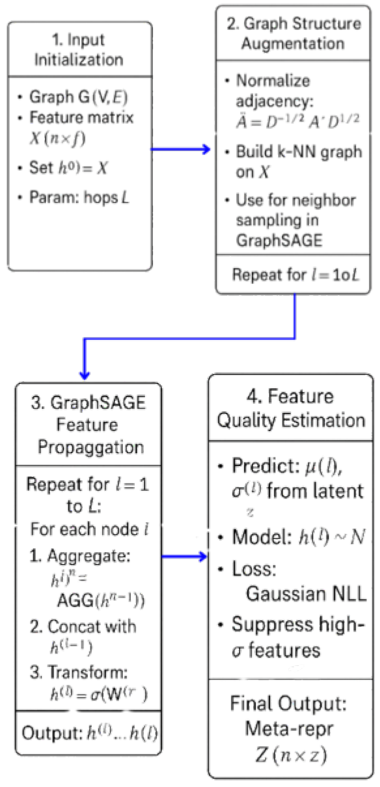
Nodes or features with high variance (i.e., high uncertainty in σ_1) are down weighted or discarded, while features with low variance are retained as reliable. This variance-aware mechanism allows the model to suppress noisy signals even in the context of inductively learned embeddings.

5.2.5. Pseudocode: GraphSAGE-Augmented MQE

```
Input:
    ▪ $G = (V, E)$: Input graph with nodes and edges
    ▪ $X \in R^{N\times F}$: Node feature matrix
    ▪ $L$: Number of GraphSAGE layers (i.e., how many hops of neighborhood to aggregate)
    ▪ $Z_{dim}$: Dimension of meta-representation vector $Z$
    ▪ epochs: Number of training iterations
    ▪ AGG_FUNC: Aggregation function (e.g., mean, max, or LSTM)

Output ⇒ $Z \in R^{N\times Z_{dim}}$: Final node-level meta-representations capturing reliable information

Step 1: Initialization
    For each node $i \in V$:
        Set initial node embedding: ${h_i}^{(0)} = X_i$        Initialize latent meta-representation: $Z_i \sim N(0,0.01)$
        For each layer $l = 1\ to\ L$:
            Initialize the trainable weight matrix $W^{(l)}$
            Initialize two MLPs for each layer:
                ${E_\mu}^{(l)}(Z_i) \to {\mu_i}^{(l)}$        ${E_\sigma}^{(l)}(Z_i) \to {\sigma_i}^{(l)}$

Step 2: GraphSAGE-Based Feature Propagation
    For each layer $l = 1\ to\ L$:
        For each node $i \in v$:
```

```
                    Sample a fixed-size set of neighbors $N(i)$
                    Aggregate neighbors' embeddings from the previous layer:
                        $h^{(l)}{}_{N(i)} = AGG\_FUNC\ (\{h_j{}^{(l-1)} \mid j \in N(i)\})$
                        Concatenate with the node's own previous embedding and apply transformation:
                        $h_i{}^{(l)} = \sigma(W^{(l)} \cdot\ CONCAT(h_i{}^{(l-1)}, h_{N(i)}{}^{(l)}$
          After all $L$ layers, store all node embeddings: $\{h^{(0)}, h^{(1)}, \dots, h^{(L)}\}\}$

Step 3: Feature Quality Estimation (Gaussian Modeling)
          For each training epoch:
               For each layer $l = 0\ to\ L$:
                    For each node $i \in V$:
                         Estimate feature distribution parameters: $\mu_i{}^{(l)} = E_u{}^{(l)}(Z_i), \sigma^{(l)} = E_\sigma{}^{(l)}(Z_i)$
                         Compute the Gaussian negative log-likelihood loss:
                              $L_i{}^{(l)} = \frac{(h_i{}^{(l)} - \mu_i{}^{(l)})^2}{2(\sigma_i{}^{(l)})^2} + log\ log\ \left(\sigma_i{}^{(l)}\right)$
          Compute total loss over all nodes and layers: $L_{total} = \sum_{i=1}^{N} \quad \sum_{l=0}^{L} \quad L_{(i)}{}^{(l)}$
          Use gradient descent to update all model parameters, including:
                         Weight matrices $W^{(l)}$
                         $MLPs\ E_\mu{}^{(l)}, E_\sigma{}^{(l)}$
                         Meta-representations $Z$
```




1. Input Initialization
• Graph G (V, E)
• Feature matrix X (n×f)
• Set h^0) = X
• Param: hops L
2. Graph Structure Augmentation
• Normalize adjacency: Ä = D^-1/2 A' D^1/2
• Build k-NN graph on X
• Use for neighbor sampling in GraphSAGE
Repeat for l = 1oL
3. GraphSAGE Feature Propaggation
Repeat for l = 1 to L:
For each node i
1. Aggregate: h^i)^n = AGG(h^(n-1))
2. Concat with h^(l-1)
3. Transform: h^(l) = σ(W^(r)
Output: h^(l)...h(l)
4. Feature Quality Estimation
• Predict: μ(l), σ^(l) from latent z
• Model: h(l) ~ N
• Loss: Gaussian NLL
• Suppress high-σ features
Final Output: Meta-repr Z (n×z)

6. Evaluation Plan

In the next phase of this project, we will implement and evaluate the Multi-Hop Feature Quality Estimation (MQE) framework in two forms: the standalone version and an augmented version that incorporates GraphSAGE-based propagation. Our objective is to assess how well each variant learns robust, noise-aware node representations under a variety of graph conditions. Experiments will be conducted on five benchmark datasets: Cora, CiteSeer, PubMed, Amazon Computers, and Amazon Photos. Each dataset will be split into 10% training, 10% validation, and 80% testing, which is in line with standard practice. Node classification will serve as the primary downstream task to evaluate representation quality.

6.1. Testing Plan

We will evaluate the MQE framework under a range of scenarios, each designed to highlight different aspects of model behavior. First, we will establish a performance baseline by training on clean, uncorrupted datasets. To test robustness, we will inject synthetic Gaussian and uniform noise into node features, varying both the intensity and the proportion of affected nodes. To further examine generalization under widespread corruption, we will increase the proportion of noisy nodes while keeping the noise level fixed. We will assess the accuracy of MQE's internal noise modeling by comparing its estimated feature variances to the ground-truth noise levels. Additionally, we will conduct ablation studies to evaluate the impact of removing key components such as augmentation and multi-hop propagation. Finally, we will compare the runtime, scalability, and adaptability of both MQE variants, especially under dynamic or large-scale graph conditions.

6.2. Evaluation Metrics

Metric	Description	Expected Result	Expected Superior Variant
Classification Accuracy	Measures the predictive quality of node embeddings using logistic regression.	Both variants are expected to perform similarly on clean datasets.	Both (MQE and MQE augmented with GraphSAGE)
Robustness to Noise	Measures performance degradation under varying levels and types of feature noise.	MQE should degrade more gracefully under increasing noise levels.	MQE (Standalone)
Generalization under α Variation	Test stability as the proportion of noisy nodes increases (e.g., 10% to 90%).	MQE is expected to remain robust even under widespread corruption.	MQE (Standalone)
Noise Estimation Accuracy	Compares MQE's predicted feature variances to ground-truth noise levels.	High correlation is expected in both variants.	Both (MQE and MQE augmented with GraphSAGE)

Ablation Performance Drop	Measures performance reduction when components like augmentation or multi-hop propagation are removed.	Removing components should significantly reduce performance in matrix-based models.	MQE (Standalone)
Runtime	Compare training and inference time across datasets.	GraphSAGE-based propagation is expected to be faster due to neighborhood sampling.	MQE augmented with GraphSAGE
Scalability (Qualitative)	Evaluates memory and computational efficiency on large graphs.	A graphSAGE-based variant is expected to scale more efficiently.	MQE augmented with GraphSAGE
Inductive Generalization	Tests whether the model can handle nodes not seen during training.	Only GraphSAGE-based MQE supports this capability.	MQE augmented with GraphSAGE
Support for Dynamic Graphs	Evaluates adaptability to evolving graph structures during inference.	GraphSAGE's sampling strategy supports graph updates efficiently.	MQE augmented with GraphSAGE

7. References

1) *Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). DeepWalk: Online learning of social representations. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 701–710.* DeepWalk.pdf
2) *Grover, A., & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 855–864.* node2vec.pdf
3) *Hamilton, W., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. Advances in Neural Information Processing Systems, 30.* GraphSAGE.pdf
4) *Veličković, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., & Hjelm, R. D. (2019). Deep Graph Infomax. International Conference on Learning Representations (ICLR).* DGI.pdf
5) *Zhu, Z., Xu, D., Yu, Y., Zhang, B., & Lin, Y. (2021). GRACE: Graph contrastive learning with augmentations. Proceedings of the* GRACE.pdf
6) *Zhu, Z., Yu, Y., Xu, D., Lin, Y., & Yang, H. (2022).COSTA: Covariance-Preserving Contrastive Graph Representation Learning.Advances in Neural Information Processing Systems (NeurIPS), 35.* COSTA.pdf
7) *Tan, Q., Liu, N., Huang, X., Chen, R., Choi, S.-H., & Hu, X. (2022). MGAE: Masked Autoencoders for Self-Supervised Learning on Graphs. arXiv preprint arXiv:2201.02534.* MGAE.pdf
8) *Hou, Y., Zheng, S., Xu, J., Shen, H., & Cheng, X. (2022). GraphMAE: Self-Supervised Masked Graph Autoencoders. arXiv preprint arXiv:2205.10803.* GraphMAE.pdf
9) *Li, S., Liu, Y., Chen, Q., Webb, G. I., & Pan, S. (2024). Noise-Resilient Unsupervised Graph Representation Learning via Multi-Hop Feature Quality Estimation. Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM), pp. 1255–1265.* MQE.pdf
10) *Hamilton W., Ying, R., & Leskovec, J. (2017). Representation Learning on Graphs: Methods and Applications.* Graph Representation Learning.pdf
11) *ipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. ICLR 2017.* GNNs / Normalized Adjacency / GCN.pdf
12) *Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A Comprehensive Survey on Graph Neural Networks. IEEE Transactions on Neural Networks and Learning Systems.* Feature Propagation.pdf
13) *Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph Attention Networks. ICLR 2018.* Graph Attention Networks.pdf
14) *Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space.* Skip-Gram (Word2Vec).pdf

8. Appendix

8.1. Appendix A: Tools Used

⇒ *Canva*

⇒ *ChatGPT*

⇒ *Claude*

⇒ *Gemini*

⇒ *Stackoverflow*

⇒ *GeeksForGeeks*

⇒ *Wikipedia*

⇒ *PlanText UML Editor*

8.2. Appendix B: AI prompts used

- *What is the role of the parameter L (number of propagation hops) in MQE, and how is it affected when switching to GraphSAGE-based aggregation?*
- *How does GraphSAGE differ fundamentally from matrix-based feature propagation, and what implications does this have on variance modeling in MQE?*
- *Is the conditional Gaussian modeling used in MQE still valid when replacing adjacency-based propagation with GraphSAGE's learnable aggregation functions?*
- *How can the neighbor sampling mechanism of GraphSAGE be adapted to support MQE's requirement for structured multi-hop feature propagation?*
- *What are the implications of using sampled, non-deterministic neighborhoods (as in GraphSAGE) on the statistical consistency of MQE's variance modeling?*
- *Does aggregating multi-hop features from GraphSAGE layers maintain the same semantic meaning as matrix-based propagation in MQE?*
- *How should we aggregate the outputs of GraphSAGE across hops before feeding them into the Gaussian estimation modules used in MQE?*