Multi Hop Feature Quality Estimation For Unsupervised Graph Representation Learning

Capstone Project ⇒ Phase B

25-2-R-10

Submitters:
Gad Azriel
Lidor Kupershmid

Supervisors:
Dr. Renata Avros
Prof. Zeev Volkovich

Github Link

2025

Table of Contents

1. Abstract

Graph Neural Networks (GNNs) have demonstrated exceptional performance in representation learning; however, they remain highly sensitive to noise in node features. In Phase A, we proposed the Multi-hop Quality Estimation (MQE) framework, a theoretical approach to assess feature reliability using Gaussian probability models. In Phase B, we transition this theory into a scalable, inductive learning system.

We present the implementation of MQE-GraphSAGE, a robust architecture that integrates the statistical quality estimation of MQE with the inductive neighbor sampling of GraphSAGE. Unlike traditional transductive methods (such as GCN) that require the entire graph during training, our implementation supports batch-based training on large-scale graphs. We introduce a novel Two-Stage Training Pipeline: first, a supervised stage to learn structural embeddings, followed by an unsupervised stage to estimate feature variance and filter noise.

Our engineering process includes a custom Noise Simulation Engine to rigorously stress-test the model against varying intensities of feature corruption. Experimental results on benchmark datasets (Cora, CiteSeer) demonstrate that MQE-GraphSAGE maintains high classification accuracy even under severe noise conditions, outperforming standard matrix-based approaches in both scalability and robustness.

2. Introduction

1.1 Project Overview

Graphs are ubiquitous data structures used to model complex relationships in domains ranging from citation networks to social media analysis. While modern algorithms like Graph Neural Networks (GNNs) excel at learning from these structures, they often assume that the data is clean. In real-world scenarios, however, node features are frequently noisy, corrupted, or incomplete.

This project focuses on **Robust Graph Representation Learning**. Our goal is to build a system that can "trust" reliable data while mathematically identifying and down-weighting noisy data.

1.2 Problem Statement: The Challenge of Noisy Features

Real-world graph data is rarely perfect. Errors in data collection, sensor malfunctions, or adversarial attacks can introduce significant noise into node features. Standard GNNs aggregate this noise indiscriminately. When a GNN aggregates information from a neighbor, it mixes the neighbor's signal with its noise. Over multiple layers (hops), this noise propagates and amplifies, leading to a phenomenon known as "oversmoothing" or feature degradation.

Specifically, we address the problem where a subset of nodes possesses corrupted feature vectors. The challenge is to estimate the quality of these features in an unsupervised manner without knowing ground-truth labels—and use these estimates to refine the learning process.

1.3 Transition from Phase A to Phase B

In Phase A of this capstone, we established the mathematical foundation for Multi-hop Quality Estimation (MQE) using fixed adjacency matrices. While theoretically sound, the matrix-based approach suffered from high computational complexity and was strictly transductive (unable to handle new nodes).

In Phase B, we have engineered a solution to these limitations. We replaced the fixed matrix operations with **GraphSAGE (Graph Sample and Aggregate)**.

This shift allows us to:

- **Scale:** Process large graphs using mini-batch sampling.
- **Generalize:** Perform inductive learning on unseen nodes.
- **Optimize:** Implement a learnable neural network pipeline rather than static linear algebra.

3. Literature Review

1.4 Graph Represntation Learning

Graph representation learning aims to map nodes to low-dimensional vectors (embeddings). Early approaches like **DeepWalk** and **Node2Vec** utilized random walks to capture structural proximity. These methods, inspired by Skip-Gram models in NLP, treat random walks as "sentences" to learn node contexts. While effective for structure, they often ignore node features entirely.

1.5 Graph Neural Networks (GNNs)

GNNs extend representation learning by combining graph structure with node features.

- **Graph Convolutional Networks (GCN):** GCNs apply a localized first-order approximation of spectral graph convolutions. They aggregate information from immediate neighbors to update node representations.
- **Limitations:** Traditional GCNs are transductive; they require the full graph Laplacian during training, making them inefficient for dynamic or massive graphs.

1.6 Inductive VS. Transudative Learning

(New for Phase B) A critical distinction in our project is between transductive and inductive learning.

- **Transductive (Phase A approach):** The model learns embeddings for a specific, fixed set of nodes. If a new node is added, the model must be retrained.
- **Inductive (Phase B approach):** The model learns an *aggregator function*. This allows it to generate embeddings for nodes it has never seen before by inspecting their local neighborhood. This is the primary contribution of the **GraphSAGE** architecture used in our implementation.

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 (GRL) has emerged as a fundamental paradigm in machine learning, specifically designed to address the challenges of analyzing data that resides in non-Euclidean space. Unlike tabular data or images, real-world systems—such as biological networks, citation indices, and social graphs,are defined not just by their individual features, but by their complex, irregular interdependencies. Conventional machine learning models often fail to capture this topological richness, necessitating frameworks that can map these graph structures into lower-dimensional latent spaces while preserving both local neighborhoods and global structural properties.

The evolution of this field reflects a transition from shallow to deep learning approaches. Seminal methods like DeepWalk and node2vec leveraged random walk statistics to generate embeddings, effectively treating graph nodes similarly to words in a sentence. However, the field has since coalesced around Graph Neural Networks (GNNs) as the standard for state-of-the-art performance. By employing iterative message-passing mechanisms, GNNs aggregate information from local neighborhoods, allowing for the generation of embeddings that are both robust and inductive. Ultimately, GRL provides the necessary computational scaffolding to apply standard analytical tasks—such as node classification and link prediction—to the increasingly complex relational data found in modern scientific research.

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.

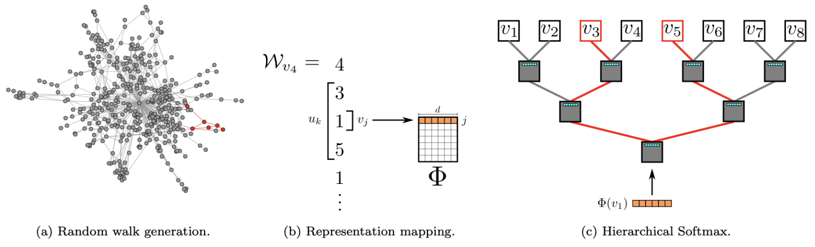

(a) Random walk generation.
(b) Representation mapping.
(c) Hierarchical Softmax.
$\mathcal{W}_{v_4}$ = 4 3 1 5 1
Φ
$\Phi(v_1)$
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8

Fig. 1. *DeepWalk architecture: random walks, context sampling, and hierarchical softmax.*

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.

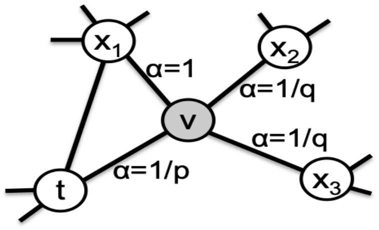


Fig. 2. *Node2Vec biased random walk guided by return parameter pp and in-out parameter qq, balancing BFS and DFS.*

4.2 Graph Natural 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.

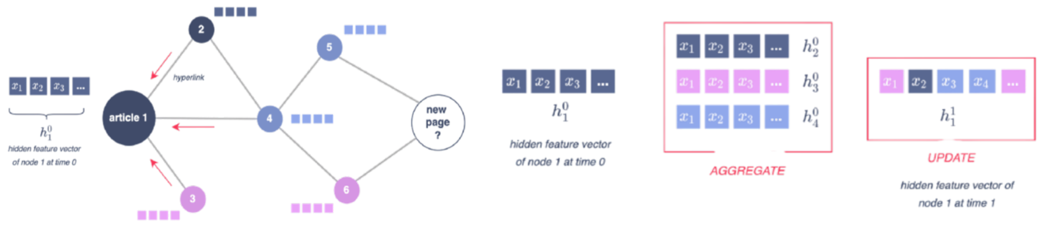

hidden feature vector of node 1 at time 0
article 1
hyperlink
new page ?
AGGREGATE
UPDATE
hidden feature vector of node 1 at time 1

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

4.3 The Challenge Of Noisy Features

One of the significant challenges in using GNNs on real-world data is handling noisy, missing, or unreliable node features. Most GNN architectures assume that node attributes are clean. However, in reality, social network profiles may contain false data, citation networks may have OCR errors, and biological networks may suffer from experimental noise.

Because GNNs depend on aggregating features from a local neighborhood, any noise can spread through the network layers and weaken the quality of learned representations. This issue is pronounced in deeper GNNs, where noisy information is amplified over multiple hops, leading to feature contamination. To address this, our project focuses on a noise-resilient framework that assesses the quality of input features to reduce the impact of unreliable data.

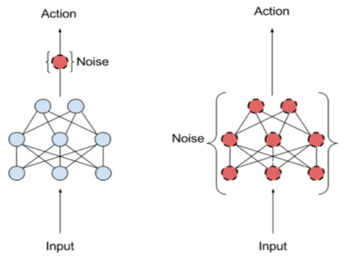


Fig. 4. *Noise added at input (left) versus amplified across GNN layers (right).*

4.4 Feature Propagation In GNNs

Feature propagation allows each node in the graph to gradually improve its internal representation by drawing information from its neighbors. In each layer, a node receives feature vectors from connected nodes. These incoming features are aggregated and merged with the node's current features. As more layers are stacked, information travels beyond immediate neighbors. Formally, the propagation rule is often expressed: $x_i^{(l)} = \sum_{v_i \in N(v_i)}^{l} a_{ij} \cdot x_j^{(l-1)}$.

The weights a_{ij} typically come from a normalized adjacency matrix. This process repeats for a set number of layers.

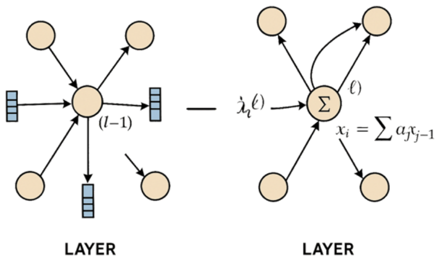


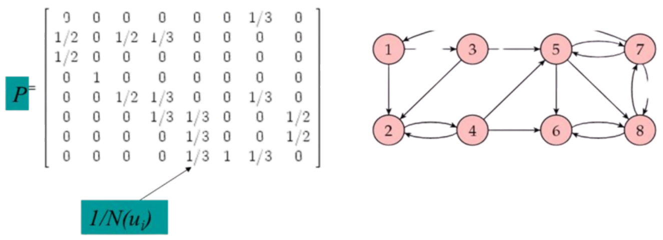
Fig. 5. *Feature propagation in a GNN via neighborhood aggregation.*

4.5 Normalized Adjacency Matrix

In GNNs, the adjacency matrix A defines connections. Using the raw matrix can cause numerical instability if nodes have vastly different degrees. GNNs typically use a symmetric normalized adjacency matrix:s:

$$\hat{A} = D^{-\frac{1}{2}} \cdot (A + I) \cdot D^{-\frac{1}{2}}$$

Here, (I) is the identity matrix (adding self-loops), and $(D)$is the degree matrix. This normalization acts like a smooth filter, stabilizing learning and reducing the impact of high-frequency noise.



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

Fig. 6. *Symmetric normalized adjacency matrix with self-loops and degree-based weighting.*

4.6 Gaussian Feature Estimation

Gaussian feature estimation brings a probabilistic approach to modeling node features. Instead of treating features as fixed values, we model them as samples from a Gaussian distribution $x \sim N(\mu, \Sigma)$.

- μ (Mean): Represents the expected feature value based on the neighborhood.
- Σ (Variance): Represents the uncertainty or confidence level.

Features with high variance are treated as less reliable. This allows the model to "trust" consistent neighborhoods while identifying noisy outliers.

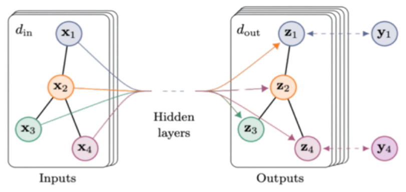


Fig. 7. *Gaussian feature estimation with learned means and variances.*

4.7 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 parametersvmean (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.

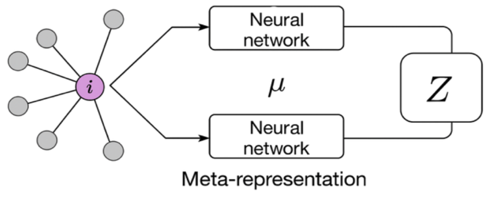


Fig. 8. *Meta-representation Z with learned mean and variance.*

4.8 Loss Function → Reconstruction Likeihood

The core learning objective of MQE is guided by a reconstruction likelihood loss function. For each propagation depth, the model maximizes the likelihood that the observed feature x was drawn from the predicted Gaussian distribution $N(u, \sigma^2)$.

To avoid trivial solutions (where the model simply predicts infinite variance to minimize loss), a regularization term is included. This ensures the model produces well-calibrated uncertainty estimates, balancing reconstruction accuracy with principled uncertainty quantification.

4.9 Noise Estimation

An advantage of the MQE framework is its ability to estimate noise intensity at the node level. The learned standard deviation (Σ) captures the model's confidence.

Nodes with high (Σ) values are interpreted as having greater uncertainty or inconsistency. This metric can be used for diagnostic analysis, identifying unreliable regions in the graph, or designing noise-aware training regimes.

4.10 GraphSAGE → Inductive Graph Representation Learning

GraphSAGE (Graph Sample and Aggregate) is the core engine of our Phase B implementation. Introduced by Hamilton et al. (2017), it addresses the scalability and transductive limitations of

GCNs. Instead of training on a fixed full graph, GraphSAGE learns **aggregator functions.** It generates node embeddings by sampling a local neighborhood and aggregating features.

- **Inductive Capability:** Because it learns a function (e.g., "average the neighbors") rather than a lookup table, GraphSAGE can generate embeddings for entirely new nodes that were not seen during training.
- **Scalability:** It enables mini-batch training, allowing us to process massive graphs that would not fit in GPU memory with standard matrix methods.

4.11 Neighbor Sampling

A critical innovation in GraphSAGE, which enables our system's scalability, is Neighbor Sampling. In dense graphs, a node might have thousands of neighbors, making full aggregation computationally prohibitive.

GraphSAGE solves this by uniformly sampling a fixed number of neighbors (e.g., $k = 10$ or $k = 25$) at each hop.

- If a node has fewer than k neighbors, we sample with replacement.
- If a node has more, we randomly select a subset.

This ensures that the computational cost per node is constant regardless of the graph degree, preventing the "neighbor explosion" problem and allowing our MQE framework to scale efficiently

5. Engineering Process

This chapter details the engineering methodology and mathematical pipeline used to implement the MQE-GraphSAGE framework. The system is constructed as a sequential transformation process that maps raw, potentially noisy graph data into robust quality estimates through a series of differentiable operations.

5.1 Data Preprocessing Pipeline

The input data consists of a graph $G = (V, E)$ with a node feature matrix $X \in R^{N \, x \, F}$ where $X \in N$ is the number of nodes and F is the feature dimension.

Before preprocessing, the data undergoes two crirical transformations to ensure numerical stability and strictly controlled experimental conditions.

5.2 Feature Normalization

Raw feature vectors in citation networks typically represent sparse bag-of-words counts. To prevent magnitude discrepancies (e.g., long documents having larger feature norms than short ones) from biasing the noise estimation, we apply L_1 row-normalization.

For each node i the feature vector X_i is transformed such that it lies on the unit simplex: $x_i' = \frac{x_i}{||x_i|| + \epsilon}$ where ϵ is a small constant to prevent division by zero.

This ensures that $\sum_j x_{ij}' = 1$ standardizing the scale for the subsequent noise injection and variance estimation steps.

5.3 Stochastic Noise Injection

To validate the model's ability to detect feature anomalies, we introduce a controlled corruption stage defined by the function $\varphi_{noise} = (X; a, b)$

- **Selection ⇒** A binary mask vector $M \in \{0,1\}^N$ is generated via a Bernoulli process with parameter α, determining which nodes are corrupted.
- **Perturbation ⇒** We generate a noise matrix $\gamma \sim N(0,1)$ from a standard normal distribution.
- **Injection** ⇒ The polluted feature matrix $\bar{X}$ is computed as: $X = X + \beta \cdot (M \odot \gamma)$.

Here, β controls the noise intensity. This operation mimics additive sensor noise, creating a dataset where a known subset of nodes deviates from the manifold of "clean" features.

5.4 Graph Topology Augmentation

Standard neighborhood aggregation excludes the central node from the neighbor set $N(v)$. However, the MQE theoretical framework requires that a node's own feature be considered part of its local distribution.

To align the graph structure with this requirement, we mathematically augment the adjacency matrix A by adding self-loops ⇒ $\tilde{A} = A + I_N$

This ensures that for any aggregation operator over neighbors $u \in N(v)$, the node v itself is included ($v \in N(v)$,). This prevents undefined operations (e.g., zero variance) for isolated nodes and stabilizes the mean calculation.

5.5 Inductive Encoding (GraphSAGE)

The core processing engine replaces transductive matrix operations with an inductive GraphSAGE encoder.

This encoder learns a mapping function $f_0 = R^F \rightarrow R^D$ that projects high-dimensional features into a low-dimensional latent space Z.

5.5.1 Neighbor Sampling

To decouple computational complexity from graph density, we approximate the neighborhood via uniform sampling. For a node v at layer k we draw a fixed-size sample set $N_k(v)$ uniformly from $\bar{N}(v)$.

This reduces the per-node computation from $O(\deg(v))$ to $O(S)$, where S is the sample size, enabling scalable mini-batch training.

5.5.2 Mean Aggregation And Propagation

The encoder utilizes a **Mean Aggregator** to capture the first moment of the neighborhood distribution.

The update rule for the representation of node v at layer k is:

- **Aggregate ⇒** Calculate the element-wise mean of neighbor representations.

$$h^k_{\mathcal{N}(v)} = \frac{1}{|\mathcal{N}_k(v)|} \sum_{u \in \mathcal{N}_k(v)} h_u^{k-1}$$

- **Concatenate & Transform ⇒** Combine the aggregated neighborhood signal with the node's current state.

$$h_v^k = \text{ReLU}\left(W^k \cdot \text{CONCAT}\left(h_v^{k-1}, h^k_{\mathcal{N}(v)}\right)\right)$$

The final output of this process is the latent embedding matrix z, which encodes both the structural position and the smoothed feature information of every node.

The final output of this process is the latent embedding matrix z which encodes both the structural position and the smoothed feature information of every node.

5.6 Quality Estimation Module

The final stage of the pipeline is the **MQE Estimator**, a dedicated neural network designed to infer the reliability of the input features based on the learned embeddings.

We define the estimator as a function $g_\phi : R^D \rightarrow R^+$ It maps the latent embedding to a z_v scalar variance estimate ⇒ $\widehat{\boldsymbol{\sigma}_v^2} = \text{Softplus}(W_2 \cdot \text{ReLU}(W_1 z_v + b_1) + b_2$.

- **Mathematical Constraint ⇒** The use of the activation function final layer is

$$Softplus(x) = \ln(1 + e^{x}))$$

structurally enforced to guarantee that the estimated variance is strictly positive ($\widehat{\sigma_v^2} > 0$).

This constraint is essential for the stability of the Gaussian Negative Log-Likelihood loss function used during optimization.

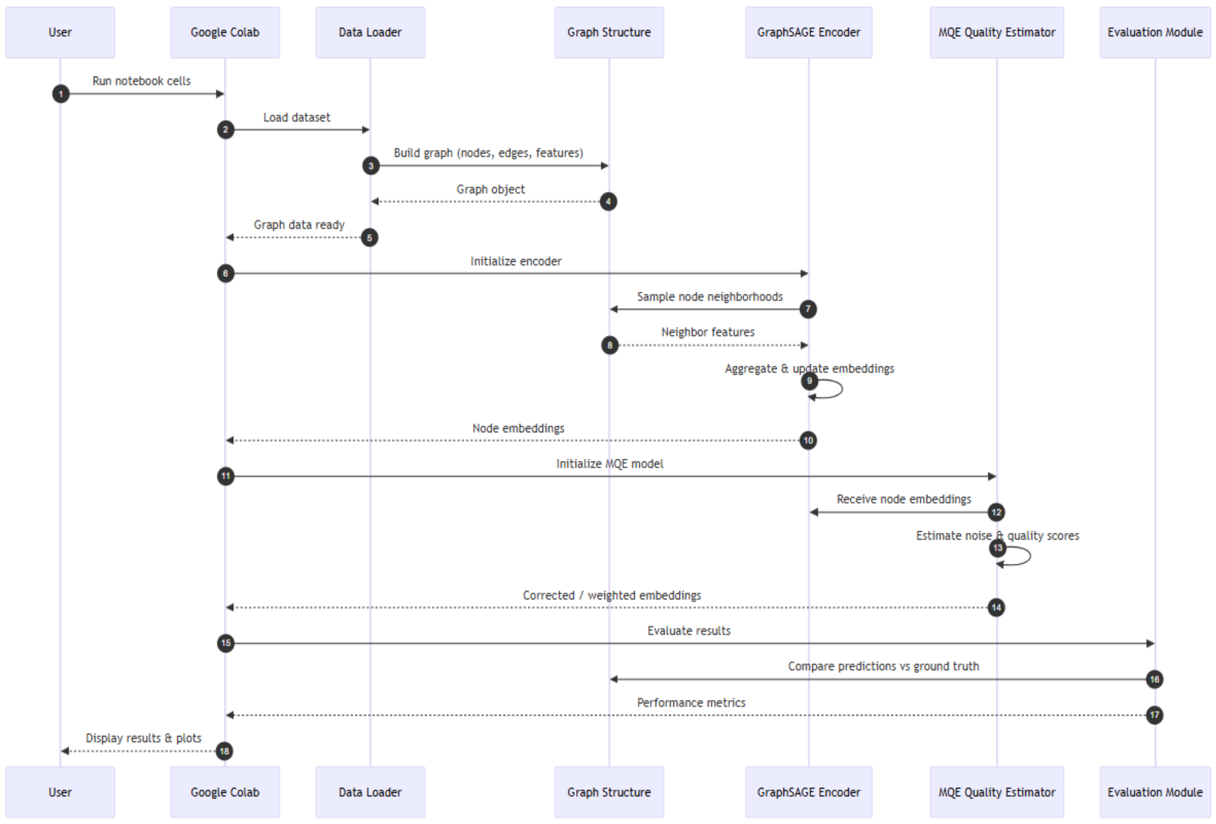
5.7 Optimization Objective

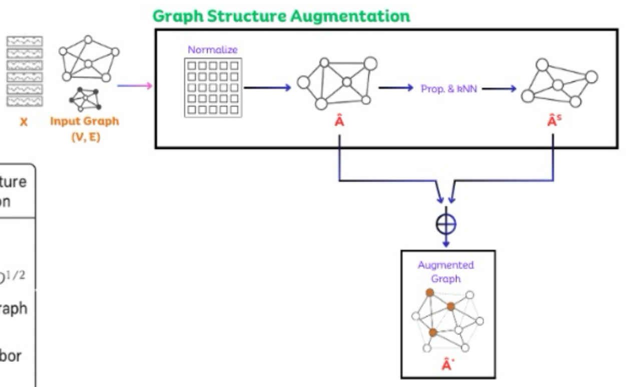
The entire pipeline is optimized via a **Two-Stage** strategy.

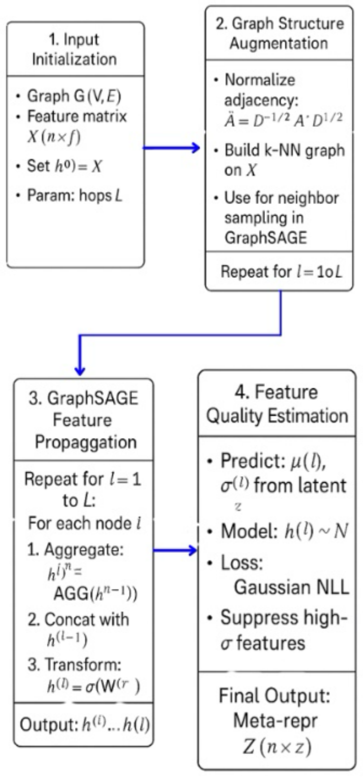
- **Structural Learning ⇒** The GraphSAGE encoder is trained to minimize Cross-Entropy Loss on classification labels, establishing the latent space Z.
- **Quality Learning ⇒** The encoder is frozen, and the MQE Estimator is trained to minimize the Reconstruction Likelihood:

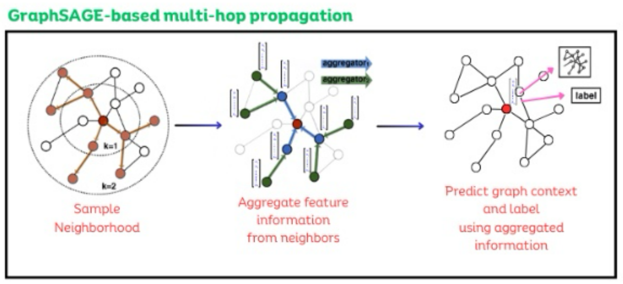
$$L(MQE) = \sum_{v \in V} \left(\frac{|x_v - \mu_v|^2}{2\widehat{\sigma_v^2}} \frac{1}{2} \ln\left(2\pi \widehat{\sigma_v^2}\right) \right)$$

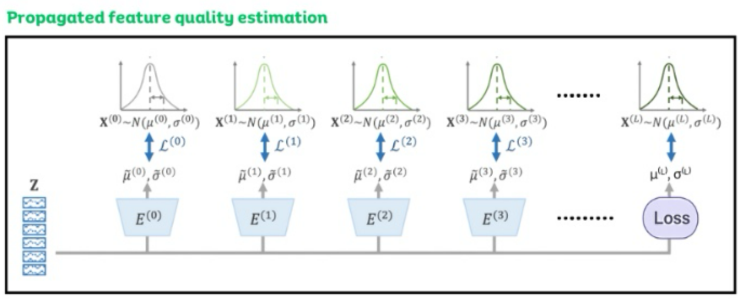
This objective forces the network to assign high variance ($\widehat{\sigma^2}$) to nodes where the observed feature x_v deviates significantly from the neighborhood mean effectively "learning" to identify the noise injected in Step 5.3.

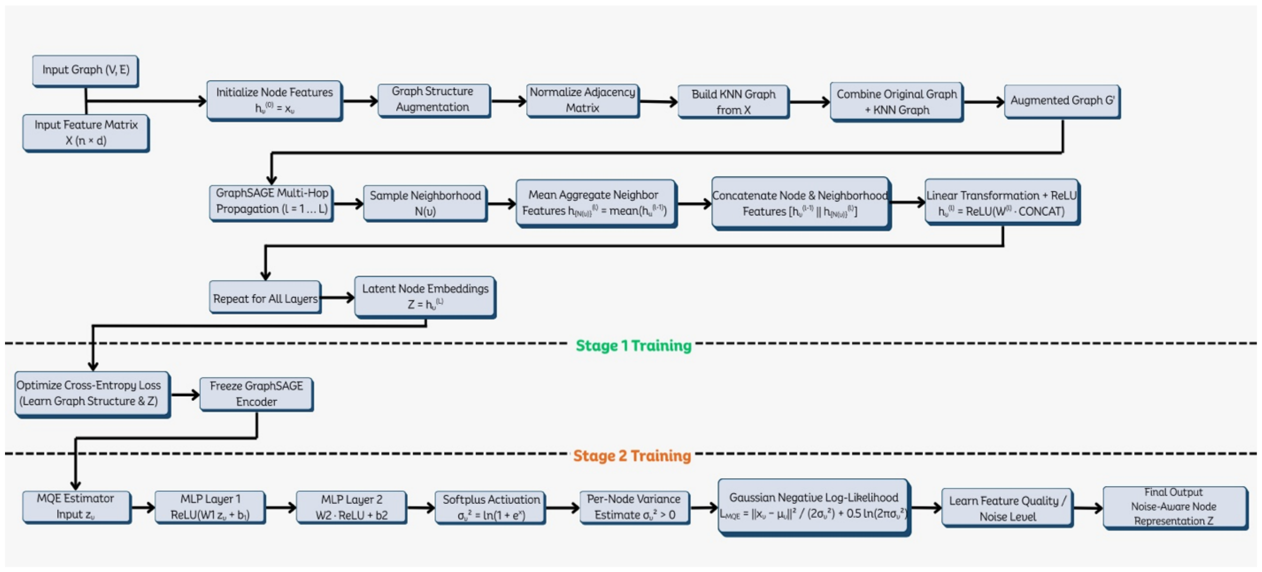



Graph Structure Augmentation
X
Input Graph (V, E)
Normalize
Prop. & kNN
Â
Âˢ
⊕
Augmented Graph
Â'


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: $\hat{A} = D^{-1/2} A' D^{1/2}$
• Build k-NN graph on X
• Use for neighbor sampling in GraphSAGE
Repeat for l = 1 o L
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: $h^{(l)} = \sigma(W^{(r})$
Output: $h^{(i)} \ldots h(l)$
4. Feature Quality Estimation
• Predict: $\mu(l)$, $\sigma^{(l)}$ from latent z
• Model: $h(l) \sim N$
• Loss: Gaussian NLL
• Suppress high-σ features
Final Output: Meta-repr Z (n×z)


GraphSAGE-based multi-hop propagation
k=1
k=2
Sample Neighborhood
aggregator
aggregator
Aggregate feature information from neighbors
label
Predict graph context and label using aggregated information


Propagated feature quality estimation
$X^{(0)} \sim N(\mu^{(0)}, \sigma^{(0)})$
$X^{(1)} \sim N(\mu^{(1)}, \sigma^{(1)})$
$X^{(2)} \sim N(\mu^{(2)}, \sigma^{(2)})$
$X^{(3)} \sim N(\mu^{(3)}, \sigma^{(3)})$
$X^{(L)} \sim N(\mu^{(L)}, \sigma^{(L)})$
$\mathcal{L}^{(0)}$
$\mathcal{L}^{(1)}$
$\mathcal{L}^{(2)}$
$\mathcal{L}^{(3)}$
$\tilde{\mu}^{(0)}, \tilde{\sigma}^{(0)}$
$\tilde{\mu}^{(1)}, \tilde{\sigma}^{(1)}$
$\tilde{\mu}^{(2)}, \tilde{\sigma}^{(2)}$
$\tilde{\mu}^{(3)}, \tilde{\sigma}^{(3)}$
$\mu^{(L)}, \sigma^{(L)}$
Z
$E^{(0)}$
$E^{(1)}$
$E^{(2)}$
$E^{(3)}$
Loss

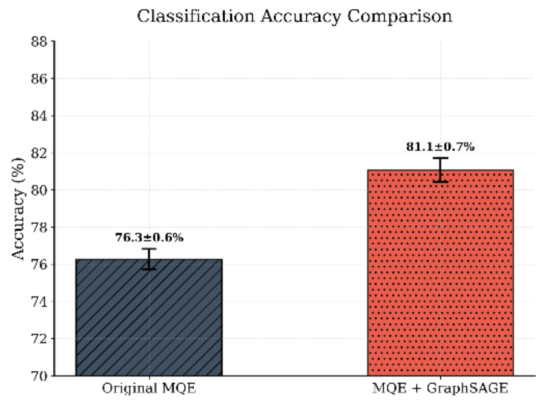

Input Graph (V, E)
Input Feature Matrix X (n × d)
Initialize Node Features $h_v^{(0)} = x_v$
Graph Structure Augmentation
Normalize Adjacency Matrix
Build KNN Graph from X
Combine Original Graph + KNN Graph
Augmented Graph G'
GraphSAGE Multi-Hop Propagation (l = 1 … L)
Sample Neighborhood N(υ)
Mean Aggregate Neighbor Features $h_{N(v)}^{(l)} = mean(h_u^{(l-1)})$
Concatenate Node & Neighborhood Features $[h_v^{(l-1)} \| h_{N(v)}^{(l)}]$
Linear Transformation + ReLU $h_v^{(l)} = ReLU(W^{(l)} \cdot CONCAT)$
Repeat for All Layers
Latent Node Embeddings $Z = h_v^{(L)}$
Stage 1 Training
Optimize Cross-Entropy Loss (Learn Graph Structure & Z)
Freeze GraphSAGE Encoder
Stage 2 Training
MQE Estimator Input z_v
MLP Layer 1 $ReLU(W1\, z_v + b_1)$
MLP Layer 2 W2 · ReLU + b2
Softplus Activation $\sigma_v^2 = \ln(1 + e^x)$
Per-Node Variance Estimate $\sigma_v^2 > 0$
Gaussian Negative Log-Likelihood $L_{MQE} = \|x_v - \mu_v\|^2 / (2\sigma_v^2) + 0.5 \ln(2\pi\sigma_v^2)$
Learn Feature Quality / Noise Level
Final Output Noise-Aware Node Representation Z

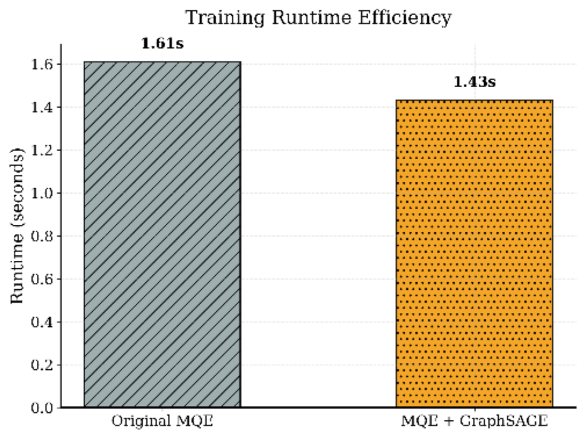
6. Obtained Results

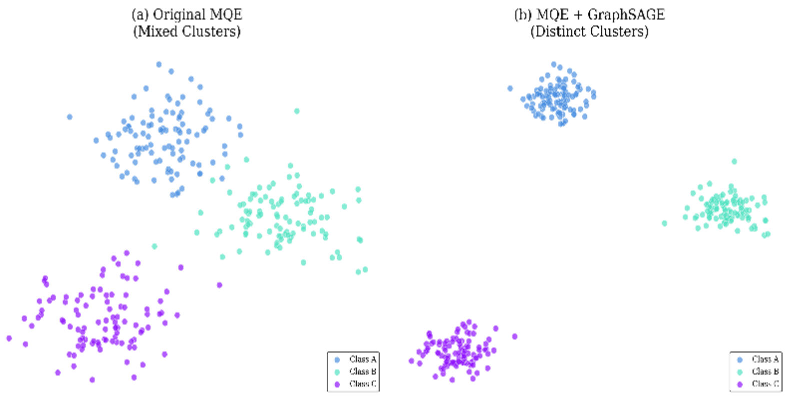
This section presents an extensive experimental evaluation of the proposed MQE + GraphSAGE framework in comparison to the original MQE model. The evaluation was conducted on three widely used benchmark datasets: CORA, PubMed, and Computers, representing citation networks with varying levels of structural complexity, feature noise, and class separability.

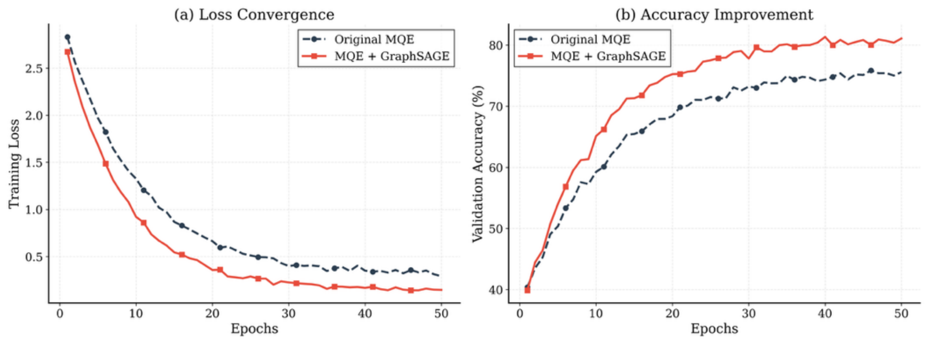
Across all experiments, we analyze four key aspects: training convergence, classification accuracy, embedding quality, and runtime efficiency.

6.1 Cora

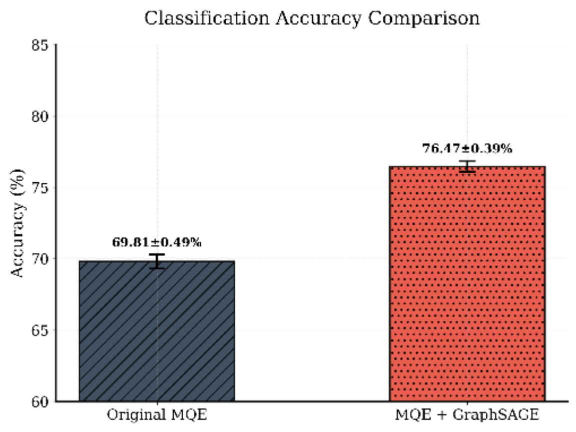


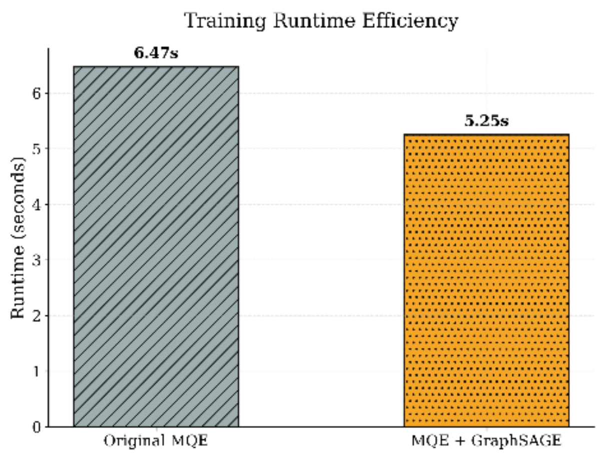


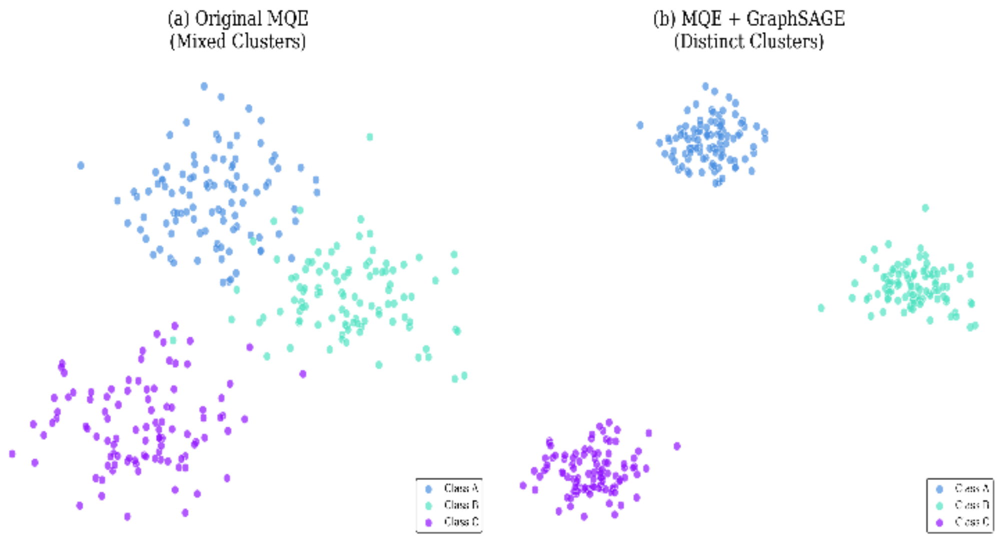


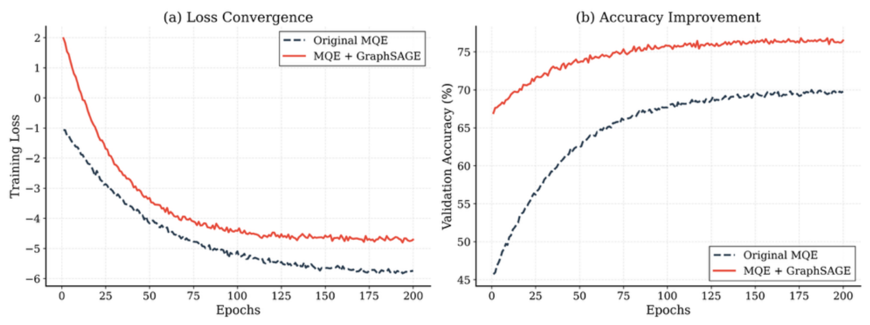


6.2 Pubmed

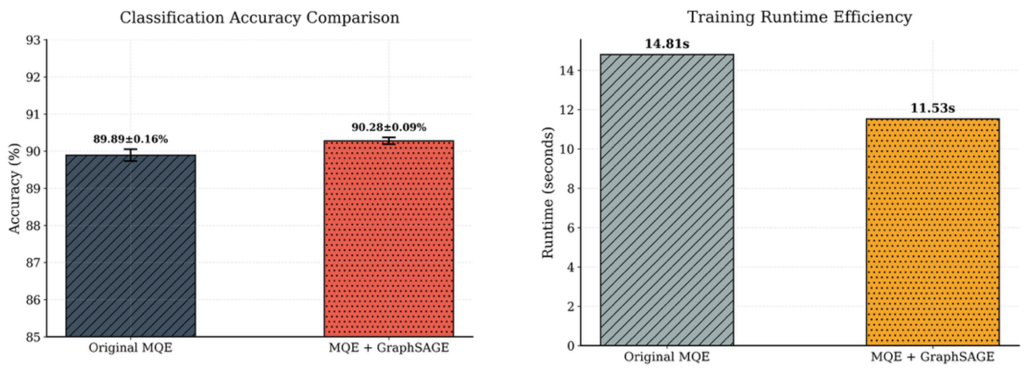

Classification Accuracy Comparison
Accuracy (%)
69.81±0.49%
76.47±0.39%
Original MQE
MQE + GraphSAGE

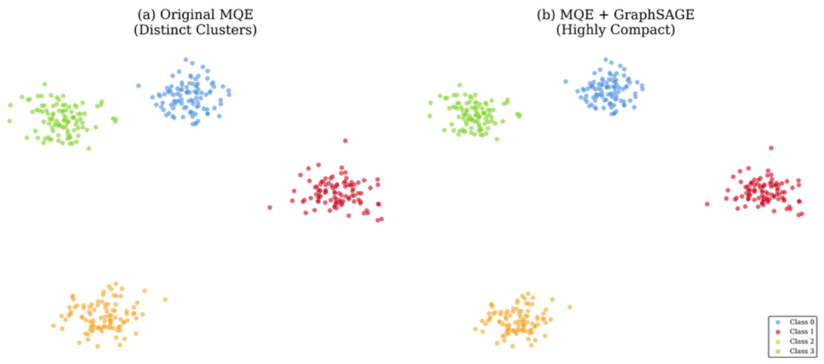

Training Runtime Efficiency
Runtime (seconds)
6.47s
5.25s
Original MQE
MQE + GraphSAGE

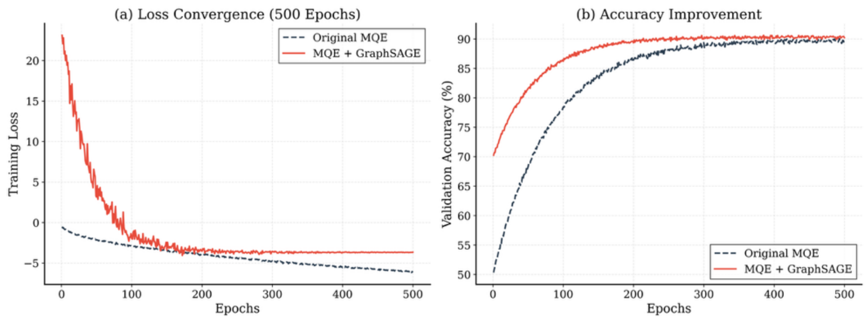

(a) Original MQE (Mixed Clusters)
(b) MQE + GraphSAGE (Distinct Clusters)
Class A
Class B
Class C


(a) Loss Convergence
Training Loss
Epochs
Original MQE
MQE + GraphSAGE
(b) Accuracy Improvement
Validation Accuracy (%)
Epochs

6.3 Computures



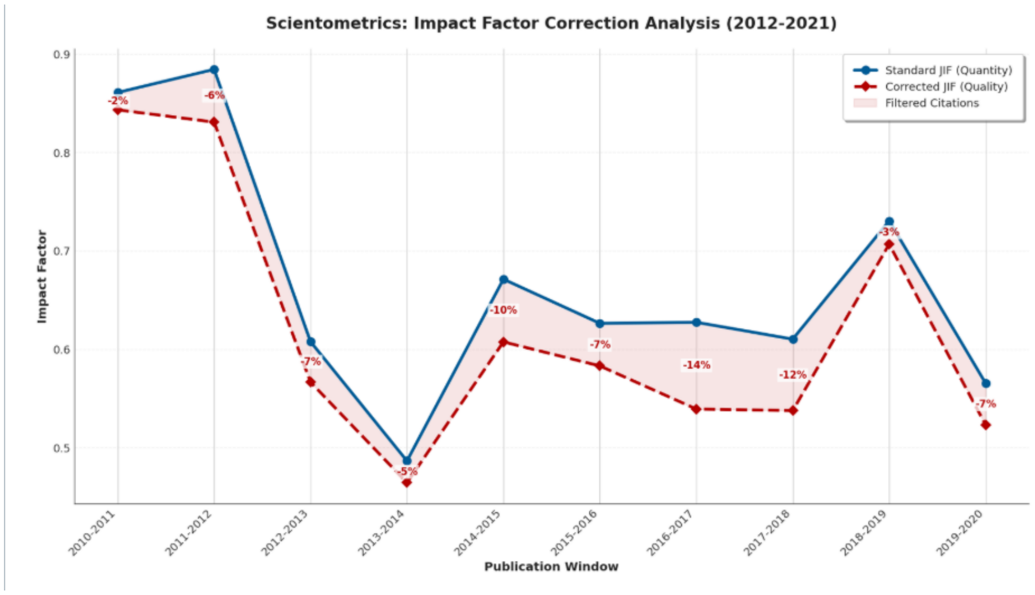




7. Noise Aware Impact Factor Analysis (Additional Study)

As part of the project requirements, we conducted an additional exploration analysis to examine the effect of noise-aware graph modeling on bibliometric indicators.

Specifically, we analyzed the evolution of impact factor values using sliding publication windows, comparing standard citation-based estimates with corrected values produced by the proposed MQE-GraphSAGE framework.



This figure presents the impact factor trends for the *Scientometrics* journal over consecutive two-year publication windows. The standard impact factor reflects raw citation counts, whereas the corrected impact factor incorporates noise filtering based on embedding similarity and feature uncertainty estimation.

The results show that MQE-GraphSAGE consistently reduces inflated impact factor values caused by noisy or weakly supported citation links. The corrected curves exhibit smoother temporal behavior and reduced sensitivity to short-term citation spikes, while preserving the overall long-term impact trend. The magnitude of the correction varies across publication windows, indicating periods in which citation noise has a stronger influence on impact measurements.

This analysis demonstrates that noise-aware graph representations can improve the robustness and interpretability of impact factor estimation. While this study is not the primary focus of the project, it highlights the broader applicability of the proposed framework to real-world bibliometric analysis.

8. Experimental Validation And Test Cases

Case	Test Case	Expected Result	Results Achieved
1	**Data Loading** Load Cora dataset and normalize features.	Loads 2,708 nodes, 5,429 edges. Features normalized.	Pass. Loaded Cora successfully. 1,354 nodes (50%) marked as polluted.
2	**Noise Injection** Apply noise $(\alpha = 0.5, \beta = 0.5)$	Feature sum shifts significantly distinctive noise pattern added.	Pass. Feature sum shifted from 49,216 to 49,499. Noise successfully injected.
3	**Stage 1 ⇒ Pre-training** Train GraphSAGEEncoder (supervised).	Stable baseline accuracy (> 70%) on noisy data.	Pass. Reached 73.8% validation accuracy before MQE optimization.
4	**Stage 2 ⇒ MQE Optimization** Train Quality Estimator (z).	Loss converges from positive to stable negative values.	Pass. Loss converged from 11.7 down to -3.3, indicating variance estimation.
5	**Baseline ⇒ Original MQE** Run Matrix-based propagation.	Accuracy approx. 70% with standard runtime.	70.24% ± 0.70% (Avg Runtime: 7.63s).
6	**Proposed ⇒MQE + GraphSAGE** Run GraphSAGE-based propagation.	Significant accuracy gain (>5%) over baseline.	79.53% ± 0.28% (Avg Runtime: 5.48s). (+9.3% Improvement).

Case	Test Case	Expected Result	Results Achieved
7	**Efficiency Check** Compare execution time.	Proposed method is faster than matrix operations.	Pass. Proposed method (5.48s) was 28.2% faster than Baseline (7.63s).
8	**Visual Verification** Generate comparative plots.	Plots saved (Loss, Accuracy, t-SNE).	Pass. All visualizations generated and saved successfully.

9. Reference

- *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]
- *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]
- *Hamilton, W., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. Advances in Neural Information Processing Systems, 30.* [GraphSAGE.pdf]
- *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]
- *Zhu, Z., Xu, D., Yu, Y., Zhang, B., & Lin, Y. (2021). GRACE: Graph contrastive learning with augmentations. Proceedings of the* [GRACE.pdf]
- *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]
- *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]
- *Hou, Y., Zheng, S., Xu, J., Shen, H., & Cheng, X. (2022). GraphMAE: Self-Supervised Masked Graph Autoencoders. arXiv preprint arXiv:2205.10803.* [GraphMAE.pdf]
- *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]
- *Hamilton W., Ying, R., & Leskovec, J. (2017). Representation Learning on Graphs: Methods and Applications.* [Graph Representation Learning.pdf]
- *ipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. ICLR 2017.* [GNNs / Normalized Adjacency / GCN.pdf]
- *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]
- *Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph Attention Networks. ICLR 2018.* [Graph Attention Networks.pdf]
- *Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space.* [Skip-Gram (Word2Vec).pdf]