# Multi Hop Feature Quality Estimation For Unsupervised Graph Representation Learning

Capstone Project ⇒ **Maintainer Guide**

25-2-R-10

Submitters:
Gad Azriel
Lidor Kupershmid

Supervisors:
Dr. Renata Avros
Prof. Zeev Volkovich

Github Link

2025

# Table of Contents

# Maintaining Guide

MQE with GraphSAGE for Robust Graph Representation Learning

# 1. General Project Description

## 1.1 Project Goal

This project extends the original Multi-hop feature Quality Estimation (MQE) framework by integrating a GraphSAGE-based graph neural network for neighborhood propagation.

The motivation behind this extension is to improve robustness, classification accuracy, and runtime efficiency when learning node representations under noisy feature conditions.

The original MQE framework relies on matrix-based multi-hop propagation. In this project, that propagation mechanism is replaced with a learned GraphSAGE encoder, while preserving the MQE quality estimation principle and evaluation protocol.

## 1.2 High-Level Workflow

The system operates according to the following logical pipeline:

1. Load a benchmark graph dataset following the original MQE setup
2. Normalize node features and preprocess the graph structure
3. Optionally inject synthetic noise into node features
4. Execute Original MQE using matrix-based propagation
5. Execute MQE + GraphSAGE using neural neighborhood aggregation
6. Train MQE quality estimators for multi-hop representations
7. Evaluate learned embeddings using downstream node classification
8. Compare accuracy and runtime between the two approaches

All stages are executed automatically through a single experiment script.

# 2. Folder and File Structure

## 2.1 Main Files

| File | Purpose |
|---|---|
| MQE_GraphSAGE.ipynb | Main notebook coordinating environment setup and execution |
| utils.py | Utility functions for data loading, preprocessing, noise injection, and evaluation |
| model_graphsage.py | Definitions of GraphSAGE and MQE neural models |
| train_graphsage.py | Training logic for both Original MQE and MQE + GraphSAGE |
| run_comparison.py | Entry point for running comparative experiments |
| visualization.py | Generation of plots and visual analysis |

## 2.2 Data and Results Folders

**data/**

- Logical directory used for storing graph datasets following the MQE dataset structure.

**mqe_graphsage/best_params/**

- Directory containing datasets and configuration files downloaded from the official MQE GitHub repository.

**Results**

- Results are generated dynamically and include:
  - ⇒ Printed accuracy and runtime summaries
  - ⇒ Logs from training and evaluation
  - ⇒ Visualization outputs (if enabled)

# 3. Dependencies and Execution Environment

The project depends on the following Python libraries:

- PyTorch
- PyTorch Geometric
- NumPy
- SciPy
- scikit-learn
- Matplotlib
- Seaborn

All dependencies are installed automatically when running the project in Google Colab.

## 3.1 Execution Environment

- Intended execution platform: Google Colab
- GPU acceleration is recommended but optional
- No manual environment configuration is required

# 4. Code Architecture and Logical Components

## 4.1 Utility Layer (utils.py)

This module contains shared helper functions used across the project, including:

- Dataset loading and preprocessing
- Feature normalization
- Synthetic noise injection
- Graph adjacency normalization
- Evaluation via logistic regression

This file acts as the foundational utility layer and is imported by all major components.

## 4.2 Model Layer (model_graphsage.py)

This module defines the core neural architectures:

- GraphSAGEEncoder ⇒ performs multi-layer neighborhood aggregation
- MQEQualityEstimator ⇒ estimates feature reconstruction mean and variance
- MQE_GraphSAGE ⇒ combined GraphSAGE propagation with MQE estimation
- MQENet ⇒ original MQE estimator used as a baseline

Changes in this file directly affect representation quality and model behavior.

## 4.3 Training Layer (train_graphsage.py)

This file implements two distinct training pipelines:

- Original MQE training using matrix-based propagation
- MQE + GraphSAGE training, which follows a two-stage process:
  1. Train GraphSAGE using supervised node classification
  2. Apply MQE quality estimation on learned embeddings

Training stability and performance are primarily controlled

## 4.4 Experiment Runner (run_comparison.py)

This script serves as the main experimental controller. It is responsible for:

- Parsing hyperparameters
- Selecting datasets
- Injecting noise
- Running multiple trials
- Aggregating accuracy and runtime metrics

This is the primary file to modify when changing experimental settings.

# 5. Maintenance and Configuration Points

## 5.1 Dataset Configuration

- Download the best_params datasets from MQE official Github repository as described in the User Guide.
- Move all datasets (Cora,Pubmed,Citerceer,Computers,Photo) to mqe_graphsage/bestparams path in Colab Notebook.
- Dataset selection is controlled via the "--data-name" argument
- Supported datasets must conform to the MQE dataset format
- Adding new datasets requires extending the data loading logic

## 5.2 Noise and Robustness Parameters

Key robustness parameters include:

- **alpha** ⇒ proportion of noisy nodes
- **beta** ⇒noise magnitude
- **noise_type** ⇒ noise distribution (normal or uniform)

These parameters are central to evaluating robustness behavior.

# 6. Extending the Project

The modular design allows straightforward extensions, such as:

- Replacing GraphSAGE with alternative GNNs (GCN, GAT)
- Modifying the quality estimation mechanism
- Supporting larger or dynamic graphs
- Adding new downstream evaluation tasks

Each extension can typically be implemented without modifying the entire pipeline.

# 7. Summary for Future Maintainers

- The project is modular and well-structured
- Experimental control is centralized in run_comparison.py
- The core innovation lies in replacing matrix-based propagation with GraphSAGE
- MQE quality estimation and evaluation remain consistent with the original framework

This guide provides sufficient detail for future students or researchers to maintain, modify, or extend the project confidently.