# Framework for GNNs on TUDataset

Erik Deinzer and Felix Glatzel

July 14, 2025

Project Report for

**Prof. Rizzi**

**Prof. De Santis**

Sapienza University of Rome

July 14, 2025

# Contents

# 1   Introduction

Graph-structured data is increasingly prevalent across a wide range of domains, including chemistry, biology, social networks, and recommendation systems. Traditional machine learning techniques often struggle to handle such data due to its non-Euclidean nature. Graph Neural Networks (GNNs) have emerged as a powerful class of models designed to learn representations from graph data by leveraging the underlying structure of nodes and edges.

This project explores the application of GNNs to graph classification tasks using real-world datasets from the TUDataset collection. Specifically, we design and implement a modular and extensible pipeline capable of loading, preprocessing, training, and evaluating models on multiple datasets, including MUTAG, PROTEINS, and ENZYMES. These datasets are commonly used benchmarks in the GNN research community and represent various challenges in terms of graph size, sparsity, heterogeneity, and domain-specific semantics.

Our objectives are threefold: (1) to deepen our understanding of how GNNs can be applied to graph classification problems; (2) to gain practical experience with PyTorch Geometric (PyG) and the TUDataset API; and (3) to analyze and compare the performance of different GNN architectures across datasets with diverse structural characteristics.

In addition to implementing a working framework, we provide a detailed analysis of the structure and nature of each dataset, including an examination of what the nodes and edges represent, whether the graphs are small or large, sparse or dense, and whether they are homogeneous or heterogeneous. We also survey relevant literature to identify key papers that have used these datasets and report their benchmark results for comparison.

This report presents our methodology, findings, and conclusions. It is structured as follows: Section 2 provides background on GNNs and related work; Section 3 introduces the datasets; Section 4 outlines the pipeline implementation; Section 5 presents the experiments and results; and Section 6 concludes the report and suggests possible improvements.

# 2   Methods

## 2.1   Datasets

In the context of graph classification, several properties are used to describe and compare graph datasets. These characteristics help determine how challenging a dataset is and what modeling strategies may be appropriate.

**Small vs. Large Graphs**   This refers to the number of nodes and edges per individual graph in the dataset.

- **Small graphs** usually have tens of nodes and edges (e.g., MUTAG), making them computationally light and less structurally complex.

- **Large graphs** can have hundreds or thousands of nodes and edges, requiring more memory and processing power (e.g., datasets like REDDIT-BINARY or COLLAB).

**Sparse vs. Dense Graphs**   This characterizes the ratio of the actual number of edges to the maximum possible number of edges.

- A graph is **sparse** if it has relatively few edges compared to the number of nodes (common in molecular or protein datasets).

- A graph is **dense** if it has many edges and a high degree of node connectivity.

Formally, sparsity can be approximated by the edge density: Density $= \frac{2 \cdot |E|}{|V| \cdot (|V| - 1)}$ where $|E|$ is the number of edges and $|V|$ is the number of nodes. With Density $D \geq 0.4$, a graph is denoted as dense in the following, a graph with $D \leq 0.1$ as very sparse and a graph with $D \approx 1$ as fully connected..

**Homogeneous vs. Heterogeneous Graphs**

- **Homogeneous graphs** contain only one type of node and one type of edge. Most classical GNN benchmarks (like MUTAG) fall in this category.

- **Heterogeneous graphs** include multiple types of nodes and/or edges, each potentially carrying different semantic information. For example, in biological datasets, nodes may represent different molecule components or protein substructures with diverse attributes.

**Attributed vs. Non-Attributed Graphs**

- In **attributed graphs**, nodes (and sometimes edges) carry feature vectors or labels, which can be used as input to GNN models.

- **Non-attributed graphs** only provide connectivity information, and node/edge features must be inferred or embedded manually.

**Labeled vs. Unlabeled Graphs**

- **Labeled graphs** provide a target class (e.g., "mutagenic" or "non-mutagenic") for each graph, necessary for supervised learning.

- **Unlabeled graphs** are more suitable for unsupervised or self-supervised learning approaches.

**Multi-class vs. Binary Classification**   This refers to the number of target classes:

- **Binary classification** tasks (e.g., MUTAG, PROTEINS) distinguish between two labels.

- **Multi-class classification** tasks (e.g., ENZYMES) require assigning one label from more than two categories.

To evaluate the performance of Graph Neural Networks (GNNs) in graph classification tasks, we employ three well-established datasets from the TUDataset collection: **MUTAG**, **PROTEINS**, and **ENZYMES**. These datasets are drawn from domains such as chemistry and bioinformatics and are widely used in benchmarking GNN architectures due to their distinct structural properties and classification challenges. In the following sections, we present a detailed description of each dataset, including the semantics of nodes and edges, the size and density of the graphs, and the nature of their labels.

## MUTAG

The MUTAG dataset consists of 188 chemical compounds, each modeled as an undirected graph. The classification task is to predict whether each compound exhibits mutagenic activity on a specific bacterium strain (*Salmonella typhimurium*). This is a binary classification problem.

**Graph Structure:** Each graph represents a molecule where:

- **Nodes** correspond to atoms in the molecule, such as Carbon (C), Oxygen (O), Nitrogen (N), etc. Each node is associated with a discrete label that identifies the atom type.

- **Edges** represent chemical bonds between atoms. These are undirected and unweighted, with no bond-type information explicitly provided in the basic version of the dataset.

**Characteristics:**

- **Number of graphs:** 188

- **Average number of nodes per graph:** 17.93

- **Average number of edges per graph:** 19.79

- **Mean Density:** $D \approx 0.096$

- **Classes:** 2 (mutagenic, non-mutagenic)

- **Type:** Small, very sparse, homogeneous, labeled, attributed

**Scientific Usage:** MUTAG has been used extensively as a benchmark in kernel-based and neural network-based graph learning models. Notable references include:

- Weisfeiler-Lehman subtree kernel [9]

- Graph Convolutional Networks (GCN) [5]

- Graph Isomorphism Network (GIN) [11]

## PROTEINS

The PROTEINS dataset is derived from the protein data bank and contains 1,113 protein graphs. The classification task is to determine whether each protein is an enzyme or a non-enzyme, resulting in a binary classification problem.

**Graph Structure:** Each graph models the secondary structure of a protein:

- **Nodes** correspond to secondary structure elements (SSEs), such as alpha helices or beta sheets. Nodes are labeled with discrete types and may carry additional structural features.

- **Edges** connect SSEs that are in close spatial proximity or connected sequentially. These edges are undirected and encode structural relationships.

**Characteristics:**

- **Number of graphs:** 1,113

- **Average number of nodes per graph:** 39.06

- **Average number of edges per graph:** 72.82

- **Mean Density:** $D \approx 0.098$

- **Classes:** 2 (enzyme, non-enzyme)

- **Type:** Medium-sized, very sparse, heterogeneous, labeled, attributed

**Scientific Usage:** PROTEINS is a common benchmark in both classical and deep graph learning literature. It was used in:

- Deep Graph Kernels [12]

- Message Passing Neural Networks (MPNNs) [8]

- Geometric deep learning benchmarks [1]

## ENZYMES

ENZYMES contains 600 protein graphs that are classified into one of six enzyme classes as defined by the Enzyme Commission (EC) number classification. This is a multi-class classification problem.

**Graph Structure:** Each graph represents a protein, modeled similarly to the PROTEINS dataset:

- **Nodes** denote SSEs in the protein structure, labeled by type and possibly enriched with additional feature information.

- **Edges** indicate spatial or sequential proximity between SSEs. These are undirected and generally sparse.

**Characteristics:**

- **Number of graphs:** 600

- **Classes:** 6 (corresponding to EC enzyme classes)

- **Average number of nodes per graph:** 32.63

- **Average number of edges per graph:** 62.14

- **Mean Density:** $D \approx 0.12$

- **Type:** Medium-sized, sparse, heterogeneous, labeled, attributed

**Scientific Usage:** ENZYMES presents a more complex classification challenge due to its multi-class nature and greater intra-class variability. It has appeared in:

- GIN benchmark experiments [11]

- TUDataset benchmarking [8]

- Comparative studies between graph kernels and deep models [12]

## 2.2 Summary Table

| Dataset | # Graphs | Avg. Nodes | Avg. Edges | Classes | Graph Type[1] | Domain |
|---------|----------|------------|------------|---------|---------------|--------|
| MUTAG | 188 | 17.93 | 19.79 | 2 | S, SP, HO | Chemistry |
| PROTEINS | 1113 | 39.06 | 72.82 | 2 | M, SP, HE | Biology |
| ENZYMES | 600 | 32.63 | 62.14 | 6 | M, SP, HE | Biology |

Table 1: Overview of the main properties of the selected TUDatasets.
(S=Small, SP=Sparse, HO=Homogenous, M=Medium, HE=Heterogenous)

## 2.3 Graph Neural Network Architectures

Graph Neural Networks (GNNs) are designed to process graph-structured data by leveraging the relationships (edges) between entities (nodes). The key idea is to iteratively update node representations by aggregating information from their neighbors. In this section, we discuss four prominent GNN architectures: Graph Convolutional Networks (GCN), Graph Isomorphism Networks (GIN), Graph Attention Networks (GAT), and GraphSAGE. Each of these models defines a different message passing and aggregation strategy.

## 2.4 Graph Convolutional Network (GCN)

Graph Convolutional Networks (GCNs) were introduced in [5] as an extension of the convolution operation to graph-structured data. GCNs perform a form of *neighborhood averaging*, where each node updates its representation based on the features of its neighbors and itself.

The forward propagation rule for a single GCN layer is:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{1}$$
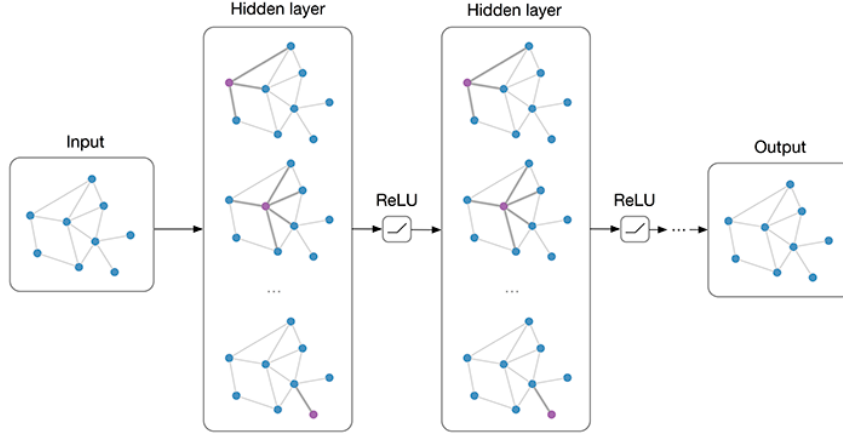


Figure 1: Graph Convolutional Neural Network [4].

**Explanation of Symbols:**

- $\mathbf{H}^{(l)}$ is the matrix of node features at layer $l$; each row is the feature vector for one node.

- $\mathbf{H}^{(0)}$ is the input feature matrix (e.g., atom types).

- $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix of the graph with added self-loops.

- $\tilde{\mathbf{D}}$ is the diagonal degree matrix of $\tilde{A}$, where $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

- $\mathbf{W}^{(l)}$ is a trainable weight matrix at layer $l$.

- $\sigma$ is a non-linear activation function (typically ReLU).

The normalization $\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ ensures that feature aggregation is scaled properly, especially for nodes with high degree. GCNs are efficient and perform well on many tasks, but they are limited in expressivity due to their linear nature and tendency to oversmooth node features across layers.

## 2.5 Graph Isomorphism Network (GIN)

The Graph Isomorphism Network (GIN), proposed in [11], is a GNN architecture specifically designed to match the discriminative power of the Weisfeiler-Lehman (WL) test  a classical test for graph isomorphism.

The update rule in GIN is:

$$\mathbf{h}_v^{(l+1)} = \text{MLP}^{(l)}\left((1+\epsilon)\cdot\mathbf{h}_v^{(l)} + \sum_{u\in\mathcal{N}(v)}\mathbf{h}_u^{(l)}\right) \tag{2}$$
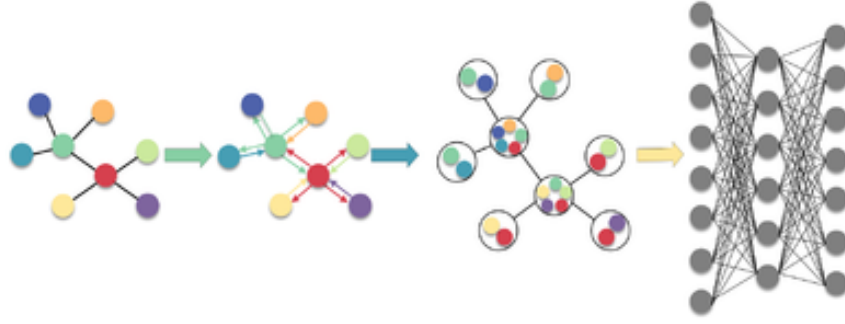
Figure 2: Node update process of a GIN layer [7].

**Explanation of Symbols:**

- $\mathbf{h}_v^{(l)}$ is the feature vector of node $v$ at layer $l$.

- $\mathcal{N}(v)$ denotes the set of neighboring nodes of $v$.

- $\epsilon$ is a learnable or fixed scalar that determines the importance of the central node's own features.

- $\text{MLP}^{(l)}$ is a multi-layer perceptron applied after aggregation.

Unlike GCNs, which average the features, GIN performs a *sum aggregation* followed by a non-linear transformation, which retains more unique structural information. This design choice makes GIN theoretically more powerful for distinguishing non-isomorphic graphs. GIN performs particularly well in graph classification tasks such as MUTAG and ENZYMES.

## 2.6   Graph Attention Network (GAT)

Graph Attention Networks (GATs), introduced in [10], apply an attention mechanism to graphs. Instead of treating all neighbors equally, GAT learns to assign different importance (attention scores) to each neighbor.

The update rule for GAT is:

$$\mathbf{h}_v^{(l+1)} = \sigma \left( \sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(l)} \cdot \mathbf{W}^{(l)} \mathbf{h}_u^{(l)} \right) \tag{3}$$

**Explanation of Symbols:**

- $\mathbf{h}_v^{(l)}$ and $\mathbf{h}_u^{(l)}$ are the feature vectors of node $v$ and its neighbor $u$.

- $\mathbf{W}^{(l)}$ is a trainable weight matrix.

- $\alpha_{vu}^{(l)}$ is the attention coefficient from node $v$ to node $u$, learned via:

$$\alpha_{vu}^{(l)} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}^{(l)}\mathbf{h}_v^{(l)} \,\|\, \mathbf{W}^{(l)}\mathbf{h}_u^{(l)}]\right)\right)}{\sum_{k \in \mathcal{N}(v)} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}^{(l)}\mathbf{h}_v^{(l)} \,\|\, \mathbf{W}^{(l)}\mathbf{h}_k^{(l)}]\right)\right)}$$

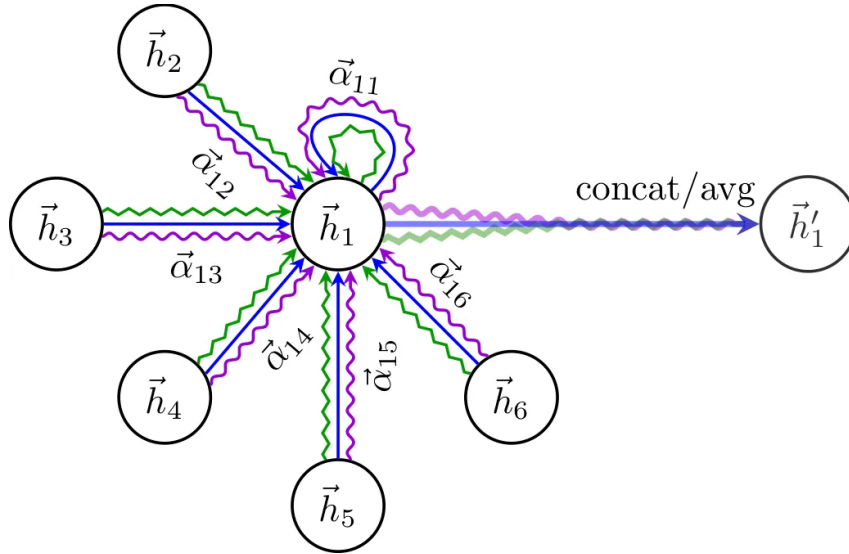- **a** is a trainable attention vector; $\|$ denotes vector concatenation.



Figure 3: Graph Attention Mechanism [2].

By learning attention scores, GAT can focus on the most relevant parts of a node's neighborhood, which is useful for datasets where not all connections are equally informative. It is particularly effective in social network data, though its performance in small molecular graphs is also competitive.

## 2.7   GraphSAGE

GraphSAGE (Graph Sample and Aggregate), introduced in [3], was designed for *inductive learning*, i.e., making predictions on unseen graphs. Unlike full-neighborhood aggregation, GraphSAGE samples a fixed-size neighborhood and aggregates features using various strategies (mean, max-pool, LSTM).

The generic update rule is:

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\mathbf{W}^{(l)} \cdot \mathrm{AGG}^{(l)}\left(\left\{\mathbf{h}_v^{(l)}\right\} \cup \left\{\mathbf{h}_u^{(l)} \mid u \in \mathcal{N}(v)\right\}\right)\right) \tag{4}$$



1. Sample neighborhood    2. Aggregate feature information    3. Predict graph context and label
                              from neighbors                     using aggregated information
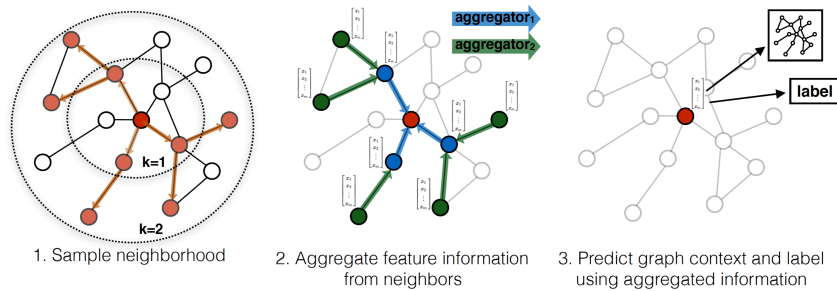
Figure 4: Graphsage Node update [6].

**Explanation of Symbols:**

- $\text{AGG}^{(l)}$ is an aggregation function (e.g., mean, LSTM, max-pooling).

- $\mathbf{h}_v^{(l)}$ is the current node embedding at layer $l$.

- $\mathbf{W}^{(l)}$ is a trainable weight matrix.

- $\sigma$ is a non-linear activation function.

GraphSAGE is particularly scalable for large graphs, such as citation or social networks, due to its sampling approach. It also enables efficient inference on previously unseen nodes and graphs.

# References

[1] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[2] Giuseppe Futia. Graph attention networks under the hood, 2021.

[3] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[4] Thomas Kipf. Graph convolutional networks, 2016.

[5] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2017.

[6] Jure Leskovec. Graphsage: Inductive representation learning on large graphs, 2017.

[7] Tong Lu, Sizu Hou, and Yan Xu. Fault line selection algorithm for distribution networks based on adapglgin network. *IET Generation, Transmission Distribution*, 2023.

[8] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

[9] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Lee, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. In *Advances in Neural Information Processing Systems*, volume 24, 2011.

[10] Petar Velikovi, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[11] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

[12] Pinar Yanardag and S V N Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374, 2015.