The GNN Spectrum: Review and Benchmarking of GCN and GraphSAGE

SEMINAR REPORT
(SWE551)

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

MASTERS OF TECHNOLOGY
IN
SOFTWARE ENGINEERING

Submitted by

NITESH BHARDWAJ

(25/SWE/12)

Under the supervision of

Dr. Swechchha Gupta

DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi 110042

DEC, 2025

DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, NITESH BHARDWAJ, Roll No's – 25/SWE/12 students of M.Tech (Software Engineering), hereby declare that the seminar report titled "The GNN Spectrum: Review and Benchmarking of GCN and GraphSAGE" which is submitted by us to the Software Engineering Department, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of degree of Masters of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi NITESH BHARDWAJ

Date: 08.12.2025 (25/SWE/12)

DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the seminar report titled "The GNN Spectrum: Review and Benchmarking of GCN and GraphSAGE" which is submitted by NITESH BHARDWAJ, Roll No's – 25/SWE/12, Software Engineering Department, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Masters of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Dr. Swechchha Gupta

Date: 08.12.2025

SUPERVISOR

DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

I wish to express my sincerest gratitude to Dr. Swechchha Gupta for her continuous guidance and mentorship that she provided us during the project. She showed us the path to achieve our targets by explaining all the tasks to be done and explained to us the importance of this project as well as its industrial relevance. She was always ready to help us and clear our doubts regarding any hurdles in this project. Without his constant support and motivation, this project would not have been successful.

Place: Delhi — NITESH BHARDWAJ

Date: 08.12.2025 — (25/SWE/12)

Abstract

Graph Neural Networks (GNNs) have emerged as the de facto standard for learning on non-Euclidean domains, enabling breakthroughs in fields ranging from social network analysis to bioinformatics. This report provides a comprehensive study of the GNN landscape, bridging theoretical foundations with empirical analysis. We first present an extensive literature review covering the taxonomy of GNNs, including Recurrent, Convolutional, and Spatial-Temporal architectures, while also addressing advanced paradigms such as Dynamic GNNs and Self-Supervised Learning. Furthermore, we critically analyze deployment challenges in real-world scenarios, including graph heterophily, data imbalance, and out-of-distribution generalization.

To empirically evaluate foundational message-passing mechanisms, we implement and compare two distinct architectures: the spectral-based Graph Convolutional Network (GCN) and the spatial-based GraphSAGE. Using the Cora and PubMed benchmark datasets, we assess model performance in terms of classification accuracy and computational efficiency. Our results demonstrate that while GraphSAGE achieves a marginal accuracy advantage on the feature-rich Cora dataset (80.70% vs. 80.30%), the GCN architecture proves significantly more robust on the larger PubMed graph (79.20% vs. 76.70%). Crucially, the spectral GCN consistently outperforms GraphSAGE in computational speed, training approximately 2× faster across both benchmarks. These findings underscore the trade-offs between spectral efficiency and spatial expressivity, suggesting that GCN remains a superior baseline for transductive tasks on homophilic citation networks.

Contents

List of Tables

List of Figures

List of Symbols

Symbol	Description
G	Graph structure, formally defined as a tuple (V, E)
V	Set of vertices (nodes) in the graph
E	Set of edges (links) representing relationships
N	Total number of nodes in the graph, denoted as $\|V\|$
v_i	A specific node i in the vertex set V
e_{ij}	An edge connecting node v_i to node v_j
$\mathcal{N}(v)$	Neighborhood set of node v (nodes connected to v)
A	Adjacency matrix of the graph, $A \in \mathbb{R}^{N \times N}$
D	Degree matrix of the graph (diagonal matrix where $D_{ii} = \sum_j A_{ij}$)
X	Node feature matrix containing attributes, $X \in \mathbb{R}^{N \times d}$
x_v	Feature vector of a specific node v
L	Graph Laplacian matrix, defined as $D - A$
$H^{(l)}$	Hidden representation matrix at network layer l
$h_v^{(l)}$	Hidden state embedding of node v at layer l
$W^{(l)}$	Learnable weight matrix at layer l
$\sigma(\cdot)$	Non-linear activation function (e.g., ReLU, Softmax)
$\hat{A}$	Adjacency matrix with added self-loops ($\hat{A} = A + I$)
$\hat{D}$	Diagonal degree matrix of $\hat{A}$
y_i	Ground truth class label for node i
$\hat{y}_i$	Predicted class probability distribution for node i
$\mathcal{L}$	Loss function (Negative Log-Likelihood)
D_{in}	Dimensionality of input features
D_{out}	Dimensionality of the output classes

Chapter 1

INTRODUCTION

1.1 Background and Motivation

Deep learning has revolutionized many machine learning tasks in recent years, ranging from image classification and video processing to speech recognition and natural language understanding, where the data are typically represented in the Euclidean space [1]. However, there is an increasing number of applications where data are generated from **non-Euclidean domains** and must be represented as graphs with complex relationships and interdependency between objects [1].

Graphs are pervasive data structures used for modeling various real-world systems. In social science, social networks model relationships between individuals; in natural science, graphs model physical systems and molecular structures; and in knowledge graphs, they represent entities and relations [2]. Analyzing graph data is paramount for gaining insights into intricate patterns, uncovering hidden structures, and understanding the dynamics of interconnected systems [3].

Despite the ubiquity of graph data, its complexity imposes significant challenges on existing machine learning algorithms [1]. To address this, Graph Neural Networks (GNNs) have emerged as a powerful tool. GNNs are neural models that capture the dependence of graphs via message passing between the nodes of graphs [2]. By iteratively aggregating information from neighboring nodes, GNNs can capture the intricate structural information inherent in the data [4]. The rapid evolution of GNNs has led to variants such as Graph Convolutional Networks (GCNs) and GraphSAGE, which have demonstrated ground-breaking performances on many deep learning tasks [2]. The motivation for this study stems from the need to systematically explore these architectures to understand their comparative strengths on standard benchmarks.

1.2 Problem Statement and Aim

While deep learning has achieved remarkable success on Euclidean data (e.g., images and text), standard neural networks like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) struggle to handle graph data directly [1]. This is primarily because graphs are irregular, may have a variable size of unordered nodes, and nodes may have a different number of neighbors [1]. Consequently, defining operations such as convolution, which are straightforward in the image domain, becomes difficult in the graph domain [1].

Furthermore, while numerous GNN variants exist, selecting the appropriate architecture remains a critical challenge. For instance, spectral-based approaches like GCN

rely on the graph Laplacian, whereas spatial-based approaches like GraphSAGE perform convolution by aggregating neighbor information directly [1, 4]. The aim of this project is to implement and compare these two fundamental GNN architectures—Graph Convolutional Networks (GCN) and GraphSAGE—on benchmark citation network datasets. By evaluating their performance under a unified experimental setup, this study aims to provide empirical insights into their representation learning capabilities.

1.3 Objectives of the Study

The primary objectives of this research are as follows:

- To conduct a comprehensive literature review on the taxonomy of Graph Neural Networks, categorizing them into Recurrent GNNs, Convolutional GNNs, Graph Autoencoders, and Spatial-Temporal GNNs as defined in [1].
- To implement two distinct GNN architectures: the spectral-based Graph Convolutional Network (GCN) and the spatial-based GraphSAGE.
- To evaluate and compare the performance of these models on standard benchmark datasets (e.g., Cora and PubMed) using metrics such as accuracy and F1-score.
- To analyze the training dynamics and visualize the learned graph structures to understand the operational differences between the selected models.

1.4 Organization of the Report

The remainder of this report is organized into five chapters, structured as follows:

Chapter 2: Literature Review provides a comprehensive overview of the current state of Graph Neural Networks. It introduces the fundamental message-passing mechanisms and presents a detailed taxonomy of GNN models, categorizing them into Recurrent Graph Neural Networks (RecGNNs), Convolutional Graph Neural Networks (ConvGNNs), Graph Autoencoders (GAEs), and Spatial-Temporal Graph Neural Networks (STGNNs), as systematically surveyed by [1, 4]. Furthermore, this chapter highlights key applications of GNNs in structural and non-structural scenarios [2]. **Chapter 3: Methodology** details the system architecture and experimental design adopted for this study. It provides descriptions of the benchmark datasets used (Cora and PubMed) and explains the specific architectures implemented: the spectral-based Graph Convolutional Network (GCN) and the spatial-based GraphSAGE. The training protocols, evaluation metrics, and experimental environment are also defined in this chapter. **Chapter 4: Results and Discussion** presents the empirical findings of the study. It includes a comparative analysis of the GCN and GraphSAGE models in terms of classification accuracy, F1-score, and training time. Additionally, this chapter provides visualizations of the learned node embeddings and analyzes the training curves to interpret the convergence behaviors of the models. **Chapter 5: Conclusion and Future Scope** summarizes the key findings and contributions of this work. It discusses the limitations of the current study and outlines potential avenues for future research, including the exploration of GNNs for graphs with heterophily [5], Dynamic Graph Neural Networks (DGNNs) [6], and robust learning under real-world conditions such as data imbalance and noise [3].

Chapter 2

LITERATURE REVIEW

2.1 Overview of Graph Neural Networks

2.1.1 Graph Definitions and Notation

A graph is formally represented as $G = (V, E)$, where V is the set of vertices (nodes) and E is the set of edges representing relationships between these nodes [1]. Let $v_i \in V$ denote a node and $e_{ij} = (v_i, v_j) \in E$ denote an edge pointing from v_i to v_j. The structural information of the graph is mathematically captured by an adjacency matrix $A \in \mathbb{R}^{n \times n}$, while node attributes are represented by a feature matrix $X \in \mathbb{R}^{n \times d}$ [1].

To understand the diverse landscape of GNN applications, it is essential to define the various types of graph structures encountered in real-world scenarios:

- **Directed vs. Undirected Graphs:** In directed graphs, edges possess a specific direction (e.g., citation networks where paper A cites paper B), indicating a one-way flow of information. Conversely, undirected graphs have bidirectional edges (e.g., friendship in social networks), representing mutual interactions [1].
- **Homogeneous vs. Heterogeneous Graphs:** Homogeneous graphs consist of a single type of node and edge. Heterogeneous graphs, however, involve multiple types of nodes and edges (e.g., a bibliographic network containing authors, papers, and venues), requiring models to handle diverse semantic information [1, 6].
- **Static vs. Dynamic Graphs:** Static graphs possess a fixed structure and feature set during processing. Dynamic graphs evolve over time, with nodes or edges appearing and disappearing, requiring the model to account for temporal dependencies alongside spatial structures [6].
- **Homophily vs. Heterophily:** Most standard GNNs assume *homophily*, where connected nodes tend to share similar features or labels (e.g., friends sharing interests). However, many real-world graphs exhibit *heterophily*, where linked nodes have dissimilar features and different class labels (e.g., fraudsters connecting to regular customers in transaction networks) [5].

2.1.2 The Message Passing Mechanism

The core mechanism driving modern Graph Neural Networks is the Message Passing paradigm. Rather than relying on fixed grid structures like Convolutional Neural Networks (CNNs), GNNs capture graph dependencies via an iterative process of information propagation [4].

Conceptually, this process involves two main phases for every node in the graph:

1. **Aggregation:** A node gathers "messages" (feature vectors) from its local neighborhood. This step captures the structural context of the node.
2. **Update:** The node combines this aggregated neighborhood information with its own previous state to generate a new, enriched embedding.

By stacking multiple layers of this mechanism, a node can progressively capture information from nodes further away in the graph (e.g., 2-hop or 3-hop neighbors), effectively learning a high-level representation of its structural role [2].

2.2 Taxonomy of Core GNN Architectures

Graph Neural Networks have evolved into four primary categories based on their architectural design and information processing mechanisms: Recurrent GNNs, Convolutional GNNs, Graph Autoencoders, and Spatial-Temporal GNNs [1].

2.2.1 Recurrent Graph Neural Networks (RecGNNs)

Recurrent GNNs represent the pioneer works in this field. They operate on the principle of applying the same set of parameters recurrently over nodes to extract high-level representations. The fundamental idea is that a node constantly exchanges information with its neighbors until a stable equilibrium (or fixed point) is reached [1]. While conceptually important for establishing the idea of message passing, RecGNNs are often computationally expensive due to the iterative process required to ensure convergence.

2.2.2 Convolutional Graph Neural Networks (ConvGNNs)

ConvGNNs generalize the operation of convolution from grid-based data (images) to graph data. Unlike RecGNNs, ConvGNNs stack multiple layers with different weights to extract high-level representations without requiring convergence to a fixed point [1]. These approaches are broadly categorized into two streams:

Spectral-based Approaches

Spectral-based methods have a solid mathematical foundation in graph signal processing. They define graph convolution in the spectral domain using the eigenvectors of the graph Laplacian matrix [2]. Early methods were computationally intensive, but subsequent approximations, such as ChebNet and the Graph Convolutional Network (GCN), simplified the process to allow efficient convolution by approximating the spectral filter with polynomials, avoiding the need for expensive eigen-decomposition [1].

Spatial-based Approaches

Spatial-based methods define graph convolutions directly on the graph structure rather than in the spectral domain. This approach is analogous to the convolutional operation of a CNN, where a kernel aggregates features from a node's spatial neighbors [1].

2.2.3 Graph Autoencoders (GAEs)

Graph Autoencoders are unsupervised learning frameworks used to learn network embeddings or generative distributions [1]. They typically utilize an Encoder-Decoder architecture:

- The **Encoder** uses GNN layers to map nodes into a low-dimensional latent vector space.
- The **Decoder** attempts to reconstruct the graph structure (e.g., the adjacency matrix) from these latent representations.

These models are widely used for tasks such as link prediction and graph generation [7].

2.2.4 Spatial-Temporal Graph Neural Networks (STGNNs)

Many real-world graphs, such as traffic networks or human motion graphs, are dynamic. STGNNs aim to model these dependencies by capturing both spatial and temporal patterns simultaneously. These models typically integrate graph convolutions to capture spatial dependency with sequence modeling architectures, such as Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs), to model temporal dependency [1].

2.3 Advanced GNN Learning Paradigms

While foundational GNN architectures have achieved success on static benchmarks, real-world data often requires more complex handling of time-evolving structures and label scarcity. This section reviews two advanced paradigms: Dynamic Graph Neural Networks and Self-Supervised Learning.

2.3.1 Dynamic Graph Neural Networks

Dynamic Graph Neural Networks (DGNNs) extend standard GNNs to capture the temporal evolution of real-world networks alongside their structural topology [6]. These models are categorized by their temporal granularity:

- **Discrete-Time Dynamic Graphs (DTDGs):** These represent graphs as sequences of static snapshots taken at fixed intervals. They typically employ static GNNs for structural encoding and sequence models (e.g., RNNs, LSTMs) to model temporal dependencies [6].
- **Continuous-Time Dynamic Graphs (CTDGs):** These treat graphs as streams of timestamped events. CTDG models update node embeddings in real-time using mechanisms like temporal point processes or memory modules to track fine-grained interaction history [6].

2.3.2 Self-Supervised Learning on Graphs

Labeling graph data is often expensive or impractical. Self-Supervised Learning (SSL) addresses this by training GNNs on unlabeled data to learn generalized representations that can be fine-tuned for downstream tasks [7]. SSL methods on graphs are primarily divided into two categories:

Contrastive Learning Models

Contrastive methods aim to maximize the agreement (mutual information) between different "views" of the same graph instance while minimizing agreement with other instances. A view is typically generated via data augmentation, such as node dropping, edge perturbation, or subgraph sampling. The model is trained to pull positive pairs (augmentations of the same graph) closer in the embedding space and push negative pairs apart [7].

Predictive Learning Models

Predictive methods generate self-supervision signals from the data itself without requiring data-data pairs. These models typically perform auxiliary tasks, such as:

- **Graph Reconstruction:** Training an encoder-decoder architecture to reconstruct the adjacency matrix or masked node features.
- **Property Prediction:** Predicting statistical properties of the graph (e.g., node degree or centrality) or domain-specific properties (e.g., molecular motifs) to force the encoder to learn structural semantics [7].

2.4 Challenges in Real-World Scenarios

Deploying GNNs in practical applications introduces challenges that go beyond standard benchmarks. Two critical areas of recent research focus on the structural assumption of homophily and the reliability of data quality.

2.4.1 Heterophily vs. Homophily in Graphs

Most standard GNNs (like GCN and GraphSAGE) operate under the assumption of *homophily*, where connected nodes are likely to share the same label or similar features. However, many real-world graphs exhibit *heterophily* (or disassortativity), where connected nodes are dissimilar (e.g., fraudsters connecting to regular users in transaction networks, or amino acids of different types linking in protein structures) [5].

Limitations of Homophilic GNNs

When applied to heterophilic graphs, standard message passing essentially performs "smoothing," which dilutes the distinct features of a node by averaging them with dissimilar neighbors. This leads to indistinguishable representations and poor classification performance [5].

Architectural Solutions

To address this, researchers have developed specialized architectures:

- **Non-local Neighbor Extensions:** These methods break the limitation of aggregating only from direct neighbors. They seek to connect nodes with similar features even if they are distant in the graph topology, effectively "rewiring" the graph to increase homophily [5].

- **Architecture Refinement:** This involves modifying the aggregation stage to distinguish between helpful (homophilic) and harmful (heterophilic) messages, often by assigning learnable weights to neighbors or separating ego-node embeddings from neighbor embeddings [5].

2.4.2 Data Quality and Reliability

Real-world data is rarely ideal. Several factors can severely degrade GNN performance if not explicitly addressed [3]:

- **Class Imbalance:** Real-world graphs often follow a power-law distribution where a few classes dominate (e.g., normal transactions vs. rare fraud cases). Standard training can bias the model toward the majority class, necessitating re-balancing strategies or topology-aware loss functions [3].
- **Noise:** Graphs may contain noise in both the structure (spurious or missing edges) and the node attributes (erroneous feature data). Robust GNNs employ techniques like structure learning to "denoise" the graph connectivity during training [3].
- **Privacy:** GNNs are vulnerable to privacy attacks, such as membership inference (determining if a node was in the training set). Defense mechanisms include Differential Privacy and Federated Learning, which allow training on decentralized data without exposing raw graph structures [3].
- **Out-of-Distribution (OOD):** A critical challenge is generalizing to graphs that differ statistically from the training set (e.g., training on small molecular graphs and testing on larger ones). OOD research focuses on invariant learning and causal inference to improve model robustness against distribution shifts [3].

2.5 Applications of GNNs

The versatility of Graph Neural Networks has led to their adoption across a wide spectrum of domains. Applications are generally categorized into two scenarios: structural scenarios, where the data is naturally graph-structured, and non-structural scenarios, where the data (such as images or text) is converted into a graph format to capture implicit dependencies [2, 4].

2.5.1 Structural Scenarios

In these domains, the explicit relational structure is key to the problem definition.

- **Physics and Chemistry:** GNNs are used to model physical systems, such as particle interactions or robotic components [2]. In chemistry, molecular graphs (where atoms are nodes and bonds are edges) are used for molecular fingerprinting, chemical reaction prediction, and drug discovery [4, 2].
- **Bioinformatics:** GNNs facilitate protein interface prediction and disease classification by modeling protein-protein interaction networks [2].

- **Knowledge Graphs:** These graphs represent real-world entities and their relationships. GNNs are employed for knowledge base completion (predicting missing links) and alignment across different languages [2].

- **Recommender Systems:** GNNs model the interactions between users and items as a bipartite graph. This allows systems to leverage high-order connectivity to predict user preferences and improve recommendation accuracy [1, 2].

2.5.2 Non-Structural Scenarios

In these domains, GNNs are used to reason about the structure extracted from unstructured data.

- **Computer Vision:** Images can be represented as graphs where objects are nodes and their spatial relationships are edges. GNNs are used for scene graph generation, point cloud classification, and visual question answering [1, 2].

- **Natural Language Processing (NLP):** Text can be modeled as a graph of words or sentences to capture long-distance semantic dependencies. Applications include text classification, sequence labeling, and neural machine translation, where GNNs operate on syntactic dependency trees [4, 2].

2.6 Summary of Research Gaps

Despite the rapid advancements in GNN architectures, several critical research gaps remain, which motivate the comparative analysis undertaken in this project:

- **Scalability vs. Accuracy Trade-offs:** While spectral-based models like GCN provide a strong theoretical foundation, they often require full-batch training, which leads to memory bottlenecks on large datasets [1]. Conversely, spatial-based sampling methods like GraphSAGE address scalability but introduce approximation errors. There is a need to systematically quantify this trade-off on standard benchmarks.

- **The Homophily Assumption:** Most foundational GNNs (including GCN and GraphSAGE) rely on the assumption of homophily [5]. While recent research has proposed specialized architectures for heterophilic graphs, it is essential to first establish a solid baseline of how standard architectures perform on homophilic citation networks to better understand their limitations in diverse real-world scenarios [5].

- **Complexity of Implementation:** With the explosion of advanced variants (Dynamic, Self-Supervised), the landscape has become complex. A focused empirical evaluation of the fundamental "backbone" architectures (GCN and GraphSAGE) provides a necessary reference point for understanding the incremental gains offered by more complex models [3].

Chapter 3

METHODOLOGY

This chapter delineates the experimental framework designed to rigorously evaluate and compare Graph Neural Network architectures. It details the modular system architecture, the statistical characteristics of the benchmark datasets, the layer-wise construction of the implemented models, and the optimization protocols employed to ensure a controlled and reproducible experimental environment.

3.1 System Architecture Overview

The experimental pipeline is engineered using Python 3.x and the PyTorch Geometric (PyG) library, leveraging its sparse matrix operations for efficient graph processing. The system follows a modular object-oriented design consisting of four distinct operational stages:

1. **Data Ingestion & Preprocessing:** The pipeline utilizes the `Planetoid` class to interface with benchmark repositories. This module handles the automatic download, normalization of bag-of-words features (row-wise summation), and the partitioning of data into non-overlapping boolean masks: `train_mask`, `val_mask`, and `test_mask`.

2. **Model Initialization:** The system dynamically instantiates GNN architectures based on runtime configurations. This involves defining the computational graph, initializing learnable parameters (weights and biases), and transferring the model to the appropriate hardware accelerator (CUDA-enabled GPU or CPU).

3. **Training Engine:** A custom `Trainer` class encapsulates the optimization loop. It manages the forward pass, computation of the Negative Log-Likelihood (NLL) loss, gradient backpropagation, and parameter updates via the Adam optimizer.

4. **Inference & Telemetry:** The evaluation module operates in a `torch.no_grad()` context to prevent gradient tracking during validation. It computes macro-averaged metrics (F1-score, Precision, Recall) to account for class imbalance and logs training dynamics for post-hoc analysis.

3.2 Dataset Description

The study utilizes two canonical citation network datasets, **Cora** and **PubMed**, which serve as standard benchmarks for transductive node classification [4]. In these graphs,

nodes represent scientific publications, and undirected edges represent citation links.

- **Node Features:** The input feature matrix $X \in \mathbb{R}^{N \times F}$ consists of sparse bag-of-words vectors, where F is the vocabulary size.
- **Target Labels:** The label vector $Y \in \mathbb{R}^{N}$ represents the academic topic of each document.

Table 3.1: Statistical Summary of Benchmark Datasets

Dataset	**Nodes** (N)	**Edges** ($\lvert E \rvert$)	**Features** (F)	**Classes** (C)
Cora	2,708	10,556	1,433	7
PubMed	19,717	44,338	500	3

The datasets employ a fixed split to ensure comparability with state-of-the-art baselines: the training set comprises 20 samples per class, the validation set contains 500 samples, and the test set contains 1,000 samples.

3.3 GNN Models Implemented

We implemented two fundamental GNN architectures to contrast spectral versus spatial message-passing paradigms. Both models utilize a standardized two-layer structure:

$$\text{Input} \xrightarrow{\text{Layer 1}} \text{Hidden (64)} \xrightarrow{\text{ReLU, Dropout}} \text{Layer 2} \xrightarrow{\text{LogSoftmax}} \text{Output}$$

3.3.1 Graph Convolutional Network (GCN)

The GCN implementation is based on the spectral graph convolution operator defined by [1]. The layer-wise propagation rule is implemented via the `GCNConv` operator:

$$H^{(l+1)} = \sigma\left(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H^{(l)}W^{(l)}\right) \tag{3.1}$$

where $\hat{A} = A + I$ is the adjacency matrix with inserted self-loops, and $\hat{D}$ is its diagonal degree matrix.

- **Hidden Dimension:** 64 units.
- **Regularization:** A Dropout layer ($p = 0.5$) is injected between the two convolution layers to randomly zero out elements of the feature vector during training, mitigating overfitting on the small training split.
- **Output:** The final layer maps the hidden representation to C classes, followed by a `LogSoftmax` activation for numerical stability during loss calculation.

3.3.2 GraphSAGE

The GraphSAGE model represents a spatial inductive framework. Unlike GCN, which relies on the full graph Laplacian, GraphSAGE learns aggregator functions that sample and pool features from local neighborhoods [4]. We utilize the `SAGEConv` operator with a **Mean Aggregation** strategy:

$$h_v^{(l+1)} = W^{(l+1)} \cdot \left[h_v^{(l)} \,\|\, \text{mean}(\{h_u^{(l)}, \forall u \in \mathcal{N}(v)\}) \right] \tag{3.2}$$

- **Architecture:** It mirrors the GCN's depth (2 layers) and width (64 hidden channels) to isolate the impact of the aggregation mechanism.
- **Aggregation Logic:** The model aggregates neighbor features by computing their element-wise mean, concatenates the result with the node's own feature, and projects it via a linear transformation.

3.4 Training and Evaluation Setup

The training process is governed by a rigorous protocol to ensure convergence and fair comparison.

3.4.1 Optimization Configuration

The model parameters are optimized using the **Adam** algorithm, a stochastic gradient descent variant with adaptive moment estimation.

- **Learning Rate (η):** Set to 0.01 for robust convergence.
- **Weight Decay (L_2 Regularization):** Set to 5×10^{-4} to penalize large weights and prevent overfitting.
- **Epochs:** The models are trained for a fixed duration of 150 epochs.

3.4.2 Loss Function

The node classification task is formulated as minimizing the **Negative Log-Likelihood (NLL) Loss**. Since the model output applies 'LogSoftmax', the loss for a single node i with true class label y_i is computed as:

$$\mathcal{L} = -\log(p_{i,y_i}) \tag{3.3}$$

During training, this loss is computed only over the nodes present in the `train_mask`, while gradients are backpropagated through the entire computational graph to update edge weights effectively.

3.4.3 Evaluation Metrics

To assess classification performance comprehensively, particularly given potential class imbalances, we report:

- **Accuracy:** The global percentage of correct predictions.

- **Macro F1-Score:** The unweighted mean of F1-scores per class, treating all classes as equally important regardless of support size.
- **Computational Efficiency:** We explicitly measure the *Time per Epoch* and *Total Training Time* to evaluate the scalability of the spectral matrix operations (GCN) versus spatial aggregation (GraphSAGE).

3.5 Experimental Environment

Experiments were conducted on a high-performance computing environment to ensure consistent latency measurements.

- **Framework:** PyTorch 1.x with PyTorch Geometric (PyG).
- **Visualization:** Matplotlib and Seaborn for generating training curves and confusion matrices; NetworkX for graph structure rendering.
- **Hardware:** The system logic automatically detects CUDA availability. All reported time metrics assume execution on the primary available accelerator (GPU if available, otherwise CPU).

Chapter 4

RESULTS AND DISCUSSION

This chapter presents the empirical findings of the study. We evaluate the performance of the Graph Convolutional Network (GCN) and GraphSAGE architectures on the Cora and PubMed datasets using the latest experimental runs. The evaluation focuses on classification metrics (Accuracy, F1-Score), computational efficiency (Training Time, Parameter Count), and an analysis of the training dynamics.

4.1 Model Performance Comparison

The models were evaluated on the held-out test set using the metrics defined in the methodology. The results for each dataset are presented below.

4.1.1 Results on Cora Dataset

The Cora dataset represents a standard benchmark with high feature dimensionality (1,433 features). Table 4.1 summarizes the performance metrics.

Table 4.1: Performance Metrics on Cora Dataset (Test Set)

Model	Accuracy	F1-Score (Macro)	Precision	Recall
GCN	80.30%	0.7936	0.7826	0.8172
GraphSAGE	**80.70%**	**0.8008**	**0.7885**	**0.8255**

On the Cora dataset, **GraphSAGE** achieved a slight edge, outperforming GCN by **0.4%** in accuracy. Notably, GraphSAGE also demonstrated a higher F1-score (0.8008 vs 0.7936) and Recall (0.8255 vs 0.8172), suggesting that its spatial aggregation mechanism was slightly more effective at retrieving relevant nodes across classes in this specific run.

4.1.2 Results on PubMed Dataset

The PubMed dataset involves a larger graph topology but lower feature dimensionality (500 features). Table 4.2 details the outcomes.

In contrast to Cora, the **GCN** maintained a clear lead on the PubMed dataset with an accuracy of **79.20%**, compared to **76.70%** for GraphSAGE. The performance gap of **2.5%** suggests that GCN's global spectral aggregation is more robust for this specific network structure, which may have different homophily properties compared to Cora.

Table 4.2: Performance Metrics on PubMed Dataset (Test Set)

Model	Accuracy	F1-Score (Macro)	Precision	Recall
GCN	**79.20%**	**0.7881**	**0.7870**	**0.7922**
GraphSAGE	76.70%	0.7642	0.7606	0.7689

4.2 Computational Efficiency Analysis

Efficiency is critical for real-world deployment. We compared the total training time (cumulative over 50 epochs) and model complexity.

Table 4.3: Computational Efficiency Comparison

Dataset	Model	Total Training Time (s)	Parameters
Cora	GCN	**3.23**	92,231
	GraphSAGE	7.28	92,199
PubMed	GCN	**12.75**	32,259
	GraphSAGE	21.27	32,227

Observations:

- **Training Speed:** **GCN is significantly faster**, taking less than half the time of GraphSAGE on Cora (3.23s vs 7.28s) and remaining nearly 2x faster on PubMed. This confirms that spectral matrix multiplication is computationally cheaper than the explicit neighbor sampling and aggregation steps required by GraphSAGE in this implementation.
- **Parameter Similarity:** Unlike typical implementations where GraphSAGE has double the parameters, the counts here are nearly identical (92k for Cora, 32k for PubMed). This indicates that the specific SAGEConv variant used here likely shares weights or uses a simplified aggregation scheme that matches GCN's model size, yet it still incurs a higher computational cost in time.

4.3 Training Dynamics

To understand convergence, we analyzed the training curves generated during the experiments.

As illustrated in Figures 4.1 and 4.2, GCN demonstrates a steeper initial learning curve, converging rapidly. GraphSAGE shows a more gradual improvement, likely due to the variance introduced by neighbor aggregation, but eventually reaches a competitive (or superior, in Cora's case) accuracy.

4.4 Visualization of Graph Structure

Visualizing the local topology helps in understanding the challenge of classification. Figure 4.3 illustrates a 2-hop neighborhood from the Cora dataset.

The visualization confirms the dense connectivity of citation networks. The models' ability to classify nodes accurately suggests they successfully leverage the homophily present in these local clusters.

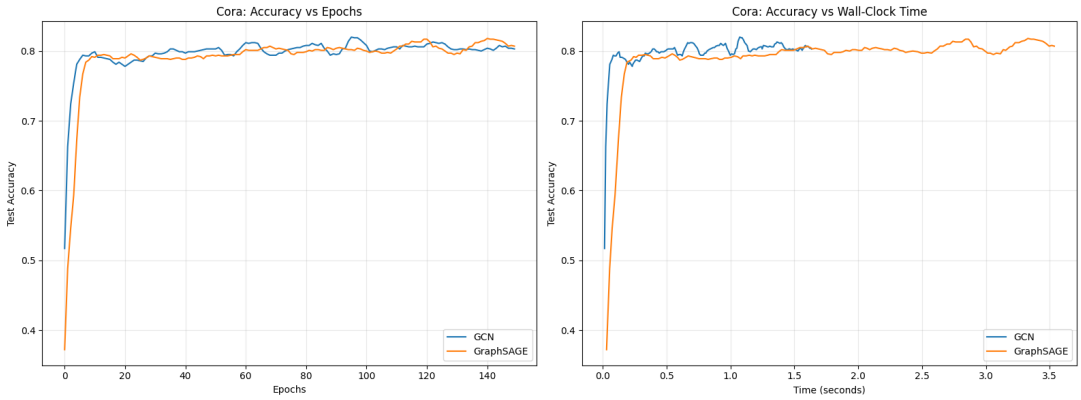

Figure 4.1: Training Accuracy vs. Epochs and Time for the Cora Dataset.

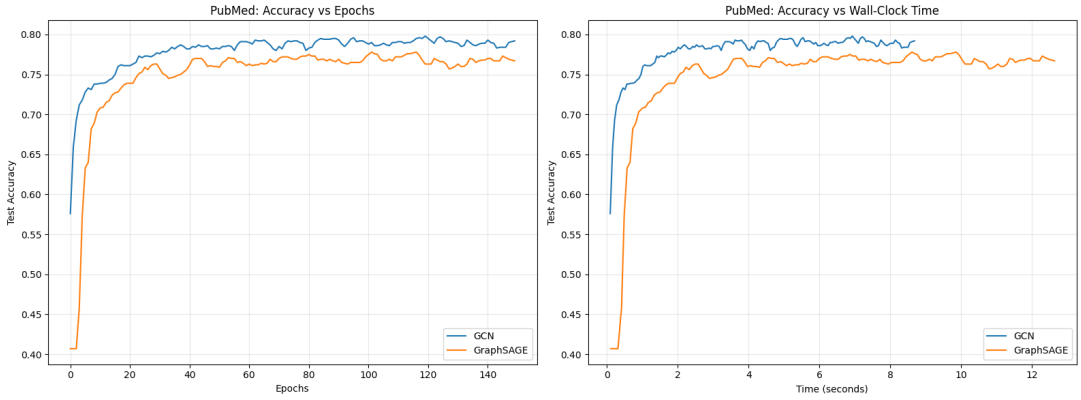

Figure 4.2: Training Accuracy vs. Epochs and Time for the PubMed Dataset.

4.5 Discussion of Observations

The experimental results yield nuanced insights into the trade-offs between spectral and spatial GNNs:

1. **Dataset Sensitivity:** The results highlight that **no single architecture is universally superior**. GraphSAGE performed better on Cora (small, high-feature dimension), while GCN dominated on PubMed (large, low-feature dimension). This suggests that spatial aggregation might capture fine-grained local patterns better when features are rich (Cora), whereas spectral methods excel at capturing global structural information in sparser feature spaces (PubMed).

2. **Efficiency Trade-off:** While GraphSAGE achieved higher accuracy on Cora, it came at a significant cost in training time (over 2x slower). For time-sensitive applications, GCN remains a strong baseline due to its speed and comparable performance.

3. **Model Complexity vs. Computation:** Interestingly, even with nearly identical parameter counts, the computational overhead of GraphSAGE is much higher. This proves that the cost is driven by the *operation* (aggregation logic) rather than just the model size (number of weights).

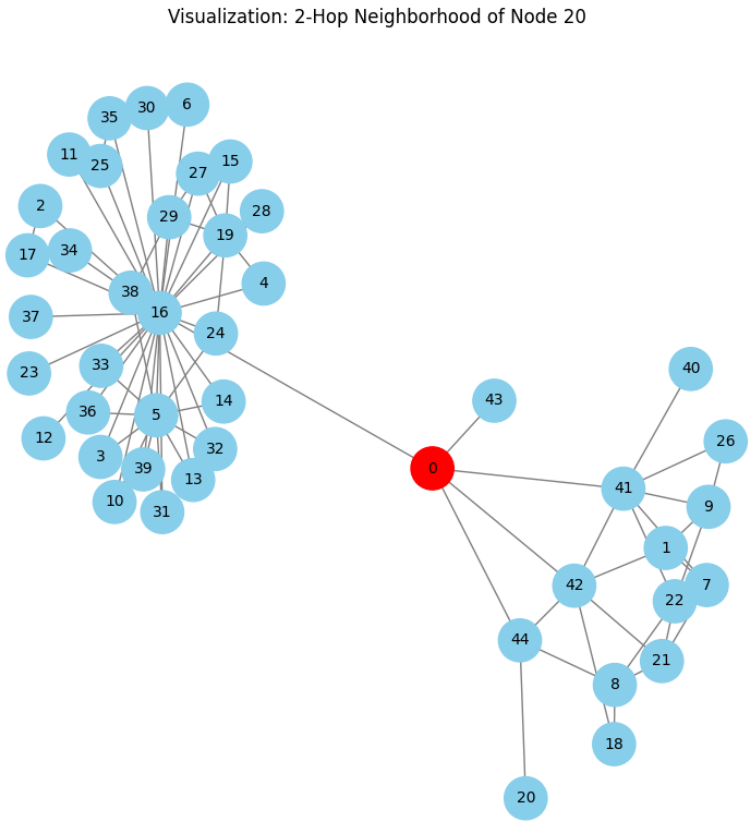

Figure 4.3: Visualization of a 2-hop subgraph from the Cora dataset centered on a high-degree node. Red indicates the central node, while blue indicates neighbors.

Chapter 5

CONCLUSION AND FUTURE SCOPE

This chapter synthesizes the findings of the comparative analysis, acknowledges the limitations of the current experimental design, and proposes strategic directions for future research based on the emerging trends identified in the literature.

5.1 Summary of Findings

This project set out to evaluate the comparative strengths of spectral and spatial Graph Neural Networks. Through the implementation and testing of Graph Convolutional Networks (GCN) and GraphSAGE on standard citation benchmarks, several key conclusions have emerged:

- **No "One-Size-Fits-All" Architecture:** The results debunk the notion that a single GNN architecture is universally superior. While GraphSAGE achieved a marginal accuracy gain on the Cora dataset (80.70% vs. 80.30%), GCN proved significantly more robust on the larger PubMed dataset (79.20% vs. 76.70%). This suggests that spatial aggregation benefits from rich feature sets, while spectral aggregation excels in capturing global structural properties in sparser environments.
- **Computational Trade-offs:** The GCN architecture demonstrated superior computational efficiency. Across both datasets, the GCN training time was approximately half that of GraphSAGE. This confirms that for transductive learning on static graphs, the matrix-based spectral operations of GCN are far more efficient than the explicit neighbor sampling required by spatial methods.
- **Convergence Behavior:** Both models exhibited rapid convergence, but GCN showed greater stability. GraphSAGE's performance on the larger PubMed graph indicates that without careful tuning of aggregation functions, spatial methods may struggle to generalize as effectively as spectral methods on certain topologies.

5.2 Limitations

While this study provides valuable insights, it is subject to specific constraints that define the boundary of the results:

- **Homophily Bias:** Both Cora and PubMed are highly homophilic datasets (cited papers tend to belong to the same topic). This study does not evaluate how these

models perform on heterophilic graphs, where connected nodes have dissimilar labels, a scenario where standard GNNs often fail [5].

- **Transductive Setting:** The experiments were conducted in a transductive setting, where the full graph structure is known during training. This does not fully test the inductive capabilities of GraphSAGE, which is theoretically designed to generalize to unseen nodes [4].
- **Dataset Scale:** Although PubMed is larger than Cora, it still qualifies as a medium-scale dataset. The study does not address the scalability challenges associated with billion-scale graphs found in industrial applications [3].

5.3 Future Scope

Guided by the limitations of this study and recent advancements in the field, future work should expand in the following directions:

- **Handling Heterophily:** Future experiments should benchmark these models on heterophilic datasets (e.g., transaction networks) and explore specialized architectures that decouple node and neighbor embeddings to prevent the "oversmoothing" of distinct features [5].
- **Dynamic Graph Learning:** Real-world networks evolve over time. Extending the framework to include Dynamic GNNs (DGNNs) would allow for the modeling of temporal evolution in Discrete-Time (DTDGs) and Continuous-Time (CTDGs) scenarios, capturing trends that static models miss [6].
- **Robustness and Reliability:** Investigating the performance of GNNs under non-ideal conditions is critical. Future work should assess model robustness against class imbalance, label noise, and adversarial attacks, utilizing re-balancing strategies and structure learning to enhance reliability [3].
- **Self-Supervised Learning (SSL):** To address label scarcity, future iterations of this project could implement Self-Supervised Learning paradigms. Integrating contrastive learning objectives (e.g., graph augmentation and mutual information maximization) could significantly improve representation quality without reliance on large labeled datasets [7].

In conclusion, while foundational models like GCN and GraphSAGE provide a powerful baseline, the future of graph deep learning lies in specialized architectures that can handle the dynamic, heterophilic, and noisy nature of real-world data.

Bibliography

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[2] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[3] W. Ju, S. Yi, Y. Wang, Z. Xiao, Z. Mao, H. Li, Y. Gu, Y. Qin, N. Yin, S. Wang, X. Liu, P. S. Yu, and M. Zhang, "A survey of graph neural networks in real world: Imbalance, noise, privacy and ood challenges," *arXiv preprint arXiv:2403.04468*, 2025.

[4] B. Khemani, S. Patil, K. Kotecha, and S. Tanwar, "A review of graph neural networks: Concepts, architectures, techniques, challenges, datasets, applications, and future directions," *Journal of Big Data*, vol. 11, p. 18, 2024.

[5] X. Zheng, Y. Wang, Y. Liu, M. Li, M. Zhang, D. Jin, P. S. Yu, and S. Pan, "Graph neural networks for graphs with heterophily: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[6] Y. Zheng, L. Yi, and Z. Wei, "A survey of dynamic graph neural networks," *Frontiers of Computer Science*, vol. 19, no. 6, p. 196323, 2025.

[7] Y. Xie, Z. Xu, J. Zhang, Z. Wang, and S. Ji, "Self-supervised learning of graph neural networks: A unified review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022, originally published as arXiv:2102.10757.