EUROPEAN JOURNAL OF EMERGING ARTIFICIAL INTELLIGENCE

Graph Neural Networks: A Foundational Guide for the Applied ML Engineer

Dr. Jalen M. Korvic
Department of Computer Engineering Nordwell Institute of Technology, Finland

V0LUME02 ISSUE02 (2025)

Published Date: 11 July 2025 // Page no.: - 01-10

ABSTRACT

Graph Neural Networks (GNNs) have emerged as a powerful class of deep learning models designed to handle the complexities of graph-structured data. Their ability to learn from relational information has led to state-of-the-art performance on a wide array of tasks, from social network analysis to molecular chemistry. However, for machine learning engineers new to this domain, the steep learning curve and the sheer variety of GNN architectures can be daunting. This article provides a comprehensive introduction to GNNs, grounding the discussion in the intuitive encoder-decoder framework. We focus on three foundational GNN architectures—Graph Convolutional Networks (GCN), GraphSAGE, and Graph Attention Networks (GATv2)—to build a concrete understanding of their mechanisms. Through a series of extensive experiments on thirteen homogeneous graph datasets, we systematically investigate how GNN performance is influenced by fundamental graph properties, particularly homophily, and varying training conditions. We compare these models against established baselines, including Multilayer Perceptrons (MLP) and DeepWalk, to highlight the unique advantages of GNNs. Our findings reveal that architectural choices are critical; more flexible models like GraphSAGE excel on low-homophily graphs, whereas simpler, more rigid models like GCN are highly effective on high-homophily graphs, especially in low-data regimes. Furthermore, we demonstrate that hyperparameter tuning offers the most significant performance gains in moderately difficult learning scenarios. By combining theoretical explanations with empirical evidence and qualitative analyses of the GNN learning process, this work serves as a practical and accessible starting point for engineers looking to effectively develop and deploy GNNs.

Keywords: Graph neural networks, Graph representation learning, Deep learning, Encoder-decoder models, Node classification, Hyperparameter tuning.

1. Introduction

1.1 Broad Background and Historical Context

In our increasingly interconnected world, a vast amount of data is inherently relational. From the intricate web of friendships on social media platforms to the complex interactions between proteins in a biological system, relationships between entities are often as important as the entities themselves. Graphs provide a natural and powerful mathematical framework for representing such relational data, where entities are modeled as nodes and their relationships as edges. For decades, analyzing these structures has been a cornerstone of fields ranging from sociology and physics to computer science.

However, the rise of modern machine learning, particularly deep learning, initially presented a challenge for graph data. The spectacular success of models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) was built on their ability to process regular, grid-like data structures such as images (2D grids of pixels) and text (1D sequences of words). Graphs, in contrast, are irregular. They lack a natural ordering of nodes, the neighborhood of each node can vary in size, and they do not possess a straightforward coordinate system. These properties render standard deep learning architectures incompatible with raw graph data, preventing them from directly exploiting the rich relational information encoded in the graph's topology.

Early attempts to bridge this gap involved a two-stage approach. First, graph-based features were manually engineered using traditional graph theory metrics like node centrality or clustering coefficients. These hand-crafted features were then fed into conventional machine learning models. While sometimes effective, this method was suboptimal because the feature engineering process was separate from the model's training, meaning the features were not fine-tuned for the specific downstream task.

A significant breakthrough came with the development of graph representation learning, a paradigm focused on automatically learning low-dimensional vector representations, or "embeddings," for nodes in a graph [2]. The goal was to produce embeddings that capture the graph's structural information. Seminal methods like DeepWalk [16] and node2vec [65] pioneered this approach by treating random walks on the graph as sentences and applying techniques from natural language processing, such as Skip-Gram, to learn node embeddings. These shallow embedding methods proved highly effective but came with their own limitations. They were primarily *transductive*, meaning they could only generate embeddings for nodes seen during training, making them unsuitable for dynamic graphs or predictions on entirely new graphs. Furthermore, they were often unable to incorporate node attributes (features associated with each node), relying solely on the graph's edge structure.

1.2 Critical Literature Review

Graph Neural Networks (GNNs) emerged as a transformative solution to these challenges, unifying feature learning and graph topology within a single, end-to-end deep learning framework [5, 6]. GNNs operate directly on the graph structure and can naturally incorporate both node and edge attributes, learning representations that are optimized for a given task. This capability is rooted in a powerful relational inductive bias, allowing them to generalize to unseen nodes and graphs [1]. The core idea behind most modern GNNs is a neighborhood aggregation or message-passing scheme, where each node iteratively updates its feature vector by aggregating information from its neighbors [45, 74]. After several iterations, a node's embedding effectively summarizes information from its multi-hop neighborhood.

The GNN landscape has grown rapidly, leading to a proliferation of architectures and a wealth of literature attempting to categorize them [7, 8, 9]. These surveys often group GNNs based on their underlying mathematical principles, such as spectral approaches that build on graph signal processing [69, 81] or spatial approaches that define operations directly in the node domain [17, 77]. A broader perspective frames GNNs within the context of geometric deep learning, which seeks to generalize deep learning to non-Euclidean data like graphs and manifolds [4]. This rich theoretical foundation has spurred the development of countless models and applications.

GNNs have been successfully applied to a diverse range of problems. In social networks, they are used for node classification to predict user attributes or interests [13, 14] and for link prediction to forecast future connections [19]. In recommender systems, GNNs model the interactions between users and items to provide personalized recommendations [12, 21]. The field of bioinformatics has also seen significant impact, with GNNs predicting drug-disease associations [24], measuring disease similarity [25], and, most notably, accelerating drug discovery by predicting molecular properties from their graph representations [42, 46]. Other applications include fake news detection [11], knowledge graph completion [26, 27], entity resolution [22, 23], and traffic prediction [37, 38].

Despite their success, GNNs are not without challenges. One well-known issue is *oversmoothing*, where stacking too many GNN layers causes node representations to become indistinguishable, degrading performance [10]. Another is the challenge of modeling graphs with low *homophily* (or high *heterophily*), where connected nodes tend to be of different classes or have dissimilar features—a scenario that violates the assumptions of many basic GNN models [71, 80].

To navigate this complex landscape, it is helpful to adopt a unifying framework. The encoder-decoder framework provides an elegant and powerful lens through which to understand a wide variety of GNN-based models [55]. In this view, the GNN acts as an *encoder* that maps each node in a graph to a low-dimensional embedding. This embedding is then passed to a simpler *decoder* model that uses it to make a prediction for the specific task at hand, such as node classification [15], link prediction [56], or graph classification [54].

1.3 Research Gap

The explosion of GNN research has produced numerous survey papers. However, a gap exists for material tailored to machine learning engineers and practitioners who are new to the field. Existing surveys are often highly theoretical, focusing on broad categorizations that can be abstract and difficult for newcomers to grasp [3, 4], or they are narrowly focused on a specific application or weakness, assuming significant prior knowledge [11, 12]. There is a pressing need for a concrete and concise introduction that bridges theory with practice, enabling engineers to develop an intuitive understanding of how GNNs behave.

Specifically, there is a lack of systematic, comparative studies that explore how foundational GNN architectures perform across datasets with varying structural properties, such as homophily. While many papers introduce novel architectures, they often benchmark them on a limited set of standard, high-homophily datasets. It is often unclear to a practitioner which model to choose for a new problem, especially if the underlying graph does not conform to these standard assumptions. Furthermore, the practical impact of hyperparameter tuning—a critical step in any deep learning workflow—is not always well-documented in a comparative context. Engineers need to know not just *that* tuning is important, but *when* it is most critical and *which* hyperparameters are likely to yield the most significant performance gains under different conditions.

1.4 Objectives and Hypotheses

This article aims to fill this gap by providing a foundational and empirically grounded introduction to GNNs. Our primary goal is to equip machine learning engineers with the practical knowledge and intuition needed to effectively apply GNNs to their own problems. To achieve this, we set the following objectives:

1. **To provide a clear, foundational understanding of GNNs** through the encoder-decoder framework, focusing on three seminal and widely used models: Graph Convolutional Network (GCN), GraphSAGE, and Graph Attention Network v2 (GATv2).

2. **To empirically evaluate and compare the performance** of these GNNs against non-graph-based (Multilayer Perceptron) and feature-agnostic (DeepWalk) baselines on a diverse collection of node classification tasks.

3. **To systematically investigate the impact of key architectural choices and hyperparameters** on model performance, specifically examining how performance varies across graphs with high and low homophily and under different training data regimes.

Based on these objectives, we formulated the following hypotheses:

- **Hypothesis 1:** GNN architectures that offer greater flexibility in their aggregation mechanism, such as the attention-based GATv2 and the concatenation-based GraphSAGE, will outperform the more rigid, convolution-based GCN on complex, low-homophily graphs where the standard neighborhood-averaging assumption is violated.

- **Hypothesis 2:** The performance benefits gained from extensive hyperparameter tuning will be most pronounced in moderately challenging learning scenarios (e.g., low-homophily graphs with a sufficient amount of training data). In contrast, in very easy (high-homophily, high data) or very hard (low-homophily, low data) scenarios, performance will be less sensitive to fine-tuning.

- **Hypothesis 3:** The optimal GNN architecture will depend heavily on graph characteristics. Specifically, deeper models with more message-passing layers will be beneficial for high-homophily graphs to capture broader neighborhood consensus, while shallower models augmented with additional non-relational processing layers (pre- or post-processing) will be more effective for low-homophily graphs to avoid aggregating conflicting information.

By testing these hypotheses through a carefully designed set of experiments, we aim to deliver not just a theoretical overview, but a practical playbook for understanding and using GNNs.

2. Methods

2.1 Research Design

To address our objectives and test our hypotheses, we employed a comprehensive, multi-faceted experimental design centered on the task of node classification. The overall design is a comparative study evaluating multiple graph learning models across a spectrum of datasets and training conditions.

The **independent variables** in our study were:

1. **Model Architecture:** We evaluated five distinct models: three canonical GNNs (GCN, GraphSAGE, GATv2), a Multilayer Perceptron (MLP) that uses only node features, and DeepWalk, a shallow embedding method that uses only the graph structure.

2. **Dataset Characteristics:** Datasets were categorized based on two key complexity measures: *edge homophily* (high vs. low) and the *signal-to-noise ratio (SNR)* of their node features.

3. **Training Set Size:** We simulated both data-rich and data-scarce scenarios by using two distinct training set sizes: 80% and 1% of the total labeled nodes.

4. **Hyperparameters:** We systematically tuned a range of architectural and training hyperparameters, including the number of hidden dimensions, the number and type of layers, layer connectivity, aggregation functions, and learning rate.

The primary **dependent variable** was **node classification accuracy**, measured on a held-out test set. All experiments were conducted within a **transductive learning** paradigm. In this setting, the model has access to the entire graph structure and the features of all nodes (including validation and test nodes) during training, but only the labels of the training nodes are used to compute the loss. This is a common evaluation setting for GNNs on static graphs [15].

2.2 Participants / Sample (Datasets)

We selected thirteen publicly available, open-source datasets to ensure a diverse range of graph structures, complexities, and domains. These datasets are all homogeneous graphs, meaning they consist of a single type of node and a single type of edge. They were strategically divided into two groups based on their edge homophily level.

Edge homophily is defined as the fraction of edges in the graph that connect two nodes of the same class [71]. A high value indicates that nodes tend to connect to similar nodes, while a low value indicates a tendency towards heterophily. The **signal-to-noise ratio (SNR)** was used as a measure of node feature quality, quantifying the separability of classes in the feature space. It was calculated as the ratio of the squared distance between class mean feature vectors (signal) to the sum of within-class feature variances (noise).

The datasets were:

- **Group 1: High Homophily Datasets (Homophily > 0.7)**
    - **Citation Networks:** Cora, CiteSeer, PubMed, and DBLP, where nodes are publications and edges are citations [101, 102].
    - **Co-purchase Networks:** Amazon Computers and Amazon Photo, where nodes are products and edges indicate they are frequently bought together [96].
    - **Co-author Network:** CoauthorCS, where nodes are authors and edges represent co-authorship on a publication [96].
- **Group 2: Low Homophily Datasets (Homophily < 0.3)**
    - **Webpage Networks:** WikipediaSquirrel, WikipediaChameleon, and WikipediaCrocodile, where nodes are Wikipedia pages and edges are mutual links between them. These were originally node regression datasets, which we converted to five-class classification problems by binning the target variable [41].
    - **Academic Networks:** Cornell and Wisconsin, which are smaller webpage networks from the WebKB collection [103].
    - **Co-occurrence Network:** Actor, where nodes are actors and edges indicate co-occurrence on the same Wikipedia page [103].

All datasets were used in their standard form. For the regression datasets that we converted, the node features and graph structure were used as-is, with only the target variable being transformed to create discrete classes.

2.3 Materials and Apparatus

Software and Hardware: All experiments were implemented in Python using the PyTorch deep learning framework and the PyTorch Geometric (PyG) library, which provides efficient implementations of many GNNs and standard graph datasets [108]. For systematic hyperparameter tuning, we leveraged the GraphGym platform [104]. Experiments were conducted on a standard workstation equipped with an Intel Core i9 CPU, 64GB of RAM, and an NVIDIA RTX 3090 GPU, although GPU usage was not strictly necessary for the smaller datasets.

Model Architectures:

- **GCN (Graph Convolutional Network):** We used the architecture proposed by Kipf and Welling [15]. Its message-passing layer performs a linear transformation on neighbor node features, aggregates them via summation (weighted by normalized node degrees), and adds a self-loop before applying a non-linear activation.
- **GraphSAGE (Graph Sample and Generalize):** We used the mean-aggregator variant from Hamilton et al. [17]. A key distinction of GraphSAGE is its two-stage update: first, it aggregates neighbor embeddings, and second, it concatenates the aggregated vector with the node's own current embedding before passing it through a fully connected layer. This concatenation provides a form of skip connection.
- **GATv2 (Graph Attention Network v2):** We opted for GATv2 over the original GAT because it is a more expressive and typically better-performing variant [78]. GATv2 computes attention coefficients for each neighboring node, indicating the importance of that neighbor's message. Unlike the original GAT, the attention function is designed to be fully dynamic, allowing the ranking of neighbors to change based on the query node. The neighbor messages are then linearly transformed, weighted by their attention scores, and summed.
- **MLP (Multilayer Perceptron):** Our MLP baseline was a standard fully connected neural network that processes each node's feature vector independently, completely ignoring the graph structure.
- **DeepWalk:** This baseline represents shallow, structure-only methods [16]. It first learns node embeddings using random walks and the Skip-Gram model and then trains a logistic regression classifier on top of these fixed embeddings.

For the initial baseline comparison, all GNNs were configured with two message-passing layers and 16 hidden dimensions. The MLP was a three-layer network. For subsequent experiments, these architectures were varied as part of the hyperparameter tuning process.

2.4 Data Collection Procedure

To ensure the reliability of our results, each experiment (a specific model-dataset-hyperparameter configuration) was run **25 times with different random seeds** for weight initialization and data shuffling.

Data Splitting: For each run, the dataset's nodes were partitioned into training, validation, and test sets. We investigated two primary conditions for the training set size: an **80% split** to model a data-rich scenario and a **1% split** to model a data-scarce scenario. In both cases, the remaining labeled nodes were split evenly between the validation and test sets.

Hyperparameter Tuning: Our tuning process was designed to be systematic and interpretable. It proceeded in two main stages:

1. **Baseline Experiments:** We first ran all models with a fixed, "off-the-shelf" configuration to establish baseline performance. Training was run for a maximum of 200 epochs with early stopping based on validation accuracy.

2. **Greedy Tuning Process:** To investigate the impact of specific hyperparameters, we performed a greedy search, following the methodology inspired by You et al. [104]. We started with a reasonably well-performing GNN configuration (128 hidden dimensions) and then iteratively tuned one hyperparameter at a time, fixing it to its best-performing option before moving to the next. The tuning order was: (1) number of message-passing layers, (2) number of post-processing layers, (3) number of pre-processing layers, (4) layer connectivity (skip connections), (5) aggregation function within a layer, and (6) learning rate. The best option for each was determined by the highest average test accuracy over 25 runs on the validation set. All tuned models were trained for a fixed 400 epochs to observe long-term learning dynamics.

2.5 Data Analysis

The collected data were analyzed using both quantitative and qualitative methods.

Quantitative Analysis: The primary evaluation metric was **node classification accuracy**. Results were reported as the **mean accuracy and 95% confidence interval** across the 25 runs. To analyze the relationship between the two dataset complexity measures, we calculated the **Pearson correlation coefficient** between dataset homophily and SNR.

Qualitative Analysis: To gain a deeper, more mechanistic understanding of how GNNs learn, we analyzed the evolution of node embeddings. We tracked the **signal** and **noise** energy of the embeddings throughout the training process and across the different layers of the GNNs. Following the formulation from recent work on GNN dynamics [107], we defined:

- **Signal Energy:** The average squared Euclidean distance between the mean feature vectors of different classes. A higher signal indicates better class separability.

- **Noise Energy:** The average intra-class variance of the feature vectors. Lower noise indicates that nodes of the same class have more compact representations.

By plotting these two metrics, we could visualize how GNN layers transform the feature space, either by enhancing the signal (pulling class clusters apart) or by reducing the noise (making clusters tighter). This provided a qualitative picture of what happens when GNNs learn.

3. Results

3.1 Preliminary Analyses

Before evaluating model performance, we first examined the properties of our datasets and the computational costs of the models. A Pearson correlation analysis between edge homophily and the signal-to-noise ratio (SNR) of node features across our thirteen datasets revealed a moderately strong positive correlation ($r = 0.62$, $p < 0.05$), excluding the two small outlier datasets, Cornell and Wisconsin. This indicates that graphs with high homophily also tend to have node features that are more easily separable by class, suggesting that low-homophily datasets often present a dual challenge of complex edge structure and noisy features.

Next, we measured the average CPU time required for training and evaluation on the Cora dataset, a common benchmark. The graph convolutional models, GCN (2.26 min) and GraphSAGE (2.23 min), exhibited computational costs comparable to the MLP baseline (2.58 min). The attentional network, GATv2, was moderately slower (4.03 min) due to the overhead of computing attention coefficients. The shallow embedding model, DeepWalk, was by far the slowest (57.96 min), as it requires a separate, computationally intensive random-walk sampling and embedding generation phase before classifier training.

A baseline comparison of "off-the-shelf" model architectures with 16 hidden dimensions yielded clear performance patterns dependent on graph homophily and training size.

With 80% of data for training on high-homophily graphs, all GNNs performed well, with GATv2 (86.95% accuracy) and GCN (85.31%) leading. On low-homophily graphs, however, GraphSAGE was the clear winner (56.69%), significantly outperforming GATv2 (44.89%) and GCN (37.69%), and even surpassing the feature-only MLP (54.97%). In the low-data regime (1% training), GNNs demonstrated a distinct advantage on high-homophily datasets over the MLP (e.g., GCN 72.57% vs. MLP 46.52%), highlighting their ability to effectively leverage edge information. Conversely, on low-homophily datasets with little data, all models struggled, with GraphSAGE (34.33%) holding a slight edge. DeepWalk performed competitively on high-homophily graphs but failed on low-homophily graphs, where edge structure is a poor indicator of class.

3.2 Main Findings (Hyperparameter Effects)

Effect of Hidden Dimensions: We systematically increased the number of hidden dimensions from 16 to 256 for the GNNs. On high-homophily graphs, increasing dimensions yielded consistent and significant performance improvements. For example, moving from 16 to 128 hidden dimensions improved GraphSAGE's accuracy by over 4.5% in the 80% training condition. In contrast, on low-homophily graphs, increasing dimensions provided only marginal gains. This suggests that simply increasing model capacity is not sufficient to overcome the challenge of complex graph structures; architectural modifications are necessary. Overfitting was not observed to be a major issue, even on the smallest datasets, likely due to the transductive learning setting where test features provide a regularizing effect.

Effect of Training Epochs: The analysis of learning curves over 400 epochs revealed divergent behaviors. On high-homophily datasets, GNN performance generally stabilized or continued to improve gradually over the full training period. However, for low-homophily datasets, the untuned GNN models often exhibited a sharp drop in accuracy after an initial peak around 25 epochs. This suggests that continued message passing on these graphs aggregated conflicting information, degrading the quality of node representations—a phenomenon related to oversmoothing.

Effect of Full Hyperparameter Tuning: We conducted a greedy hyperparameter tuning process, starting from a baseline with 128 hidden dimensions, and measured the improvement at each step. The results starkly depended on the training and dataset conditions, which we categorized as "easy" (high homophily, 80% training), "hard" (low homophily, 1% training), and "medium" (the other two combinations).

- In the **easy condition**, tuning provided minimal benefit (+0.57% for GCN). The off-the-shelf models were already highly effective.
- In the **hard condition**, tuning was detrimental, with performance dropping for all models (e.g., -3.89% for GraphSAGE). With very little training signal, the tuning process likely overfit to the small training set, finding brittle configurations that did not generalize.
- The most significant gains were observed in the **medium conditions**. On low-homophily graphs with 80% training data, tuning improved GCN accuracy by a remarkable +22.98% and GATv2 by +16.39%. On high-homophily graphs with 1% training data, GraphSAGE improved by +6.67%.

Analysis of the selected hyperparameters revealed clear patterns. For high-homophily graphs, the tuning process consistently favored deeper models with a large number of message-passing layers (averaging around 7 layers). For low-homophily graphs, the optimal models had fewer message-passing layers (around 3-4) but a greater number of pre- and post-processing layers, which refine node features without relying on neighborhood aggregation. The tuning process also recovered the expected stable learning curves for low-homophily datasets, resolving the performance drop seen in untuned models.

3.3 Exploratory Findings (Qualitative Learning Analysis)

To visualize the learning process, we tracked the signal and noise energy of the node embeddings. Across all conditions, GNN training was characterized by a rapid increase in signal energy (improved class separability) while noise energy (within-class variance) remained relatively stable or decreased. This contrasts with MLP models, where both signal and noise plateaued quickly, typically within 100 epochs.

Analyzing the energy flow through the layers of the tuned GNNs provided further insight.

- **On high-homophily graphs**, pre-processing layers initiated a reduction in noise. The subsequent message-passing layers further amplified the signal and reduced the noise, as aggregating features from same-class neighbors effectively smoothed the embeddings towards their class-centric representation.
- **On low-homophily graphs**, the message-passing layers presented a challenge. For GCN and GATv2, which rely on additive aggregation, this step actually *decreased* the signal energy, as conflicting information from different-class neighbors was mixed. However, for GraphSAGE, the signal energy continued to increase

through the message-passing layers. Its use of concatenation for the update function was critical, as it allowed the model to preserve the original node's information while incorporating neighborhood context, preventing the signal from being washed out.

Post-processing layers consistently contributed to further signal enhancement across all models and conditions, refining the learned embeddings for the final classification task. In contrast, stacking more than three layers in MLP models provided no additional benefit to signal or noise reduction, highlighting the architectural advantage of GNNs in progressively refining representations.

4. Discussion

4.1 Interpretation

Our experimental results provide a clear and empirically supported narrative about the behavior of foundational GNN architectures. The findings underscore that there is no single "best" GNN; instead, the optimal choice is deeply intertwined with the underlying characteristics of the graph data and the specifics of the learning task.

The stark performance difference between models on high- and low-homophily graphs is one of the most critical takeaways. The success of GCN on high-homophily graphs can be attributed to its strong, simplifying assumption that neighboring nodes are similar. Its weighted averaging acts as a powerful regularizer, smoothing node representations and effectively leveraging neighborhood consensus when the signal is clean. This makes it particularly potent in low-data regimes, where the graph structure provides a much-needed inductive bias that a feature-only model like MLP lacks.

Conversely, this same mechanism becomes a liability on low-homophily graphs. Aggregating features from neighbors that belong to different classes introduces noise and conflates distinct representations, a problem evidenced by the drop in signal energy we observed. Here, the architectural flexibility of GraphSAGE proves decisive. Its two-stage update mechanism, which concatenates the original node's embedding with the aggregated neighborhood vector, functions as a powerful skip connection. This ensures that a node's core identity is never lost, allowing the model to learn how to balance self-information against neighborhood information. This explains why GraphSAGE consistently outperformed other models on low-homophily datasets. GATv2, while more flexible than GCN due to its learned attention weights, still relies on a weighted sum for aggregation, making it susceptible to the same information-mixing problem, albeit to a lesser extent.

The results from our hyperparameter tuning experiments add another layer of practical insight. The finding that tuning is most beneficial in "medium difficulty" conditions is intuitive. In easy scenarios (high homophily, high data), the problem is simple enough that even a default model performs near the ceiling. In hard scenarios (low homophily, low data), there is insufficient signal in either the features or the graph structure for the model to learn from. In this case, complex architectural tuning risks overfitting to the tiny training set, leading to worse generalization, as we observed. The medium difficulty settings are the sweet spot where the problem is challenging, but enough signal exists for architectural enhancements to make a meaningful difference. The discovery that optimal models for low-homophily graphs favor fewer message-passing layers and more pre/post-processing layers confirms our hypothesis. These additional layers allow the model to learn more powerful feature transformations independent of the noisy neighborhood information, a crucial step before the final classification.

Our qualitative analysis of signal and noise energy provides a mechanistic explanation for these phenomena. We visually confirmed that GNNs learn by progressively separating class clusters in the embedding space. This analysis concretely illustrated why GCN struggles with low homophily (signal degradation during message passing) and why GraphSAGE succeeds (signal preservation via concatenation). This energy-based perspective offers a powerful tool for diagnosing and understanding GNN behavior beyond simple accuracy metrics.

4.2 Comparison with Literature

The findings of this study align with and provide empirical support for several key themes in the broader GNN literature. The central role of homophily in determining GNN performance is a well-established concept, and our results are consistent with surveys and studies dedicated to GNNs for heterophilous graphs [71, 80]. These works have similarly identified that standard message-passing schemes can fail when the homophily assumption is violated and have proposed solutions that, like GraphSAGE's concatenation, aim to preserve the root node's information.

Our use of the encoder-decoder framework to structure the analysis resonates with foundational texts on graph representation learning [2, 55]. This perspective clarifies that the primary role of the GNN encoder is to produce high-quality embeddings. Our results demonstrate how different encoder architectures (GCN, GATv2, GraphSAGE) produce embeddings of varying quality depending on the input graph, which in turn impacts the performance of the downstream decoder. The observed performance degradation in deep models on low-homophily graphs can also be viewed through the lens of *oversmoothing* [10].

While oversmoothing is typically discussed in the context of very deep networks, our results suggest that on graphs with complex, conflicting neighborhood signals, the effect can manifest with even a small number of layers, leading to the rapid performance drop we observed during training.

The superior performance of GNNs over the MLP baseline, particularly in low-data settings, empirically validates the concept of relational inductive bias [1]. The graph structure provides a powerful prior that guides the learning process, enabling generalization from very few labeled examples—a feat the feature-only MLP cannot achieve. Furthermore, the systematic exploration of the GNN design space, including the number and type of layers, echoes the approach taken by large-scale studies on GNN architecture design [104], but our work provides a more granular analysis of *why* certain designs are preferable under specific conditions. Our qualitative analysis of learning dynamics also connects to emerging theoretical work that models GNNs as gradient flows on an energy landscape, providing an experimental counterpart to these mathematical formulations [107].

4.3 Strengths and Limitations

This study has several strengths. Its primary strength lies in its systematic and comparative approach, evaluating a focused set of foundational models across a wide spectrum of thirteen datasets with diverse characteristics. This breadth allows for more generalizable conclusions than studies based on a few benchmark datasets. The dual focus on both quantitative accuracy and qualitative analysis of learning dynamics provides a richer, more nuanced understanding of model behavior. Finally, the paper's explicit goal of serving as a starting point for engineers, with clear connections between theory, experiments, and practical recommendations, fills a crucial gap in the existing literature.

However, the study is also subject to several limitations. First, our analysis is confined to the **transductive node classification task**. The findings may not directly generalize to other important graph learning tasks, such as link prediction, graph classification, or time-series forecasting on dynamic graphs. Second, all experiments were performed in a transductive setting. Performance in a fully **inductive setting**, where the test set consists of entirely unseen nodes or graphs, might differ, as the regularizing effect of having access to test features during training would be absent. Third, our study focused exclusively on **small- to medium-sized, homogeneous graphs**. The scalability and performance of these models on massive, industrial-scale graphs or on more complex heterogeneous graphs (with multiple node and edge types) remain open questions not addressed here. Lastly, our hyperparameter tuning process, while systematic, was based on a greedy search. A more exhaustive search, such as Bayesian optimization or random search, might have uncovered even better-performing configurations, though our goal was interpretability over finding the absolute peak performance.

4.4 Implications

Despite these limitations, our findings have significant implications for both practitioners and researchers.

Practical Implications: For machine learning engineers, this study offers a clear, actionable guide for approaching a new GNN problem.

1. **Start with Data Analysis:** Before selecting a model, analyze the graph's homophily. This single metric can strongly guide initial model selection.

2. **Model Selection Heuristics:** For high-homophily graphs, a simple GCN is a strong and efficient baseline. For low-homophily graphs, start with a model that has an explicit mechanism to preserve root node identity, like GraphSAGE.

3. **Strategic Hyperparameter Tuning:** Do not invest heavily in tuning for problems that are either very easy or appear intractably hard with very little data. Focus tuning efforts on moderately complex problems, paying close attention to the number of message-passing layers and the inclusion of pre/post-processing layers, as these architectural choices can unlock significant performance gains.

Theoretical Implications: Our work provides robust empirical validation for several core theoretical concepts in the GNN field. The signal-and-noise analysis offers a simple yet powerful framework for visualizing and reasoning about the flow of information through GNN layers. This approach could be refined and extended to become a standard diagnostic tool for understanding and debugging new GNN architectures. The results also highlight the critical role of skip connections—whether explicit or implicit, as in GraphSAGE's concatenation—in enabling deeper and more powerful graph models, reinforcing a key lesson from the broader deep learning literature.

4.5 Conclusion and Future Directions

Graph Neural Networks represent a fundamental shift in our ability to apply deep learning to relational data. In this article, we have sought to demystify these powerful models for machine learning engineers by grounding our discussion in the intuitive encoder-decoder framework and a series of systematic experiments. Our results demonstrate that the effectiveness of a GNN is not an intrinsic property of the model itself, but rather emerges from the interplay between its architecture, the characteristics of the graph, and the availability of training data. By understanding these dynamics, practitioners can move beyond treating GNNs as black boxes and begin to make informed, principled design choices.

Looking forward, several exciting research avenues remain. Extending this systematic analysis to the **inductive learning setting** is a critical next step, as it more closely mirrors many real-world deployment scenarios. A similar investigation on **large-scale, heterogeneous graphs** would provide invaluable insights for industrial applications. Further research could also explore the interplay between multiple hyperparameters more deeply using advanced optimization techniques. Finally, developing more sophisticated theoretical tools and qualitative analyses, like the signal-noise framework, will continue to be essential for unlocking the full potential of the next generation of graph neural networks.

References

[1] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner et al., "Relational inductive biases, deep learning, and graph networks," arXiv preprint arXiv:1806.01261, 2018.

[2] W. L. Hamilton, "Graph representation learning," Synthesis Lectures on Artifical Intelligence and Machine Learning, vol. 14, no. 3, pp. 1–159, 2020.

[3] I. Chami, S. Abu-El-Haija, B. Perozzi, C. R´e, and K. Murphy, "Machine learning on graphs: A model and comprehensive taxonomy," Journal of Machine Learning Research, vol. 23, no. 89, pp. 1–64, 2022.

[4] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veliˇckovi´c, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," arXiv preprint arXiv:2104.13478, 2021.

[5] L. Wu, P. Cui, J. Pei, L. Zhao, and X. Guo, "Graph neural networks: foundation, frontiers and applications," in Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2022, pp. 4840–4841.

[6] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," IEEE transactions on neural networks and learning systems, vol. 32, no. 1, pp. 4–24, 2020.

[7] 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.

[8] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," IEEE Transactions on Knowledge and Data Engineering, 2020.

[9] Y. Zhou, H. Zheng, X. Huang, S. Hao, D. Li, and J. Zhao, "Graph neural networks: Taxonomy, advances, and trends," ACM Transactions on Intelligent Systems and Technology (TIST), vol. 13, no. 1, pp. 1–54, 2022.

[10] T. K. Rusch, M. M. Bronstein, and S. Mishra, "A survey on oversmoothing in graph neural networks," arXiv preprint arXiv:2303.10993, 2023.

[11] H. T. Phan, N. T. Nguyen, and D. Hwang, "Fake news detection: A survey of graph neural network methods," Applied Soft Computing, vol. 139, p. 110235, 2023.

[12] C. Gao, Y. Zheng, N. Li, Y. Li, Y. Qin, J. Piao, Y. Quan, J. Chang, D. Jin, X. He et al., "A survey of graph neural networks for recommender systems: Challenges, methods, and directions," ACM Transactions on Recommender Systems, vol. 1, no. 1, pp. 1–51, 2023.

[13] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," in Social network data analytics. Springer, 2011, pp. 115–148.

[14] S. Ahmad, M. Z. Asghar, F. M. Alotaibi, and I. Awan, "Detection and classification of social media-based extremist affiliations using sentiment analysis techniques," Human-centric Computing and Information Sciences, vol. 9, no. 1, pp. 1–23, 2019.

[15] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in International Conference on Learning Representations, 2017.

[16] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 701–710.

[17] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 1025–1035.

[18] X. Jiang, Q. Wang, and B. Wang, "Adaptive convolution for multi-relational learning," in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Minneapolis, Minnesota: Association for Computational

Linguistics, Jun. 2019, pp. 978–987. [Online]. Available: https://aclanthology.org/N19-1103

[19] B. Pandey, P. K. Bhanodia, A. Khamparia, and D. K. Pandey, "A comprehensive survey of edge prediction in social networks: Techniques, parameters and challenges," Expert Systems with Applications, vol. 124, pp. 164–181, 2019.

[20] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," Computer, vol. 42, no. 8, pp. 30–37, 2009.

[21] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: a survey," ACM Computing Surveys, vol. 55, no. 5, pp. 1–37, 2022.

[22] S. Shekhar, D. Pai, and S. Ravindran, "Entity resolution in dynamic heterogeneous networks," in Companion Proceedings of the Web Conference 2020, 2020, pp. 662–668.

[23] B. Li, W. Wang, Y. Sun, L. Zhang, M. A. Ali, and Y. Wang, "Grapher: Token-centric entity resolution with graph convolutional neural networks," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 05, 2020, pp. 8172–8179.

[24] Z. Yu, F. Huang, X. Zhao, W. Xiao, and W. Zhang, "Predicting drug–disease associations through layer attention graph convolutional network," Briefings in Bioinformatics, vol. 22, no. 4, p. bbaa243, 2021.

[25] J. Gao, X. Zhang, L. Tian, Y. Liu, J. Wang, Z. Li, and X. Hu, "Mtgnn: Multi-task graph neural network based few-shot learning for disease similarity measurement," Methods, 2021.

[26] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A review of relational machine learning for knowledge graphs," Proceedings of the IEEE, vol. 104, no. 1, pp. 11–33, 2015.

[27] S. Arora, "A survey on graph neural networks for knowledge graph completion," arXiv preprint arXiv:2007.12374, 2020.

[28] N. R. Smith, P. N. Zivich, L. M. Frerichs, J. Moody, and A. E. Aiello, "A guide for choosing community detection algorithms in social network studies: The question alignment approach," American journal of preventive medicine, vol. 59, no. 4, pp. 597–605, 2020.

[29] Z. Yang, R. Algesheimer, and C. J. Tessone, "A comparative analysis of community detection algorithms on artificial networks," Scientific reports, vol. 6, no. 1, pp. 1–18, 2016.

[30] S. Bandyopadhyay and V. Peter, "Unsupervised constrained community detection via self-expressive graph neural network," in Uncertainty in Artificial Intelligence. PMLR, 2021, pp. 1078–1088.

[31] D. Jin, Z. Liu, W. Li, D. He, and W. Zhang, "Graph convolutional networks meet markov random fields: Semi-supervised community detection in attribute networks," in Proceedings of the AAAI conference on artificial intelligence, vol. 33, no. 01, 2019, pp. 152–159.

[32] C. Wang, C. Hao, and X. Guan, "Hierarchical and overlapping social circle identification in ego networks based on link clustering," Neurocomputing, vol. 381, pp. 322–335, 2020.

[33] G. Tauer, K. Date, R. Nagi, and M. Sudit, "An incremental graphpartitioning algorithm for entity resolution," Information Fusion, vol. 46, pp. 171–183, 2019.

[34] S. Maddila, S. Ramasubbareddy, and K. Govinda, "Crime and fraud detection using clustering techniques," Innovations in Computer Science and Engineering, pp. 135–143, 2020.

[35] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mane, D. Fritz, D. Krishnan, F. B. Vi´egas, and M. Wattenberg, "Visualizing dataflow graphs of deep learning models in tensorflow," IEEE transactions on visualization and computer graphics, vol. 24, no. 1, pp. 1–12, 2017.