



UNIVERSIDADE FEDERAL DO RIO GRANDE - FURG
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

Master Dissertation

**Bridging the Gap: A Systematic Framework for
Benchmarking Graph Neural Networks**

Willian Borges Lemos

Master Disstertation presented to the Programa de Pós-Graduação em Computação of the Universidade Federal do Rio Grande - FURG, in partial fulfillment of the requirements for the degree: Master in Computer Engineering

Advisor: Prof. Dr. Silvia Silva da Costa Botelho
Co-advisor: Prof. Dr. Marcelo de Rita Pias

Rio Grande, 2023



Master Dissertation

Bridging the Gap: A Systematic Framework for Benchmarking Graph Neural Networks

Willian Borges Lemos

Banca examinadora:

Prof. Dr. Leonardo Ramos Emmendorfer

Prof. Dr. Marcelo de Gomensoro Malheiros

Profa. Dra. Larissa Astrogildo de Freitas

Prof. Dr. Marcelo de Rita Pias (Coadvisor)

Prof. Dr. Silvia Silva da Costa Botelho (Advisor)

ACKNOWLEDGEMENTS

First, I would like to express my deep gratitude to my advisers, for their patient guidance, for the criticism of my work and for the tips in writing my dissertation. Second, I extend my acknowledgement to the entire Text of Things project team. We all work together, aiming for a world where technology is where it is needed. In particular, I want to thank three members of this wonderful team. Vanessa Lopes, Lucas Ribeiro, and Marcelo Pias , thank you for all your help, support and companionship during these years when we worked closely together. Also, I want to thank Wanessa Ribeiro. Your kindness and support inspired me every moment of this journey.

RESUMO

LEMOS, Willian Borges. **Bridging the Gap: A Systematic Framework for Benchmarking Graph Neural Networks**. 2023. 90 f. Dissertação (Master) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

O crescimento exponencial dos dados apresentou oportunidades e desafios para organizações e indivíduos que buscam extrair insights e padrões significativos. Nos últimos anos, o campo de análise de redes complexas tem recebido crescente atenção como solução para identificar padrões e estruturas em grafos. A técnica conhecida como Graph Embedding é uma etapa necessária para preencher a lacuna entre dados de grafos e algoritmos de aprendizado de máquina, permitindo o processamento e análise eficientes de estruturas de grafos. No entanto, o campo de graph embedding permanece fragmentado, com falta de uma compreensão teórica consensual. Este trabalho visa abordar essas questões fornecendo uma visão abrangente do estado da arte atual em graph embedding e sua influência na análise de redes complexas. O trabalho desenvolve um framework unificado para descrever e comparar várias técnicas de incorporação de grafos, destacando as forças e fraquezas de cada abordagem em fatores comuns.

Palavras-chave: Deep Learning, Graph Embedding, Graph Neural Networks, Matrix Factorization.

ABSTRACT

LEMOS, Willian Borges. **Bridging the Gap: A Systematic Framework for Benchmarking Graph Neural Networks**. 2023. 90 f. Thesis (Master) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

The exponential growth of data has presented both opportunities and challenges for organizations and individuals seeking to extract meaningful insights and patterns. In recent years, the field of complex network analysis has gained increasing attention as a solution to identifying patterns and structures in graphs. Graph embedding is a necessary step to bridge the gap between graph data and machine learning algorithms, allowing for the efficient processing and analysis of graph structures. However, the field of graph embedding remains fragmented, with a lack of a consensual theoretical understanding and a unified framework to describe and compare different techniques. This work aims to address these issues by providing a overview of the current state of the art in graph embedding and its influence on complex network analysis. The paper develops a unified framework for describing and comparing various graph embedding techniques, highlighting the strengths and weaknesses of each approach on common factors.

Keywords: Deep Learning, Graph Embedding, Graph Neural Networks, Matrix Factorization .

LIST OF FIGURES

Figure 1	Transformer Architecture used in OpenAi’s LLM model.	25
Figure 2	Basic operation of a GCN with neighborhood aggregation function and GNN layers. Adapted from Nathaniel [44].	28
Figure 3	ANEMONE Architecture has GCN (GNN) as the backbone of the application.	30
Figure 4	Architecture diagram.	35
Figure 5	Examples of charts automatically generated by the benchmark results.	39
Figure 6	Cora Dataset	44
Figure 7	t-SNE Visualization of GCN embedding for Cora dataset	47
Figure 8	t-SNE visualization of GraphSAGE embeddings for the dataset Cora	49
Figure 9	t-SNE visualization of GAT embeddings for the dataset Cora. . . .	51
Figure 10	Accuracy sensibility between models related to the number of layers.	52
Figure 11	Accuracy sensibility between models related to the number of neurons.	53
Figure 12	Accuracy sensibility between models related to the learning rate. . .	54
Figure 13	Accuracy sensibility between models related to the dropout.	55

LIST OF TABLES

Table 1	Notations and their explanations used	16
Table 2	Summary of Works	32
Table 3	Summary of Works	33
Table 4	Summary of GCN performance extracted from the main results table.	45
Table 5	Summary of GraphSAGE performance	48
Table 6	Summary of GAT performance	49
Table 7	Summary of Network Metrics for Original Datasets.	56
Table 8	Summary of Setup Parameters and Node Classification Accuracy. .	57
Table 9	Summary of metrics distortion	58
Table 10	Network Metrics Summary Table for Setups 7_b, 28_b, and 55_b. . .	59
Table 11	Comparison of Matrix Factorization (MF), Random Walk (RW), and Deep Learning without Random Walk (DL NO-RW) in key common characteristics	63

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DL	Deep Learning
GAN	Generative Adversarial Network
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
ML	Machine Learning
NN	Neural Network
ReLu	Rectified Linear Units

SUMMARY

1	Preface	12
2	Introduction	13
2.0.1	Formal definitions	14
2.1	Graphs	15
2.1.1	Directed and Undirected Graphs	17
2.1.2	Weighted and Unweighted Graphs	17
2.1.3	Static and Dynamic Graphs	17
2.2	Graph Embedding	18
2.3	Problem Definition and Objectives	19
3	Background and Related Work	21
3.1	Graph Embedding	21
3.1.1	Matrix Factorization	22
3.1.2	Deep Learning	23
3.2	Related Work	29
4	Systematic Benchmark and Analysis	34
4.0.1	Experiment Objectives	35
4.1	Systematic benchmark Tool	35
4.1.1	Automated Preparation	36
4.1.2	Parameterization and Hyperparameter Exploration	36
4.1.3	Results Compilation And Insightful Visualisations	38

5	Benchmark Experiment	41
5.1	Setups	41
5.2	Datasets	42
5.2.1	Classification Datasets: Cora, Citeseer, and Pubmed	43
5.2.2	Network Analysis Datasets	44
5.3	GCN	45
5.4	GraphSAGE	47
5.5	GAT	48
5.6	Overall Analysis	51
5.6.1	Number of Layers	51
5.6.2	Number of Neurons	52
5.6.3	Learning Rate	53
5.6.4	Dropout	54
5.7	Conclusions	55
5.8	Network Analysis	56
5.8.1	Network Metrics in Original Datasets	56
5.8.2	Experimental Setups	57
5.8.3	Steps to reproduce	60
5.9	Field Gaps and discussion	60
5.10	Discussion	61
5.10.1	Future directions	63
6	Conclusion	64
	References	65
7	Article I - GraphLED: A graph-based approach to process and visualise linked engineering documents	72
8	Article II - Exploring the Impact of Embedding Techniques on Graph Anomaly Detection: A Comparative Study	85

1 PREFACE

Except where otherwise stated in the text, this dissertation results from my work. No part of my dissertation has already been or is currently being submitted for a degree or diploma or any other qualification at any other university. Parts of the results presented in this dissertation have been published or are in a review process:

[Appendix A] - GraphLED: A graph-based approach to process and visualise linked engineering documents 7.

[Appendix B] - Exploring the Impact of Embedding Techniques on Graph Anomaly Detection: A Comparative Study 8.

Complete copies of these manuscripts have been included in this dissertation as appendices.

The links for the preliminary experiments can be found on:

<https://github.com/Willblemos/Bridging-the-Gap-A-Systematic-Framework-for-Benchmarking-Graph-Neural-Networks>

2 INTRODUCTION

Driven by the constant technological development Fang et al. [23], explosive growth in volume, velocity, and variety of data is at the core of modern science. The expansion of information has enriched human life; however, this growth poses challenges in obtaining helpful information with speed and accuracy Mao et al. [40], with an estimated 85% of information in companies stored in an unstructured form Blumberg and Atre [8]. With Big Data and the proliferation of advanced data processing techniques, various organisations are increasingly aware that they can process and analyse their large volumes of data to capture value for their businesses and employees George et al. [24], driving the transition from storing their documents in physical form to electronic form Matsubayashi et al. [41].

In recent work focused on the industrial sector, automated document digitisation has proven to be highly effective in the task of obtaining and structuring unstructured source data; however, with improvements in information storage capacity, there is a growing need to manage the quality of the stored information Kwon et al. [37]. In Madnick and Wang [39], four components of data quality are described: completeness, accuracy, suitability, and consistency.

Graph representation learning has been garnering increased attention within the data mining community, emerging as a promising area that has piqued the interest of researchers. It is recognized as an area of potential significance in data mining Akoglu et al. [5], holding promise for reducing losses incurred by institutions and individuals Adewumi and Akinyelu [1]. This growing interest stems from the fact that uncovering valuable patterns in data can lead to valuable insights across a diverse array of application domains Kumar [36] Aggarwal [2]. These domains encompass various fields, such

as fraud detection in credit cards Bhattacharyya et al. [7], Bolton et al. [9], assessing industrial damage Ahmed et al. [3], countering email and web spam Castillo et al. [12], identifying malware and spyware Invernizzi et al. [30], and enhancing surveillance systems Damjanovic et al. [16]. Furthermore, graph mining, rooted in pattern analysis, often unveils intricate individual or group behaviour patterns and their interconnections Sultana and Palaniappan [50].

One approach that has gained increasing attention in recent years is graph anomaly detection, also known as graph outlier detection Koutra and Faloutsos [34]. This is the task of identifying unusual or unexpected patterns in a graph, which may indicate anomalies such as fraud, cyberattacks, or system failures Chandola et al. [13]. The rise of graph anomaly detection is closely linked to the exponential growth of data and the increasing use of graphs to represent and analyze complex systems Fan and Han [22]. However, graphs have a complex structure that makes applying traditional machine learning algorithms for anomaly detection challenging. This is where graph embedding comes in as a necessary step to bridge the gap between graph data and machine learning algorithms.

Graph embedding aims to learn low-dimensional representations of graph structures, allowing for efficient processing and analysis Kipf and Welling [32]. Despite its importance, the field of graph embedding remains fragmented, with distinct subfields within the machine learning and data mining communities that may need a more explicit connection to one another Hamilton et al. [28]. Current approaches to graph embedding are frequently evaluated using benchmarks that emphasize different aspects of graph structures. As a result, it can be challenging to compare and understand the strengths and weaknesses of different approaches, hindering the development of a broader understanding. Consensual theoretical knowledge and a unified framework are needed to describe and compare the other techniques Cai et al. [11].

2.0.1 Formal definitions

This section introduces the fundamental concepts of graph embedding, followed by a formal definition of the problem of graph embedding.

- **graph:** A graph is a collection of vertices and edges that connect pairs of vertices. A graph G can be represented as $G = (V, E)$, where V is the set of vertices and

E is the set of edges.

- **Subgraph:** A subgraph of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$, where the edges in E' connect only vertices in V' .
- **Adjacency Matrix:** An adjacency matrix A is a square matrix of size $n \times n$, where n is the number of vertices in a graph. For an undirected graph, $A_{ij} = A_{ji} = 1$ if there exists an edge between vertices i and j , and $A_{ij} = A_{ji} = 0$ otherwise. For a directed graph, $A_{ij} = 1$ if an edge exists from vertex i to vertex j , and $A_{ij} = 0$ otherwise.
- **Directed graph:** A directed graph is a graph in which the edges have a direction. Each edge connects an ordered pair of vertices (u, v) , where u is the source vertex, and v is the destination vertex.
- **Undirected graph:** An undirected graph is a graph in which the edges do not have a direction. Each edge connects an unordered pair of vertices (u, v) .
- **Heterogeneous Network:** A heterogeneous network is a network that consists of multiple types of nodes and edges. Each node and edge type may have its attributes, and the nodes and edges may be connected differently.
- **Embedding Function:** An embedding function $f : V \rightarrow \mathbb{R}^d$ maps each vertex $v \in V$ to a d -dimensional vector $f(v)$. The embedding function can be learned using deep learning, matrix factorization, or random walks.
- **Embedded graph:** An embedded graph is a graph where each vertex is associated with a d -dimensional vector obtained by applying an embedding function. That is, an embedded graph $G' = (V', E')$ is obtained by applying the embedding function $f : V \rightarrow \mathbb{R}^d$ to the original graph $G = (V, E)$, where $V' = \{f(v) : v \in V\}$ and $E' = \{(f(u), f(v)) : (u, v) \in E\}$.

2.1 Graphs

Graphs are mathematical structures that model relationships between objects. In a graph, the objects are represented as vertices (or nodes), and the connections are represented as edges that connect pairs of vertices. Graphs can model real-world phenomena,

Notation	Explanation
$G = (V, E)$	A graph G with set of nodes V and set of edges E
$G' = (V', E')$	A subgraph of G with set of nodes $V' \subseteq V$ and set of edges $E' \subseteq E$
A	The adjacency matrix of a graph G
$\{j \rightarrow i\}$	The set of neighboring nodes j pointed to node i
$W^{(l)}$	The weight matrix
$h_i^{(l)}$	The attribute vector for a node i
n	The number of nodes in G
Z	The predicted class
σ	The activation function

Table 1: Notations and their explanations used

such as social networks, transportation networks, and molecular structures. The study of graphs, also known as graph theory, has a rich history that dates back to the 18th century, with the famous problem of the Seven Bridges of Königsberg posed by Euler [21]. The problem asks whether it is possible to walk through the city of Königsberg, crossing each of its seven bridges once and only once. Euler solved the problem by representing the city as a graph and showing that the situation was impossible to solve. In social network analysis, graphs have been used to predict social ties, detect communities, and classify users based on their interests Wang and Tang [56]. In bioinformatics, graphs have been used to predict protein functions, classify genes, and analyze biological networks Zitnik et al. [63]. In recommender systems, graphs have been used to recommend items to users based on their interactions with the system Ying et al. [59]. A graph G can be defined as a pair $G = (V, E)$, where V is the set of nodes or vertices, and E is the set of edges that connect them. An edge e is defined as a pair of nodes (u, v) , where $u, v \in V$. In a graph, the nodes represent objects, and the edges represent relationships or connections between those objects. One important concept in graph theory is the degree of a node. The degree d_v of a node v is defined as the number of connected edges. A node with degree 0 is called an isolated node, while a node with a high degree is called a hub node, as it connects multiple edges in the graph. Another fundamental concept in graph theory is the notion of paths and cycles. A path in a graph is a sequence of edges that connect a sequence of distinct nodes. Conversely, a cycle is a path that starts and ends at the same node. Paths and cycles are essential in many applications of graph theory, including network routing and cycle detection in

graphs. Graphs can also be classified based on their properties. For example, graphs can be directed or undirected, weighted or unweighted, and static or dynamic.

2.1.1 Directed and Undirected Graphs

Directed graphs, also known as digraphs, are graphs where the edges have a direction associated with them. In a directed graph, the edges are represented by ordered pairs of nodes, (u, v) , where u is the source node and v is the destination node. The concept of in-degree and out-degree of a node is introduced in directed graphs. In-degree is the number of edges pointing towards a node, whereas out-degree is the number of edges originating from a node. In an undirected graph, the edges do not have a direction and can be traversed in both directions. For example, a social network can be represented as an undirected graph, where the edges connect users who are friends. On the other hand, a web page can be described as a directed graph, where the edges point from pages that link to other pages.

2.1.2 Weighted and Unweighted Graphs

Graphs can also be weighted or unweighted, depending on whether each edge has a weight associated with it. In a weighted graph, the weights can represent various attributes, such as the distance between nodes, the strength of a relationship, or the similarity between nodes. In contrast, an unweighted graph has no values associated with its edges. For example, a transportation network can be represented as a weighted graph, where the weights represent the distance between the nodes. By representing the transportation network as a weighted graph, where the weights of the edges represent the distances between the nodes, we can use graph algorithms to find the shortest path between two locations. On the other hand, a social network can be represented as an unweighted graph, where the edges represent relationships between users.

2.1.3 Static and Dynamic Graphs

In a static graph, the nodes and edges do not change over time. In a dynamic graph, the nodes and edges can change over time. A static graph is formally defined as a graph whose topology remains fixed over time, i.e., the set of nodes and edges does not change. On the other hand, a dynamic graph can change over time, meaning that the set of nodes and edges may change. In a dynamic graph, the graph's topology can

change as nodes and edges are added or removed. A dynamic graph can be represented as $G = (V, E, t)$, where t represents time, and the graph may change over time. For example, social networks are dynamic graphs because new users can join, and existing users can form new connections or disconnect.

2.2 Graph Embedding

Graph embedding is a technique to learn vector representations of nodes or graphs in a given graph dataset. The goal of graph embedding is to map each node or graph in the dataset to a lower-dimensional vector space such that the structural information of the graph is preserved. Graph embedding has become increasingly popular in recent years due to its ability to improve performance on graph-based tasks such as node classification, link prediction, and community detection.

The field of graph embedding has experienced significant growth in recent years, with the development of new techniques and the exploration of new applications, for instance, in social network analysis, graph embedding has been used to predict social ties, detect communities, and classify users based on their interests Wang and Tang [56]. In bioinformatics, graph embedding has been used to predict protein functions, classify genes, and analyse biological networks Zitnik et al. [63]. Graph embedding has been used in recommender systems to recommend items to users based on their interactions with the system Ying et al. [59]. Graph embedding is closely related to representation learning, concerned with learning low-dimensional vector representations of high-dimensional objects, such as images, text, and graphs. In particular, graph embedding can be seen as a special case of representation learning, where the input data is a graph.

Formally, given a graph $G = (V, E)$ with n nodes and m edges, the goal of graph embedding is to learn a function $f : V \rightarrow \mathbb{R}^d$ that maps each node v_i to a d -dimensional vector x_i , such that the pairwise similarities between the vectors x_i and x_j preserve some of the structural properties of the graph. The pairwise similarities can be measured using different metrics, such as cosine similarity, dot product, or Euclidean distance.

One way to learn graph embeddings is through deep learning techniques, such as neural networks, which have shown great success in various tasks, including image and speech recognition, natural language processing, and graph analysis. In graph embedding, deep learning models can be used to learn representations of nodes by processing local and global information about the graph structure and additional features associated with each node, such as node attributes and labels. Another popular approach to graph embedding is based on random walks, which is used by the DeepWalk algorithm Perozzi et al. [47] and its variants. The idea behind random walk-based methods is to generate node sequences by performing random walks on the graph and then using these sequences to train a skip-gram model Mikolov et al. [42] that learns node embeddings based on the co-occurrence of nodes in sequence.

2.3 Problem Definition and Objectives

Graph embedding techniques are widely used in machine learning and data mining, with applications ranging from social network analysis to anomaly detection. However, despite their popularity, there currently needs to be a widely accepted standard for evaluating and comparing the performance of different graph embedding techniques that respect these methods' full applicability. A common evaluation method is necessary for researchers and practitioners to determine which strategies are most effective for different types of graph data or to compare results across various studies in a way that fits into the big picture of the field.

The problem scope addressed in this work is evaluating graph embedding quality, network distortions, and their impact on graph classification tasks. The objective is to develop a framework for assessing and comparing techniques based on standard evaluation metrics and benchmarks. This framework will accommodate various graph data types and application domains. It will allow researchers and practitioners to assess the performance of different techniques consistently and transparently. By providing a standard evaluation method for graph embedding that respects the full spectrum of applicability and provides a comparison, this work aims to facilitate deeper understanding and further progress in this crucial area of machine learning and data mining. Among the specific objectives, we can highlight:

- Identify the universal characteristics related to the applicability of graph embed-

ding models.

- Develop a fair way to compare different graph embedding models.
- Develop a benchmark setup that is appropriate for different types of graphs and tasks.
- Develop a framework to evaluate and compare graph embedding models.
- Include a set of best practices for evaluating different models and tools and resources for implementing the benchmarks and evaluating the models.

3 BACKGROUND AND RELATED WORK

This section provides an overview of key concepts in the application domain that serve as the guiding line for this work.

3.1 Graph Embedding

Recently, graph embedding has emerged as a promising approach for data mining on graphs. Graph embedding is a technique that maps the nodes and edges of a graph to a low-dimensional vector space, preserving the structural and semantic information of the graph. By leveraging the properties of the vector space, graph embedding can be used to perform various tasks, such as graph classification, link prediction, and anomaly detection. These techniques can be broadly classified into two categories: Shallow and Deep. Shallow graph embedding techniques are based on matrix factorisation and graph algorithms such as Random walks. These methods are efficient in small-medium size graphs but may not capture the complex relationships and structure of graphs well. Examples of shallow graph embedding techniques include Laplacian Eigenmaps Belkin and Niyogi [6], Locally Linear Embedding (LLE) Roweis and Saul [49], and Graph Factorization Ahmed and Mahoney [4]. In contrast, deep graph embedding techniques leverage the power of deep learning and neural networks to capture the complex relationships and structure of graphs accurately. Although they require more computational resources, they provide higher accuracy than shallow embedding techniques. Examples of deep graph embedding methods include Graph Convolutional Networks (GCN) Kipf and Welling [32], Graph Attention Networks (GAT) Veličković et al. [53], and Graph Autoencoders Kipf and Welling [33].

3.1.1 Matrix Factorization

Matrix factorisation has become popular for extracting meaningful information from graph datasets. For example, graph Laplacian Eigenmaps (GLE) is a method introduced by Belkin and Niyogi [6] that maps nodes in a graph to a low-dimensional representation using the eigenvectors of the graph Laplacian matrix. The Laplacian matrix encodes the global structure of the graph, and by computing its eigenvectors, GLE can preserve the global structure of the graph in the low-dimensional representation. However, the computational complexity of GLE grows at least proportionally to the square of the number of nodes, making it unsuitable for processing large-scale networks efficiently.

Laplacian Eigenmaps aim to keep the embedding of two nodes close when the weight W_{ij} is high. Specifically, they minimise the following objective function:

$$\varphi(Y) = \frac{1}{2} \sum_{i,j} |Y_i - Y_j|^2 W_{ij} = \text{tr}(Y^T L Y), \quad (1)$$

where Y_i and Y_j are the low-dimensional embeddings of nodes i and j , respectively. The weight W_{ij} measures the similarity between nodes i and j . The Laplacian matrix L of graph G is a graph representation that encodes the global structure of the graph. The objective function is subject to the constraint that $Y^T D Y = I$ to eliminate trivial solutions, where D is the diagonal matrix of node degrees. To obtain the low-dimensional embeddings, we can compute the eigenvectors of the normalised Laplacian $L_{\text{norm}} = D^{-1/2} L D^{-1/2}$ and take the d eigenvectors corresponding to the d smallest eigenvalues. Here, d is the desired dimensionality of the embedding. The eigenvectors are then used to construct the embedding matrix Y .

Node Proximity Matrix Factorization (NPMF) is a popular technique for obtaining node embeddings from the node proximity matrix of a graph, introduced by Cai et al. [10]. The node proximity matrix captures the similarity between nodes in the graph, and NPMF aims to find a low-dimensional representation of the nodes that preserves this similarity. Unlike GLE, NPMF models can capture the graph's local and global structure, making it a versatile approach for various graph types. Compared to Graph Laplacian Eigenmaps, NPMF is more flexible and can handle multiple graph types efficiently.

3.1.2 Deep Learning

With the rise of deep learning, graph embedding has been an active area of research, and numerous models have been proposed to perform this task. These models can be broadly categorised into two groups: random walk methods and non-random walk methods.

3.1.2.1 *Deep learning with random walk*

The basic concept of random walk graph embedding is to capture the connectivity patterns of a graph by generating node sequences using random walks and then using these sequences to compute node embeddings that reflect the graph's structural information. DeepWalk Perozzi et al. [47], Node2vec Grover and Leskovec [26], and LINE Tang et al. [52] are popular examples of these graph embedding techniques, known as Deep Learning graph embedding with Random walk. DeepWalk and Node2vec employ a decoder based on the inner product, similar to matrix factorisation approaches. DeepWalk considers random walks as sentences and learns node representations through a skip-gram model, while Node2vec has a more flexible tradeoff between local and global structure information by controlling random walks with a parameter that determines whether to prioritise breadth-first or depth-first sampling. LINE, in particular, extends upon the previous models by more explicitly incorporating local and global structure information through factorising first- and second-order similarities. Nevertheless, it still uses random walks as the foundation for generating node sequences and learning node embeddings, making it a natural extension of DeepWalk and node2vec. However, these methods are inherently transductive, meaning they can only generate embeddings for nodes present during the training phase. They cannot be used to generate embeddings for new nodes, making them less suitable for dynamic graphs or use cases where new nodes are frequently added.

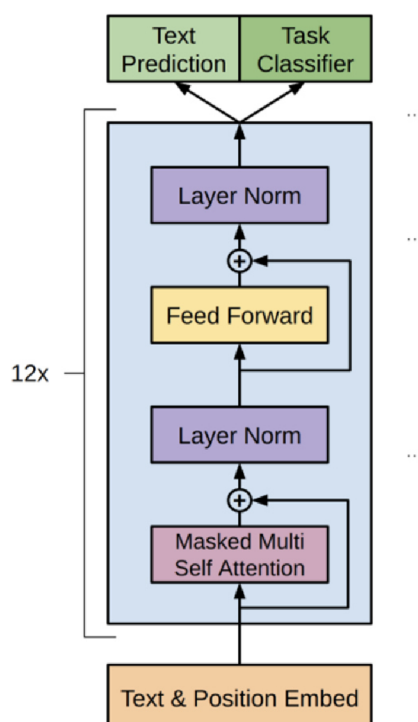
3.1.2.2 *Word Embedding*

Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences. From computer vision to natural language processing (NLP), the idea of selectively focusing on specific ele-

ments of data has led to significant advancements. Following this notion of attention mechanisms, the field of word embedding has witnessed substantial progress. Word embedding techniques, integral to understanding semantic relationships in textual data, have benefited from the concept of attention. The ability to capture intricate contextual nuances and semantic meanings has been greatly enhanced. Recent developments, including models like BERT Devlin et al. [18] and the foundational work of Mikolov et al. [42], have harnessed attention mechanisms to refine the representation of words and sentences, amplifying the effectiveness of word embedding. Moreover, attention mechanisms have transcended word embedding and made a profound impact on Large Language Models (LLMs). LLMs, exemplified by OpenAI's latest model, GPT-4, have redefined the landscape of natural language understanding. These models, capable of generalizing knowledge from extensive training data, are equipped with sophisticated attention-based architectures. GPT-4, for instance, leverages attention mechanisms and the transformer architecture to excel in natural language understanding tasks.

The architectural flow of these models, as shown in figure 1, can be delineated into the following phases: In the initial phase, input text is transformed into numerical vectors. Additional position embeddings are incorporated to encode the sequential order of words, enabling the model to discern the context and ordering of words, a critical aspect of language processing. Then, a Masked Multi-Self-Attention phase empowers the model to assign varying levels of importance to individual words in the input, adapting dynamically to contextual relevance. Following self-attention, layer normalization is applied. This step enhances the stability and efficiency of the training process by ensuring consistent statistical properties across layers, thereby facilitating the smooth flow of gradients during training. After attention mechanisms, the data undergoes further processing through a feed-forward neural network. Comprising fully connected layers and non-linear activation functions, this network enables the model to capture intricate relationships between words and extract higher-level features. Subsequent to the feed-forward network, another layer normalization step is employed. This additional normalization contributes to the model's training stability and convergence. In the final phase, the model generates text predictions or conducts text classification, contingent upon the specific task at hand.

Figure 1: Transformer Architecture used in OpenAi's LLM model.



Source: Adapted from Radford et al. [48]

3.1.2.3 Deep learning without random walk

Autoencoders and Graph Convolutional Networks (GCNs) are popular non-random walk methods for graph embedding. Both of these methods aim to capture the structural information of a graph, but they differ in their approach. An autoencoder consists of two components: an encoder that maps the input graph data to a lower-dimensional representation and a decoder that maps the lower-dimensional representation back to the original data. The encoder and decoder are trained to reconstruct the original data as accurately as possible, minimising the reconstruction error of the output. Adopting an autoencoder for graph embedding is similar to node proximity matrix factorisation in neighbourhood preservation, where the reconstruction process will make the nodes with similar neighbourhoods with similar embedding. The rise of autoencoders in graph embedding came from their ability to handle non-linear relationships in the graph data, learn hierarchical representations of graph data, and handle variable-sized input graphs.

Additionally, autoencoders can embed both node and graph-level information in the same representation, which is useful for tasks requiring information from both levels of abstraction. For example, the work Kipf and Welling [33] used a variational autoencoder for graph representation learning and achieved promising results on a range of graph datasets. Pan et al. [46] also used autoencoders for graph embedding, achieving state-of-the-art performance on a variety of graph classification tasks. However, Autoencoders also can be extremely costly to train, and they are also strictly transductive.

Graph Convolutional Networks were introduced in Kipf and Welling [32] to extend convolutional neural networks to graph-structured data. GCNs effectively capture local and global structures in graphs since they can encode structure information of the graphs by aggregating information from their local neighbourhood. GCNs rely on parameter sharing and neighbourhood aggregation, incorporating the graph structure directly into the encoder and leveraging node attributes. This allows GCNs to generate embeddings for absent nodes during the training phase. The work by Kipf and Welling [32] showed that GCNs achieved state-of-the-art performance on several benchmark graph classification tasks. On the other hand, GCNs can be limited in their ability to learn complex non-linear relationships between nodes. This is where work such as the "Deep Graph Infomax" by Veličković et al. [54] can prove helpful, as they extend the basic GCN framework to allow for more expressive representations through deep architectures and unsupervised objectives.

Formally, a GCN takes as input a graph $G = (X, A)$, where X is a feature matrix that describes the attributes of the nodes and A is the adjacency matrix that describes the relationships between the nodes. At each layer l of the GCN, the feature vectors $h_i^{(l)}$ of the nodes i are transformed by a function f , which takes into account the feature vectors of their neighbouring nodes $h_j^{(l)} : j \rightarrow i$ according to the graph structure. This function can be written as:

$$h_i^{(l+1)} = f(h_i^{(l)}, h_j^{(l)} : j \rightarrow i) \quad (2)$$

where $j \rightarrow i$ denotes the set of neighbouring nodes j pointed to node i . The function f can be learned by optimising a loss function that measures the difference between the predicted and true labels, described as:

$$\text{loss} = \sum_{i=1}^p (t_i - o_i) \quad (3)$$

One way to define the function f is to perform spectral graph convolutions, which are defined as the multiplication of a signal $x \in \mathbb{R}^N$ with a filter $g_\theta = \text{diag}(\theta)$ parameterised by $\theta \in \mathbb{R}^N$ in the Fourier domain, i.e.:

$$g_\theta * x = U g_\theta U^T x \quad (4)$$

where U is the normalised eigenvector matrix of the Laplacian matrix A . This approach can be generalised to a GCN by defining a layer as follows:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}) \quad (5)$$

where $\tilde{D}$ is the diagonal degree matrix of $\tilde{A}$, defined as $\tilde{D}ii = \sum_j \tilde{A}_{ij}$ and $\tilde{A}$ is $\tilde{A} = A + I_n$. The weight matrix $W^{(l)}$ is learnable, and σ is a non-linear activation function, usually ReLU. we can simplify this function by separating the pre-processing step $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, rewriting the GCN layer function as:

$$H^{(l+1)} = \sigma(\hat{A} H^{(l)} W^{(l)}) \quad (6)$$

For prediction, GCN uses a softmax layer combined with a matrix of filtered parameters Θ , which will predict n nodes of the class Z in l layers, described as:

$$Z = \text{softmax}(\hat{A} H^{(l-1)} \Theta^{(l)}) \quad (7)$$

As depicted in Figure 2, the illustration outlines the basic operation of a Graph Convolutional Network (GCN), showcasing the neighborhood aggregation function and the Graph Neural Network (GNN) layers.

Posteriorly Hamilton et al. [27] presented GraphSAGE, a framework for scalable graph representation learning that can generate embeddings for nodes that have not been previously seen in the graph. This is achieved by sampling and aggregating feature information from a node's local neighbourhood, which is then used to compute a low-dimensional embedding for that node. GraphSAGE builds upon the GCN model by incorporating the node features of each node in addition to its previous layer information

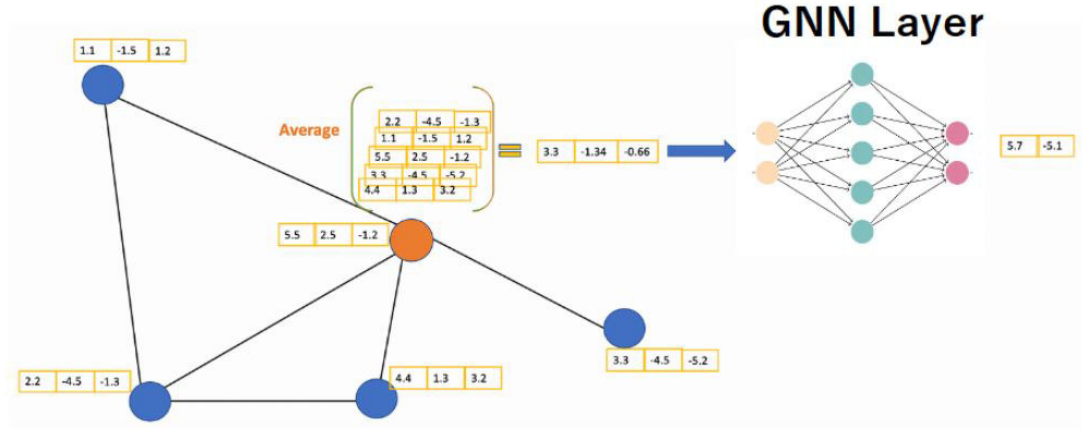


Figure 2: Basic operation of a GCN with neighborhood aggregation function and GNN layers. Adapted from Nathaniel [44].

to update its representation. The update equation used is given by:

$$\hat{h}_i^{(l+1)} = \text{ReLU} \left(U^{(l)} \cdot \text{Concat} \left(h_i^{(l)}, \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} h_j^{(l)} \right) \right), \quad (8)$$

where $U^{(l)} \in \mathbb{R}^{d \times 2d}$ is a weight matrix that maps the concatenated input features to a lower-dimensional space, and $\mathcal{N}(i)$ represents the set of neighbouring nodes of node i . The operation $\text{Concat}(h_i^{(l)}, \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} h_j^{(l)})$ concatenates the feature vector of the central node $h_i^{(l)}$ with the average of its neighboring nodes' feature vectors $\frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} h_j^{(l)}$. The central node features $h_i^{(l)}$ and the neighborhood features $h_j^{(l)}$ undergo different transformations. The ReLU function applies an element-wise rectified linear unit activation function to the output of the matrix multiplication. The node features are then projected onto the l_2 -unit ball to ensure that their magnitudes do not grow too large and to facilitate training stability, which is given by:

$$h_i^{(l+1)} = \frac{\hat{h}_i^{(l+1)}}{|\hat{h}_i^{(l+1)}|_2}. \quad (9)$$

Graph Attention Networks (GATs) are another recent development in graph embed-

ding. GATs were introduced in Velivckovic et al. [55] to learn node embeddings that consider each neighbour node's importance. GATs use a self-attention mechanism to learn a weight for each neighbour node based on its relevance to the current node. This allows the model to learn different neighbourhood structures for other nodes and better capture the heterogeneity of graph data. GATs have achieved state-of-the-art performance on various graph-related tasks, including node classification, anomaly detection, link prediction, and graph classification.

Formally, the node update equation in GAT is given by:

$$h_i^{(l+1)} = \text{Concat}_{k=1}^K \left(\text{ELU} \left(\sum_{j \in \mathcal{N}_i} e_{ij}^{(k,l)} U^{(k,l)} h_j^{(l)} \right) \right), \quad (10)$$

where $h_i^{(l)}$ represents the node representation of node i at layer l , K is the number of attention heads, and $U^{k,l} \in \mathbb{R}^{\frac{d}{K} \times d}$ are the projection matrices for each attention head. The $e_{ij}^{k,l}$ are the attention coefficients for each head, which determine the importance of each neighbour node j for updating the representation of node i .

The attention coefficients $e_{ij}^{k,l}$ are computed as follows:

$$e_{i,j}^{k,l} = \frac{\exp(\hat{e}_{i,j}^{k,l})}{\sum_{j' \in \mathcal{N}_i} \exp(\hat{e}_{i,j'}^{k,l})} \quad (11)$$

where $\hat{e}_{i,j}^{k,l}$ is the attention score computed as the output of a LeakyReLU activation function applied to the concatenation of the projection of node features $h_i^{(l)}$ and $h_j^{(l)}$ with trainable parameter vector $\mathbf{V}^{k,l}$:

$$\hat{e}_{i,j}^{k,l} = \text{LeakyReLU} \left(\mathbf{V}^{k,l} \text{Concat}(\mathbf{U}^{k,l} h_i^{(l)}, \mathbf{U}^{k,l} h_j^{(l)}) \right) \quad (12)$$

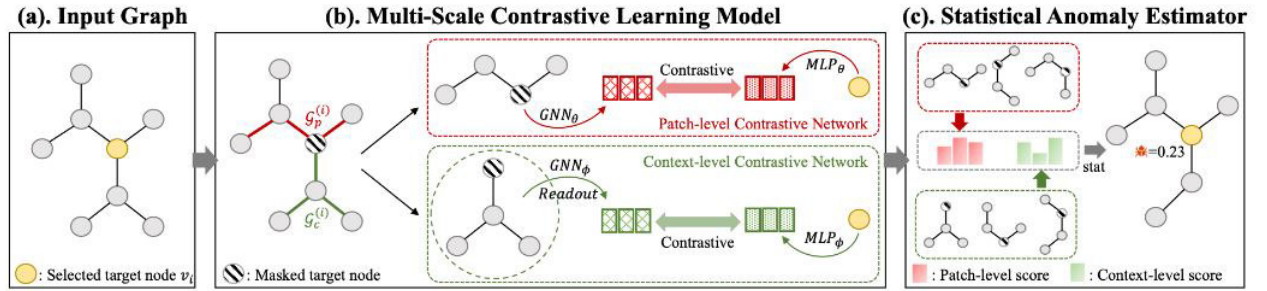
In summary, GAT learns to update each node's representation by aggregating information from its neighbours in a way tailored to each attention head. This allows GAT to capture complex patterns in the graph topology and learn meaningful node representations.

3.2 Related Work

Graph mining techniques have made significant progress recently, relying on graph embedding models such as Graph Neural Networks (GNNs). Traditional machine lear-

ning methods are unsuitable for graph data, necessitating the embedding of these structures. Several state-of-the-art models, such as Jin et al. [31], Liu et al. [38], and Ding et al. [19], use a GNN model as a core component, utilizing GCN/Kipf and Welling [32] layers to extract low-dimensional embeddings, which are then combined with graph learning methods like contrastive learning. An example of such architecture is depicted in Figure 3. However, despite being a vital component of these models, there needs to be more precise reasoning on how these embedding layers are chosen beyond their popularity and flexibility or their impact on the overall process. This is because straightforward methods have yet to be established to measure the effect of the graph embedding process in classification tasks. Consequently, a meaningful way to measure and compare different embedding techniques could significantly benefit the field, allowing researchers to select methods better suited to their specific applications.

Figure 3: ANEMONE Architecture has GCN (GNN) as the backbone of the application.



Source: ANEMONE: Graph Anomaly Detection with Multi-Scale Contrastive Learning Jin et al. [31]

Recent studies have focused on evaluating the quality of graph embeddings and their impact on graph mining. One way to assess the quality of graph embeddings is through intrinsic evaluation methods, which measure the ability of the embedding to preserve the structural information of the graph. Some work, such as Sun et al. [51], discuss the impact of the quality of graph embeddings in their model's performance when limited embeddings lead to false positive or false negative results in node classification tasks. However, it needs a consensus on theoretical foundations, and the studies are disconnected from one another, with models being evaluated using various benchmarks that emphasize different aspects. Cai et al. [11], and Hamilton et al. [28] provide a valuable overview of graph embedding methods, but they may only comprehensively evaluate some existing techniques. Additionally, there needs to be established common

ground for comparison, and they do not necessarily explore the tradeoffs between different embedding methods, such as their characteristics and applicability to other use cases. Nonetheless, both works serve as essential references for graph embedding.

The work of Xu et al. [58] proposes a theoretical framework for neural graph networks based on multiset aggregation, which involves combining the feature vectors of a node’s neighbours into a multiset. However, the model’s expressiveness is constrained by its fixed feature construction scheme, which does not adapt to the data distribution. Moreover, the framework is limited to measuring and comparing the expressive power of GNNs based on the concept of multisets.

Chen et al. [14] offers a comprehensive historical overview of network embedding algorithms, outlining key concepts and categorizing their main characteristics. However, the paper does not assess or compare the efficacy of the methods concerning the factors mentioned above.

Dwivedi et al. [20] propose a benchmark platform for graph neural networks (GNNs) that tests several methods’ performance. The paper provides an insightful review of the main techniques, concepts, and definitions. However, although the benchmark platform offers helpful guidance for comparing model scalability, it does not address the embedding behaviour of these methods concerning the graph structure or their practical applicability.

Wei and Zhao [57] offer a valuable timeline perspective of graph learning techniques and their growth over time. Their paper presents an intuitive taxonomy based on graph inputs and elements (i.e., node, edge, and graph structure). However, the work does not explicitly link graph features to model characteristics or address the advantages and issues of the different models. The tables 2 and 3 summarizes the key works in the field.

Work	Year	Research Aim	Open Gap
Hamilton et al. [28]	2017	Representation Learning Review	• Cover techniques for node and subgraphs representation learning. • Does not provide a systematic comparison for methods based on their common characteristics. • Does not include any benchmark experiments or frameworks for evaluating the effectiveness of these methods.
Cai et al. [11]	2018	Graph Embedding Classification	• Proposes two taxonomies of graph embedding methods. • Focus on the problem settings of graph inputs and outputs. • Does not include any benchmark experiments or frameworks for evaluating the effectiveness of these methods.
Xu et al. [58]	2018	Representation Learning	• Covers techniques for node and subgraphs representation learning. • Does not provide a systematic comparison for methods based on their common characteristics. • Does not include any benchmark experiments or frameworks for evaluating the effectiveness of these methods.
Cai et al. [11]	2018	Graph Embedding Classification	• Proposes a theoretical framework for graph neural networks based on the concept of multiset aggregation. • The model's expressiveness is constrained by its fixed feature construction scheme. • The framework is limited to measuring and comparing the expressive power of GNNs solely based on the concept of multisets.

Table 2: Summary of Works

Work	Year	Research Aim	Open Gap
Chen et al. [14]	2018	Network Embedding	• Provides historical overview, important concepts and categorizes key characteristics of network embedding algorithms • Lacks evaluation or comparison of works and methods in key factors mentioned. • Does not include any benchmark experiments or frameworks for evaluating the effectiveness of these methods.
Dwivedi et al. [20]	2020	Benchmarking GNN's	• Proposes benchmark platform to test performance, generability, and scalability of GNN methods. • Does not address embedding behavior concerning the graph structure or practical applicability.
Wei and Zhao [57]	2022	Graph Learning	• Provides timeline perspective of graph learning techniques and a taxonomy based on graph elements. • Lacks a direct link between graph features and model characteristics. • Does not address the advantages and drawbacks of different models.

Table 3: Summary of Works

4 SYSTEMATIC BENCHMARK AND ANALYSIS

In recent years, graph embedding has emerged as a powerful technique for addressing multiple tasks involving connected networks. Graph Convolutional Networks (GCN), GraphSAGE, and Graph Attention Networks (GAT) are three popular methods for graph embedding. However, comparing the performance of these methods can be challenging due to variations in implementation and experimental conditions. The state-of-the-art models for graph embedding exhibit specificity and lack of uniformity across implementations. Compatibility issues, diverse data formats, and transformation requirements further complicate comparing and evaluating these models. Models may expect different input data formats, such as matrices, graphs, tensors, or other representations, creating a barrier to direct comparisons. To address these challenges, a dedicated platform has been developed that enables the evaluation of these methods under consistent conditions. This platform facilitates importing and adapting various models and datasets, streamlining transformations necessary for applying diverse datasets to multiple models. This generic and parameterised benchmark platform promotes the sharing of parameters and experimental setups, enabling the training and testing of various models across different datasets, layer configurations, neuron counts, dropout rates, optimisers, and epochs. For the conducted experiments, the Google Colab Pro platform was utilized, featuring a machine equipped with 2x Intel(R) Xeon(R) CPU @ 2.20GHz and 12GB of RAM. All the mentioned code and results are available at the corresponding `GitHubRepositoryLink`, along with an easily executable version on Google Colab.

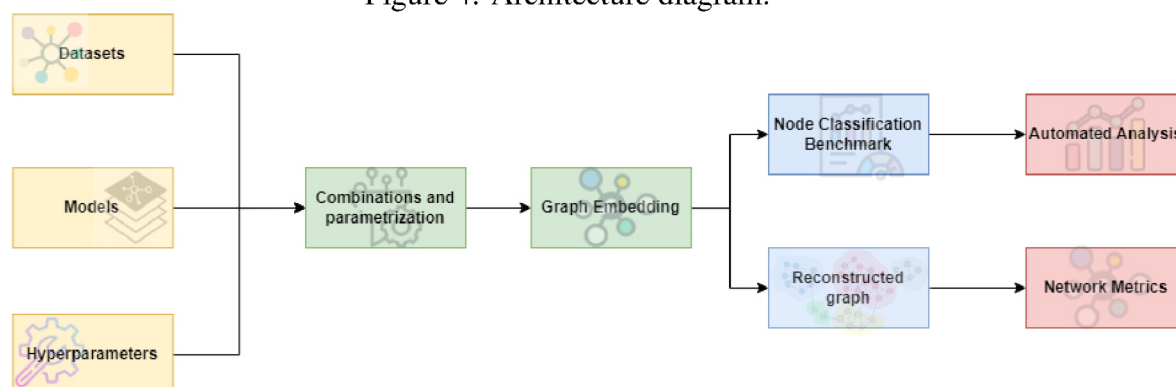
4.0.1 Experiment Objectives

The primary goal of the experiment is to compare the embedding representations generated by each method against the ground truth node labels. In this context, effective embeddings should position nodes with similar features close together in the embedding space while segregating nodes with different labels. Node classification is conducted based on their embeddings in 81 distinct setups. Additionally, the last embedding layer is employed for reconstructing the graph, applying network metrics, and visualising node embeddings using the t-distributed Stochastic Neighbor Embedding (t-SNE) technique. The comparison involves evaluating the models for accuracy, parameter sensitivity, datasets, visualisation, and network structure.

4.1 Systematic benchmark Tool

A dedicated framework has been developed to streamline this comparative evaluation of GNN models. This framework offers tools for benchmarking and analysing GNN models. It ensures that different models can be incorporated seamlessly, datasets can be prepared for usage across various models and provides methods for training, testing, analysis, and comparison in an automated and parameterised manner. Key features of the framework include automated preparation, parameterisation and hyperparameter exploration, model generation, training and evaluation, model agnostic, results compilation and insightful visualisations. Figure 4 illustrates the architecture diagram of the framework.

Figure 4: Architecture diagram.



Source: author

4.1.1 Automated Preparation

The benchmark framework commences with an essential step: automated dataset preparation. This phase involves transforming and formatting selected datasets – Cora, Citeseer, and Pubmed – to ensure their compatibility with subsequent model training and evaluation. During the transformation process, raw dataset information is distilled. The datasets’ raw formats are often intricate and vary across sources. This includes features, labels, edges, and additional graph-related attributes. A critical transformation is converting data into a graph-based format, specifically an adjacency matrix representation, to standardise these datasets and make them suitable for the chosen models.

The datasets are transformed into a format representing the graph’s structure and connectivity. The nodes’ features and labels are also aligned to their respective positions in the graph. This transformation facilitates seamless integration with the chosen model architectures operating on graph-based data structures. The framework imports the datasets as Graph Data objects using the PyTorch Geometric library to accommodate the various characteristics of different datasets. This format provides a uniform graph data representation, making applying the same models and procedures easier across other datasets.

The benchmark ensures that the subsequent modelling and evaluation processes are consistent and adaptable by standardising the datasets into a graph-based format. This approach also leverages the strengths of graph neural networks designed to operate on graph-structured data. Using graph-based data structures enables effective capture of node interactions and connectivity patterns, which is crucial for accurate graph classification tasks.

4.1.2 Parameterization and Hyperparameter Exploration

The benchmarking framework employs a parameterisation and hyperparameter exploration strategy to assess model performance across various configurations systematically. This systematic approach not only aids in identifying optimal model architectures but also provides insights into the sensitivity of model performance to different hyperparameter values. In the spirit of this flexibility, the framework introduces parameterisation for several key hyperparameters, including:

- **Number of Layers:** A list of potential values, such as (1, 2, 3), allowing users to

explore different model depths.

- **Number of Neurons per Layer:** Users can provide a list of neuron counts, such as (16, 32, 64), offering the freedom to evaluate various levels of model complexity.
- **Learning Rate:** A set of learning rates, like (0.01, 0.001, 0.0001), for probing the model’s sensitivity to this critical training parameter.
- **Dropout Rate:** A range of dropout values, like (0.5, 0.2, 0.1), facilitating examination of regularisation effects.

For each hyperparameter, the benchmark considers all possible combinations of values from the provided lists. This approach results in a set of setups, each characterised by a distinct combination of hyperparameter values. Then, for each combination, the following steps are executed:

4.1.2.1 Model Creation

The benchmark instantiates model architectures based on the hyperparameters provided. Three model architectures – Graph Convolutional Network (GCN), GraphSAGE, and Graph Attention Network (GAT) – are employed in this benchmark. These models are implemented as separate functions that create instances of the corresponding model class from the PyTorch Geometric library. The chosen architectures are renowned for their effectiveness in graph-based classification tasks.

4.1.2.2 Training and Evaluation

The model is trained and evaluated using the provided dataset. The dataset is divided into training and testing subsets. The model learns to predict node labels during training through gradient descent optimisation. The training and testing losses, as well as accuracies, are recorded throughout the training process. The exploration of hyperparameters generates many results, revealing trends and patterns in the model’s behaviour across different settings. This iterative process enables the identification of hyperparameter combinations that yield the best performance on the given dataset.

4.1.2.3 *Model Agnostic*

Notably, the framework is designed to be model-agnostic, facilitating the inclusion of additional models by implementing their corresponding creation functions without significant modifications to its core structure. This modularity enhances the benchmark's flexibility and adaptability, extending the experimentation to other graph-based models compatible with the dataset. Using multiple datasets, models, and hyperparameters ensures a robust evaluation of the chosen models' performance across various scenarios.

4.1.3 **Results Compilation And Insightful Visualisations**

After conducting the exhaustive hyperparameter exploration and training process, the benchmark framework compiles the results from all setups, models, and datasets into a dataframe. The aggregated results and accompanying analyses are saved to a CSV file for further examination and sharing. This approach not only enhances the accessibility of results but also promotes collaborative research and knowledge sharing within the graph-based machine learning community. This consolidated representation fits for analysing the performance of various graph-based models under diverse conditions, with a scalable approach that can isolate and export different process steps. Let's delve into the critical components of this results compilation and analysis phase, as illustrated in Figure 5.

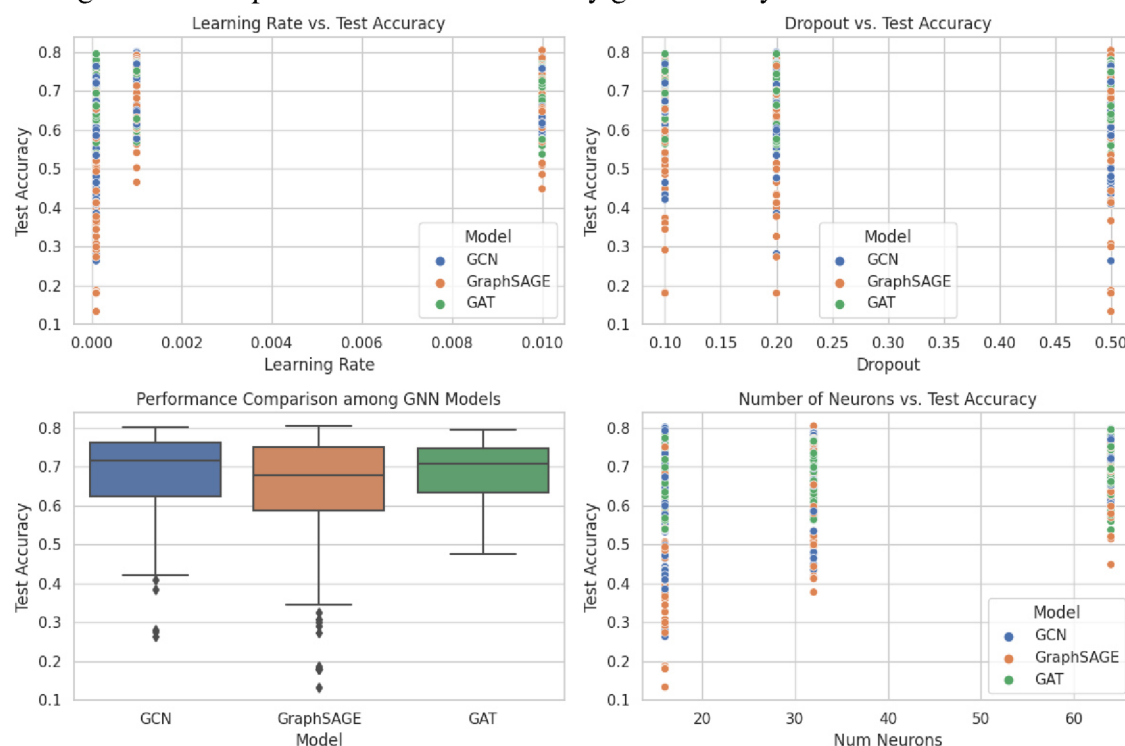
4.1.3.1 *Hyperparameter Sensitivity Analysis*

The benchmark employs a sensitivity analysis to elucidate the intricate relationship between hyperparameters and model performance. This analysis systematically investigates how varying individual hyperparameters impact the mean test accuracy of different models. The benchmark generates line plots for each hyperparameter, showcasing how the model's performance changes across different values of that specific hyperparameter. This approach enables users to identify optimal hyperparameter configurations that lead to promising model performance. Figure 10 shows an example of this.

4.1.3.2 Model Performance Landscape

A pivotal aspect of the benchmark analysis involves dissecting the model's performance from various angles. Through scatter plots, the framework provides a view of the trade-offs between mean train accuracy and mean test accuracy for different models and learning rates. Each scatter plot is distinctively colour-coded by model type, granting users an intuitive understanding of how each model type performs under different learning rates. This approach allows users to navigate the intricate landscape of model performance and select configurations that align with their specific requirements.

Figure 5: Examples of charts automatically generated by the benchmark results.



Source: author

4.1.3.3 Network Metrics Evaluation

The benchmark goes beyond accuracy metrics by incorporating an evaluation of network-related metrics. These metrics offer insights into the graph's structural characteristics and representation within the embedding space. Three key metrics are utilised:

- **Assortativity Coefficient:** This metric assesses the correlation of nodes' degrees in the graph. Positive assortativity suggests that nodes with similar degrees tend to be connected, while negative assortativity indicates that nodes of different degrees are connected. It provides insights into the graph's organisation and connectivity patterns.
- **Global Efficiency:** Global efficiency quantifies how efficiently information is transmitted across the graph. A higher global efficiency signifies efficient information flow between nodes and highlights the graph's overall connectivity.
- **Transitivity:** Transitivity measures the extent to which nodes in the graph tend to form triangles. It reveals the presence of clustering within the graph and indicates the potential for community structure.

These metrics collectively enhance the analysis by providing additional dimensions to the evaluation process. By comparing these metrics between the original and embedded graphs, users can gain insights into how the embedding process may impact the graph's structural properties.

4.1.3.4 Visualizing Embeddings

The framework also visualises the final embeddings using t-SNE (t-distributed stochastic neighbour embedding). This visualisation technique projects high-dimensional embeddings into a two-dimensional space, providing an intuitive representation of data relationships. Visualising the embeddings allows users to separate different classes within the embedding space and identify potential clusters or patterns. Figure 7 shows an example of this.

The benchmarking framework transforms raw performance metrics into actionable insights through these analyses and visualisations. Users can make informed decisions by leveraging the insights derived from the sensitivity analysis, understanding the trade-offs revealed by the performance landscape, and evaluating the impact of network metrics on the embedding process. This quantitative results and qualitative insights empower researchers, practitioners, and developers to select the best-suited models and hyperparameters for their specific graph-based machine learning tasks.

5 BENCHMARK EXPERIMENT

An exploratory experiment was conducted to demonstrate the utility and effectiveness of the developed benchmark framework, leveraging the tool’s capabilities to evaluate various Graph Neural Network (GNN) architectures. This experiment serves as a testament to the framework’s versatility and its potential to streamline the process of model comparison and hyperparameter tuning. The primary aim of this experiment was to assess the performance of different GNN models across a range of hyperparameters and datasets, highlighting on their strengths and weaknesses under varying conditions. For access to the compiled results, they are available at the following link: [Benchmarkresultscompilation](#).

5.1 Setups

In pursuing a comprehensive assessment, a diverse set of GNN models were examined across multiple datasets. The datasets selected for this experiment included Cora, CiteSeer, and Pubmed. Each dataset represents distinct real-world scenarios and challenges, ensuring a robust evaluation of the models’ adaptability and generalization capabilities.

This empirical exploration summarizes for researchers and practitioners choosing GNN architectures and hyperparameters aligned with their specific problem domains. Furthermore, the experiment showcases the utility of the benchmark framework in simplifying and automating the evaluation process. The capability to seamlessly compare diverse models and their hyperparameter configurations promotes informed decision-making and expedites the model selection process.

To systematically explore the hyperparameter space, a range of values was predefined for each of the following hyperparameters:

- **Number of Layers:** The values (1, 2, 3) represent shallow, moderately deep, and deep architectures. Shallow networks may capture simpler patterns, while deeper networks could capture more complex relationships within the data.
- **Number of Neurons per Layer:** The values (16, 32, 64) were selected to cover a spectrum of model complexities. Lower values may prevent overfitting, while higher values could increase the capacity for capturing intricate features.
- **Learning Rate:** We considered (0.01, 0.001, 0.0001) as learning rate options. This range accounts for various learning dynamics, allowing models to converge effectively without diverging.
- **Dropout Rate:** The options (0.5, 0.2, 0.1) were utilized to account for regularization. A higher dropout rate can mitigate overfitting, while a lower rate can ensure model stability during training.

The selection of these specific hyperparameter values was a result of careful consideration. The goal was to encompass a diverse yet meaningful range of possibilities to explore architectural configurations. These ranges are beneficial in allowing users to understand how models behave under various scenarios and to fine-tune models according to specific task requirements. The total number of combinations generated by these hyperparameter values was 81, resulting in a rich tapestry of potential model configurations. This approach enhances the experiment’s robustness and empowers users to uncover optimal model settings for specific datasets and tasks.

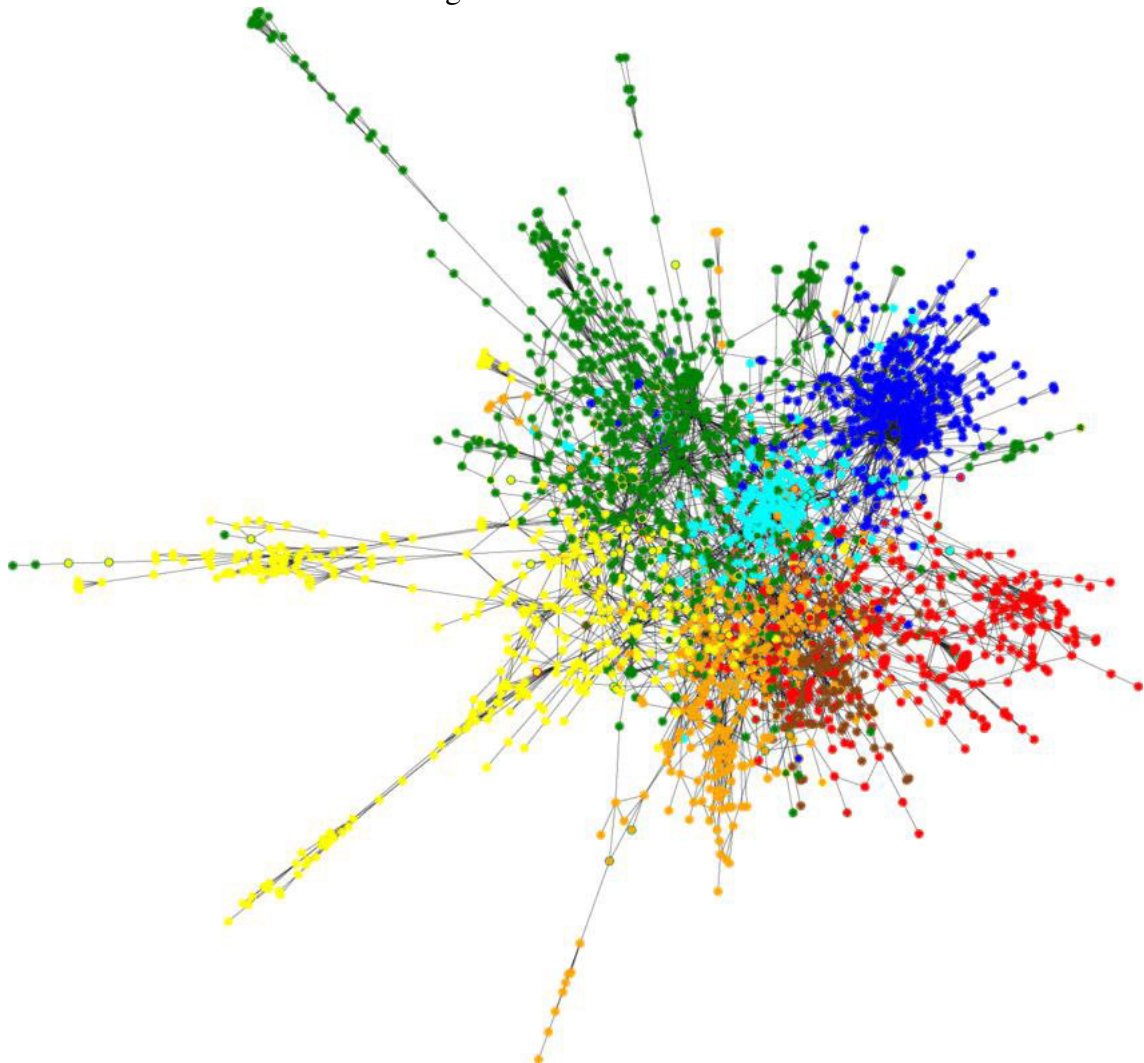
5.2 Datasets

The datasets used in the experiment play a pivotal role in evaluating the performance and robustness of the graph embedding models. Each dataset has distinct characteristics and challenges, contributing to a assessment of the models’ capabilities.

5.2.1 Classification Datasets: Cora, Citeseer, and Pubmed

- Cora:** The Cora dataset, consisting of 2,708 nodes and 10,556 edges, is a widely utilized benchmark in graph representation learning. It is based on a citation network involving scientific publications in machine learning. Nodes in the Cora graph represent individual documents, and edges correspond to citation relationships between these documents. The node features represent bag-of-words representations of the documents' titles and abstracts, allowing text-based content analysis. Each document is associated with one of seven predefined classes, which makes the Cora dataset suitable for node classification tasks. Due to its moderate size, Cora is an ideal introductory dataset for evaluating graph embedding techniques. Figure 7 provides an illustration of the Cora dataset. Each color in the figure represents a distinct class or theme of articles within the Cora dataset, such as neural networks, genetic algorithms, probabilistic methods, theory, case based, reinforcement learning, and rule learning.
- Citeseer:** With 3,327 nodes and 9,228 edges, the Citeseer dataset revolves around the citation relationships among scientific papers. The dataset encompasses documents from a broader spectrum of academic disciplines. It comprises a collection of academic papers, with nodes representing documents and directed edges indicating citation relationships. The node features include bag-of-words representations of the papers' titles and abstracts. With a goal similar to Cora, Citeseer is utilized for tasks such as document categorization, where the objective is to classify articles into predefined classes based on their content. The Citeseer dataset introduces a greater diversity of topics compared to Cora.
- Pubmed:** The Pubmed dataset boasts 19,717 nodes and 88,648 edges and is also a citation network dataset focusing on biomedical and life science research articles. The dataset's nodes represent scientific papers, and edges signify citation relationships between these papers. The node features consist of word vectors derived from the titles and abstracts of the documents. Pubmed presents a unique challenge in graph representation learning with its specific domain, as it involves a distinct vocabulary and specialized content. This dataset's applicability extends to medical document classification, where models aim to categorize articles into

Figure 6: Cora Dataset



Source: author

predefined classes such as medical specialties or research areas.

5.2.2 Network Analysis Datasets

- **Pattern Dataset:** The Pattern dataset is generated with a focus on capturing intricate patterns of relationships. It is structured as a graph with nodes connected in a patterned arrangement. This synthetic dataset is valuable for assessing the models' capability to learn and preserve complex relationships present in the graph. By providing known structural patterns, the Pattern dataset is a benchmark for

evaluating how well the graph embedding models can uncover and represent intricate graph structures.

- **Cluster Dataset:** The Cluster dataset is designed to challenge the models' ability to distinguish and group nodes based on clustering patterns. It consists of nodes grouped into distinct clusters, and edges are primarily established within clusters. The Cluster dataset enables the assessment of the models' capacity to capture local structure and cluster relationships. This synthetic dataset helps evaluate the models' performance in scenarios where nodes are inherently grouped based on their features and relationships.

5.3 GCN

Graph Convolutional Networks (GCNs) have emerged as a powerful tool for graph-structured data. These models capture intricate relationships and patterns within graphs, making them useful in various applications, from social network analysis to molecular chemistry. In this analysis, we delve into the performance of GCNs across three diverse datasets and setups. Within the context of the evaluation conducted, the performance and responsiveness of the Graph Convolutional Network (GCN) architecture were meticulously examined. The analysis encompassed an exhaustive exploration of diverse hyperparameter configurations to assess the model's adaptability and efficacy across multiple datasets. Table 4 presents a summary of the performance results across the evaluated datasets and hyperparameter configurations.

Table 4: Summary of GCN performance extracted from the main results table.

Architecture	Dataset	Min Test Accuracy	Max Test Accuracy	Mean Test Accuracy	Standard Deviation
GCN	Cora	0.198	0.805	0.717	0.114
GCN	CiteSeer	0.263	0.686	0.595	0.074
GCN	Pubmed	0.492	0.775	0.723	0.068

Source: author

Incorporating the detailed performance results, we observe that the GCN's mean test accuracies across the datasets fall within the range of 0.595914 to 0.723568. Remarkably, the model exhibited versatile behaviour across these datasets. The Pubmed

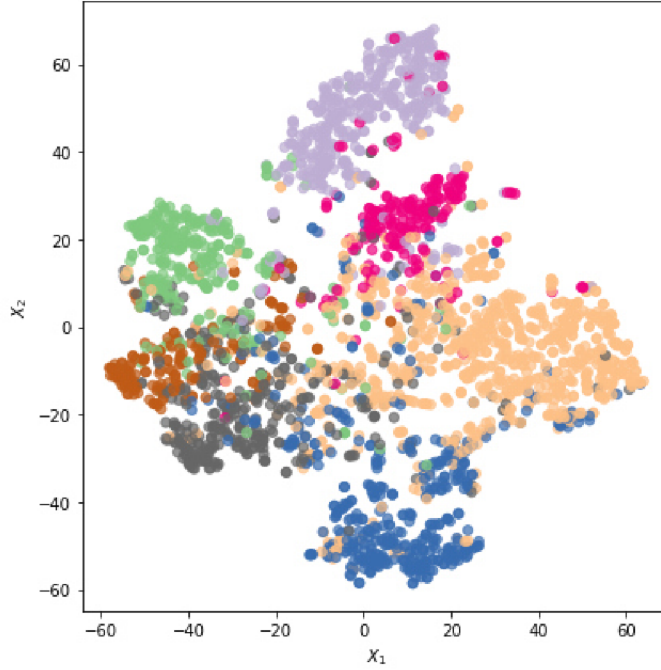
dataset yielded the highest mean test accuracy, suggesting the GCN’s adeptness in handling complex citation networks. However, the Cora dataset showcased the broadest range of performance, displaying the lowest minimum test accuracy of 0.198 and achieving the highest maximum test accuracy of 0.805. While the GCN shows versatility, dataset-specific intricacies can affect its performance.

We explored configurations that reduced model complexity in our investigation of resource-efficient setups. For instance, one noteworthy configuration involved employing a single-layer GCN with just 16 neurons, a learning rate 0.001, and a dropout rate of 0.5. Surprisingly, this simplified setup achieved a remarkable test accuracy of 0.805 on the Cora dataset. Similarly, a similar configuration with a slightly higher learning rate of 0.01 attained a test accuracy of 0.785, further showcasing the model’s robustness even under such resource-constrained conditions. These findings underscore the versatility of GCNs and their ability to perform effectively with reduced model complexity, offering insights for scenarios where computational resources are limited.

Our exploration also encountered scenarios where the model’s performance could have been better. Notably, the GCN exhibited its lowest performance on a specific configuration, achieving a test accuracy of 0.198. This setup included a GCN architecture with three layers, 16 neurons, a learning rate 0.0001, and a dropout rate of 0.5. The particularly low learning rate here significantly contributed to the poor performance. A learning rate this small could hinder the model’s ability to converge to an optimal solution within a reasonable number of training epochs. Additionally, the deeper architecture with two layers, while capable of capturing more complex patterns, might have suffered from overfitting due to the limited dataset size.

In conclusion, our analysis provides a study and evaluation of GCNs’ capabilities and limitations. The resource-efficient setup demonstrated that even with modest configurations, GCNs can deliver reasonable results, making them attractive for scenarios where resources are constrained. Our exploration of hyperparameters also emphasized their pivotal role in fine-tuning model performance, showcasing the necessity of careful configuration. As GCNs continue to be harnessed in various applications, this study contributes insights for practitioners seeking to harness the power of GCNs effectively. Figure 7 presents a t-SNE visualization of the GCN embeddings for the Cora dataset.

Figure 7: t-SNE Visualization of GCN embedding for Cora dataset



Source: author

5.4 GraphSAGE

GraphSAGE, an alternative graph neural network architecture, has gained prominence for its capabilities in handling graph-structured data. Much like GCNs, GraphSAGE models exhibit proficiency in capturing intricate relationships and patterns within graphs, rendering them tools across a spectrum of applications. In this analysis, we scrutinize the performance of GraphSAGE across three diverse datasets and various configurations, aiming to glean insights into its adaptability and effectiveness. Table 5 provides a summary of the performance results for GraphSAGE across the evaluated datasets and configurations.

Incorporating the detailed performance results in 5 , we observe that GraphSAGE demonstrates a mean test accuracy range of 0.570605 to 0.685506 across the datasets. These results exhibit a commendable level of versatility similar to GCNs. Interestingly, the Cora dataset presents the lowest minimum test accuracy of 0.179 and the highest maximum test accuracy of 0.797 among the datasets, indicating the model's capacity to

Table 5: Summary of GraphSAGE performance

Model	Dataset	Min Test Acc.	Max Test Acc.	Mean Test Acc.	Std. Deviation
GraphSAGE	Cora	0.179	0.797	0.685	0.144
GraphSAGE	CiteSeer	0.264	0.671	0.570	0.092
GraphSAGE	Pubmed	0.266	0.769	0.667	0.158

Source: author

adapt to varying data complexities.

In exploring resource-efficient configurations, we identified a cost-effective model setup for the Cora dataset. This configuration entailed employing a single-layer GraphSAGE model with 32 neurons in the initial layer, utilizing a learning rate of 0.01, and incorporating a dropout rate of 0.1. Remarkably, this simplified setup yielded a test accuracy of 0.797. This exemplifies the capacity of budget-friendly model configurations to deliver robust predictive performance in well-defined scenarios, mirroring the versatility of the GraphSAGE model under resource-constrained conditions.

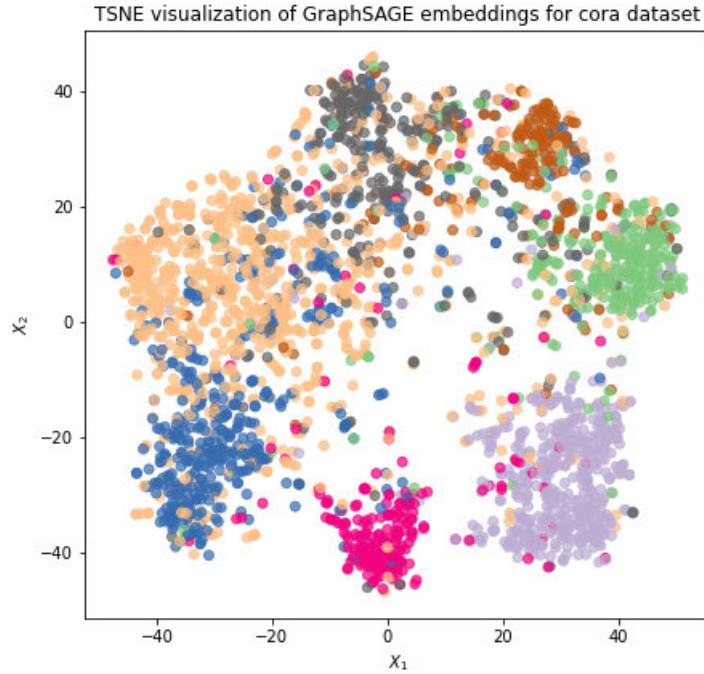
The lowest accuracy recorded in the test dataset was 0.179, observed in Experiment 34 on the Cora dataset. In this experiment, we used the GraphSAGE model with two layers: 16 neurons in the first layer, a learning rate of 0.0001, and a dropout rate of 0.2. This drop in accuracy can be attributed to the low learning rate and relatively high dropout rate, which resulted in the model underfitting the data.

On the other hand, the highest accuracy achieved in the test dataset was 0.797, observed in Experiment 32, also using the Cora dataset. In this experiment, the GraphSAGE model had two layers with 16 neurons, a learning rate of 0.001, and a dropout rate of 0.2. This experiment highlights that a relatively low learning rate can lead to exceptional performance when combined with appropriate hyperparameter settings. Figure 9 provides a t-SNE visualization of GraphSAGE embeddings for the Cora dataset.

5.5 GAT

Graph Attention Networks (GATs) have emerged as a promising graph neural network architecture, mainly known for their attention mechanisms that enable them to capture complex relationships in graph-structured data. Much like GCNs and GraphSAGE, GATs have showcased their utility across various applications. In this analysis, we meticulously assess the performance of GATs across three distinct datasets and mul-

Figure 8: t-SNE visualization of GraphSAGE embeddings for the dataset Cora



Source: author

multiple configurations, aiming to uncover their adaptability and effectiveness.

Table 6: Summary of GAT performance

Model	Dataset	Min Test Acc.	Max Test Acc.	Mean Test Acc.	Std. Deviation
GAT	Cora	0.613	0.795	0.727	0.049
GAT	CiteSeer	0.515	0.679	0.614	0.031
GAT	Pubmed	0.644	0.781	0.727	0.033

Source: author

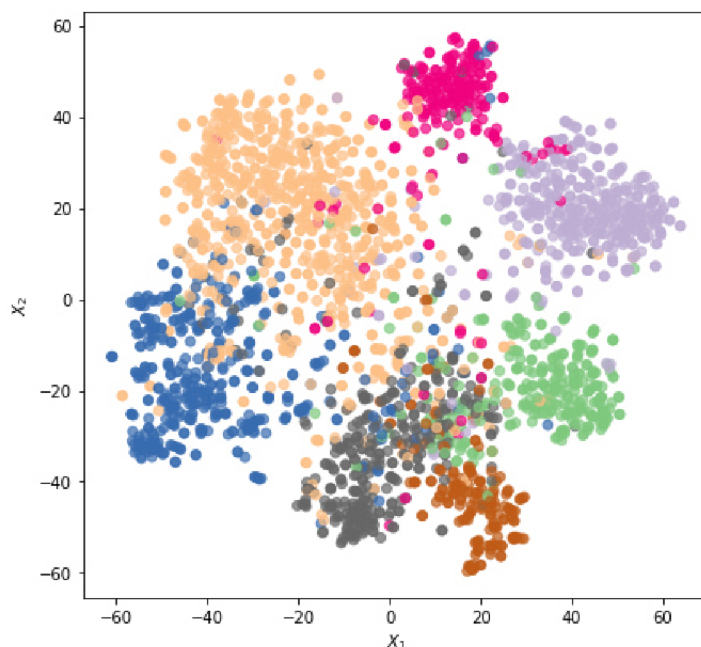
Incorporating the detailed performance results in 6, it is evident that the GAT (Graph Attention Network) has proven to be much more consistent and stable compared to other models. While GCN and GraphSAGE exhibited test accuracy ranges from 0.179 to 0.805, GAT consistently achieved a minimum accuracy of 0.515. It's worth noting that GAT showed considerably more stable mean accuracies and a significantly lower standard deviation. Across the datasets, GATs maintained a consistent mean test accuracy range of 0.614642 to 0.727556. These results underscore the robustness of GATs

in handling various data complexities. In particular, the Citeseer dataset recorded a minimum test accuracy of 0.515, while the Cora dataset reached a maximum of 0.795, further emphasizing their adaptability across diverse scenarios.

We explored configurations that reduced model complexity in our investigation of resource-efficient setups. For instance, one noteworthy configuration involved employing a single-layer GAT with 16 neurons, a learning rate of 0.001, and a dropout rate of 0.5. Surprisingly, this simplified setup achieved a remarkable test accuracy of 0.780 on the Cora dataset, further showcasing the model's robustness even under resource-constrained conditions. These findings underscore the versatility of GATs and ability to perform effectively with reduced model complexity, offering insights for scenarios where computational resources are limited.

Our exploration also encountered scenarios where the model's performance could have been better. The GAT exhibited its lowest performance on a specific configuration, achieving a test accuracy of 0.515. This setup included a GAT architecture with three layers, 16 neurons, a learning rate of 0.0001, and a dropout rate of 0.5. The high learning rate here significantly contributed to the poor performance. A high learning rate this high could hinder the model's ability to converge to an optimal solution. Additionally, the deeper architecture with three layers might have suffered from overfitting due to the limited dataset size.

Figure 9: t-SNE visualization of GAT embeddings for the dataset Cora.



Source: author

5.6 Overall Analysis

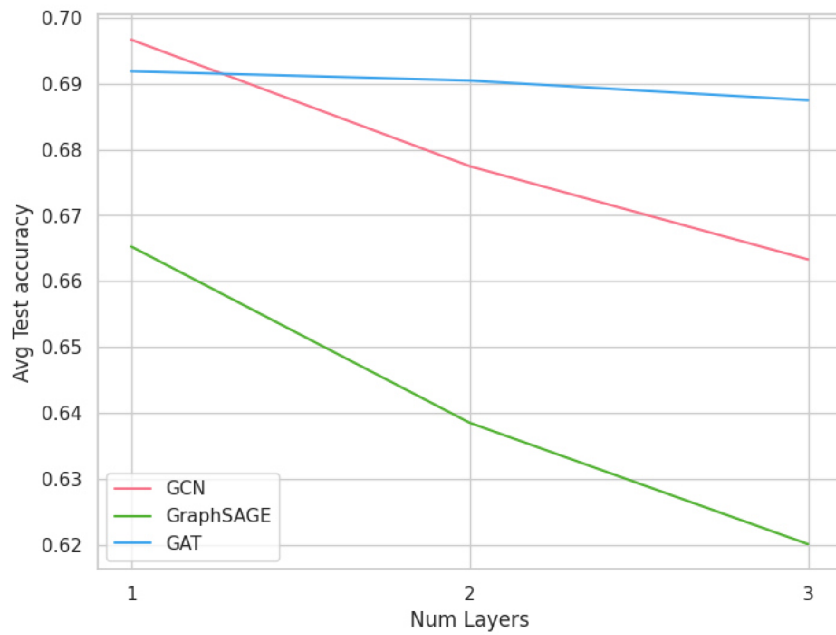
Expanding upon our previous analysis, we delve into the sensitivity of GCN, GraphSAGE, and GAT models to various hyperparameters, namely the number of layers, neurons, learning rate, and dropout rate. This investigation sheds light on how these models respond to hyperparameter variations, offering insights into their adaptability and robustness.

5.6.1 Number of Layers

Examining the impact of the number of layers on model performance reveals intriguing insights. GCN gradually declines in mean test accuracy as the number of layers increases. This observation implies that adding more layers to GCN may not necessarily improve performance, highlighting a diminishing returns effect. On the other hand,

GAT maintains relatively consistent mean test accuracies across different layer configurations, signifying its resilience to changes in this hyperparameter. GraphSAGE, while displaying a decrease in mean accuracy with more layers, also demonstrates sensitivity to the number of layers, although to a lesser extent than GCN. In Figure 10, we illustrate the accuracy sensibility between models related to the number of layers.

Figure 10: Accuracy sensibility between models related to the number of layers.

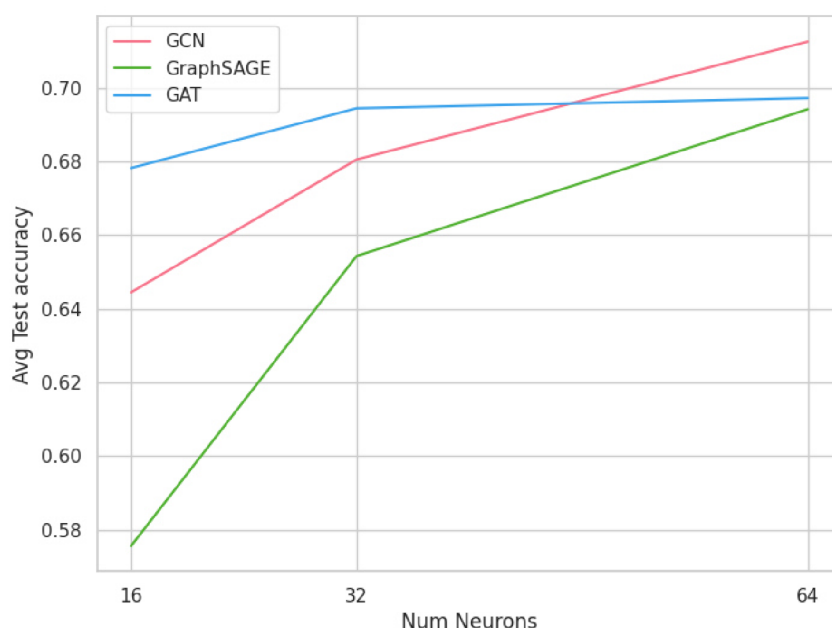


Source: author

5.6.2 Number of Neurons

The number of neurons in the initial layer is another hyperparameter with implications for model performance. GCN's mean test accuracy shows an upward trend with an increasing number of neurons. This indicates that a larger initial layer can contribute positively to GCN's performance. GraphSAGE follows a similar pattern, with the mean test accuracy improving as the number of neurons grows. GAT, however, maintains a relatively stable mean accuracy, regardless of the number of neurons, reinforcing its consistent performance irrespective of this hyperparameter. In figure 11, we illustrate the accuracy sensibility between models related to the number of neurons.

Figure 11: Accuracy sensibility between models related to the number of neurons.

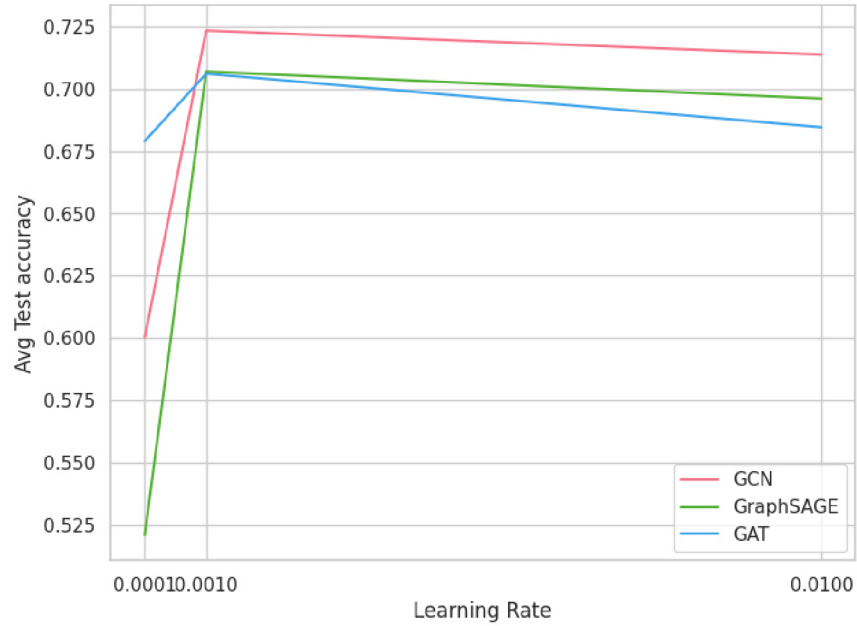


Source: author

5.6.3 Learning Rate

In examining the impact of the learning rate on model performance, distinct behaviours emerge among the three models. GCN exhibits sensitivity to changes in the learning rate, with optimal performance achieved at a moderate learning rate of 0.001, emphasizing the importance of precise learning rate tuning. In contrast, GAT demonstrates remarkable stability in mean test accuracy across various learning rates, highlighting its adaptability. GraphSAGE follows a trend similar to GCN but with slightly higher sensitivity to learning rate variations, underscoring the significance of fine-tuning this hyperparameter. In summary, while both GCN and GraphSAGE share an optimal learning rate, GCN is more sensitive to variations, while GAT maintains consistent performance, showcasing its adaptability to different learning rates. These insights provide guidance for hyperparameter tuning and illuminate the models' unique characteristics. In figure 12, we illustrate the accuracy sensibility between models related to the learning rate.

Figure 12: Accuracy sensibility between models related to the learning rate.

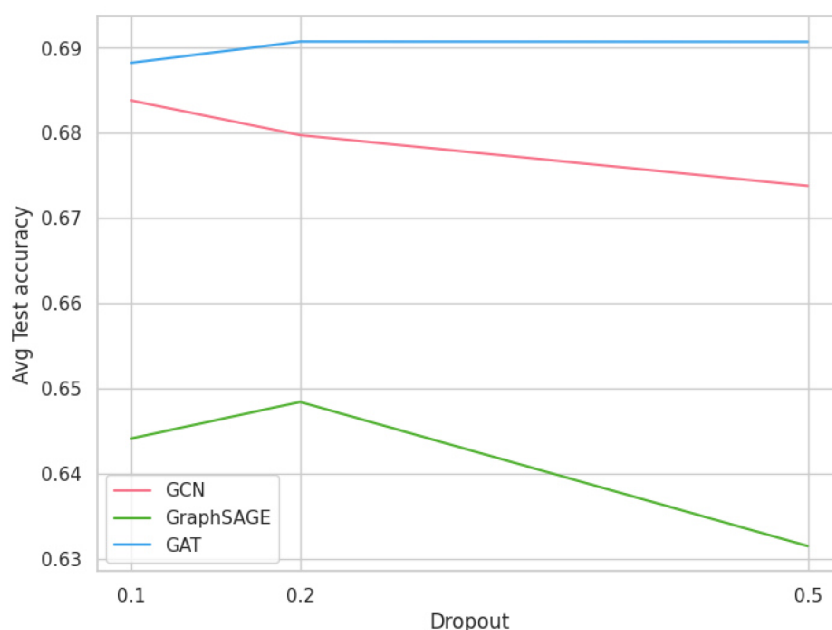


Source: author

5.6.4 Dropout

The dropout rate, a regularization hyperparameter, also impacts model performance differentially. GCN's mean test accuracy experiences a subtle decline as the dropout rate increases. This suggests that moderate dropout rates (0.1 and 0.2) are more favourable for GCN's performance. In contrast, GAT and GraphSAGE exhibit a relatively constant mean accuracy across different dropout rates, signifying their robustness to this hyperparameter. In Figure 13, we illustrate the accuracy sensibility between models related to the dropout rate.

Figure 13: Accuracy sensibility between models related to the dropout.



Source: author

5.7 Conclusions

Considering these findings, GAT is the most stable and adaptable model, with the highest mean test accuracies and the lowest standard deviations across various hyperparameter configurations. GCN demonstrates versatility, particularly in responding to the number of neurons and learning rate. It performs well with moderate learning rates and benefits from larger initial layers. GraphSAGE showcases potential for exceptional performance, particularly when meticulously tuned, although it displays sensitivity to certain hyperparameter variations, such as the number of layers. In conclusion, hyperparameter sensitivity is critical to model selection and fine-tuning. The choice between GCN, GraphSAGE, and GAT should consider their performance and their response to hyperparameters. GAT emerges as a robust choice, maintaining consistency across a wide range of hyperparameter configurations. GCN, with its versatility and compatibility, remains a dependable option. GraphSAGE, while sensitive to specific hyperparameters, holds promise for achieving outstanding results under carefully optimized

settings. Ultimately, selecting the most suitable model should align with specific project requirements, available computational resources, and the capacity for hyperparameter tuning.

5.8 Network Analysis

In this investigation, we explore network metrics and their intricate relationship with Graph Neural Network (GNN) models. Our objective is to elucidate the transformations in the structural characteristics of graphs when subjected to various GNN models and experimental setups. We also aim to discern the repercussions of these structural modifications on node classification tasks, shedding light on the subtle interplay between network metrics and GNN performance.

In this investigation, we explore network metrics and their intricate relationship with Graph Neural Network (GNN) models. Our objective is to unravel the transformations in the structural characteristics of graphs when subjected to various GNN models and experimental setups. We also aim to discern the repercussions of these structural modifications on node classification tasks, shedding light on the subtle interplay between network metrics and GNN performance.

5.8.1 Network Metrics in Original Datasets

Let us begin by summarising the key network metrics and deciphering their implications in the context of our original datasets, which encompass Cora, Citeseer, PATTER, and CLUSTER. These metrics, namely Assortativity Coefficient, Global Efficiency, and Transitivity, offer insights into the underlying network structures. Table 7 provides a summary of Network Metrics for Original Datasets.

Table 7: Summary of Network Metrics for Original Datasets.

Dataset	Assortativity Coefficient	Global Efficiency	Transitivity
Cora	-0.065	0.149	0.093
Citeseer	0.048	0.051	0.130
PATTERN	-0.033	0.711	0.429
CLUSTER	-0.012	0.651	0.315

Cora presents a negative assortativity coefficient, indicating that nodes with similar

degrees are less likely to connect. With moderate global efficiency and transitivity, Cora balances local and global network connections. Citeseer, in contrast, exhibits a positive assortativity coefficient, suggesting that nodes with similar degrees are more inclined to form connections. It showcases low global efficiency and moderate transitivity, signifying a network with a tendency for local clustering. PATTERN portrays a negative assortativity coefficient, akin to Cora, but is notably distinct regarding global efficiency and transitivity. It possesses substantially higher global efficiency and transitivity values, indicating a network facilitating efficient global information flow and pronounced community structure preservation.

5.8.2 Experimental Setups

Transitioning to examining experimental setups, we have deployed three distinct configurations: Setup 7_b, Setup 28_b, and Setup 55_b. Considering the prior benchmark results in node classification, these setups are the subject of our scrutiny. Notably, these setups encompass different performance scenarios, including one with relatively low test accuracy (Setup 7_b), one with moderate performance, and one that excels in node classification accuracy (Setup 55_b). Table 8 provides a summary of setup parameters and node classification accuracy for these configurations.

Table 8: Summary of Setup Parameters and Node Classification Accuracy.

Setup	Learning Rate	Epochs	Hidden Channels	Layers	Accuracy
7_b	0.0001	50	8	1	0.31
28_b	0.01	50	8	2	0.59
55_b	0.01	50	6	3	0.75

These setups encapsulate a spectrum of complexity levels, with the 'b'-suffixed configurations representing less intricate scenarios than the original benchmark. This distinction arises from the observation that exceedingly complex models tend to result in an Assortativity Coefficient null, alongside Global Efficiency and Transitivity values leaning towards 1. These unexpected transformations in network metrics indicate a divergence from the original network properties.

The null assortativity coefficient signifies a loss of degree correlation structure, while the heightened global efficiency and transitivity suggest an amplification of overall information flow and the preservation of community structures. These distortions directly result from the intricate operations performed by GNNs, particularly as their

complexity increases. Thus, it is imperative to delve into the root causes of these distortions and their ramifications on the performance of GNN models. Table 9 provides a summary of the average distortion of the 3 models in the datasets Cora, CiteSeer, PATTERN and CLUSTER.

Table 9: Summary of metrics distortion

Dataset	Avg. Assortativity Distortion	Avg. Global Efficiency Distortion	Avg. Transitivity Distortion
Cora	0.030	0.835	0.886
CiteSeer	0.052	0.946	0.868
PATTERN	-	0.262	0.537
CLUSTER	-	0.348	0.684

In Cora, GCN maintains a similar assortativity coefficient but significantly increases global efficiency and transitivity. This shift suggests that GCN might need to emphasise more about information flow and community structure preservation in Cora. Citeseer, on the other hand, sees GCN preserving assortativity while emphasising global efficiency and transitivity, indicating a greater focus on information flow and community structure maintenance. However, in the synthetic datasets, PATTERN and CLUSTER, GCN’s embeddings lead to substantial distortions in network properties, resulting in assortativity becoming a null value and global efficiency and transitivity approaching 1.

GraphSAGE consistently preserves assortativity across datasets while maintaining or enhancing global efficiency and transitivity. This demonstrates GraphSAGE’s proficiency in capturing global graph information. GAT impressively preserves assortativity across all datasets, which is a notable characteristic. In Cora and Citeseer, GAT effectively maintains network properties but experiences mild distortions. However, in the synthetic datasets, PATTERN and CLUSTER, GAT encounters challenges similar to GCN and GraphSAGE, with assortativity becoming a null value and global efficiency and transitivity approaching 1.

We’ve observed some noteworthy trends regarding the correlation between network property preservation (low distortion) and accuracy in the node classification benchmark. Models that excel in node classification tend to exhibit lower levels of network metric distortion. For instance, in Setup 55_b, where all models demonstrate high accuracy, it also registers the lowest network metric distortion. However, it’s worth noting that while the setup with the highest accuracy also achieved the lowest distortion, there isn’t a strict proportionality. Even setups with significantly lower accuracy levels dis-

Table 10: Network Metrics Summary Table for Setups 7_b, 28_b, and 55_b.

Dataset	Model	Assortativity	Global Efficiency	Transitivity	Setup
Cora	GCN	-0.039467	0.981966	0.975982	7_b
Cora	GraphSAGE	-0.025937	0.994850	0.991091	7_b
Cora	GAT	-0.041312	0.979594	0.972521	7_b
Citeseer	GCN	-0.009817	0.994668	0.995250	7_b
Citeseer	GraphSAGE	-0.001659	0.999978	0.999956	7_b
Citeseer	GAT	-0.002103	0.999959	0.999918	7_b
PATTERN	GCN	-	1.0	1.0	7_b
PATTERN	GraphSAGE	-0.1685450	0.92042	0.90215	7_b
PATTERN	GAT	-	1.0	1.0	7_b
CLUSTER	GCN	-	1.0	1.0	7_b
CLUSTER	GraphSAGE	-	1.0	1.0	7_b
CLUSTER	GAT	-	1.0	1.0	7_b
Cora	GCN	-	1.0	1.0	28_b
Cora	GraphSAGE	-	1.0	1.0	28_b
Cora	GAT	-0.066703	0.994048	0.989626	28_b
Citeseer	GCN	-	1.0	1.0	28_b
Citeseer	GraphSAGE	-0.000721	0.999999	0.999999	28_b
Citeseer	GAT	-0.015745	0.999261	0.998606	28_b
PATTERN	GCN	-	1.0	1.0	28_b
PATTERN	GraphSAGE	-0.6131	0.9960	0.9950	28_b
PATTERN	GAT	-	1.0	1.0	28_b
CLUSTER	GCN	-	1.0	1.0	28_b
CLUSTER	GraphSAGE	-	0.9830	1.0	28_b
CLUSTER	GAT	-	0.9160	1.0	28_b
Cora	GCN	-0.107970	0.983311	0.973674	55_b
Cora	GraphSAGE	0.041008	0.900738	0.905176	55_b
Cora	GAT	-0.118320	0.975234	0.962356	55_b
Citeseer	GCN	-0.037727	0.988331	0.980272	55_b
Citeseer	GraphSAGE	-0.000666	0.999997	0.999995	55_b
Citeseer	GAT	-0.020065	0.999460	0.998969	55_b
PATTERN	GCN	-	1.0	1.0	55_b
PATTERN	GraphSAGE	-	1.0	1.0	55_b
PATTERN	GAT	-	1.0	1.0	55_b
CLUSTER	GCN	-	1.0	1.0	55_b
CLUSTER	GraphSAGE	-	1.0	1.0	55_b
CLUSTER	GAT	-	1.0	1.0	55_b

Source: Author

play distortions relatively close to the best result. This suggests that network metric preservation, although correlated with node classification performance, is not entirely determined by it, highlighting the complex interplay between these factors. Table 10 provides the results for Network Metrics for Setups 7_b, 28_b, and 55_b.

Calculating network metrics in both the original graphs and their embeddings appeared to be resource-intensive. This process involves traversing numerous paths within the graphs and tends to be time-consuming and memory-intensive, often pushing the limits of the available resources, especially in environments like Colab Pro.

5.8.3 Steps to reproduce

The code repository is complemented by a Jupyter notebook version designed for execution on Google Colab. To replicate the experiments, navigate to the `GoogleColablink` and execute all cells. Additionally, you have the option to incorporate graph datasets by referencing them in the `'datasets'` and `'dataset_names'` variables. To tailor the experiment settings, simply adjust the values of the following variables: `num_layers_values`, `num_neurons_values`, `num_epochs`, `learning_rate_values`, and `dropout_values`.

5.9 Field Gaps and discussion

One of the main gaps in graph embedding is the need for standardised evaluation metrics. Embedding methods, such as node2vec Grover and Leskovec [26] and graph2vec Narayanan et al. [43], have been proposed to map nodes and graphs to a low-dimensional space. However, there has yet to be a consensus on evaluating the quality of these embeddings and how they impact graph discovery tasks, such as anomaly detection Zong et al. [64].

The limited scalability of algorithms is another challenge in graph embedding. Many existing methods cannot handle large graphs with millions of nodes and edges, making them unsuitable for real-world applications Ying et al. [59].

Interpretability is also one challenge. Some graph embedding algorithms, such as deep learning-based methods, tend to generate representations that are hard to interpret. This makes it difficult to understand how a particular was classified.

5.10 Discussion

Potential solutions to the gap in scalability include the use of sampling techniques, approximations, and parallelisation, as well as the use of distributed computing frameworks, some papers that address this gap are Goyal et al. [25], Pacheco et al. [45].

Another potential solution is using hybrid approaches that combine multiple methods, such as machine learning and rule-based processes, to improve the performance and robustness of graph mining systems Koyuturk et al. [35].

Researchers can explore the space using interpretable models, such as decision trees or linear models, to make graph embedding algorithms more interpretable. Additionally, techniques such as feature importance scores or partial dependence plots can be used to understand how particular nodes are classified.

Developing a standardised set of metrics and tools to evaluate the performance of graph embedding algorithms would be beneficial in creating a sense of consensus across the field. This would enable researchers to compare the strengths and weaknesses of different algorithms and their trade-offs and help guide the development of new algorithms and features.

As an extension of our study and research, we also delved into critical aspects concerning the quality of graph embedding models. This extension was accomplished through an extensive investigation, where we compared critical points of intersection among various models. To arrive at the key evaluation categories, we conducted a thorough literature review of recent advances in graph embedding and identified the most commonly used evaluation criteria across different studies. We aimed to identify the key factors essential for assessing the effectiveness of graph embedding models and to develop a framework that understands the strengths and limitations of different approaches. After analysing the existing literature, we identified six key categories that are essential for evaluating graph embedding models:

1. preserving local structure,
2. preserving global structure,
3. capturing complex relationships,
4. flexibility,

5. capturing long-range relationships, and
6. scalability to large graphs.

These categories were chosen based on their importance in various graph-related tasks, such as link prediction, community detection, network visualisation, and classification. By evaluating graph embedding models based on these categories, we aim to understand the strengths and limitations of different approaches and their trade-offs.

5.10.0.1 The Keys

Local structure (1) preservation is vital for task prediction and community detection while preserving global structure (2) is better suited for network visualisation and classification problems.

Complex relationships (3) between nodes, such as non-linear and higher-order relationships, are essential in knowledge graph completion and relational learning. However, capturing such relationships may come at the cost of computational efficiency and scalability to large graphs.

Flexibility (4) is a consideration for dynamic graph structures, such as social networks and web graphs. Models that can quickly adapt to changing relationships between nodes may not be as computationally efficient as models optimised for specific graph structures.

Long-range relationships(5) are essential for link prediction and recommendation systems but may not capture fine-grained local relationships.

Scalability (6) to large graphs is helpful for big data applications and real-time systems. However, computational efficiency may come at the cost of capturing complex relationships or higher-level structures in the graph.

Table 11 presents a comparative study based on extensive research on several papers and studies, evaluating the performance of Matrix Factorization (MF), Random Walk (RW), and Deep Learning without Random Walk (DL NO-RW) models in the six keys mentioned above. The table shows the performance of each model in six evaluation categories, with "Good" indicating high performance and "Limited" indicating relatively lower performance.

Category	1	2	3	4	5	6	References
MF	Good	Good	Limited	Limited	Limited	Limited	[60],[61],[29],[26],[17]
RW	Good	Good	Good	Limited	Limited	Limited	[47],[52],[28],[62],[27]
DL NO-RW	Good	Good	Good	Good	Limited	Limited	[11],[32],[53],[15],[11]

Table 11: Comparison of Matrix Factorization (MF), Random Walk (RW), and Deep Learning without Random Walk (DL NO-RW) in key common characteristics

5.10.1 Future directions

This benchmark framework transcends traditional performance metrics by converting them into practical insights. Delving into the sensitivity analysis, unearthing performance trade-offs, and assessing the influence of network metrics on the embedding process equips users with a deeper understanding of model behaviour. This synthesis of quantitative results and qualitative observations empowers researchers, practitioners, and developers to make informed choices when selecting models and hyperparameters tailored to their specific graph-based machine learning requirements.

For future directions, it is imperative to explore more scalable methods for the network analysis component, delving deeper into this analysis to uncover intricate and intrinsic relationships regarding network distortions and their repercussions, including a broader range of network metrics and exploring the correlations and impacts associated with each. Furthermore, expanding this experimental framework to encompass other types of tasks, such as link prediction and anomaly detection, can provide insights into the versatility of graph-based machine learning models. Additionally, considering the ever-evolving landscape of machine learning, exploring methods for incorporating domain-specific knowledge or graph-specific features could be another avenue to enhance model performance and interpretability.

6 CONCLUSION

In summary, this work has extensively explored the current landscape in graph embedding. It has achieved this through an in-depth analysis of existing research, identifying common characteristics, and connecting various approaches to graph embedding.

Our presented framework is a valuable tool for benchmarking and dissecting GNN models. It simplifies the integration of diverse models, ensures dataset readiness for various models, and streamlines the entire process, from automated preparation to parameterization, hyperparameter exploration, model creation, training, evaluation, and comparison. Alongside these features, it offers insightful visualizations that transform raw performance metrics into actionable insights.

As we peer into the future, there's exciting potential in developing more scalable methods for network analysis. This could involve a deeper dive into the intricate relationships within network distortions and their far-reaching impacts. Expanding our experiment's horizons to encompass tasks like link prediction and anomaly detection promises further insights into the framework's versatility.

Furthermore, future iterations of our framework promise to guide the development of novel models that can address the limitations and trade-offs of existing approaches. This might pave the way for creating more efficient graph neural network models, ideally suited for various applications. Ultimately, these advancements have the potential to leave a significant mark on the field, propelling progress, bolstering reproducibility, and nurturing the growth of more potent and efficient models.

REFERENCES

- [1] Adewumi, A. O. and Akinyelu, A. A. (2017). A survey of machine-learning and nature-inspired based credit card fraud detection techniques. *International Journal of System Assurance Engineering and Management*, 8(2):937–953.
- [2] Aggarwal, C. C. (2013). Outlier ensembles: position paper. *ACM SIGKDD Explorations Newsletter*, 14(2):49–58.
- [3] Ahmed, M., Mahmood, A. N., and Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31.
- [4] Ahmed, N. and Mahoney, M. W. (2013). Distributed graph factorization. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 567–575. ACM.
- [5] Akoglu, L., Tong, H., and Koutra, D. (2015). Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688.
- [6] Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396.
- [7] Bhattacharyya, S., Jha, S., Tharakunnel, K., and Westland, J. C. (2011). Data mining for credit card fraud: A comparative study. *Decision support systems*, 50(3):602–613.
- [8] Blumberg, R. and Atre, S. (2003). The problem with unstructured data. *Dm Review*, 13(42-49):62.

- [9] Bolton, R. J., Hand, D. J., et al. (2001). Unsupervised profiling methods for fraud detection. *Credit scoring and credit control VII*, pages 235–255.
- [10] Cai, D. W., He, X., Han, J., and Ren, J. (2017). Unsupervised deep embedding for clustering analysis. *arXiv preprint arXiv:1706.02263*.
- [11] Cai, H., Zheng, V. W., and Chang, K. C.-C. (2018). A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637.
- [12] Castillo, C., Donato, D., Gionis, A., Murdock, V., and Silvestri, F. (2007). Know your neighbors: Web spam detection using the web topology. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 423–430.
- [13] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58.
- [14] Chen, H., Perozzi, B., Al-Rfou, R., and Skiena, S. (2018). A tutorial on network embeddings. *arXiv preprint arXiv:1808.02590*.
- [15] Cui, P., Wang, W., and Zhu, W. (2018). A comprehensive survey on graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637.
- [16] Damjanovic, U., Fernandez, V., Izquierdo, E., and Martinez, J. M. (2008). Event detection and clustering for surveillance video summarization. In *2008 Ninth International Workshop on Image Analysis for Multimedia Interactive Services*, pages 63–66. IEEE.
- [17] Demmel, J., Dumitriu, I., and Holtz, O. (2007). Fast linear algebra is stable. *Numerische Mathematik*, 108(1):59–91.
- [18] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

- [19] Ding, K., Li, J., Bhanushali, R., and Liu, H. (2019). Deep anomaly detection on attributed networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 594–602. SIAM.
- [20] Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking graph neural networks.
- [21] Euler, L. (1736). *Solutio problematis ad geometriam situs pertinentis*. Lausannae.
- [22] Fan, X. and Han, J. (2019). Outlier detection on graphs: A survey. *ACM Computing Surveys (CSUR)*, 52(1):1–44.
- [23] Fang, J., Xu, D., et al. (2020). How the internet influences the development of modern market economy. *American Journal of Industrial and Business Management*, 10(05):1002.
- [24] George, G., Haas, M. R., and Pentland, A. (2014). Big data and management.
- [25] Goyal, A., Akoglu, L., and Liu, H. (2018). Scalable anomaly detection in graphs. In *Proceedings of the VLDB Endowment*, volume 11, pages 2049–2060. VLDB Endowment.
- [26] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.
- [27] Hamilton, W. L., Ying, R., and Leskovec, J. (2017a). Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035.
- [28] Hamilton, W. L., Ying, R., and Leskovec, J. (2017b). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.
- [29] He, X. and Chua, T.-S. (2017). Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 355–364.

- [30] Invernizzi, L., Comparetti, P. M., Benvenuti, S., Kruegel, C., Cova, M., and Vigna, G. (2012). Evilseed: A guided approach to finding malicious web pages. In *2012 IEEE symposium on Security and Privacy*, pages 428–442. IEEE.
- [31] Jin, M., Liu, Y., Zheng, Y., Chi, L., Li, Y.-F., and Pan, S. (2021). Anemone: Graph anomaly detection with multi-scale contrastive learning. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, pages 3122–3126.
- [32] Kipf, T. N. and Welling, M. (2016a). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.
- [33] Kipf, T. N. and Welling, M. (2016b). Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*.
- [34] Koutra, D. and Faloutsos, C. (2013). Outlier detection in graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 85–93. ACM.
- [35] Koyuturk, M., Akoglu, L., and Faloutsos, C. (2010). Robust anomaly detection in graphs. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, pages 924–933. ACM.
- [36] Kumar, V. (2005). Parallel and distributed computing for cybersecurity. *IEEE Distributed Systems Online*, 6(10).
- [37] Kwon, O., Lee, N., and Shin, B. (2014). Data quality management, data usage experience and acquisition intention of big data analytics. *International journal of information management*, 34(3):387–394.
- [38] Liu, Y., Li, Z., Pan, S., Gong, C., Zhou, C., and Karypis, G. (2021). Anomaly detection on attributed networks via contrastive self-supervised learning. *IEEE transactions on neural networks and learning systems*, 33(6):2378–2392.
- [39] Madnick, S. E. and Wang, Y.-Y. R. (1992). *Introduction to the TDQM research program*. Massachusetts Institute of Technology.

- [40] Mao, X., Yang, H., Huang, S., Liu, Y., and Li, R. (2019). Extractive summarization using supervised and unsupervised learning. *Expert Systems with Applications*, 133:173–181.
- [41] Matsubayashi, K., Yamashita, A., Nonaka, H., and Konno, Y. (2018). A research on document summarization and presentation system based on feature word extraction from stored informations. In *2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 60–63. IEEE.
- [42] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.
- [43] Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., and Jaiswal, S. (2017). graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*.
- [44] Nathaniel, J. (2021). Recent advances in graph convolutional network (gcn): A hitchhiker guide to the latest advances in gcn, in plain english. <https://towardsdatascience.com/recent-advances-in-graph-convolutional-network-gcn-a-hitchhiker-guide-> [Online; accessed 20-Nov-2023].
- [45] Pacheco, J., Farajtabar, M., Gharavi, S., and Bayati, M. (2020). Scalable graph anomaly detection: A survey. *ACM Computing Surveys*, 53(2):1–37.
- [46] Pan, W., Long, G., Liu, J., and Yang, Q. (2018). Adversarially regularized graph autoencoder for graph embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1175–1184.
- [47] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM.
- [48] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

- [49] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.
- [50] Sultana, N. and Palaniappan, S. (2018). A survey on online social network anomaly detection. *International Journal of Innovative Science and Research Technology*, 3(3):243–257.
- [51] Sun, Y., Wang, X., Chen, J., and Liu, H. (2019). Anomaly detection in dynamic graphs with attention-based graph neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 832–841. ACM.
- [52] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. (2015). Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077.
- [53] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.
- [54] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2019). Deep graph infomax. *International Conference on Learning Representations*.
- [55] Velivckovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations*.
- [56] Wang, X. and Tang, K. (2017). Community detection in graphs. *Wiley Interdisciplinary Reviews: Computational Statistics*, 9(3):e1399.
- [57] Wei, S. and Zhao, Y. (2022). Graph learning: A comprehensive survey and future directions. *arXiv preprint arXiv:2212.08966*.
- [58] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *International Conference on Learning Representations*. Accepted as poster.
- [59] Ying, R., He, R., Chen, K., Eksombatchai, C., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems.

Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 974–983.

- [60] Zhang, S., Yao, L., Sun, A., and Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM computing surveys (CSUR)*, 52(1):1–38.
- [61] Zhang, S., Yao, L., Tay, Y., Xu, X., Zhang, X., and Zhu, L. (2018). Metric factorization: Recommendation beyond matrix factorization. *arXiv preprint arXiv:1802.04606*.
- [62] Zhang, Z., Cui, P., and Zhu, W. (2020). Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):249–270.
- [63] Zitnik, M., Agrawal, M., and Leskovec, J. (2018). Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466.
- [64] Zong, Y., Chen, J., Wang, X., and Liu, H. (2018). Anomaly detection on graphs with gated graph neural networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2215–2224. ACM.

7 ARTICLE I - GRAPHLED: A GRAPH-BASED APPROACH TO PROCESS AND VISUALISE LINKED ENGINEERING DOCUMENTS

This article presents GraphLED, a system designed to handle data processing, graph-based modeling, and visually appealing representation of associated documents. In February 2023, it was submitted to the ArXiv Repository with the DOI: <https://doi.org/10.48550/arXiv.2302.08905>. The graph-based approach employed by GraphLED enables better comprehension of interconnected information, as the graph structure offers a promising tool for modeling the underlying data properties of engineering documents. The findings of this research have the potential to enhance the dependability and robustness of industrial production systems by facilitating the automated summarization of large volumes of documents and their connections.

Roles: Willian B. de Lemos: Conceptualization, Modeling, Programming, ETL Design, Query Architecture, Analysis, Writing - Review and Editing; Lucas de A. M. Ribeiro: Conceptualization, Programming, Simulations, Writing - Review and Editing; Vanessa T. da Silva: Conceptualization, Programming, Analysis, Writing - Original Draft, Writing - Review and Editing; Marcelo R. Pias: Conceptualization, Analysis, Writing - Original Draft, Writing - Review and Editing; Sílvia S. da C. Botelho: Writing - Review and Editing; Nelson L. D. Filho: Writing - Review and Editing.

GRAPHLED: A GRAPH-BASED APPROACH TO PROCESS AND VISUALISE LINKED ENGINEERING DOCUMENTS

Vanessa T. da Silva, Lucas de A. M. Ribeiro, Willian B. de Lemos,
Sílvia S. da C. Botelho, Nelson L. D. Filho, Marcelo R. Pias

Universidade Feral do Rio Grande (FURG)

Rio Grande, RS, Brazil

{vanessa.telles, lucasribeiro, willianlemons.b, silviacb, dmtnldf, mpias}@furg.br

ABSTRACT

The architecture, engineering and construction (AEC) sector extensively uses documents supporting product and process development. As part of this, organisations should handle big data of hundreds, or even thousands, of technical documents strongly linked together, including CAD design of industrial plants, equipment purchase orders, quality certificates, and part material analysis. However, analysing such records is daunting for users because it gets complicated to sift through hundreds of documents to establish valuable relationships. This paper addresses how knowledge extracted from linked engineering documents contributes to industrial digitalisation under IT/OT convergence. The proposed GraphLED is a system tasked with data processing, graph-based modelling, and colourful visualisation of related documents. The graph-based approach ensures an improved understanding of linked information because the graph structure offers a promising tool to model the underlying data properties of engineering documents. Preliminary system validation indicates quality improvements are possible in the OCR-based data (85.9% of ambiguous text data removed). This work has the potential to benefit the industry by improving the reliability and resilience of industrial production systems through automated summaries of large quantities of documents and their linkage.

Keywords graph-based processing, document processing, industrial digital transformation, industrial IO, graph big data

1 Introduction

Ubiquitous data creation has grown at a rapid pace, with big data collection providing tangible digital transformation benefits in several industries [1]. However, data remains document-centric in mixed analogue, and digital formats in more traditional sectors, including architecture, engineering and construction (AEC). Organisations often deal with hundreds, or even thousands, of technical documents such as CAD design of industrial factory plants, fleet design, equipment purchase orders, quality certificates, and material analysis [2].

The management of technical documents with efficient tools leads to efficient coordination of critical day-to-day activities throughout AEC process lifetime [3]. To this end, continuous data processing and visualisation bring considerable benefits to the AEC end-users: (i) support timely inspection of engineering projects, (ii) develop automated document compliance models with existing standards and societies' guidelines and (iii) promote information exchange among key stakeholders (suppliers, clients, standards societies). However, AEC documents are inherently complex because diverse unstructured topics usually link many documents to convey the information. Conventional state-of-the-art approaches that handle structured data (e.g. search by keywords in relational databases) are not feasible in this case [4].

The relationship between the tables is defined in advance, so users can only obtain limited relevant knowledge [5]. Paper-based documents further complicate the matter where scanned handwritten reports, technical specifications, engineering drawings, schematics, stamps, signatures and so on [6] make it challenging to retrieve information efficiently, even in its simplest form. For instance, a *databook* – a collection of engineering documents that specify project events – comprises many pieces of data (i.e. topics) that coherently link with data present in other databooks. This interconnection

creates a complex yet powerful *network* of related engineering documents that offer several features. For instance, the end-users can visually inspect the material traceability of equipment parts and their compliance with current technical specifications (e.g. ASTM - American Society for Testing and Materials). Today the visualisation and inspection of a large dataset of documents become challenging and error-prone if the user is not assisted by big data tools.

A key high-level question in this context is: *how could knowledge, namely engineering document summaries and their relationships, extracted from noisy OCR processed engineering documents contribute to industrial digitalisation and resilience of production systems?*

The knowledge extraction is expected to be carried out efficiently and close to real-time. Equally important in this process is the exploitation of the cross-reference data between documents to obtain new information, a task that is inherently difficult for humans because of the document network depth and broadness [7].

This paper sheds some light to answer the question above. The proposed approach considers a simple yet promising mechanism to extract and structure the data for processing, knowledge extraction, and visualisation from diverse engineering databooks. The GraphLED is a system tasked with data processing, graph-based modelling, and visualisation of related documents. The graph-based approach ensures an improved understanding of linked information because the graph structure offers a first-hand tool to represent and model the underlying data properties [8].

The system assumes a moderate level of correctness in the close-to-real-time knowledge extraction from scanned documents, particularly cross-reference data. For instance, ambiguous data from OCR processing introduces errors that should be handled accordingly.

This paper is structured as follows: Section 2 discusses related work. Section 3 explores user requirements and GraphLED design choices. The system architecture is discussed in Section 4. Finally, sections 5 and 6 present the validation results and draw the paper's conclusions.

2 Related Work

Engineering databooks are complex documents. The user's acceptable level of data correctness associated with the knowledge extraction remains an open challenge, particularly considering the output quality obtained from the available OCR tools (e.g. Tesseract, Google OCR). The issue is beyond the OCR tools, where low-quality scanning of documents remains the norm in the industry. Conventional approaches that handle structured data are not very robust in this case, as noisy data in document table cells lead to limited relevant knowledge [4] [5].

Several applications use graph-based analysis to extract knowledge and visualise the results. For example, [9] seeks to create public awareness of UN members voting patterns. The approach uses social knowledge graphs and similarity measures to analyse the publicly available information on the member votes. Because graphs lend themselves to intuitive visualisation, this work explored visual aids from the data. However, such an approach is less robust to data noise since stages of acquisition, treatment, and handling of the data require a prior pre-processed and ready-for-use dataset. Linked engineering AEC documents establish connections (e.g. purchase order number, quality certificate ID) that can be followed and analysed through automated software tools [10].

The graph-based analysis explores algorithms on graph data structures [11]. For instance, the graph-based document representation system (CGDR) provides classification reports intended for document forensic analysis [12]. A total of 1500 files were used in the study to classify 16 types of reports. The method structures the information as graph nodes and uses hierarchical text classification to set the reading order. However, the proposed model is not the ideal solution as it still depends on human experts to identify the useful features for each type of report.

A more automated and adaptive approach could benefit the application in many ways. Also, the data should be inputted in an advanced processing stage (e.g. augmented, cleaned) for the system to work. The work in [13] proposes a graph-based method for inductive text classification in documents.

The system builds individual graphs for each document fed into a Graph Neural Network (GNN) that learns fine-grained word representations based on their local structures. The proposed approach can effectively produce embeddings for unseen words in new documents. However, the noisy data limit the robustness of the proposed work. [14] introduces knowledge graphs and ontologies to cope with the complex network of decisions in construction documents. Other related work explores methods and algorithms in automated analysis of engineering drawings, commonly used in the oil and gas engineering industry.

Authors in [15] acknowledge the document quality as being the backbone of the problem, given that a poor-quality drawing (e.g. a paper that has been scanned numerous times) is virtually impossible to digitise unless a robust set of pre-processing methods are used.

The proposed work addressed the problem using a deep neural network with satisfactory improvements. However, the approach remains model-centric (i.e. model hyperparameter tuning) with little effort in improving the raw data quality through pre-processing, cleaning, augmentation or re-shaping. Existing work explores to a great extent digitalisation of documents through OCR (Optical character recognition) based techniques [16] and [17]. The benefits of the links between documents to unveil very relevant patterns in knowledge extraction are not sufficiently explored in the previous work.

The prior art acknowledges the challenge of achieving a satisfactory level of data correctness regarding close-to-real-time knowledge extraction from scanned documents, particularly cross-reference data. Data noise propagates throughout the knowledge extraction processing pipeline to visualisation and user engagement.

The previous work primarily addresses the correctness problem using human-intensive correction [4] [5] or, more recently, using model-centric deep neural networks [15], where the job is to fine-tune model hyperparameters as opposed to improve input data quality. This paper advances the state-of-the-art as follows:

- Proposes and validates data cleaning, and augmentation techniques (i.e. data-centric methodology) in noisy OCR scanned documents to overcome the so-called *garbage in, garbage out* error propagation.
- Proposes a system architecture to provide a large scale graph-based tool for close-to-real-time knowledge extraction and representation, assisting users in the difficult task of sifting through hundreds to thousand of documents. The system validation focuses on the performance and scalability issues.

3 System Design Choices

The proposed work is part of a larger technology-driven project that aims at digitalising industrial documents (OCR-based) grouped in databooks units. The work realises the vision of building digital twins tailored for the Oil&Gas industry to benefit users whose roles include inspection of documentation of materials, equipment, construction processes and quality control as part of the IT/OT convergence trend. A series of meetings and workshops were held with AEC documentation inspectors in 2021 that elicited the following list of user requirements:

- Ability to upload and extract key information from a large quantity of scanned databooks through OCR-based systems. The expected correctness level is minimal ambiguity in the knowledge-extracted data.
- Capability to visualise, manipulate and analyse databooks through intuitive and easy-to-use interfaces.
- Carry out usual inspection tasks (Completeness and Conformance) in an automated fashion.
- Assert that specific information extracted from a databook complies with existing technical specifications and standards (e.g. ASTM B16).
- Discover where and how the product and process were made (traceability, anomaly detection).

3.1 Databooks

The data space of interest comprises technical documents that make references to other documents. For instance, an equipment purchase order document can have relationships with complementary documents describing the chemical and physical properties of the materials used.

Another example is the collection of information that documents the construction process of offshore oil rigs and platforms. In this context, a databook is a group of related documents covering relevant aspects of a specific product or process cycle, from material production, batch production, product conformance validation and purchase order description.

For instance, the purchase of a bolt for a high-pressure gas pipe is described in a databook. A series of documents specify the bolt part's key aspects, including the manufacturing process, material properties, ultra-sound testing, and visual, and dimensional characteristics.

3.2 Design Choices

The system should check the completeness and compliance of databooks with relevant standards, databook traceability information and anomaly detection in case of data inconsistency and potential fraud. Non-functional requirements include low-latency operation, real-time performance, data coherence and integrity.

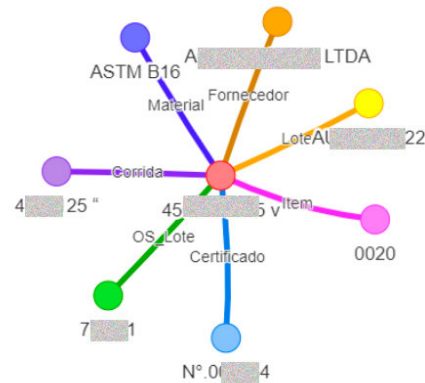


Figure 1: Complete Graph: documents linked through topics (e.g. batch number OS_LOTE).

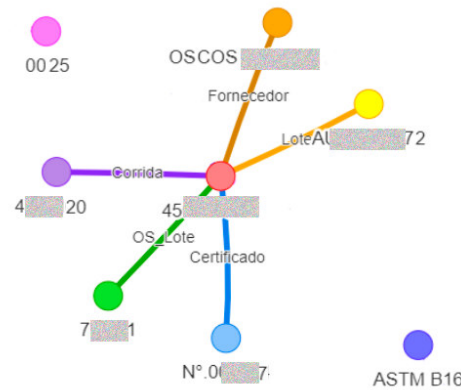


Figure 2: Incomplete Graph: databook with missing links.

3.2.1 Completeness and conformance

Inspecting completeness comprises checking whether all required documents are readily available in the databook. In addition, conformance to standards and guidelines is carried out to check several compliance rules such as material properties thresholds, dimensional specifications and visual inspection.

3.2.2 Traceability

This system functionality ensures that a product or process has records that prove the manufacturing path to its inception. The traceability inspection searches all databook documents referenced in the dataset to analyse the linked and source data compatibility. In addition, incomplete databooks or non-conformant documents should also be flagged as faults in the traceability verification.

3.2.3 Data Coherence

Data inconsistency is inevitable as the documents are scanned, OCR (Optical Character Recognition) processed to extract relevant text, and structured in an automated way. For instance, word ambiguities, either because of language variability or OCR quality issues, challenge the proposed system where the same data element has different meanings. In this case, the system should strive to maintain consistency and coherence at all costs.

3.2.4 Linked Documents

Graph-based structures and models provide a design choice to specify data elements and relationships of technical documents. Also, several graph algorithms can be explored to fulfil the document verification needs, including

completeness, conformance and traceability. For instance, centrality techniques traverse a graph to compute such properties as popular elements and bottlenecks.

The visual appeal of graph-based data representation offers a more intuitive data exploration to document inspectors. Figs. 1 and 2 shows an example of a databook sample, represented as a graph, extracted from a real-world dataset (Oil&Gas industry, identifiers have been crossed-out). In this case, a graph $G(V, E)$ comprises a set of vertices V (databook individual documents) and a set of edges E that interconnects the vertices. Fig. 1 shows an example of a graph with a complete databook, given that there is a path connecting all documents through the central node. In contrast, Fig. 2 presents an incomplete graph where documents are missing, and two documents are isolated, thus compromising the intra-databook traceability. It is important to note that these documents relate to other documents through specific relationships (e.g. same quality assurance laboratory).

4 The GraphLED System

The GraphLED system architecture (as shown in Fig. 3) is modularised to provide interoperability in the implementation of the pipeline functions from data insertion and knowledge extraction to graph visualisation and processing.

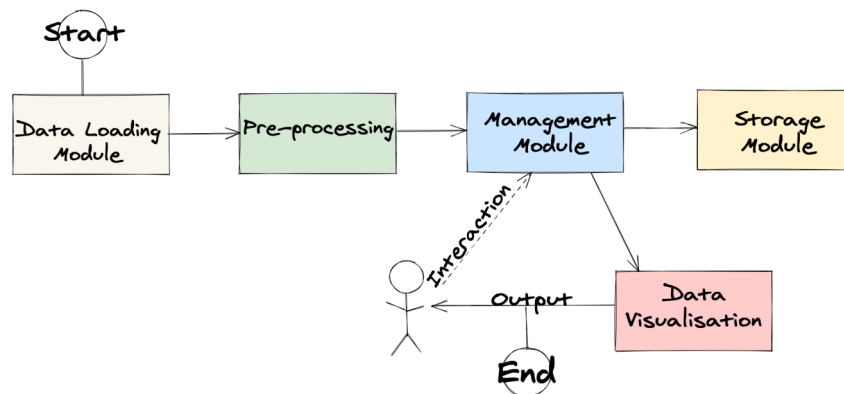


Figure 3: System architecture modules. The GraphLED explores a colourful visual representation using bright colours (i.e. analogy to LEDs)

4.1 Data Loading Module

OCR tools (e.g. Tesseract, Google image reader) convert scanned documents into raw text under a certain quality expectation. To minimise the OCR-related errors, GraphLED adopts the steps proposed in [18], where the generated raw text data is pre-processed using noise reduction, binarisation and skew reduction.

The outputted document is fed into a *form understanding* process that classifies documents using a form-based layout similarity scheme of document models previously defined. This module output is a JSON file containing each document represented as a set of blocks with the extracted words, their bounding boxes, and whether there is a link to a piece of data in another document.

4.2 Pre-Processing

Data is inherently noisy in this application scenario. This module handles the data cleaning and wrangling to improve data quality by transforming the input *dataset* into an intermediate representation suitable for the graph generation. Pre-processing tasks are performed at coarse-grained (documents) and fine-grained (document words) levels. A summary of the challenges is as follows: (1) Data ambiguity: Data is represented differently, but it semantically represents the same information; (2) Irrelevant data: data with small intrinsic value for a graph-based analysis and (3) Over-relevant data: identifying the more likely data to become nodes or edges in the graph generation phase.

A sequence of operations is applied to the data to decrease noise levels. For instance, it is crucial to filter out *Stopwords* prior to using disambiguation functions. Disambiguation works through similarity comparisons between data instances, using the following filters:

- Levenshtein filter [19] relies on the edit distance between two strings of comparable size (same proportion) to minimise false positives. A low acceptance threshold is also used to detect very similar instances.
- The Longest Common Subsequence (LCS) filter [20] detects similarity through the longest common subsequence (in the exact relative order) between any two strings.
- Sequence Matcher filter (LCS variation) uses junk-free contiguous subsequences as a similarity metric.

Fig.4 presents the pipeline for the ambiguity correction sub-module, where filters are applied one after another, covering the primary forms of ambiguity found in the dataset. Another module maps the modifications performed in the filtering steps to instances of the original document for data provenance purposes. It is essential to keep track of input data, transformed data and processes, thus providing a historical record of the data and its origins.

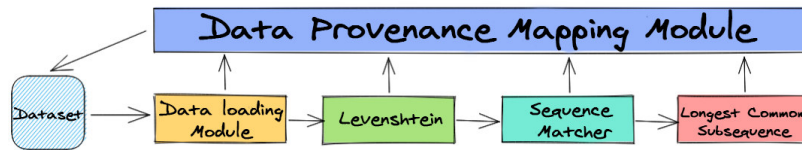


Figure 4: Proposed ambiguity module architecture.

With a large amount of data, exploiting helpful information can be daunting. Nevertheless, once we decrease the noise with filtered data, graph-based algorithms can attempt to spot patterns to detect relevant information based on past data behaviours. The detection of relevant data nodes exploits graph centrality algorithms for trend identification. Such algorithms enable the monitoring of a set of indicators: (a) the transitive influence of nodes (Eigenvector Centrality), (b) node influence on the flow of the graph (Betweenness Centrality) and (c) the number of incoming and outgoing relationships from a node (Degree Centrality).

The criterion for defining the relevant features is the aggregate sum of the scores produced by the metrics above.

4.3 Management Module

This module deals with system integration and its proper operation. It exchanges information through RestFUL APIs (Fig. 5), namely *graph-api*, that allows *query* creation considering the user parameters informed via HTTP requests.

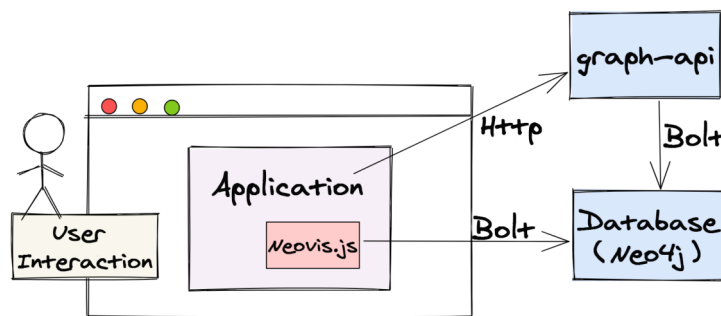


Figure 5: Management Module. Data is exchanged with the Neo4j graph database through the Bolt protocol

4.4 Storage Module

Graph-based storage and query engine are essential functions in the GraphLED system. The following criteria were used for the selection of a suitable graph database system:

- Maturity and availability of documentation and support from the developer team and community.

- Easy integration with other existing tools and provision of programming language integration (e.g. Python, Java and JavaScript).
- Satisfactory performance in moderate data volume.
- Easy-to-use, understandable and maintainable graph query language.
- Schema-free: flexibility and agility in structuring graphs.

Unlike traditional databases (SQL-like), graph-oriented databases offer a few advantages, but the primary one is the optimisation of breadth-first searches, which is a highly recommended procedure to find non-obvious relationships between entities. A few choices were considered, including cloud-based Amazon Neptune, Azure CosmosDB and community-based Neo4j. The latter fulfilled the project requirements at the time. Also, the upcoming query language standards (Apache Gremlin and SparkQL) can soon make the GraphLED system graph database-agnostic.

4.5 Data Visualisation

Graphs lend themselves to intuitive data visualisation. This module uses open-source toolkits, particularly the Neovis library, to render graph nodes and edges that model databooks, associated documents, and their relationships. In addition, the GraphLED visual tool interacts directly with the Bolt protocol's graph database for improved performance. However, it has been noted that graph visualisation can be a bottleneck as the number of graph nodes increases.

The designed user interfaces mitigate potential bottlenecks through (i) simple searches aimed at node-edge-node traversals, in which the user can specify the nodes' classes.

The system offers (ii) advanced *queries* in *Cypher* language, returning to the user complex and elaborated results. Properties calculated (iii), such as degree centrality, are also presented in a table format at the bottom of the GUI.

4.6 User Interface

The system *beta* version (Fig. 6) offers aesthetic and improved user experience. Customised GUI elements were added, allowing users to choose between the various types of engineering documents, and create, visualise and delete databook/document graphs.

An exploration form field has been placed in the side menu so users can write complex graph traversal searches, choosing between nodes and connections types. In addition, a simple search box was added to complement node centrality information, where users can choose the following properties: *Betweenness*, *Node Degree*, *Closeness* and *Eigenvector centrality*. The centrality provides hints to the user on structural anomalies, such as incomplete graphs, missing relationships, and others. Given that databook's documents are modelled as graphs, the centrality properties answer the questions of completeness, conformance, and traceability.

5 System Validation Results

5.1 Data Quality

The validation is aimed at understanding and further improving the input data quality.

The data pipeline implemented takes in a scanned document and passes it through an OCR stage. Then the OCR outputted file is classified and stored. The validation dataset used 81 PDF scanned databook files, totalling *1.43 GB* of individual documents. It is important to note that a databook is a set of engineering-driven documents with various purposes ranging from purchase order specifications to material testing certificates.

The benchmarking employed two document categories, labelled as *easy document* and *difficult document*. The labels indicate how difficult it is for the Tesseract OCR system to process scanned documents considering visual and document structural aspects such as page inclination, scrawls and complex internal elements (e.g. tables with few lines, few pages).

Further validation experiments were undertaken on a small set of labelled data to analyze the performance of data noise filtering and disambiguation techniques (Pre-processing module). In addition, many recognized data entities in the graph have been used as a comparison metric because it conveys the amount of duplicate/ambiguous nodes in the dataset. The data comprises parts suppliers to oil platforms (the dataset has 226 rows). Specialists analyzed the dataset and found 17 distinct suppliers (out of 226). However, the graph generated where each supplier is a node contained 128 different suppliers.

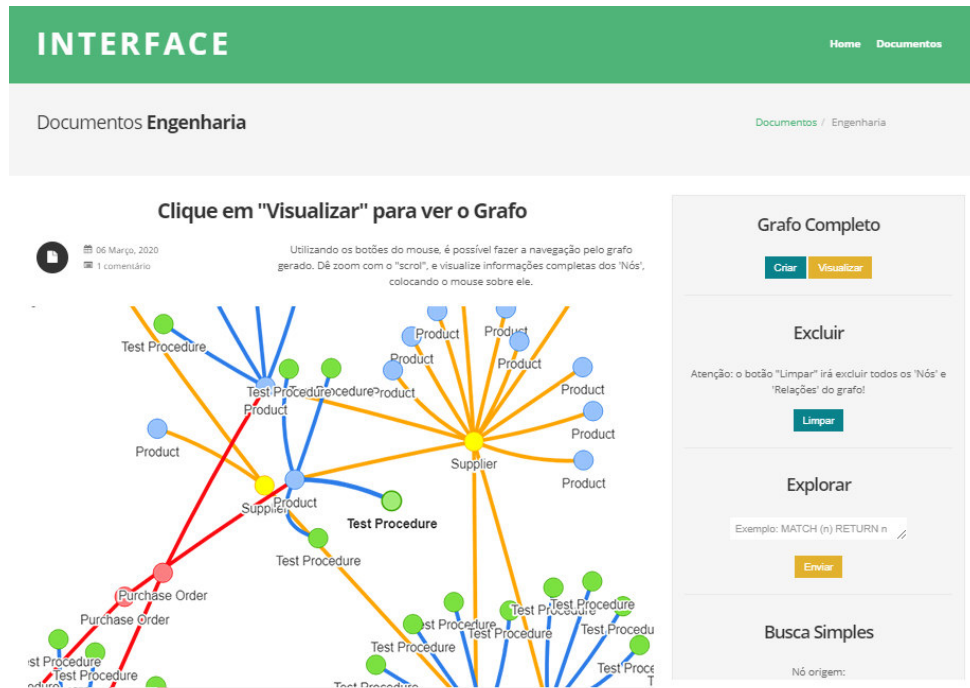


Figure 6: Beta version.

Further analysis found that such a discrepancy was due to ambiguous nodes in the graph, caused by 128 grammatical variations of the 17 suppliers. Some examples of variations include the terms: "supplier-A": "A-supplier", "supplier-A.inc", "group-A-supplier". The key idea for dealing with this issue is to identify variations of the same entity (a supplier) in the dataset and standardize them in a single form of writing. The ambiguity removal algorithm was developed to accomplish this task. The algorithm produced a list of 18 distinct suppliers, just one supplier off the specialist's list, as is summarised in Figure 7. As a result, the algorithm removed 110 ambiguous nodes in the graph - total removal of 99.09% of ambiguous nodes and an 85.93% reduction in the number of nodes from 128 to 18 suppliers.

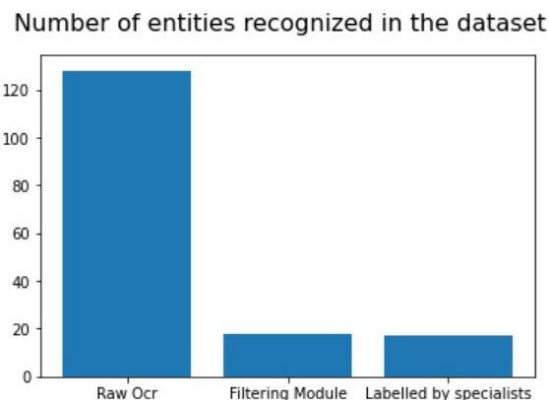


Figure 7: Comparison between recognized entities by the OCR system, after the filter, and by specialists.

The results were classified considering the similarity (OCR text versus ground-truth original text) into: *100% Accuracy* (total hit), *Partially accurate* (partial hit) and *Inconsistency* for lack of any relation between the OCR and ground-truth text. In the *easy document* context, as shown in Fig. 8, the system managed to identify 85.9% of the data elements inside the document with total hits, 12.72% as partial hits and 1.58% as inconsistent (total error). However, the results for the *difficult document* were not satisfactory: 25.67% of total hits, 24.32% of partial hits and 50.01% of inconsistencies.

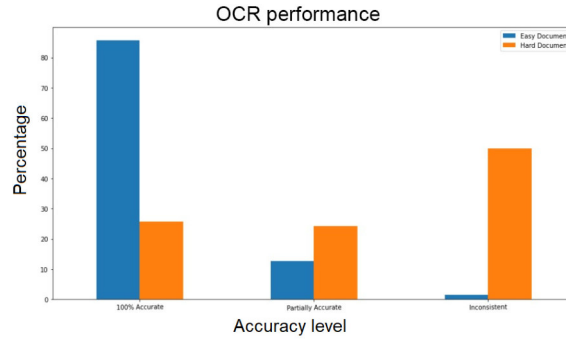


Figure 8: OCR performance: easy and difficult documents.

It is clear from these results that noise and ambiguity in the data propagate through the OCR-processed documents. Such errors are not helpful for graph construction from the OCR data.

More complex forms of ambiguity, such as entity standardization and acronym identification, were handled manually and will be addressed in an automated way in the next release of the GraphLED system.

5.2 Scaling the GraphLED

As a quantitative reference, an analysis of the Read/Write rate of graph nodes, edges and properties was carried out in the Neo4j database. We chose the *graph-workload* [21], an open-source tool for running loading tests, which is capable of generating random loads of data and Cypher queries to create graph nodes, edges and properties in a configurable and scalable way.

Table 1 lists the graph database setup parameters, including a Heap Initial Size value equal to 4GB of DDR4 RAM, limited by a Heap Max Size of 8GB.

The experiment parameters used are: $n = 1000$, where “n” is the target number of queries in the database, and $concurrency = 10$ is the number of queries executed in parallel. The data used are generated randomly (at runtime) or deterministically (hard coded), having no relation with an input dataset. The selected subgraph patterns, e.g. the Hub-Spoke (Figure 1) and the N-Ary Tree pattern, are also observed in structures of operational industry documents.

Table 1: Graph Database Setup Configuration.

Setting	
Heap Initial Size	4 GB
Heap Max Size	8 GB
Page Cache Size	4 GB
Neo4j Version	3.5.14

The complete graph could be generated with 143,602 nodes and 87,427 edges, with an approximate execution time of 42286 ms (Table 2). Figure 11 shows the rendering of a graph sample limited to 10,000 nodes. We observe the following graph patterns and operations:

- Hub-Spoke Structural Patterns can be spotted where a central node connects a "leaf" in a shape that resembles a dandelion. This type of node is represented in yellow (Figure 9).
- N-Ary Tree Pattern, with nodes in red representing an unbalanced N-Ary tree structure, where the large level of the sub-graph gives the main characteristic, are shown in red (Figure 10).
- Large set of 1-on-1 single linked structures to populate the graph where nodes are depicted in blue, pink and green in Figures 10 and 11). In addition, some of such sub-graphs could exhibit other properties: a large number of indexable attributes (Index Heavy), sub-graphs of nodes with a large number of characters (Fat Node Append), sub-graphs with nodes connected through random links (Random Linkage), sub-graphs with both random nodes and links (Raw Write) and sub-graphs created from a MERGE operation.

- Other types of operations (Aggregate Read, Long Path Read and Random Access Read) are allowed as Read queries directly performed to the database. Such operations were included only to simulate a production environment with real-world users when both Write and Read operations can coincide.

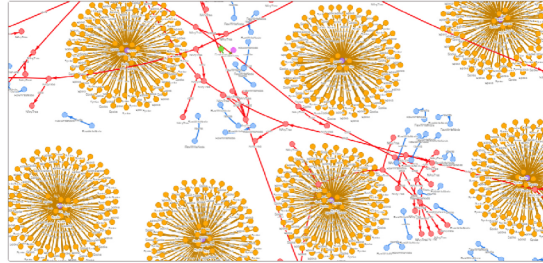


Figure 9: Hub-Spoke Subgraph Pattern.



Figure 10: N-Ary Tree Subgraph Pattern.

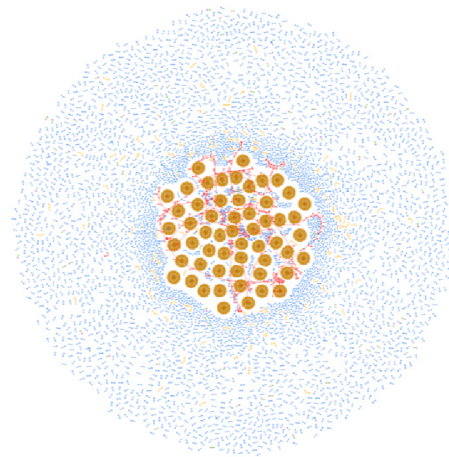


Figure 11: Subgraph Sample with Ten Thousand Nodes.

6 Conclusions

The GraphLED system addresses the needs of Oil&Gas inspection in *big data* engineering contexts.

This work contributes to a data-centric mechanism to address the correctness level in knowledge extracted from noisy OCR-based documents. The proposed data cleaning and augmentation achieved satisfactory results in removing ambiguity in the data.

The GraphLED system offers a graph-based tool for close-to-real-time knowledge extraction and representation.

Table 2: Graph Database Running Results.

Pattern	RUNS	AVG	MIN	MAX
Fat Node Append	1050	73.03	18	1990
NArY Tree	1003	54.72	4	1993
Merge Write	985	38.24	4	1991
Random Linkage	1010	31.90	5	249
Hub-Spoke	481	60.74	20	719
RawWrite	2933	41.51	3	1754
Index Heavy	1002	41.10	4	1960
Aggregate Read	552	18.25	2	126
Random Access Read	952	37.44	7	114
Long Path Read	62	24.66	2	127
Total (units)	10030	-	-	-
Average (ms)	-	42.16	6.9	1102.3

References

- [1] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE industrial electronics magazine*, 11(1):17–27, 2017.
- [2] Robert Amor and M Clift. Documents as an enabling mechanism for concurrent engineering in the construction industry. In *1st international conference on Concurrent Engineering in Construction, CEC*, volume 97, 1997.
- [3] Pai Zheng, Chun-Hsien Chen, and Suiyue Shang. Towards an automatic engineering change management in smart product-service systems—a dsm-based learning approach. *Advanced Engineering Informatics*, 39:203–213, 2019.
- [4] Kei Matsubayashi, Akihiro Yamashita, Hidetoshi Nonaka, and Yohko Konno. A research on document summarization and presentation system based on feature word extraction from stored informations. In *2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 60–63. IEEE, 2018.
- [5] Qing Guan, Fuli Zhang, and Enli Zhang. Application prospect of knowledge graph technology in knowledge management of oil and gas exploration and development. In *2019 2nd International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 161–166. IEEE, 2019.
- [6] Christian M Dahl and Emil Nørmark Sørensen. Document digitization and machine learning. 2019.
- [7] Eduard Hovy, Roberto Navigli, and Simone Paolo Ponzetto. Collaboratively built semi-structured content and artificial intelligence: The story so far. *Artificial Intelligence*, 194:2–27, 2013.
- [8] Elena Simperl, Oscar Corcho, Marko Grobelnik, Dumitru Roman, Ahmet Soylu, María Jesús Fernández Ruíz, Stefano Gatti, Chris Taggart, Urška Skok Klima, Annie Ferrari Uliana, Ian Makgill, and Till Christopher Lech. Towards a knowledge graph based platform for public procurement. In Emmanouel Garoufallou, Fabio Sartori, Rania Siatry, and Marios Zervas, editors, *Metadata and Semantic Research*, pages 317–323, Cham, 2019. Springer International Publishing.
- [9] Manjai Lee and Byung-Won On. Social graph visualization techniques for public data. *Journal of the HCI Society of Korea*, 10(1):5–17, 2015.
- [10] Masaki Eto. Extended co-citation search: Graph-based document retrieval on a co-citation network containing citation context information. *Information Processing & Management*, 56(6):102046, 2019.
- [11] Trevor Strohman, W Bruce Croft, and David Jensen. Recommending citations for academic papers. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 705–706, 2007.
- [12] Ghulam Mujtaba, Liyana Shuib, Ram Gopal Raj, Retnagowri Rajandram, Khairunisa Shaikh, and Mohammed Ali Al-Garadi. Classification of forensic autopsy reports through conceptual graph-based document representation model. *Journal of biomedical informatics*, 82:88–105, 2018.
- [13] Yufeng Zhang, Xueli Yu, Zeyu Cui, Shu Wu, Zhongzhen Wen, and Liang Wang. Every document owns its structure: Inductive text classification via graph neural networks. *arXiv preprint arXiv:2004.13826*, 2020.
- [14] Mads Holten Rasmussen, Maxime Lefrançois, Pieter Pauwels, Christian Anker Hviid, and Jan Karlshøj. Managing interrelated project information in aec knowledge graphs. *Automation in Construction*, 108:102956, 2019.

- [15] Carlos Francisco Moreno-García and Eyad Elyan. Digitisation of assets from the oil and gas industry: Challenges and opportunities. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, volume 7, pages 2–5, 2019.
- [16] Carlos Francisco Moreno-García, Eyad Elyan, and Chrisina Jayne. New trends on digitisation of complex engineering drawings. *Neural computing and applications*, 31(6):1695–1712, 2019.
- [17] Carlos Francisco Moreno-García, Xavier Cortés, and Francesc Serratosa. A graph repository for learning error-tolerant graph matching. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 519–529. Springer, 2016.
- [18] Gabriel L Santos, Vanessa T Silva, Laura A Dalmolin, Ricardo N Rodrigues, Paulo LJ Drews, and Nelson L Duarte Filho. A form understanding approach to printed and structured engineering documentation. In *2021 34th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 330–337. IEEE, 2021.
- [19] Rishin Haldar and Debajyoti Mukhopadhyay. Levenshtein distance technique in dictionary lookup methods: An improved approach. *arXiv preprint arXiv:1101.1232*, 2011.
- [20] Jörg Tiedemann. Automatic construction of weighted string similarity measures. In *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.
- [21] David Allen. Graph workload. <https://github.com/moxious/graph-workload>, 2019.

8 ARTICLE II - EXPLORING THE IMPACT OF EMBEDDING TECHNIQUES ON GRAPH ANOMALY DETECTION: A COMPARATIVE STUDY

This earlier paper reflects the author's early work on this project and serves as a valuable precursor to the present work. The present work builds upon the insights gained from the earlier study and expands upon its findings to provide a more comprehensive evaluation of graph embedding quality and comparison. Specifically, this work highlights key indicators that are discussed in detail to evaluate and compare models. These indicators offer a practical and effective means of assessing the performance of graph embedding techniques, and they can be used to inform the development of more robust and accurate models.

Exploring the Impact of Embedding Techniques on Graphs: A Comparative Study

Willian Borges de Lemos,
 Marcelo Rita Pias,
 Silvia Silva da Costa Botelho
 Computer Science Center (C3)
 Federal University of Rio Grande (FURG)
 Rio Grande, Brazil

Abstract—The exponential growth of data has presented both opportunities and challenges for organizations and individuals seeking to extract meaningful insights and patterns. In recent years, graph embedding has emerged as a promising method to bridge the gap between graph data and machine learning algorithms. Graph embedding allows for efficient processing and analysis of graph structures by representing graph data in a low-dimensional space. However, the graph embedding field remains fragmented, lacking a consensual theoretical understanding and a unified framework to describe and compare different techniques. This paper addresses these issues by providing a comprehensive overview of the current state of the art in graph embedding. The paper develops a unified framework for describing and comparing various graph embedding techniques, highlighting the strengths and weaknesses of each approach on common factors.

Index Terms—graph-based processing, Graph Neural Networks, Graph embedding, Random Walk, Matrix Factorization.

I. INTRODUCTION

In recent years, the growth of data has been exponential, with vast amounts of information being generated and stored in various formats, including structured and unstructured data [1], [2]. This explosion of data has presented both opportunities and challenges for organizations and individuals seeking to extract meaningful insights and to identify patterns and anomalies [3].

One key challenge in the field of machine learning and data mining is analyzing complex data structures like graphs, which have a rich topology and can be challenging to analyze using traditional machine learning algorithms. Graphs have emerged as an increasingly popular way to represent and analyze a wide variety of systems, including social networks, biological networks, and transportation networks. By representing the structure of these systems as nodes and edges, graphs enable the modeling of complex relationships to represent and analyze complex systems [4]. However, graphs have a complex structure that makes applying traditional machine learning algorithms challenging. This is where graph embedding comes in as a necessary step to bridge the gap between graph data and machine learning algorithms.

The goal of graph embedding is to learn low-dimensional representations of graph structures, allowing for efficient processing and analysis [5]. Despite its importance, the field of

graph embedding remains fragmented, with distinct subfields within the machine learning and data mining communities that may not have a clear connection to one another [6]. Current approaches to graph embedding are frequently evaluated using diverse benchmarks that emphasize different aspects of graph structures. As a result, it can be challenging to compare and understand the strengths and weaknesses of different approaches, hindering the development of a broader understanding. Furthermore, there is a lack of a consensual theoretical understanding and a unified framework to describe and compare the different techniques [7].

This survey aims to address these issues by providing a comprehensive overview of the current state of the art in graph embedding. Our goal is to develop a unified framework for describing and comparing the various techniques, and to highlight the strengths and weaknesses of each approach on common factors. We also discuss a number of important applications and future directions for work in this field. By offering a comprehensive view of graph embedding, we hope to facilitate deeper understanding and further progress in this crucial area of machine learning and data mining.

II. RELATED WORK

The field of graph embedding has seen a surge in interest in recent years, leading to numerous attempts to survey the existing methods. [6] reviews the various approaches to embed individual nodes and entire (sub)graphs, but does not compare their strengths and weaknesses or identify common factors related to this field. [8] focuses exclusively on knowledge graph embedding, while [7] provides a detailed analysis of graph embedding techniques and proposes a taxonomy based on problem settings. However, this analysis is limited to the problem setting based on inputs and outputs, and does not provide a comprehensive comparison of the strengths and weaknesses of different techniques on common factors. Despite the notable contributions of these works, there still exists a lack of a consensual theoretical understanding in the field of graph embedding. The development of consistent theoretical frameworks is crucial in order to guide future innovations and progress in the field. In this work, we provide a comprehensive overview of the field of graph embedding. By reviewing a diverse range of models, we aim to identify

common characteristics and connect different approaches to deepen our understanding of this field. The goal of this work is to build a bridge between different methods and contribute to the development of a unified understanding of the field.

III. GRAPH EMBEDDING

Recently, graph embedding has emerged as a promising approach for graph mining and representation learning. Graph embedding is a technique that maps the nodes and edges of a graph to a low-dimensional vector space, preserving the structural and semantic information of the graph. By leveraging the properties of the vector space, graph embedding can be used to perform various tasks, such as graph classification, link prediction, and anomaly detection. These techniques can be broadly classified into two categories: Shallow and Deep. Shallow graph embedding techniques are based on matrix factorization and graph algorithms such as Random walks. These methods are efficient in small-medium size graphs, but they may not capture the complex relationships and structure of graphs well. Examples of shallow graph embedding techniques include Laplacian Eigenmaps [9], Locally Linear Embedding (LLE) [10], and Graph Factorization [11]. In contrast, deep graph embedding techniques leverage the power of deep learning and neural networks to accurately capture the complex relationships and structure of graphs. Although they require more computational resources, they provide a higher level of accuracy compared to shallow embedding techniques. Examples of deep graph embedding techniques include Graph Convolutional Networks (GCN) [5], Graph Attention Networks (GAT) [12], and Graph Autoencoders [13].

A. Matrix Factorization

Matrix factorization has risen as a popular tool in meeting this demand, providing an efficient way to extract meaningful information from large datasets.

Graph Laplacian Eigenmaps (GLE), introduced by [9], is a method that maps nodes in a graph to a low-dimensional representation using the eigenvectors of the graph Laplacian matrix. The Laplacian matrix is a graph representation that encodes the local and global structure of the graph. By computing the eigenvectors of the Laplacian matrix, GLE can obtain a mapping that preserves the local structure of the graph. However, their complexity grows at least proportionally to the square of the number of nodes, making them unsuitable for processing large-scale networks efficiently.

Node Proximity Matrix Factorization (NPMF), introduced by [14], on the other hand, factorizes the node proximity matrix of a graph to obtain node embeddings. The node proximity matrix encodes the similarity between nodes in the graph, aiming to find a low-dimensional representation of the nodes that preserves this similarity. Unlike GLE, Node Proximity Matrix Factorization models can capture both local and global structure of the graph. In General, Node Proximity Matrix Factorization is considered more flexible in terms of

the types of graphs it can handle compared to Graph Laplacian Eigenmaps.

B. Deep Learning

With the rise of deep learning, graph embedding has been an active area of research, and numerous models have been proposed to perform this task. These models can be broadly categorized into two groups: random walk methods and non-random walk methods.

1) *Deep Learning with random walk*: The basic concept of random walk graph embedding is to capture the connectivity patterns of a graph through generating node sequences using random walks, then using these sequences to compute node embeddings that reflect the graph's structural information. DeepWalk, Node2vec, and LINE are popular examples of these shallow graph embedding techniques, known as Deep Learning graph embedding with Random walk. DeepWalk and Node2vec employ a decoder based on the inner product, similar to matrix factorization approaches. DeepWalk considers random walks as sentences and learns node representations through a skip-gram model, while Node2vec has a more flexible trade-off between local and global structure information by controlling random walks with a parameter that determines whether to prioritize breadth-first or depth-first sampling. LINE, in particular, extends upon the previous models by incorporating both local and global structure information more explicitly through factorizing first- and second-order similarities. Nevertheless, it still uses random walks as the foundation for generating node sequences and learning node embeddings, making it a natural extension of DeepWalk and node2vec. However, these methods are inherently transductive, meaning they can only generate embeddings for nodes that were present during the training phase. They cannot be used to generate embeddings for new nodes, making them less suitable for dynamic graphs or use cases where new nodes are frequently added.

2) *Deep Learning without random walk*: Autoencoders and Graph Convolutional Networks (GCN's) are two popular non-random walk methods for graph embedding. Both of these methods aim to capture the structural information of a graph, but they differ on the approach. An autoencoder consists of two components: an encoder that maps the input graph data to a lower-dimensional representation, and a decoder that maps the lower-dimensional representation back to the original data. The encoder and decoder are trained together to reconstruct the original data as accurately as possible, minimizing the reconstruction error of the output. The idea of adopting autoencoder for graph embedding is similar to node proximity matrix factorization in terms of neighborhood preservation, where the reconstruction process will make the nodes with similar neighborhood with similar embedding. The rise of autoencoders in graph embedding came from their ability to handle non-linear relationships in the graph data, their ability to learn hierarchical representations of graph data, and their ability to handle variable-sized input graphs. Additionally, autoencoders can be used to embed both node

and graph-level information in the same representation, which is useful for tasks that require information from both levels of abstraction. For example, the work [13], used a variational autoencoder for graph representation learning, and achieved promising results on a range of graph datasets. [15] also used autoencoders for graph embedding, achieving state-of-the-art performance on a variety of graph classification tasks. However, Autoencoders also can be extremely costly to train, and they are also strictly transductive.

Graph Convolutional Networks (GCN's), on the other hand, have been shown to be effective in capturing local and global structures in graphs, since it can encode structure information of the graphs by aggregating information from its local neighborhood. GCN's relies in parameter sharing and neighborhood aggregation, incorporating the graph structure directly into the encoder and leveraging node attributes. This allow GCN's to generate embeddings for nodes that were not present during the training phase. For example, the work by [5] showed that GCN's achieved state-of-the-art performance on several benchmark graph classification tasks. On the other hand, GCN's can be limited in their ability to learn complex non-linear relationships between nodes. This is where works such as the "Deep Graph Infomax" by [16] can prove useful, as they extend the basic GCN framework to allow for more expressive representations through deep architectures and unsupervised objectives.

IV. FRAMEWORK

In this work, we propose a framework for describing and comparing graph embedding models based on a careful analysis of the strengths and limitations of existing approaches. To arrive at the key categories of evaluation, we conducted a thorough literature review of recent advances in graph embedding and identified the most commonly used evaluation criteria across different studies. We aimed to identify the key factors essential for assessing the effectiveness of graph embedding models and to develop a framework that comprehensively understands the strengths and limitations of different approaches. After analysing the existing literature, we identified six key categories that are essential for evaluating graph embedding models:

- 1) preserving local structure,
- 2) preserving global structure,
- 3) capturing complex relationships,
- 4) flexibility,
- 5) capturing long-range relationships, and
- 6) scalability to large graphs.

These categories were chosen based on their importance in various graph-related tasks, such as link prediction, community detection, network visualisation, and classification. By evaluating graph embedding models based on these categories, we aim to provide a comprehensive understanding of the strengths and limitations of different approaches and their tradeoffs.

A. The Keys

Local structure (1) preservation is vital for task prediction and community detection, while preserving global structure (2) is better suited for network visualisation and classification problems.

Complex relationships (3) between nodes, such as non-linear and higher-order relationships, are essential in knowledge graph completion and relational learning. However, capturing such relationships may come at the cost of computational efficiency and scalability to large graphs.

Flexibility (4) is a consideration for dynamic graph structures, such as social networks and web graphs. Models that can quickly adapt to changing relationships between nodes may not be as computationally efficient as models optimised for specific graph structures.

Long-range relationships(5) are important for link prediction and recommendation systems but may not capture fine-grained local relationships.

Scalability (6) to large graphs is helpful for big data applications and real-time systems. However, computational efficiency may come at the cost of capturing complex relationships or higher-level structures in the graph.

Table I presents a comparative study based on extensive research on several papers and studies, evaluating the performance of Matrix Factorization (MF), Random Walk (RW), and Deep Learning without Random Walk (DL NO-RW) models in the six keys mentioned above. The table shows the performance of each model in six evaluation categories, with "Good" indicating high performance and "Limited" indicating relatively lower performance.

V. GAPS

One of the main gaps in graph embedding is the lack of standardised evaluation metrics for embedding quality and its impact on classification and mining task. Embedding methods, such as node2vec [20] and graph2vec [27], have been proposed to map nodes and graphs to a low-dimensional space, making it easier to detect anomalies. However, there has yet to be a consensus on evaluating the quality of these embeddings, and how they impact in graph mining performance [28].

The limited scalability of algorithms is another challenge in graph embedding. Many existing methods cannot handle large graphs with millions of nodes and edges, making them unsuitable for real-world applications. This gap should be addressed to make graph embedding more practical and useful [29].

Interpretability is also one challenge. Some graph embedding algorithms, such as deep learning-based methods, tend to generate representations that are hard to interpret.

VI. DISCUSSION

Potential solutions to the gap in scalability include the use of sampling techniques, approximations, and parallelisation, as well as the use of distributed computing frameworks, as well as the use of graph embedding techniques, which can map the nodes and edges of a graph into a low-dimensional vector

Category	1	2	3	4	5	6	References
MF	Good	Good	Limited	Limited	Limited	Limited	[17], [18], [19], [20], [21]
RW	Good	Good	Good	Limited	Limited	Limited	[22], [23], [6], [24], [25]
DL NO-RW	Good	Good	Good	Good	Limited	Limited	[7], [5], [12], [26], [7]

TABLE I: Comparison of Matrix Factorization (MF), Random Walk (RW), and Deep Learning without Random Walk (DL NO-RW) models in graph-based representation learning, evaluated on six key metrics: 1- preservation of the local structure, 2- preservation of the global structure, 3- handling of complex relationships, 4- flexibility, 5- preservation of long-range relationships, and 6- scalability.

space, making it easier to process and analyse the graph data [7]. Some papers that address this gap are [30], [31].

The development of a standardised set of metrics and tools to evaluate the performance of graph embedding algorithms would be beneficial in creating a sense of consensus across the field. This would enable researchers to compare the strengths and weaknesses of different algorithms and their tradeoffs and help guide the development of new algorithms and features.

Researchers can explore the space using interpretable models, such as decision trees or linear models, to make graph embedding algorithms more interpretable.

VII. CONCLUSIONS

Graph mining is a complex problem that can be approached using various methods, such as statistical, rule-based and machine learning methods. Machine learning approaches have shown promising results in identifying anomalies and patterns in graph data. However, there are still gaps in the representation learning and embedding techniques used in these methods. Recent studies have focused on evaluating the quality of graph embeddings and its impact on graph representation learning, proposing intrinsic and extrinsic evaluation methods, such as link prediction evaluation and classification performance evaluation respectively, as ways to evaluate the quality of the embedding. This paper provides a comprehensive overview of the current state of the art in graph embedding and highlights the important relationship between these two fields and their impact. Through an in-depth analysis of existing research, the paper identifies common characteristics and connects different approaches to graph embedding. The paper also presents a unified framework for describing and comparing various graph embedding techniques, allowing for a more consistent and thorough understanding of the field. This framework serves as a useful tool for researchers, practitioners, and industry professionals alike to navigate the rapidly-evolving landscape of graph embedding and data mining. The insights and perspectives provided in this paper are valuable for guiding future innovations and progress in this crucial area of machine learning and data mining.

REFERENCES

- [1] T. Hey, S. Tansley, and K. Tolle, "The data deluge: an e-science perspective," *International Journal of Grid and Distributed Computing (IJGDC)*, vol. 2, no. 3, pp. 1–15, 2009.
- [2] J. Leskovec and A. Krevl, "Graph data management: challenges and opportunities," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1265–1276, 2014.
- [3] G. George, M. R. Haas, and A. Pentland, "Big data and management," 2014.
- [4] X. Fan and J. Han, "Outlier detection on graphs: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–44, 2019.
- [5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
- [6] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.
- [7] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.
- [8] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.
- [9] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003.
- [10] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.
- [11] N. Ahmed and M. W. Mahoney, "Distributed graph factorization," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 567–575.
- [12] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.
- [13] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.
- [14] D. W. Cai, X. He, J. Han, and J. Ren, "Unsupervised deep embedding for clustering analysis," *arXiv preprint arXiv:1706.02263*, 2017.
- [15] W. Pan, G. Long, J. Liu, and Q. Yang, "Adversarially regularized graph autoencoder for graph embedding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, 2018, pp. 1175–1184.
- [16] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Deep graph infomax," *International Conference on Learning Representations*, 2019.
- [17] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM computing surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.
- [18] S. Zhang, L. Yao, Y. Tay, X. Xu, X. Zhang, and L. Zhu, "Metric factorization: Recommendation beyond matrix factorization," *arXiv preprint arXiv:1802.04606*, 2018.
- [19] X. He and T.-S. Chua, "Neural factorization machines for sparse predictive analytics," in *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2017, pp. 355–364.
- [20] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.
- [21] J. Demmel, I. Dumitriu, and O. Holtz, "Fast linear algebra is stable," *Numerische Mathematik*, vol. 108, no. 1, pp. 59–91, 2007.
- [22] 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*. ACM, 2014, pp. 701–710.
- [23] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

- [24] Z. Zhang, P. Cui, and W. Zhu, “Deep learning on graphs: A survey,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 249–270, 2020.
- [25] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *Advances in neural information processing systems*, vol. 30, 2017.
- [26] P. Cui, W. Wang, and W. Zhu, “A comprehensive survey on graph embedding: Problems, techniques, and applications,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.
- [27] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, “graph2vec: Learning distributed representations of graphs,” *arXiv preprint arXiv:1707.05005*, 2017.
- [28] Y. Zong, J. Chen, X. Wang, and H. Liu, “Anomaly detection on graphs with gated graph neural networks,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2215–2224.
- [29] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, “Graph convolutional neural networks for web-scale recommender systems,” in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 974–983.
- [30] A. Goyal, L. Akoglu, and H. Liu, “Scalable anomaly detection in graphs,” in *Proceedings of the VLDB Endowment*, vol. 11, no. 13. VLDB Endowment, 2018, pp. 2049–2060.
- [31] J. Pacheco, M. Farajtabar, S. Gharavi, and M. Bayati, “Scalable graph anomaly detection: A survey,” *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–37, 2020.