UNIVERSIDADE FEDERAL DE VIÇOSA

Beyond Node-Centric Graph Learning: Advancing Edge-Aware GNNs for Enhanced Representation Learning

Wagner de Almeida Júnior
Doctor Scientiae

VIÇOSA - MINAS GERAIS
2026

WAGNER DE ALMEIDA JÚNIOR

Beyond Node-Centric Graph Learning: Advancing Edge-Aware GNNs for Enhanced Representation Learning

Thesis submitted to the Computer Science Graduate Program of the Universidade Federal de Viçosa in partial fulfillment of the requirements for the degree of *Doctor Scientiae*.

Adviser: Alex Borges Vieira

VIÇOSA - MINAS GERAIS
2026

Ficha catalográfica elaborada pela Biblioteca Central da Universidade Federal de Viçosa - Campus Viçosa

T
A447b
2026

Almeida Junior, Wagner de, 1988-
Beyond node-centric graph learning: advancing edge-aware GNNs for enhanced representation learning / Wagner de Almeida Junior. – Viçosa, MG, 2026.
1 tese eletrônica (116 f.): il. (algumas color.).

Texto em inglês.
Orientador: Alex Borges Vieira.
Tese (doutorado) - Universidade Federal de Viçosa, Departamento de Informática, 2026.
Referências bibliográficas: f. 105-116.
DOI: https://doi.org/10.47328/ufvbbt.2026.209
Modo de acesso: World Wide Web.

1. Redes neurais (Computação). 2. Conectividade de grafos. 3. Aprendizado do computador. I. Vieira, Alex Borges, 1978-. II. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD 22. ed. 006.32

Bibliotecário(a) responsável: Euzébio Luiz Pinto CRB-6/3317

WAGNER DE ALMEIDA JÚNIOR

Beyond Node-Centric Graph Learning: Advancing Edge-Aware GNNs for Enhanced Representation Learning

Thesis submitted to the Computer Science Graduate Program of the Universidade Federal de Viçosa in partial fulfillment of the requirements for the degree of *Doctor Scientiae*.

APPROVED: January 23, 2026.

Assent:

Wagner de Almeida Júnior
Author

Alex Borges Vieira
Adviser

To my family, that grew in numbers and love throughout the arduous development of this work.

ACKNOWLEDGMENTS

This work has been sponsored by the following Brazilian research agencies: Coordination for the Improvement of Higher Education Personnel (CAPES; Financing code 001), Minas Gerais State Foundation for Research Aid (FAPEMIG) and National Council of Scientific and Technological Development (CNPq).

ABSTRACT

JÚNIOR, Wagner de Almeida, D.Sc., Universidade Federal de Viçosa, January, 2026. **Beyond Node-Centric Graph Learning: Advancing Edge-Aware GNNs for Enhanced Representation Learning**. Adviser: Alex Borges Vieira.

Graph Neural Networks (GNNs) have achieved state-of-the-art performance in representation learning, yet they predominantly treat edges as auxiliary structures for message passing, systematically overlooking their potential as primary informational entities. This thesis investigates the hypothesis that explicitly modeling edge attributes enhances predictive accuracy, particularly in systems governed by physical or logical flow constraints.

We first establish a novel Taxonomy of Edge-Aware Graph Learning, categorizing existing approaches from random walks to modern Graph Transformers. Building upon this theoretical foundation, we propose AttEAGNN (Attention-based Edge-Aware GNN), a dual-stream architecture that processes node and edge features in parallel. A key innovation of our model is a magnitude-based edge attention mechanism, which acts as a learned saliency filter to dynamically weight topological importance without the computational overhead of dual-graph transformations. Extensive experiments on real-world backbone networks (Abilene and RNP) demonstrate that AttEAGNN significantly outperforms traditional node-centric baselines (GCN, GraphSAGE) in node load prediction tasks, achieving R^2 scores of up to 0.92. Error analysis reveals that our edge-aware approach is particularly effective at minimizing Root Mean Squared Error (RMSE), indicating superior capability in detecting network congestion and traffic bursts. While results on standard citation benchmarks (CORA, CITESEER, PUBMED) show competitive but mixed gains, they reinforce our finding that edge-aware learning is most critical in domains with rich, meaningful edge attributes. Finally, we address reproducibility challenges by releasing a modular, open-source framework for edge-aware GNN research.

Keywords: graphs; graph neural networks; machine learning; edge-aware graph neural networks; graph analytics

RESUMO

JÚNIOR, Wagner de Almeida, D.Sc., Universidade Federal de Viçosa, janeiro de 2026. **Além do Paradigma Centrado em Nós no Aprendizado de Grafos: Avanços em GNNs Sensíveis às Arestas para o Aprendizado de Representações**. Orientador: Alex Borges Vieira.

As Redes Neurais em Grafos (GNNs) têm alcançado desempenho de ponta em aprendizado de representações, contudo, predominantemente tratam as arestas como estruturas auxiliares para passagem de mensagens, negligenciando sistematicamente seu potencial como entidades informacionais primárias. Esta tese investiga a hipótese de que a modelagem explícita de atributos das arestas aumenta a acurácia preditiva, especialmente em sistemas governados por restrições físicas ou lógicas de fluxo. Inicialmente, estabelecemos uma nova Taxonomia de Aprendizado em Grafos Sensível às Arestas, categorizando abordagens existentes desde caminhadas aleatórias até modernos Graph Transformers. Com base nessa fundamentação teórica, propomos o AttEAGNN (Attention-based Edge-Aware GNN), uma arquitetura de fluxo duplo que processa, em paralelo, características de nós e arestas. Uma inovação central do modelo é um mecanismo de atenção em arestas baseado em magnitude, que atua como um filtro de saliência aprendido para ponderar dinamicamente a importância topológica, sem o custo computacional associado a transformações de grafo duplo. Experimentos extensivos em redes backbone reais (Abilene e RNP) demonstram que o AttEAGNN supera significativamente baselines tradicionais centrados em nós (GCN e GraphSAGE) em tarefas de predição de carga de nós, alcançando valores de R^2 de até 0,92. A análise de erro revela que a abordagem sensível às arestas é particularmente eficaz na minimização do Erro Quadrático Médio da Raiz (RMSE), indicando maior capacidade na detecção de congestionamentos de rede e rajadas de tráfego. Embora os resultados em benchmarks clássicos de citações (CORA, CITESEER, PUBMED) apresentem ganhos competitivos, porém heterogêneos, eles reforçam a constatação de que o aprendizado sensível às arestas é mais crítico em domínios onde as arestas possuem atributos ricos e semanticamente relevantes. Por fim, abordamos desafios de reprodutibilidade ao disponibilizar um framework modular e de código aberto para pesquisa em GNNs sensíveis às arestas.

Palavras-chave: grafos; redes neurais de grafos; aprendizado de máquina; redes neurais de grafos cientes de arestas; análise de grafos

LIST OF FIGURES

LIST OF TABLES

CONTENTS

Chapter 1

Introduction

As we experience the age of big data, rapid increase in data generation, collection, and processing makes it challenging to analyze and extract information from ever growing datasets. Machine learning (ML) methods are commonly applied tools to tackle this problem. Deep neural networks such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs) can achieve state-of-the-art results in big data processing. However, these most traditional ML techniques are limited by design to work well with structured data, such as texts, audio, or images.

Nonetheless, much real-world data is non-structured, complex, and inherently interconnected [Barabási and Pósfai, 2016]. For instance, computer and social networks, transportation systems, biological interactions, and financial transactions are domains where the relationships between entities can be as crucial as the entities themselves. Graphs are complex, non-Euclidean data structures where nodes represent entities and edges represent relationships, commonly used to represent this type of information. Graph-structured data analysis can enable the modeling of complex, interdependent relationships within large-scale datasets, revealing patterns and insights that traditional tabular or linear representations often miss.

In this context, graph neural networks (GNNs) [Xu et al., 2018a] are deep neural networks designed to process graph-structured data. Unlike traditional deep neural networks, GNNs use a message-passing scheme that iteratively aggregates and transforms neighboring node features, capturing information from local and global graph structures.

Graph data, however, is not exclusive to nodes, as many datasets can contain rich information on edge attributes. The problem is that most GNN methods usually overlook these features. Some datasets, like backbone networks, include data on edges that represent links. Hence, a GNN method aware of edge features can outperform traditional methods when analyzing such datasets.

In fact, it has been demonstrated that considering edge features to generate embeddings can improve accuracy not only on popular benchmark tests [Bandyopadhyay et al., 2019; Bielak et al., 2012; Jiang et al., 2019b; Gong and Cheng., 2019; Yang and Li., 2020; Jo et al., 2021], but also on many machine learning applied to real-world

tasks [Kim et al., 2019; Gao et al., 2019; Wang et al., 2019; Veličković, 2023]. In this work, we propose a deeper study of what we define as edge-aware GNNs, a new subset of GNN models that focus on edge features to solve node, graph and edge related problems. We also develop and test our own edge-aware GNN in real-world datasets.

1.1 Motivation

Graphs are powerful data structures that can represent many real-world complex systems, including social, geographical, biomedical, financial, and knowledge networks [Barabási and Pósfai, 2016]. The analysis of graph-structured data allows a greater understanding of their inherent information and makes it possible to identify, classify, predict, and infer many interactions between interconnected entities. However, analyzing such complex, high dimensional data structures is often no trivial task.

Relying on ML strategies, several researchers have tried to create techniques to reduce the inherent complexity of high-dimensional graph-structured data. Methods such as graph embedding, or graph learning [Yan et al., 2006] generate a reduced dimensional representation of graph information and enable more efficient analysis of graph structured datasets. Following that idea, graph neural networks (GNNs) were developed and have achieved state-of-the-art results in comparison with previous machine learning methods [Xu et al., 2018a], becoming a widely applied graph learning and analysis method.

However, a significant limitation of some popular GNNs and graph learning methods, in general, is that, as with most previous techniques, they focus on individual nodes or entire graph representations, usually considering only the topology and attributes of those entities. Although overlooked, edges are also important entities of graph-structured data, and can often contain valuable information about the dataset. In fact, as information becomes more interconnected, data representation can make graphs grow both in size and density (i.e. number of edges), with relationships (represented by edges) becoming as important as the entities (represented by nodes) that relate with each other. This information is mostly disregarded by the majority of graph representation techniques when generating their embeddings.

Many challenges arise when learning edge representations. Most existing popular methods, such as GNNs based, may even consider some edge features, but the edge information is only implicitly captured, based on the learned node or graph representations. Other methods aim to obtain explicit representation for edges but mostly use them to enhance node-level representation. Also, the lack of edge-centered message-passing schemes makes applying GNNs directly to edge representation impossible. This issue is tackled mainly by graph transformation techniques, which add extra

layers of complexity to the task.

Research wise, edge centered studies are also scarce and sparse, usually considering edge features for specific goals and in limited ways. For instance, one of the first edge-aware embedding methods [Gao et al., 2019] used aggregation functions on node embeddings to calculate edge vectors. Unfortunately, not only was this method based primarily on node features, but they also entirely ignored the edge attributes and topology. Following that, some efforts have been made to shift the focus to accurately representing edge information. For example, based on line graph transformation, [Bandyopadhyay et al., 2019] propose to swap edges and nodes in a graph allowing the representation of edges as nodes, taking advantage of existing message-passing schemes from traditional GNNs. Improving on this concept, [Jo et al., 2021] propose a solution based on the concept of hypergraphs to inductively generate more scalable edge representations. Another approach by [Bielak et al., 2012] relies on edge-centered random walks that can capture and embed both node and edge features.

While several previous works have comprehensively covered graph neural networks and graph learning, as [Jiang et al., 2019a; Wu et al., 2020; Xia et al., 2021; Chami et al., 2022; Wang et al., 2022; Zhou et al., 2022; Khemani et al., 2024; Khoshraftar and An, 2024; Ju et al., 2024], up until now there is no work that explicitly and deeply focus on edge features as primary entities or on edge centered tasks other than link prediction.

Our present effort is to fill that research gap. To that end we aim to provide a focused exploration of edge-aware graph learning methods. We present a taxonomy specific to edge representation techniques and seek to offer insights into the novel approaches being developed to handle edge features independently from nodes. To the best of our knowledge, prior work [Bielak et al., 2012] only briefly describes a few edge focused learning methods.

Despite such lack of prior efforts to explicitly explore edge centered graph learning methods, we present the hypothesis that edge-aware techniques could be the next step in GNN research, potentially enhancing predictive accuracy and structural representations across edge-level, graph-level, and node-level analytics tasks. In fact, some recent works point to that direction. For instance [Mirhoseini et al., 2021] presents a novel edge-aware graph embedding approach that can reduce time spent on the designing of the physical layout of chips from several months to under six hours. Furthermore, many complex graph applications, such as computing edge centrality [Newman, 2010] or information diffusion [Rogers, 2003] are edge-centric tasks and, by nature, more complex than traditional node-driven tasks [Bandyopadhyay et al., 2019].

In sum, although GNNs are the state-of-the-art graph learning methods, efforts to describe, explore and integrate edge-aware solutions are still scarce. This indicates

a viable research avenue that remains mostly unexplored, offering the potential to push the state-of-the-art in graph learning even further. In particular one may expect edge-aware GNN methods to play a particularly strong role in the analysis of large dense graphs with rich edge information.

1.2 Problem Statement

As most GNN research is mainly node or graph-centered, we found a gap in literature regarding the use of edge features, specially for general learning tasks. To tackle that issue, in this thesis, we define edge-aware GNNs, a new subset of GNN methods. Although previous works present edge centered GNN methods and show that these approaches have the potential to outperform traditional GNNs in some scenarios, it is not clear yet if edge-aware methods are indeed the next step of GNN research or a more focused tool to be used in specific datasets. Up until now, research on edge-aware GNNs is sparse, as no previous work compiles, compares and discusses different methods, thus this dissertation aims to investigate the following hypothesis:

> **Hypothesis** *Solutions that integrate both implicit and explicit edge information in graph learning methods could lead to new avenues in GNN research, contributing to enhanced* predictive accuracy and structural representations across edge-level, graph-level, and node-level analytics tasks.

Towards addressing this hypothesis we aim at compiling, discussing and analyzing results of edge-aware GNN methods. Building upon existing literature, we also intend to advance the state-of-the-art by proposing and developing our own edge-aware method. We apply our solution to a real-world task and investigate positive and negative aspects of edge-aware GNNs, providing valuable insight for future GNN research.

1.3 Objectives

We envision our investigation of the hypothesis posed in the previous section by tackling three complementary questions, as described next:

- **Research Question 1 (RQ1)**: Can edge-aware learning methods surpass traditional machine learning approaches in both computational efficiency and benchmark evaluations? To address this question, we systematically present and analyze various methods, examining their performance while highlighting their

advantages and limitations. We also calculate, detail and compare computational complexity for each presented model.

- **Research Question 2 (RQ2)**: Are edge-aware learning methods able to address real-world problems more accurately and efficiently than traditional techniques? Towards answering this question, we present real-world problems in different areas and the proposed edge-aware solutions applied to them. Focusing on the problem of predicting node loads on a backbone network, we develop our own model and compare results with some of the most popular GNN methods as well as a state-of-the-art edge-aware model.

- **Research Question 3 (RQ3)**: What are the computational trade-offs when applying edge-aware methods across different domains? Besides comparing results, we also present extensive descriptions and comparisons of computational complexity between every presented model. We provide detailed explanations to describe which factors can impact each specific edge-aware method.

- **Research Question 4 (RQ4)**: How can we promote the research on edge-aware GNNs and make it more accessible for the scientific community? Throughout our research we did not find open, flexible frameworks that allow experimenting with edge-aware methods. In fact, most of the models presented in this work do not have publicly available implementations. With that in mind we developed and made public our own testing framework, that can be used to verify our results and also modified to include different GNN models.

1.4 Contributions

The contributions achieved so far are concentrated in the first four research questions, i.e., RQ1, RQ2, RQ3 and RQ4. Towards answering these questions, we explored and compiled previous work on edge-aware related models. Also, looking forward to advance the current state-of-the-art, we developed our own edge-aware GNN model and made it available alongside an open and flexible framework that can also function as a testbed for new GNN methods. Specifically, our contributions are:

- We show a historical perspective on edge-centric learning that traces the evolution of edge representation techniques, from early graph analytics to the latest GNN-based edge-aware methods.

- We Propose a novel Taxonomy of Edge-Aware Learning Methods that defines a structured classification of edge-aware techniques, categorizing them into:

Traditional edge-centric methods, Random-walk-based edge embeddings and GNN-based edge-aware models.

- We present a comparative analysis of edge-aware vs. node-centric models that reviews limitations of standard GNNs that overlook edge features, making quantitative and qualitative comparisons of edge-aware vs. node-focused models across different tasks while also discussing scalability and performance trade-offs.
- We introduce AttEAGNN, an attention based edge-aware GNN model that separately processes node and edge features before merging them. For enhanced performance, we also develop a feature augmentation technique for datasets lacking explicit edge features.
- We apply AttEAGNN for predicting node loads in two real-world backbone networks, using both measured and artificially generated traffic data. We also apply our model to benchmark datasets, discussing and comparing all results to different node and edge centered methods.
- We provide an open-source framework for edge-aware GNN experiments, ensuring reproducibility by making the model and datasets publicly available.
- Finally, we identify open challenges in edge representation learning, highlighting over-smoothing, scalability, and computational cost as key challenges. We outline future research directions by suggesting potential improvements, such as hybrid edge-node models for better information propagation, dynamic edge representation learning for evolving graphs and new benchmarking datasets that explicitly contain edge features.

1.4.1 Publications

As this research evolved in time, partial results were published on multiple conferences and journals. Our initial concepts [Almeida et al., 2024a] were presented at the Network and Services Management Workshop (WGRS) that took place during the 2024 Brazilian Symposium of Computer Networks (SBRC). A more mature version of our model [Almeida et al., 2024b] was introduced during the 2024 XIV Brazilian Symposium on Computing Systems Engineering (SBESC). Finally, our main results were published in 2025 on the Journal of Complex Networks [Almeida et al., 2025]. We also have a survey that reviews related research and presents our proposed edge-aware taxonomy while serving as motivation to the development and testing of our own model. This survey was published on IEEE Transactions on Network Science and Engineering [Almeida et al., 2026].

1.5 Outline

This project is organized as follows: Chapter 2 introduces some theoretical background concepts needed for the full understanding of this research, as well as related work. Chapter 3 presents our analysis of edge-aware graph learning. Here we define the most common graph learning problems, and classify several edge-aware methods leading to a novel taxonomy. While Chapter 3 details generic methods, Chapter 4 shows works that apply edge-aware learning methods to real-world scenarios. We briefly explain each scenario, then show how the combined use of edge features helped to improve learning results. Chapter 5 introduces AttEAGNN, our own edge-aware GNN model that combines node and edge processing with an edge attention mechanism. Chapter 6 shows the experiments we conducted with our model, detailing datasets and presenting comparative results in different scenarios. Finally, Chapter 7 points some limitations of our work, as well as future directions and conclusions.

Chapter 2

Background and Related Work

This chapter presents the theoretical background and basic concepts used in the context of this research. First, we introduce a general overview of machine learning, focusing on deep learning. Next we briefly show graph fundamentals and finally we present graph representation learning methods, including GNNs, that combines graph structured data with machine learning algorithms.

2.1 Machine Learning Basics

In recent years, significant progress has been achieved in uncovering the properties of complex interconnected systems such as computer networks. However, studying the large variety of real-world complex network structures coming from the emergence of Big Data remains an important challenge of network science [Barabási and Pósfai, 2016]. In the last two decades, Machine Learning (ML), a subset of the artificial intelligence research field, has been used to provide large complex systems the ability to learn, make predictions, and improve from data. As a result, computationally costly and hard-to-solve tasks significantly benefit from ML techniques.

Machine Learning refers to a class of computer algorithms that can automatically learn and improve their efficiency. Most of these algorithms are, in essence, optimization problems [Sun et al., 2019]. They aim to find mathematical models capable of achieving the best possible accuracy given a particular task. Accuracy is measured using a loss function, which is evaluated and calculated at each execution of the algorithm. The model parameters are updated based on the resulting error in a way that hopefully will increase the accuracy in a process called training.

2.1.1 Categories of Machine Learning

Depending on the goal, how data is provided to the algorithm, and how accuracy is measured, ML algorithms can be classified into three main categories [Stamile et al., 2021]: supervised learning, unsupervised learning and reinforcement learning.

Supervised Learning is the paradigm used when the answer to a problem is al-

ready known, i.e., the goal is to predict a target output given input data. The dataset is composed of pairs $\langle x, y \rangle$, where x is the input value, or feature and y is the expected output value or label. A distance function measures the differences between the predicted value and the expected output.

In unsupervised learning instead, the correct answer is not known beforehand, i.e., the goal is to infer patterns such as identifying clusters of elements in the data. Only the inputs x are provided in this scenario. The algorithm must then deduct patterns and structures, attempting to find similarities. There are also semi-supervised learning methods that combines both previous strategies, using labeled and unlabeled data as inputs.

Lastly, reinforcement learning (RL) is based on training agents that take actions in some specified environment and receive rewards in order to achieve a goal. Particularly, in deep RL [Franç ois-Lavet et al., 2018], the agent chooses its actions based on a trained neural network. This method is powerful and flexible, since it can decide on how to act based on exploration alone.

Formally, in a RL problem, the environment is usually modelled as a finite markov decision process (MDP) defined as the 4-tuple (S, A, P_a, R_a) where:

- S is a finite set of states;
- A is a finite set of actions;
- $P_a(s, s')$ is the probability that an action a in state s at time t will lead to state s' at time $t + 1$;
- R_a is the reward received after transitioning from states s to s', after taking the action a.

The agent that interacts with that environment implements a policy that will result in taking an action given a set of features probabilistically obtained from the current state of the environment (an observation). In deep RL, the policy applied by the agent takes the form of a neural network.

The last part of an RL scheme is to calculate the reward obtained after taking an action and ensuring that the model adapts in regard to that value. Multiple methods have been proposed, such as policy gradient [Williams, 1992], or proximal policy optimisation(PPO) [Schulman et al., 2017]. Figure 2.1 illustrates the described RL structure.

2.1.2 Deep Learning

Developed to mimic the biological neural networks found in the human brain, deep learning (DL) is a specific set of algorithmic techniques that relies on neural networks

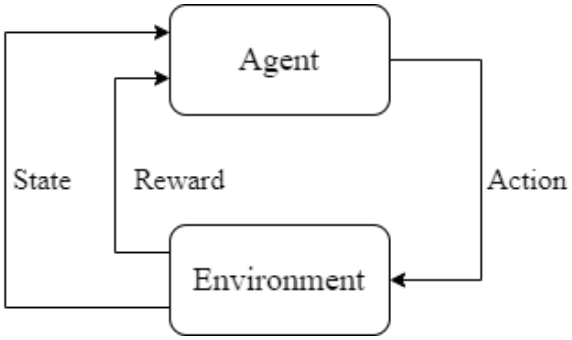


Figure 2.1: Structure of a reinforcement learning problem

(NNs) and can be used for supervised, unsupervised or reinforcement learning. The basics of DL were built upon the perceptron [Rosenblatt, 1958]. It is inspired by the structure of a neuron and it is capable of receiving inputs, performing a mathematical operation, such as a weighted sum, and then firing an output if certain criteria is met. NNs interpret input information by labeling or clustering data, and the patterns recognized are numerical. Therefore, all real world data such as images, sound, texts or time series must be mathematically represented.

A single layer of neurons can perform basic and sometimes efficient clustering, however, by definition, DL depends on the depth of the NN, i.e. how many layers of neurons compose the network. Usually, a NN must be composed of three or more layers of neurons to fall into that category.

Figure 2.2 summarizes the machine learning taxonomy presented on this section.

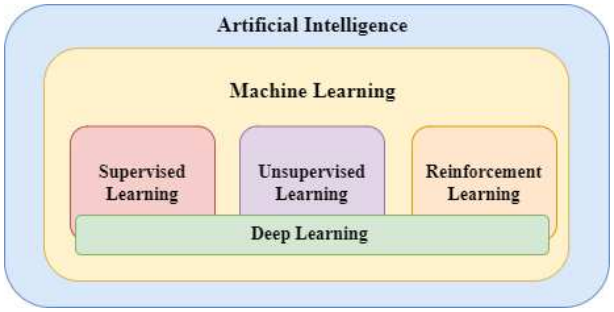
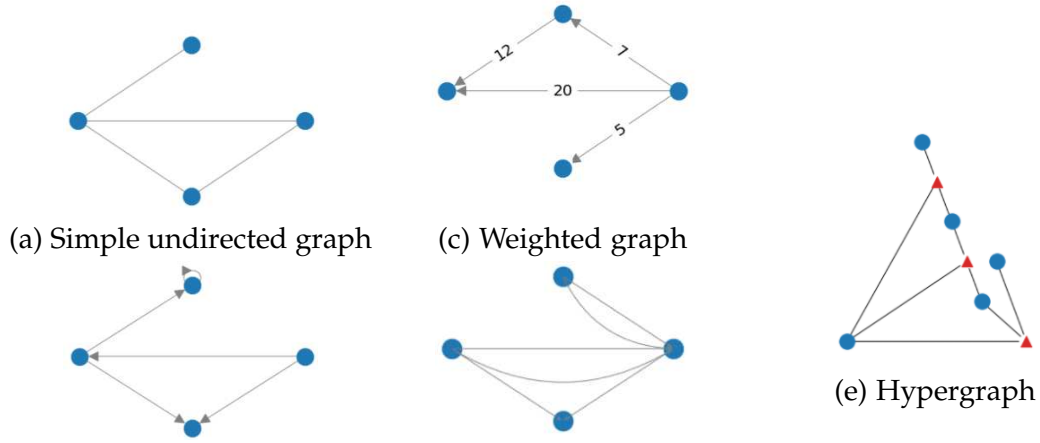


Figure 2.2: Machine Learning methods summary

2.2 Fundamentals of Graphs

Graphs are mathematical structures widely used to describe relations between entities in multiple contexts. Connections in social networks are a classic example of graph-structured data, but graphs may also represent maps, biological structures, web pages, and more.

Formally, a graph [West, 2000] (Fig. 2.3a) is defined as $G = (V, E)$, where $V =$



(a) Simple undirected graph

(c) Weighted graph

(e) Hypergraph

(b) Directed graph with self-loop

(d) Multigraph

Figure 2.3: Graph examples

$\{v_1, ..., v_N\}$ is a finite set of N nodes or vertices, and $E = \{\{v_i, v_j\} ... \{v_k, v_w\}\}$ is a finite set of pairs of vertices, whose elements $E_{ij} = \{v_i, v_j\}$ are called edges or links. The number of nodes $|V|$ represents the order of a graph and the number of edges $|E|$ is the size of the graph. The number of edges connected to a node determines its degree. In simple undirected graphs, each element of E is unordered and represents a bidirectional relationship, i.e. $\{v_i, v_j\}$ and $\{v_j, v_i\}$ represent the same edge. A directed graph (Fig. 2.3b), or digraph is defined as $G = (V, E)$, where $V = \{v_1, ..., v_N\}$ is a set of N nodes, and $E = \{\{v_i, v_j\} ... \{v_k, v_w\}\}$ is a set of ordered pairs representing the connection between $(v_i, v_j) \in V$. In this case, $\{v_i, v_j\}$ and $\{v_j, v_i\}$ represent different edges. For a node $v \in V$, the number of head ends adjacent to v is called the indegree($deg^-(v)$), while the number of tail ends adjacent to v is its outdegree($deg^+(v)$).

A weighted graph (Fig. 2.3c) is a graph whose edges have numerical weights associated with them. They may be defined by a weight matrix W such that $W_{ij} = w(e_{ij})$. The graph is unweighted otherwise. Weighted graphs may be directed or undirected. Aside from weights, edges may also contain features represented by a matrix W^e.

It is possible to generalize the graph definition to allow multiple edges to have the same pair of start and end nodes. A multigraph (Fig. 2.3d) is defined as $G = (V, E)$, where V is a set of nodes and E is a multi-set of edges. The multigraph is directed if E is a multi-set of ordered pairs, otherwise it is called undirected multigraph. A single edge, defined as an hyperedge, may also connect multiple nodes, i.e. $e \in E = \{v_i, v_j, v_k, v_w\}$, in structures called hypergraphs (Fig. 2.3e). Note that in Figure 2.3e, the hyperedges are represented as red triangles.

Graphs may yet be classified as homogeneous, when every node belongs to a single node type and every edge belongs to a single edge type, or heterogeneous, if

the number of edge types $|\mathcal{L}^e| > 1$ or the number of node types $|\mathcal{L}^v| > 1$.

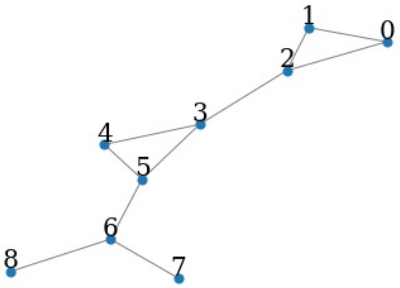
For learning tasks, graphs may be represented as an adjacency matrix A, which is an $N \times N$ matrix whose element $A_{ij} = 1$ if there is an edge connecting nodes i and j. In simple undirected graphs, the adjacency matrix is symmetric with zeros on its diagonal. On the other hand, adjacency matrices may be asymmetric in directed graphs, where a non-zero element A_{ij} indicates that there exists an edge from node i to node j. Weighted graphs follow the same rules, depending if they are directed or undirected, except that in both cases, A_{ij} may be different from 0 or 1 to represent the edge weight values.

2.3 Graph Representation Learning

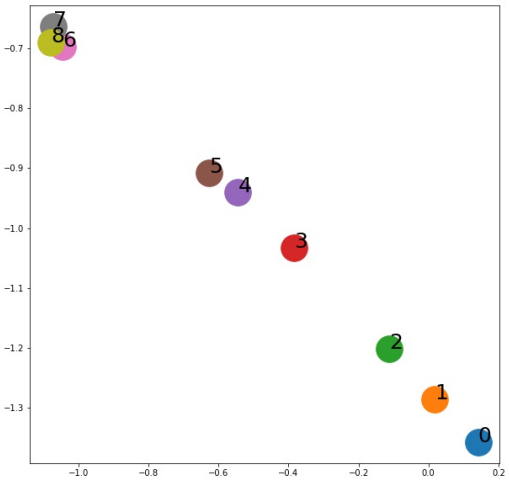
As graphs are complex structures used to represent large datasets, analytics tasks are often too complex for traditional computer algorithms. In this sense, by applying ML techniques to graphs it is possible to create algorithms to find and interpret latent patterns [Stamile et al., 2021]. For this reason, there has been an increasing interest in graph representation learning algorithms. Feature engineering may be used to process input data from graphs, producing a meaningful but compact representation of each object present in the dataset. However, new approaches based on ML algorithms can learn faithful representations of data such that structural relationships on the reduced space reflect the structure of the original graph. This process is often called graph embedding or graph representation learning.

The graph embedding task aims to learn an encoder function that maps a graph or any of its components to a low-dimensional continuous vector space. More specifically, given a graph $G = (V, E)$, the problem of graph embedding is to find a function $ENC : G \rightarrow \mathbb{R}^d$ that maps G into a predefined d-dimensional space such as $d \ll |V|$ and in which graph properties, such as topology and similarity measures [Hamilton et al., 2017b; Cai et al., 2018; Goyal and Ferrara, 2018], are preserved as much as possible.

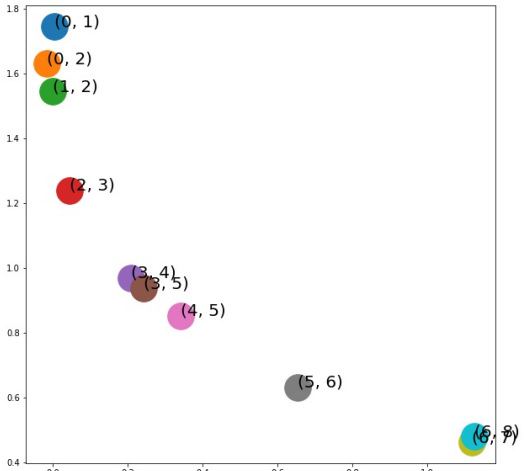
Once ENC is learned, it can be applied to the graph, and its mapping is used as a feature set for an ML algorithm. It is important to notice that ENC may be trained and applied to learn nodes, edges, and whole graph representations. Based on the task objective, three main categories of embedding can be defined: (i) node embedding, where mappings are learned for each node; (ii) edge embedding, where edges are represented on the embedding space and (iii) whole-graph embedding when an entire graph is mapped into a single vector [Cai et al., 2018]. In the dimensional space generated by ENC, objects that share similarities on the original graph will be similar in the embedding space, for instance they may have a small Euclidean distance, while dissimilar objects may have a large Euclidean distance.



(a) Example graph



(b) Node Embedding



(c) Edge Embedding

Figure 2.4: A toy example of node and edge embedding, where groups of elements with similar properties are close together at the 2D space.

Figure 2.4 presents a toy example of a graph and respective node and edge embeddings. Figure 2.4 (a) depicts an example of a simple undirected graph. Figures 2.4 (b) and (c) show the same graph embedded respectively at node and edge levels.

Many graph embedding methods have been proposed in the past few years [Goyal and Ferrara, 2018; Cai et al., 2018]. Algorithms based on Matrix Factorization [Ou et al., 2016; Ahmed et al., 2013] factorize an input matrix representing a graph into different components. The main difference between the methods in this category is how the loss function is defined and used during the optimization process. Different loss functions allow the creation of embeddings that emphasizes specific properties of the input graph.

Graph embedding approaches based on Random Walks [Perozzi et al., 2014; Grover and Leskovec, 2016] generate paths by following random nodes to create the structural context for each node, then apply techniques based on neural language models

such as SkipGram [Goldberg, 2014]. Formally, let G be a graph and v_i be one of its vertices selected as a starting point. The algorithm walks to a random neighbor of v_i. This process is repeated t times and the sequence of t nodes is a random walk of size t. These random walks are then given as input to the SkipGram model that generates the embedding.

More recently, there has been an increased interest in Graph Neural Network (GNN) techniques for graph representation learning [Li et al., 2015; Hamilton et al., 2017b; Xu et al., 2018b; Kipf and Welling, 2016; Veličković et al., 2018]. GNNs use a recursive neighborhood feature aggregation scheme, also called message passing, where each node calculates its own feature vector by aggregating the vectors from their neighbors. After k iterations of aggregation, a node is represented by its updated feature vector. This method is able to capture the structural information within the node's k-hop neighborhood. The entire graph representation can then be obtained through a pooling method [Ying et al., 2018], such as summing or averaging the vectors of all nodes in the graph. GNNs have achieved state-of-the-art accuracy levels in many tasks [Xu et al., 2018a] and are a topic of great interest both in machine learning and in network science.

2.4 Related Work

A significant limitation of some popular GNNs and graph embedding methods, in general, is that, as with most previous techniques, they focus on individual nodes or entire graph representations, usually considering only the topology and attributes of those entities. Although overlooked, edges are also important entities of graph-structured data, and can often contain valuable information about the dataset. This information is mostly disregarded by the majority of graph representation techniques when generating their embeddings.

For example, [Wu et al., 2020] presents a comprehensive survey of Graph Neural Networks, categorizing GNNs into recurrent, convolutional, autoencoder-based, and spatio temporal models. State-of-the-art GNN models are reviewed across different domains, emphasizing tasks such as node classification, graph classification, and graph generation, but edge information is mostly disregarded by the methods presented in this study. Later, [Xia et al., 2021] reviews the state-of-the-art in machine learning for graph-structured data. They specifically address GNNs applied to node and graph-level tasks, including applications in social and biological networks. [Chami et al., 2022] defines a comprehensive taxonomy of graph representation learning methods, primarily focusing on node-level representations, offering an overview of both supervised and unsupervised approaches, while [Yuan et al., 2022] presents a taxonomy specifically for GNN explainability. [Khemani et al., 2024] dis-

cusses various traditional non-edge-aware GNN models such as GCNs, GraphSAGE, and GATs, alongside their applications in domains like natural language processing, computer vision, and healthcare. It emphasizes message-passing mechanisms and highlights the evolution of GNN research, including future directions in scalability and dynamic graph learning. Following the idea of dynamic graph learning, [Khoshraftar and An, 2024] provides a broad overview of both traditional graph embedding techniques and GNN-based methods for static and dynamic graph analysis. The emphasis is on capturing node-level relationships in both static and evolving graphs, but edge features are mostly overlooked. More recently, [Ju et al., 2024] offers an in-depth review of deep graph representation learning. This work provides a comprehensive taxonomy that covers the architectures of GNNs, such as spectral and spatial convolutions, and advanced learning paradigms, such as self-supervised learning and semi-supervised learning on graphs. The primary focus is on node and graph-level embeddings and applications of GNNs in various domains, including social networks, molecular property prediction, and recommender systems. These works present a broad review of graph representation learning techniques, heavily focusing on GNN architectures, learning paradigms, and applications. However, none of them addresses edge-aware models or specific edge centered tasks.

Several works do approach edge-centered tasks, however they are mainly limited to link prediction. While [Kumar et al., 2020] does not specifically address GNNs, different methods dedicated to link prediction are discussed, including similarity-based indices, probabilistic models, dimensionality reduction approaches, and deep learning methods. It also explores applications in network reconstruction, recommender systems, and spam detection. As part of a broader task of node representation learning, [Jiang et al., 2019a] explores link prediction to optimize node and graph representation using semi-supervised learning, instead of explicitly addressing link prediction as a standalone task. [Wang et al., 2022] deals with the challenges of embedding heterogeneous graphs, where multiple types of nodes and edges are present. While this work addresses link prediction in heterogeneous graphs, its focus is predominantly on managing heterogeneity rather than isolating and enhancing edge feature representation. Our work differs by offering a dedicated exploration of edge-aware representation learning methods, even in homogeneous graphs. Also, we present different real-world applications of edge-aware models, not limited to the task of link prediction.

To the best of our knowledge, there have been no attempts to research and classify edge-aware graph learning extensively. Indeed, [Bielak et al., 2012] present a quick summary of edge representation learning techniques but focus on presenting their method. In our work, we discuss the most relevant and influential edge-aware graph learning methods and present real-world applications that account for edge informa-

tion and possible future directions in this area of research. Our work fills a crucial gap by addressing the challenges and opportunities in edge-aware graph learning.

To clearly situate our contributions within the existing literature, Table 2.1 provides a comprehensive comparison between prominent GNN surveys and this dissertation. As detailed in the table, previous works [Chami et al., 2022], [Ju et al., 2024], [Khemani et al., 2024], [Khoshraftar and An, 2024], [Wu et al., 2020], [Xia et al., 2021] have exhaustively covered node representation and general architecture types, but they predominantly categorize edges as auxiliary structural components or binary links. Even works focusing on heterogeneity [Wang et al., 2022] or link prediction [Jiang et al., 2019a], [Kumar et al., 2020] do not treat edge attributes as learnable entities with the same depth as nodes. In contrast, this work focuses explicitly on taxonomizing and modeling edges as primary informational entities.

Table 2.1: Detailed comparison of related work and GNN surveys against the proposed thesis.

Reference	Primary Scope	Treatment of Edge Features	Missing Aspect Addressed by Thesis
[Wu et al., 2020]	Comprehensive GNN Survey	Structural (Adjacency Matrix)	Explicit Edge Taxonomy
[Xia et al., 2021]	ML on Graphs (SOTA)	Auxiliary to Node Tasks	Edge-Centric Tasks
[Chami et al., 2022]	Rep. Learning Taxonomy	Node-Level Representations	Edge-Level Representations
[Yuan et al., 2022]	GNN Explainability	Feature Importance Scores	Edge-Aware Architecture
[Khemani et al., 2024]	Traditional Architectures	Message Passing Channels	Native Edge Attention
[Khoshraftar and An, 2024]	Dynamic Graph Learning	Temporal Links (Existence)	Deep Edge Attributes
[Ju et al., 2024]	Deep Graph Learning	Node & Graph Embeddings	Augmented Edge Features
[Kumar et al., 2020]	Link Prediction	Binary Classification Target	Continuous Edge Regression
[Jiang et al., 2019a]	Semi-supervised Learning	Optimization Constraint	Primary Learning Entity
[Wang et al., 2022]	Heterogeneous Graphs	Meta-path Definitions	Homogeneous Edge Attributes
This Thesis	**Edge-Aware Graph Learning**	**Primary Informational Entity**	**Unified Framework & Taxonomy**

2.5 Summary

This chapter established the theoretical foundations necessary for understanding the landscape of graphs and graph learning. We reviewed the fundamental concepts of machine learning and deep learning, highlighting the transition from traditional feature engineering to automated representation learning.

We formally defined graph structures—ranging from simple undirected graphs to complex hypergraphs—and explored how Graph Neural Networks (GNNs) leverage message-passing schemes to capture topological information. While standard GNNs and random walk-based methods have achieved state-of-the-art results in node classification and link prediction, our review of the related work reveals a critical limitation: the systematic underutilization of edge features.

As demonstrated in the literature review, most existing works predominantly focus on node-centric or whole-graph embeddings. When edges are considered, they are often used only as auxiliary data to improve node representations. This creates a

significant gap in the literature regarding methods that consider edges as entities with their own rich attributes and learnable representations.

The following chapter addresses this gap directly. We will move from these general definitions to a specific analysis of the state-of-the-art in edge-aware learning. We will introduce a novel taxonomy of edge-aware graph learning methods, categorizing techniques based on how they integrate edge information, whether through graph transformations, random walks, or native attention mechanisms.

Chapter 3

Surveiling Edge-aware Graph Learning Methods

In this chapter, we present edge-aware graph analysis methods, starting from the first formalization of edge classification up to the state-of-the-art GNN-based techniques. Our main goal here is to define and classify methods that are broadly applicable to various different types of graphs and tasks. We show the edge classification problem and how it was approached pre-machine learning, with high computational cost, low scalability, and accuracy. Then we present random walk-based edge embedding methods, which are also limited in scalability and inherently transductive in most cases. Finally, we show the state-of-the-art inductive GNN-based methods applied to edge representation learning and a summarized table that compares the accuracy of those methods. Figure 3.1 shows a graphical summary defining our taxonomy of edge-aware learning methods and classification of the methods shown in this work.

Table 3.1: Comprehensive Comparison of Edge-Aware Graph Learning Methods

Method	Edge Feature Integration	Learning Type	Primary Task
Random Walk-Based Methods			
Line2Vec [Bandyopadhyay et al., 2019]	N/A	Transductive	Edge Classification
AttrE2Vec [Bielak et al., 2012]	N/A	Inductive	Edge Classification
edge2vec [Gao et al., 2019]	N/A	Transductive	Link Prediction
GNN-Based Methods			
EGNN [Gong and Cheng., 2019]	Both	Inductive	Edge-Aware Node/Graph Class.
CensNet [Jiang et al., 2019b]	Edge-Conditioned Convolution	Inductive	Edge-Aware Node Classification
NENN [Yang and Li., 2020]	Edge-Informed Attention	Inductive	Edge & Node Classification
EGAT [Wang et al., 2021]	Edge-Informed Attention	Inductive	Edge-Aware Node/Graph Class.
DHT [Jo et al., 2021]	Neither	Inductive	Edge, Node & Graph Class.
LeL-GNN [Morshed et al., 2023]	Neither	Inductive	Link Prediction
Graphormer [Ying et al., 2021]	Edge-Informed Attention	Inductive	Edge-Aware Graph Class.
Hybrid and Task-Specific GNNs			
ELGNN [Kim et al., 2019]	Edge-Conditioned Convolution	Inductive	Edge & Node Classification
DGCNN [Wang et al., 2019]	Edge-Conditioned Convolution	Inductive	Edge-Aware Node/Graph Class.
Edge-GNN [Mirhoseini et al., 2021]	Edge-Conditioned Convolution	Inductive	Edge-Aware Node Classification
FLEE-GNN [Qu et al., 2023]	Edge-Conditioned Convolution	Inductive	Edge-Aware Node Classification
CAGNN [Tao et al., 2023]	Edge-Conditioned Convolution	Inductive	Edge-Aware Node Regression
AttEAGNN [Almeida et al., 2024b]	Edge-Informed Attention	Inductive	Edge-Aware Node Regression
ENGNN [Wang et al., 2023]	Edge-Conditioned Convolution	Inductive	Edge & Node Regression

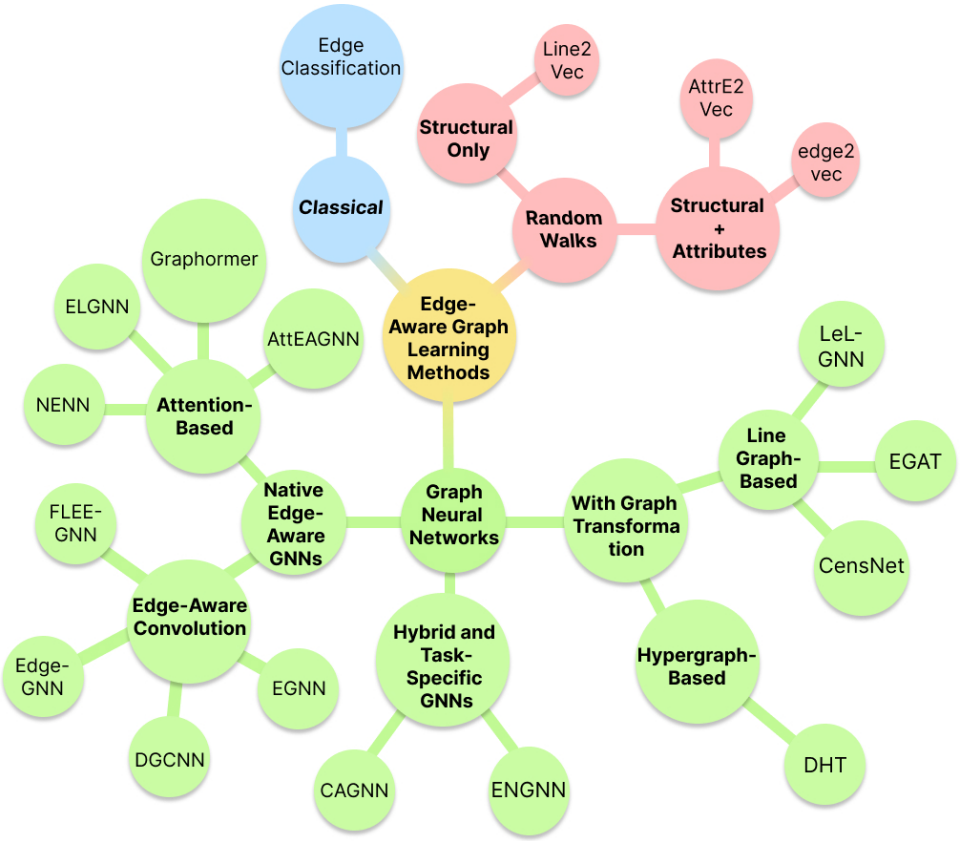


Figure 3.1: Edge-aware learning methods taxonomy

3.1 Edge-Aware Learning Tasks

Before delving into specific methods, it is essential to formally define the predictive tasks that motivate the development of edge-aware models. These tasks can be divided into two categories: those where the edge itself is the primary target of prediction (edge-centric), and those where edge information is leveraged to improve performance on traditional node- or graph-level objectives (edge-enhanced).

3.1.1 Edge-Centric Tasks

In this category, the learning objective is to predict a property of the edges themselves. The primary tasks include:

- Edge Classification: This is the task of assigning a categorical label to an edge based on its intrinsic features and its local topological context within the graph. Given a graph $G = (V, E)$ and a subset of labeled edges $E_L \subset E$, the goal is to predict the labels for the unlabeled edges $E_U = E / E_L$.
- Link Prediction: A widely studied binary classification problem that aims to determine the likelihood of an edge's existence between two nodes. Given a

graph, the model must predict whether a non-existent edge should be added or if an existing edge is valid.

- Edge Regression: This task involves predicting a continuous value associated with an edge. Examples include predicting the traffic flow on a network link, the binding affinity between two proteins, or the strength of a social connection.

3.1.2 Edge-Enhanced Node- and Graph-Level Tasks

In this category, edge features are not the final target but are used as a source of information to improve predictions at the node or graph level.

- Edge-Aware Node Classification: The objective is to predict a node's label by aggregating features not only from its neighboring nodes but also from the incident edges that connect them. The edge features provide critical context about the nature of the relationship, which can significantly refine the information passed between nodes.
- Edge-Aware Node Regression:
- Edge-Aware Graph Classification: This task involves assigning a label to an entire graph. The collective attributes and structural roles of edges provide global information that complements node features and the overall graph topology, leading to more accurate classifications.
- Edge-Aware Graph Regression:

3.2 Edge Classification

To the best of our knowledge, [Aggarwal et al., 2016] presented the first work to propose a formal general model focused on edges, specifically for the task of edge classification. Given an undirected graph $G = (V, E)$ and a set $E_l \subseteq E$ of previously labeled edges, [Aggarwal et al., 2016] defines the edge classification problem as a way to determine the unknown edge labels in $E_i = E \setminus E_l$. The proposed approach to classify edges is not based on ML techniques, instead, classical node classification methods based on the notion of **homophily** [Mewa, 2020] are adapted and used in the edge context. As the straightforward definition of node-wise homophily cannot be directly applied to edge classification, structural edge similarity is indirectly defined by node classification. Consider an edge $(i, j) \in E_i$ in a graph $G = (V, E)$ which needs to be classified. In order to determine which edges are similar to (i, j), the following steps are executed:

1. The top-k nodes more similar to i, denoted by $S_{(i)} = \{i_1...i_k\}$ are determined according to the structural similarity measured by the *Jaccard coefficient* of each node. Note that since i is the most similar node to itself, $i \in S_{(i)}$;

2. In the same manner, $S_{(j)} = \{j_1...j_k\}$ which are the top-k most similar nodes to j are determined;

3. The dominant class label on the set of edges $A_l \cap [S_{(i)} \times S_{(j)}]$ is selected and assigned as the relevant edge label for the edge(i, j).

The weighted Jaccard coefficient is used for node-wise similarity classification. The similarity between nodes i and j can be determined as the weighted average of the Jaccard coefficient of nodes $J^+(i,j)$ and the Jaccard coefficient of edges $J^-(i,j)$. The fractions of nodes incident on i belonging to each class, given by $f_i^+ = \frac{|I^+(i)|}{|I^+(i)|+|I^-(i)|}$ and $f^-(i) = 1 - f^+(i)$, are used as weights to average these two values. The averages in both cases are defined by $f^+(i,j) = (f^+(i) + f^+(j))/2$ and $f^-(i,j) = ((f^-(i) + f^-(j))/2$. Since $f^+(i,j) + f^-(i,j) = 1$, the Jaccard similarity is defined as:

$$J(i,j) = f^+(i,j) \times J^+(i,j) + f^-(i,j) \times J^-(i,j) \tag{3.1}$$

This approach, however, has a scalability issue. As the graph grows, so does the computational complexity to compare pair-wise similarity. To overcome this problem, [Aggarwal et al., 2016] also present probabilistic min-hash based algorithms [Broder et al., 1998]. Even with these algorithms, however, the results obtained using test datasets have mostly low accuracy scores, ranging from 15% in the worst scenario, to a maximum of 90% in the best case. Although this is a relevant first step in edge classification, there was still much room for improvement, and with the advent of ML techniques, more sophisticated and efficient methods were developed.

In the following, we further discuss and detail Random Walk-based methods and GNN-based methods.

3.3 Random Walk-Based Methods

Traditional embedding techniques map nodes to continuous vector representations. It has been shown that these representations outperform conventional graph algorithms [Adamic and Adar, 2003]. The first efficient node embedding approaches [Perozzi et al., 2014; Grover and Leskovec, 2016] are based on random walk (RW) techniques. However, they only marginally cover edge representation. As an example, for tasks like link prediction, where a model must be trained to classify edges as positive if they exist or negative if they do not. [Grover and Leskovec, 2016] propose

a simple aggregation function such as average or Hadamard product between two end nodes to derive the embedding vector of the corresponding edge. This is a sub-optimal approach since the original embeddings are based only on nodes and do not consider any edge attributes or structural features. Thus the need to develop methods to directly embed edges arises. The approaches presented in the next sections adapt random walk strategies and apply them specifically to edge representation learning tasks.

3.3.1 Line2Vec

Edges in a graph are not equally important. In a social network, for example, millions of people may be connected to a celebrity but still have little in common with each other. This type of connection is weaker compared to an edge that links two friends or close relatives. Also, similarly to node embeddings, edges that are topologically close to each other should have similar embeddings. In that sense, the edge representations must preserve the underlying edge semantics of the graph in a way where edge importance and edge proximity are considered.

With that in mind, [Bandyopadhyay et al., 2019] proposed **Line2Vec**, a dedicated unsupervised edge learning method which avoids aggregation of the end node embeddings. They create edge embeddings by converting the original graph into a *weighted line graph*. Given a graph $G = (V, E, W)$ where V and E are respectively the sets of nodes and edges, and W is a set of weights $w_{v_i,v_j} > 0$ associated with each edge e. A random walk strategy is then used to generate edge embeddings. Assuming $\mid E \mid = m$, the goal of the edge embedding algorithm is to learn a function $f : e \mapsto x \in \mathbb{R}^K$ that maps every edge $e \in E$ to a K dimensional space where $K < m$.

To understand the Line2Vec method, first, it is necessary to define the concept of **line graph transformation**. The line graph of any given undirected graph $G = (V, E)$, is the result of a transformation that represents each of its nodes as edges and each of its edges as nodes. In that new structure $L(G)$, two nodes will be neighbors only if their corresponding edges in the original graph G share a common endpoint node [Whitney, 1932]. Formally, in a line graph $L(G) = (V_L, E_L)$, $V_L = \{(v_i, v_j) : (v_i, v_j) \in E\}$ and $E_L = \{((v_i, v_j), (v_j, v_k)) : (v_i, v_j) \in E, (v_j, v_k) \in E\}$. Additional edges are added by the line graph transformation, so, given a node v in the original graph G with a degree of d, the line graph will have $d(d-1)/2$ edges connecting the corresponding edges in $L(G)$. Hence, the total number of edges in the line graph can be approximated by:

$$m_L = \sum_{v \in V} \binom{d_v}{2}, \tag{3.2}$$

where d_v is the degree of node v in the original graph G and $\binom{d_v}{2}$ represents the number of pairs of edges connected to node v.

A weighted approach is used in the transformation process in order to improve the random walk strategy and allow it to focus on more relevant edges represented by the nodes in the line graph. Although many of the $d(d-1)/2$ generated edges may not have considerable importance to the concerned node, they can potentially change the frequency of movement in a random walk. This may happen especially in walks involving high-degree nodes of the line graph. To overcome this issue, a strategy to ensure that the line graph $L(G)$ not only reflects the topology of G, but also that the transformation process does not affect the original graph dynamics is needed. This strategy works as follows: If a random walk started from a node $v_{ij} \equiv (v_i, v_j) \in L(G)$ traverses to $v_{jk} \equiv (v_j, v_k) \in L(G)$, that is the same as walking from node $v_j \in G$ of (v_i, v_j) to $v_k \in G$. Considering that G is an undirected graph, v_j can be chosen with a probability that is proportional to $\frac{d_{v_i}}{d_{v_i}+d_{v_j}}$, where d_{v_i} and d_{v_j} represent the degrees of the nodes connected by the edge (v_i, v_j). In this scenario, an edge has usually more importance to the node with a lower degree in comparison to the other connected nodes that have a higher degree, and the probability of v_k being selected is measured according to the edge weight of $e_{jk} \equiv (v_i, v_k) \in E$. Therefore, considering $e_{ij} \equiv (v_i, v_j)$ and $e_{jk} \equiv (v_j, v_k)$ being adjacent edges, the weight of (e_{ij}, e_{jk}) in $L(G)$ is defined as:

$$w_{(e_{ij},e_{jk})} = \frac{d_i}{d_i + d_j} \times \frac{w_{jk}}{\sum_{r \in N(v_j)} w_{jr} - w_{ij}} \tag{3.3}$$

The complete weighted line graph transformation takes $O(\mid V \mid d^2)$ time, where d is the maximum degree of a node in the graph.

Unlike typical node-based homophily, which focuses on pairwise relationships, Line2Vec introduces collective homophily to ensure that edges sharing a common node in the original graph remain close to each other in the embedding space. This concept is formalized by embedding the edges within a sphere in the line graph, where all edges incident on a common node are embedded close to each other. The goal is to ensure that all edges induced by the same node form a cluster in the embedding space, preserving both pairwise and collective proximity. To generate the edge embeddings, Line2Vec uses a skip-gram model with negative sampling, inspired by Node2Vec [Grover and Leskovec, 2016]. This model maximizes the likelihood of preserving proximity in the embedding space for edges connected via shared nodes. The optimization process seeks to maximize the log-likelihood of a random walk's context

for each node in the line graph, with the final cost function being:

$$\min_{X,R,C} \sum_{v \in V_L} \left[|NS(v)| \log \left(\sum_{v' \in N(v)} \exp(x_v \cdot x_{v'}) \right) - \sum_{v' \in NS(v)} x_{v'} \cdot x_v \right] + \alpha \sum_{u \in V} R_u^2, \quad (3.4)$$

where X is the set of embeddings, R represents the radii of the spheres for collective homophily, and C denotes the centers of the spheres. This equation ensures that edges connected to the same node remain within a small radius in the embedding space.

The time complexity of performing random walks on the line graph is determined by the number of nodes (which corresponds to the number of edges in the original graph) and the number of edges in the line graph. Assuming a fixed number of walks per node and a fixed context size, the complexity of performing random walks on the line graph is:

$$O(E \cdot d^2 \cdot \log N), \quad (3.5)$$

where E is the number of edges in the original graph, d is the maximum degree of a node in the original graph, and $logN$ is due to the skip-gram model's optimization with negative sampling. Therefore, the overall complexity of the Line2Vec method, combining the line graph construction and the random walk embedding, is:

$$O(N \cdot d^2 + E \cdot d^2 \cdot \log N) \quad (3.6)$$

Results presented by [Bandyopadhyay et al., 2019] show that in edge community detection and visualization, Line2Vec greatly outperforms pure random walk strategies. Though the accuracy in other tasks like classifying edges and identifying edge clusters is still mostly superior in comparison to traditional methods, in some datasets the results observed were similar or even lower than non-specific edge embedding methods.

Some noticeable shortcomings of line2Vec are: (i) As its node-focused predecessor methods, it is of transductive nature, i.e. the whole graph is needed in the training process and does not account for adding or removing edges and nodes to the graph. (ii) Many real-world datasets contain rich attribute data related to edges and/or nodes. Edges that are similar in structure may have significantly different features. As Line2Vec relies on only the edge structural information to generate embeddings, this rich information is completely overlooked.

3.3.2 AttrE2Vec

To overcome some of Line2Vec limitations, [Bielak et al., 2012] proposes **AttrE2Vec**, an edge-focused, unsupervised and **inductive** approach that makes use of both node and edge features to create edge embeddings. Considering a graph $G = (V, E)$ where each node $v \in V$ and each edge $e \in E = (i, j)$ has associated features $m_i \in \mathbb{R}^{d_V}$ and $f_{ij} \in \mathbb{R}^{d_E}$, a matrix $M \in \mathbb{R}^{|V| \times d_V}$ represents node features and another matrix $F \in \mathbb{R}^{|E| \times d_E}$ represents edge feature. The edge embedding goal is to learn a function $f : E \to \mathbb{R}^d$ where the dimension d to which the edges are mapped should be smaller than the original edge dimensionality d_E, i.e.: $d << d_E$. AttrE2Vec combines the information about the graph topological structure with both edge and node features: $f : (E, F, M) \to \mathbb{R}^d$.

A given edge (i, j) may provide three sources of relevant data to the embedding task: (i) the edge features m_{ij} and both its neighborhoods (ii) N_i and (iii) N_j. AttrE2Vec aims to exploit those three sources jointly. Information about the neighborhoods N_i and N_j is obtained by aggregations S_i and S_j. To capture the topological neighborhood structure, k **edge random walks** of length l starting from node i are performed. A uniform sampling approach, similar to [Perozzi et al., 2014] is used to obtain the next edge on the walk. Hence, each uth walk w_i^u started from node i is a sequence of edges:

$$\mathbf{RW}(G, k, L, i) \to \{w_i^1, w_i^2, ..., w_i^k\} \tag{3.7}$$

$$w_i^u \equiv (i, i_2), (i_3, i_4), ..., (i_{L-1}, i_L) \tag{3.8}$$

Using a walk aggregation model Agg_w, which averages over the vectors S_i^u, the edge features on each random walk are aggregated in a single vector:

$$S_i^u = Agg_w(w_i^u, F, M) \tag{3.9}$$

After that, the **neighborhood aggregation model** Agg_n summarizes the neighborhood S_i.

$$S_i = Agg_n(\{S_i^1, S_i^2, ..., S_i^k\} \tag{3.10}$$

Neighborhood aggregation model Agg_n may apply one of the following approaches:

- **average**: Simply averaging the edge attribute vectors in the random walk;
- **exponential**: Is a weighted average with the weights being exponents of the

negative position in the random walk. In this model, neighboring edges have more relevance than further way edges;

- **GRU**: Uses a Gated Recurrent Unit [Chung et al., 2014] architecture where hidden and input dimension is equal to edge attribute dimension; the aggregating process starts at the end of the random walk and moves towards the beginning and the aggregated representation is the output of the last hidden vector;
- **ConcatGRU**: Similar to GRU-based aggregation, but also uses the node feature information concatenating them with edge attributes.

The d dimensional embedding h_{ij} is obtained using an **encoder module** Enc that combines the summarized neighborhood information S_i, S_j with the edge features f_{ij}.

$$h_{ij} = Enc(f_{ij}, S_i, S_j) \tag{3.11}$$

This encoder uses a multilayer perceptron (MLP) along with an attention mechanism [Veličković et al., 2018] to ensure that the model focuses on the most relevant information from edges and their neighborhoods.

Two primary loss functions are combined to ensure accurate embeddings. First, a cosine embedding loss is used to minimize the cosine distance between embeddings of similar edges while maximizing the distance between unrelated edges. Then a mean-squared error (MSE) loss is used to reconstruct the original edge features from the embeddings, ensuring that they preserve this information. The total loss is a weighted combination of these two functions:

$$L = \lambda \cdot L_{\cos} + (1 - \lambda) \cdot L_{\text{MSE}} \tag{3.12}$$

One of the main strengths of AttrE2Vec is its ability to generalize to new, unseen edges during inference, making it an **inductive** model. Unlike Line2Vec and other transductive models that require retraining when new nodes or edges are introduced, AttrE2Vec can infer embeddings for new edges without requiring access to the entire graph structure. This property is crucial for dynamic graphs where new connections form frequently, such as social networks or evolving biological networks.

Compared to Line2Vec, AttrE2Vec has a generally smaller computational complexity, as the random walks are performed directly on the original graph to collect neighborhood information for each edge $e = (u, v)$. Considering that k random walks of length L are performed for each node in a graph with E edges, the complexity for generating these random walks is:

$$O(E \cdot k \cdot L) \tag{3.13}$$

The feature aggregation step uses an MLP with an attention mechanism. The complexity of this step is proportional to the number of edges E and the dimensionality d of the edge features:

$$O(E \cdot d), \tag{3.14}$$

Making the total computational complexity of AttrE2Vec:

$$O(E \cdot (k \cdot L + d)) \tag{3.15}$$

Experimental results using popular datasets like CORA, PUBMED, and CITESEER [Sen et al., 2008] showed that AttrE2Vec is able to surpass previous embedding methods in edge classification and clustering tasks. However, the focus on local neighborhoods due to the use of random walks can make the model overlook global graph patterns. Also, due to the feature reconstrucion loss, there is a risk of overfitting, especially in scenarios with high-dimensional feature spaces or noisy features.

Table 3.2 summarizes key aspects of both random walk methods presented in this section.

Table 3.2: Comparison between random walk based edge-aware methods.

	Line2Vec	**AttrE2Vec**
Approach	RW on Line Graph	RW on Original Graph
Attributes	Structural Information	Node and Edge Attributes
Paradigm	Transductive	Inductive
Aggregation	RW-based	RW + Aggregation
Loss	SG based on RW Proximity	Cosine + MSE Loss
Scalability	Limited by Dense Graphs	Large, Dynamic Graphs
Complexity	$O(N \cdot d^2 + E \cdot d^2 \cdot \log N)$	$O(E \cdot (k \cdot L + d))$

3.4 GNN Based Methods

Random walk-based embedding methods have several limitations. For instance, the local random walks are able to capture neighborhood information, but can ignore global patterns on the graph. Also, random walk methods can be confused by isomorphic substructures or certain heterophilous graphs (where connected nodes have

dissimilar labels or attributes). Graph density is also an issue, since random walks may not explore enough of the structure in sparse graphs, while in dense graphs they may capture too much redundant information. These limitations have been recently addressed with the adoption of neural networks and deep learning. Graph Neural Networks (GNNs) are the current state-of-the-art neural network methods for graph representation learning. However, as in the previous methods presented in this paper, initial GNN-based techniques were focused on a node and whole graph representation. Most of them completely ignored edge features and attributes while others relied on edge data only to improve node and graph representation. Nonetheless, accurately representing edges is crucial for tasks like graph generation and reconstruction, as well as graph classification tasks for which the edges are important for discrimination. Also, due to the nature of the message passing scheme, most GNNs have inherent shallow depth [Loukas, 2019]. This happens because as more layers are added, the neighborhood range is increased and node representations tend to converge to the same value. This eventually leads to over-smoothing and information loss. Therefore, researching different methods that do not rely solely on increasing depth is key to improving GNNs performance in general. In this work, we show that edge-aware models are a viable and promising research direction. The following sections present edge-aware GNN-based approaches applied to node and graph-focused tasks, as well as the most recent state-of-the-art method designed for edge-oriented tasks.

3.4.1 Integrating Edge Features in GNNs

Before presenting edge-aware GNN models, it is important to formally characterize the integration of edge features into GNNs and how they influence the core message-passing framework. For a given node i, we can generalize the update function in a GNN layer as an aggregation over its neighborhood $\mathcal{N}(i)$. Edge-aware models modify this process by making the message transmitted from a neighbor j explicitly dependent on the features of the edge e_{ij}. We can broadly categorize these models into two primary mechanisms: Edge-Conditioned Convolution and Edge-Informed Attention.

Edge-Conditioned Convolution

In the Edge-Conditioned Convolution paradigm, edge features act as dynamic weights or filters that modulate the message passed from a neighboring node. A generalized

update rule for a node i's representation h_i at layer $(l+1)$ can be expressed as:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} f_\theta(e_{ij}) \odot (W^{(l)} h_j^{(l)})\right) \tag{3.16}$$

Where $W^{(l)}$ is a learnable weight matrix, σ is a non-linear activation function, and $\odot$ denotes an element-wise operation (like multiplication). The key component is $f_\theta(e_{ij})$, a learnable function (e.g., an MLP) that transforms the edge features e_{ij} into a scalar or vector that conditions the message from node j.

Edge-Informed Attention

The Edge-Informed Attention mechanism extends the Graph Attention Network (GAT) by incorporating edge features directly into the computation of attention scores, α_{ij}. This improvement allows the model to learn the importance of a neighbor based not only on the nodes' features but also on the nature of their relationship. The general form of the attention score is:

$$\alpha_{ij} = \text{softmax}j\left(g(h_i^{(l)}, h_j^{(l)}, e_{ij})\right) \tag{3.17}$$

where g is an attention mechanism (e.g., an MLP) that takes the features of both nodes and their connecting edge as input. The node update then becomes a weighted sum of the neighbors' transformed features:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W^{(l)} h_j^{(l)}\right) \tag{3.18}$$

By framing the various models within these two formal categories, we can better understand their theoretical bases and compare how they leverage edge information to enhance graph representation learning. Table 3.1 indicates the edge feature integration mechanism of each GNN-Based edge-aware method.

3.4.2 EGNN

In recent years, several GNN models have been proposed. Inspired by the machine learning convolution technique, [Defferrard et al., 2016] and [Kipf and Welling, 2016] propose graph convolutional networks (GCNs). [Veličković et al., 2018] adapt attention mechanisms to graph learning and propose a graph attention network (GAT). Most GCN and GAT methods, however, do not fully consider edge data. In GAT, the only edge information used is whether there are or are not edges between nodes.

However, edge features may possess variable and/or continuous values. GCNs can consider real values like edge weights, but the edge features are limited to being one-dimensional. Extending both GCN and GAT methods, [Gong and Cheng., 2019] introduces Edge-enhanced Graph Neural Network (EGNN), a framework designed to incorporate multi-dimensional edge features into graph neural networks.

Each edge feature E_{ij} is a vector that represents attributes of the edge between nodes i and j in a graph. Multi-dimensional edge features are represented as an edge tensor $E \in \mathbb{R}^{|N|\cdot|N|\cdot|P|}$, where $|N|$ is the number of nodes and P the number of edge feature dimensions, or channels.

A major innovation in EGNN is the introduction of doubly stochastic normalization for edge features. Traditional GCN models use row or symmetric normalization of the adjacency matrix to ensure stable aggregation. EGNN extends this by normalizing the edge feature tensor in such a way that each row and column sums to 1, ensuring a more stable propagation of edge features across layers. Formally, the normalization process is given by:

$$E_{ijp} = \frac{\sum_{k=1}^{N} \tilde{E}_{ikp}\tilde{E}_{jkp}}{\sum_{v=1}^{N} \tilde{E}_{vkp}}, \tag{3.19}$$

where $\tilde{E}$ is the raw edge feature matrix. Edge features are also not considered static, but able to evolve as they propagate through the network. In that sense, EGNN introduces the concept of **adaptive edge features** across layers. At each layer, edge features are adjusted based on local and global information, helping to capture more complex and broad relationships between nodes and edges. This adaptiveness is a significant difference from traditional GNNs, where edge weights are usually fixed.

By encoding edge directions as multi-dimensional features, EGNN can effectively handle both undirected and directed graphs. For a directed edge connecting node i to node j, the edge feature vector E_{ij} is augmented to three channels:

- E_{ij} for the forward direction,
- E_{ji} for the reverse direction,
- $E_{ij} + E_{ji}$ for the undirected case.

This allows the model to aggregate information from forward, backward, and undirected neighborhoods, effectively capturing directional information.

EGNN was implemented in two different variants, according to the feature aggregation method. EGNN(A) uses an attention-based aggregation scheme, extending the GAT model by incorporating multi-dimensional edge features into the attention mechanism. The attention coefficients are computed as follows:

$$\alpha_{ijp} = \mathrm{DS}\left(f(X_i, X_j)E_{ijp}\right), \tag{3.20}$$

where α_{ijp} is the attention coefficient for the p-th edge feature channel, DS stands for doubly stochastic normalization and $f(X_i, X_j)$ is a similarity function (e.g., dot product) between the features of nodes i and j. The computational complexity of EGNN(A) is highly influenced by the attention computation and node updates, being specifically:

$$O(N^2Pd) + O(N^2d^2), \tag{3.21}$$

where N is the number of nodes, P is the number of edge feature channels and d is the dimensionality of node features. The first term $O(N^2Pd)$ arises from the attention computation and the second term $O(N^2d^2)$ is for the node update process based on the attention scores.

On the other hand, EGNN(C) is a convolutional-based layer derived from the GCN model. A multi-dimensional edge feature tensor E is used to aggregate node features. The update rule for node features is:

$$X^l = \sigma\left(\sum_{p=1}^{P} E_{ijp}X^{l-1}W^l\right), \tag{3.22}$$

where X^l is the node feature matrix at layer l, E_{ijp} is the normalized edge feature tensor, W^l is a learnable weight matrix and σ is a non-linear activation function such as a ReLU. Unlike EGNN(A), there is no explicit attention mechanism in EGNN(C). Instead, the edge features are directly used as weights in the convolutional aggregation, so the computational cost is only:

$$O(N^2Pd) \tag{3.23}$$

Both EGNN variants were tested on several benchmark datasets, including CORA, CITESEER and PUBMED [Sen et al., 2008] for node classification tasks, and molecular datasets (Tox21, Lipophilicity, Freesolv) [Wu et al., 2018] for graph regression tasks. Experimental results showed that EGNN consistently outperformed traditional GCN and GAT models, particularly when edge features were important for the task (e.g., molecular datasets with multi-dimensional bond features). However, the use

of multi-dimensional edge features, the attention mechanism and double stochastic normalization process introduce additional computational overhead and increase memory usage, particularly for dense graphs.

3.4.3 CensNet

Following a non-attention-based approach and relying in a line graph transformation scheme, [Jiang et al., 2019b] presents Convolution with Edge-Node Switching (CensNet), a general graph embedding framework for semi-supervised classification and regression that combines both node and edge features. Let G be a graph represented by a node adjacency matrix. The line graph $L(G)$ is constructed in a way that represents the edge adjacencies in G. The proposed CensNet method conducts graph convolution operations on both G and $L(G)$. Using node and edge features, two forward-pass propagation rules are employed on G and $L(G)$ alternately updating the embeddings. Two major graph convolutions are presented in the literature: spatial and spectral convolution. Though the CensNet framework is independent of the used convolutional method, [Jiang et al., 2019b] only implements the spectral-based.

CensNet defines two distinct propagation rules, one for nodes and one for edges, allowing for alternating updates of node and edge embeddings. The Node Layer Propagation Rule updates node embeddings using both node and edge features from the previous layer. The edge embeddings influence the node embeddings, enriching them with information about relationships between nodes, formally:

$$H_v^{(l+1)} = \sigma\left(T\Phi(H_e^{(l)}P_e)T^T \odot \tilde{A}_v H_v^{(l)} W_v\right), \tag{3.24}$$

where $H_v^{(l)}$ and $H_e^{(l)}$ are, respectively, node and edge embeddings at layer l, T is the transformation matrix that maps edges to nodes, P_e are learnable edge weights, $\tilde{A}_v$ is the normalized node adjacency matrix and σ is a non-linear activation function such as ReLU.

Similarly, the Edge Layer Propagation Rule updates edge embeddings by considering both the previous edge embeddings and the node features:

$$H_e^{(l+1)} = \sigma\left(T^T\Phi(H_v^{(l)}P_v)T \odot \tilde{A}_e H_e^{(l)} W_e\right) \tag{3.25}$$

Here, T^T is the transformation matrix that maps nodes to edges, P_v are learnable node weights and $\tilde{A}_e$ is the normalized edge adjacency matrix. This alternating process allows nodes and edges to enrich each other, leading to better embeddings.

CensNet customizes its loss function depending on the task (e.g., classification,

regression, or link prediction). For example, for node classification, CensNet uses the standard cross-entropy loss, while for graph regression, it uses the mean squared error (MSE).

The computational complexity of CensNet is influenced by the alternating update process between nodes and edges. Specifically the node layer complexity is:

$$O(E \cdot d) \tag{3.26}$$

The edge layer complexity being:

$$O(N \cdot d), \tag{3.27}$$

where E and N are, respectively the number of edges and nodes and d is the dimensionality of both node and edge embeddings. This leads to a total computational complexity of:

$$O(L \cdot (E \cdot d + N \cdot d)) \tag{3.28}$$

The need to update both nodes and edges increases the overall computational cost, particularly in large graphs. However, techniques like mini-batching are employed to handle large-scale graphs more efficiently.

Censet obtained similar results to EGNN in node classification on CORA, CITESEER and PUBMED, also outperforming non-edge-aware methods. On the Lipophilicity dataset, CensNet achieves better performance than traditional methods like Random Forest and Logistic Regression, showing its effectiveness in leveraging both node and edge information.

3.4.4 NENN

Evolving concepts from both EGNN and CensNet, another attention-based approach is proposed by [Yang and Li., 2020]. They use a hierarchical dual-level attention mechanism to incorporate both node and edge features in a graph neural network. In this method, node and edge attention layers are combined to produce more accurate embeddings, however the importance of node and edge neighborhood is learned dynamically.

There is a loss of edge information in EGNN because the original edge features are only used in the first attention layer. For each subsequent $l + 1$ layer, the attention

coefficient of two connected nodes in the $l-th$ layer is considered as an edge feature. Also, EGNN uses edge features to enhance node embeddings, but not node features to generate edge embeddings, therefore being not suitable for edge-centered tasks. On the other hand, the hierarchical dual-level attention mechanism adopted by NENN alternates the roles of nodes and edges keeping the edge features as a vector rather than a one-dimensional attention coefficient.

CensNet uses both the original graph and the line graph to embed both nodes and edges. However, the use of spectral graph convolution in layer-wise propagation makes it impossible to process large or directed graphs. The idea behind NENN is based on the spatial domain, which enables the processing of various types of graphs. NENN also does not rely on line graphs to perform alternate convolution, instead, it considers both nodes and edges neighborhoods.

The proposed hierarchical dual-level attention mechanism consists of node-level and edge-level attention layers that are stacked alternately to learn both node and edge embeddings. Considering a graph $G(V, E)$ where for each node i, the set of i's first order neighbors is defined by $\mathcal{N}_i$ and the set $\mathcal{N}_j$ represents the node-based neighbors of edge j, $\mathcal{E}_i$ is the set of edges connected to node i and $\mathcal{E}_j$ represents the set of first-order edge neighbors of edge j. The node-level attention layer is defined aiming to learn the neighboring based importance α_{ij}^e for each node. Similarly, importances β_{ij}^n and β_{ij}^e are learned for each edge in the edge-level attention layer. Based on the learned coefficients, the node and edge embeddings are aggregated and updated in order.

At the node level attention layer, the goal is to learn the importance of both node-based and edge-based neighbors for each node. For each node i, an attention score is computed for both its neighboring nodes and edges, as follows:

$$\alpha_{ij}^n = \text{softmax}_j \left(\sigma \left(a_n^T [W_n x_i \| W_n x_j] \right) \right) \tag{3.29}$$

$$\alpha_{ik}^e = \text{softmax}_k \left(\sigma \left(a_e^T [W_n x_i \| W_e e_k] \right) \right) \tag{3.30}$$

Here, α_{ij}^n denotes the attention coefficient of node j to node i, and α_{ik}^e is the attention coefficient of edge k to node i. x is the node feature vector, W_n and W_e are learnable weight matrices, a_n^T and a_e^T are attention vectors and σ is an activation function, such as a LeakyReLU. The computed coefficients are normalized using a softmax function and the node embedding is updated by aggregating the features of its neighbors using the learned attention coefficients.

Similarly, the edge level attention layer focuses on learning the importance of both

node-based and edge-based neighbors for each edge, updating edge embeddings also considering both neighboring nodes and edges. In this case, the attention coefficients are computed for edges and their neighbors:

$$\beta_{ik}^{e} = \text{softmax}_k \left(\sigma \left(q_e^T [W_e e_i \| W_e e_k] \right) \right) \tag{3.31}$$

$$\beta_{ij}^{n} = \text{softmax}_j \left(\sigma \left(q_n^T [W_e e_i \| W_n x_j] \right) \right) \tag{3.32}$$

Here, the attention coefficients β_{ik}^{e} and β_{ij}^{n} define the importance of the neighbors for updating the edge embeddings. Like in the node-level attention, the embeddings are updated by aggregating the neighboring features weighted by the learned importance coefficients.

In node-driven tasks, experiments showed that NENN was able to match or surpass EGNN and CensNet results. Table 3.3 shows a comparison of node classification tasks between traditional GNN methods and methods that incorporate edge features. In all cases, methods that consider edge features have greater accuracy than methods that do not.

After computing the attention coefficients for both node-based and edge-based neighbors, the embeddings are updated by concatenating node-based and edge-based neighbor embeddings:

$$x_i^{l+1} = \text{CONCAT}(x_{Ni}^{l}, x_{Ei}^{l}) \tag{3.33}$$

$$e_i^{l+1} = \text{CONCAT}(e_{Ni}^{l}, e_{Ei}^{l}) \tag{3.34}$$

This alternating process allows both node and edge embeddings to reinforce each other dynamically over multiple layers. Also, since NENN alternates between node-level and edge-level attention layers, the total computational complexity per layer is the sum of the node-level and edge-level complexities. Considering L layers, N nodes, E edges both with $d-$dimensional features, the computational complexity of the entire model can be defined as:

$$O(L \cdot (N + E) \cdot d^2) \tag{3.35}$$

NENN achieves high performance on molecular datasets such as Tox21 and HIV for graph classification and Lipophilicity and Freesolv for graph regression. Also, [Yang and Li., 2020] presents Table 3.3, showing accuracy of semi-supervised node classification tasks in benchmark datasets compared to previous attention-based models. EGAT, CensNet and NENN outperform traditional GNN methods and present very similar results, with NENN slightly outperforming its counterparts.

Table 3.3: Node classification accuracy (in percent) on three citation graph datasets.

Cora			
Label rate	0.5%	1%	3%
GCN	53.66 ± 0.03	62.50 ± 0.02	75.77 ± 0.01
GAT	43.45 ± 0.02	49.66 ± 0:02	57.85 ± 0.04
GraphSAGE	38.48 ± 0.04	50.51 ± 0.03	66.33 ± 0.03
EGAT	61.65 ± 0.02	68.86 ± 0.02	79.69 ± 0.03
CensNet	59.26 ± 0.02	**69.56 ± 0.04**	80.56 ± 0.02
NENN	**65.53 ± 0.02**	69.44 ± 0.02	**82.57 ± 0.02**
Citeseer			
Label Rate	0.3%	0.5%	1%
GCN	39.83 ± 0.02	48.53 ± 0.03	59.47 ± 0.03
GAT	32.53 ± 0.03	39.46 ± 0.05	47.57 ± 0.06
GraphSAGE	28.70 ± 0.04	35.80 ± 0.02	53.36 ± 0.02
EGAT	**51.64 ± 0.02**	57.56 ± 0.03	66.62 ± 0.02
CensNet	51.54 ± 0.02	59.64 ± 0.02	64.34 ± 0.02
NENN	50.66 ± 0.02	**60.64 ± 0.02**	**68.23 ± 0.02**
Pubmed			
Label Rate	0.03%	0.05%	0.1%
GCN	58.84 ± 0.04	66.23 ± 0.03	74.20 ± 0.03
GAT	51.25 ± 0.03	52.56 ± 0.02	61.55 ± 0.03
GraphSAGE	46.00± 0.02	55.42 ± 0.03	60.54 ± 0.10
EGAT	62.58 ± 0.04	64.50 ± 0.02	74.50 ± 0.02
CensNet	63.66 ± 0.02	67.87 ± 0.02	71.78 ± 0.02
NENN	**65.56 ± 0.02**	**69.50 ± 0.02**	**77.70 ± 0.03**

3.4.5 EGAT

Leveraging and enhancing the concepts of two traditional and well established models, namely GCNs and GATs, [Wang et al., 2021] later proposed a different GNN model to exploit edge information on node and graph representation.

In traditional GNN architectures, an adjacency matrix A may be either a binary matrix that indicates the node neighborhood and is used in GAT layers or a one-

dimensional positive-valued matrix of edge features used in GCN layers. This same matrix A is used in every layer of these GNNs. In contrast, an EGAT layer expands either a GCN or GAT layer and uses a *tensor* E containing multi-dimensional positive-valued edge features. Also, the edge features E are adapted at each layer before being passed to the next one.

Given a graph with N nodes and X be a $N \times F$ matrix representation of the features of the N nodes, X_{ij} represents the j^{th} feature of the i^{th} node. $X_i \in \mathbb{R}^F, \forall i \in [0, N]$ representing the F dimensional feature vector of the i^{th} node. In the same manner, let E be a $N \times N \times D$ tensor that represents edge features. Then $E_{ij} \in \mathbb{R}^D, \forall i, j \in [0, N]$ is the D-dimensional vector containing the features of the edge that connects the i^{th} and j^{th} nodes. Also, let $\mathcal{N}_i$ be the set of neighboring nodes of node i. The proposed GNN is based on a multi-layer feedforward architecture. X^0 and E^0 are the network inputs, and after passing through the first EGAT layer, X^0 is adapted to generate a new $N \times F^1$ node feature matrix X^1. Similarly, E^0 is is changed into E^1 which is then fed to the next layer as edge features. These steps are repeated at each subsequent layer, and non-linear activation functions are applied to node features X^l within each hidden layer. The node features N^L are its embedding in a F^L-dimensional space. [Wang et al., 2021] presents two GNN layer designs: (i) attention-based GNN layer blocks, that uses attention coefficients to aggregate both node and edge features and (ii) an adapted convolutional GNN layer that uses adjacency matrices. There are two attention-based blocks, one for nodes and one for edges, that update their respective embeddings in a symmetric parallel way.

The **node attention block** receives a set of node features H and edge features E as inputs and reorganizes the edge features into a form that represents the relationship between connected nodes using an **edge mapping matrix** M_E. This allows the model to capture how edge features influence the interaction between neighboring nodes. For each node i with a neighbor j, attention weights α_{ij} are computed based on both node and edge features as follows:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{h}_i \| \mathbf{h}_j \| \mathbf{e}_{ij}]))}{\sum_{k \in N(i)} \exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{h}_i \| \mathbf{h}_k \| \mathbf{e}_{ik}]))}, \tag{3.36}$$

where h_i and h_j are the feature vectors of nodes i and j and e_{ij} are the features of the edge connecting them. These attention weights are then used to aggregate neighboring node features into a new node representation. A key feature of this block is that edge features only influence the attention weights but are not integrated into the new node features, which helps to maintain clarity and separation between nodes and edges. Similarly to traditional GATs, computational cost of the node attention block is $O(E)$, where E is the number of edges in the graph.

Complementing the node attention block, an **edge attention block** focuses on updating edge features using adjacent edge information. A line graph transformation is applied to a graph G, resulting in $L(G)$, then similarly to its node counterpart, the attention mechanism is applied to $L(G)$ computing attention weights β_{pq} between adjacent edges:

$$\beta_{pq} = \frac{\exp(\text{LeakyReLU}(\mathbf{b}^T[\mathbf{e}_p \| \mathbf{e}_q \| \mathbf{h}_{pq}]))}{\sum_{k \in N(p)} \exp(\text{LeakyReLU}(\mathbf{b}^T[\mathbf{e}_p \| \mathbf{e}_k \| \mathbf{h}_{pk}]))}, \tag{3.37}$$

where e_p and e_q are the edge feature vectors and h_{pq} represents the features of the node connecting these edges. Computational complexity here is significantly higher, due to the line graph transformation, leading to:

$$O\left(\sum_{i=1}^{N} d_i^2\right), \tag{3.38}$$

where d_i is the degree of node i. This quadratic dependence on the node degrees can become expensive for dense graphs. EGAT also requires large memory resources, especially when processing dense graphs or graphs with a large number of features, as the adjacency and mapping matrices, even when stored in sparse form, can be memory-intensive. The model is also primarily designed for undirected graphs and does not naturally support directed graphs or multi-graphs. This limits its applicability in certain domains where directed relationships are important. The total computational complexity of EGAT is:

$$O(E \cdot d) + O\left(\sum_{i=1}^{N} d_i^2\right) \tag{3.39}$$

In comparison to EGNN(A) and NENN, that also use attention mechanisms, EGAT is more computationally expensive for graphs with high-degree nodes, as its complexity depends quadratically on the node degrees due to the edge attention block. This makes EGAT more suitable for sparser graphs with moderate node degrees. While EGNN(A) is more computationally expensive in graphs with many nodes and edge feature channels, as its complexity depends quadratically on the number of nodes and edge dimensions. However, EGNN(A) is particularly well-suited for tasks requiring multi-dimensional edge features. NENN scales linearly with the number of nodes and edges of the graph, making this model the most computationally efficient of the attention-based present thus far.

EGAT is particularly well-suited for applications where both node and edge features are crucial for accurate predictions, such as financial, social and biological networks. For example, on the financial related Trade-B and Trade-M datasets, EGAT achieved classification accuracies of 92.0% and 85.4%, respectively, which is significantly higher than GAT-based models that do not explicitly consider edge features.

Experiments were conducted on classifying nodes on three benchmark citation network datasets: CORA, CITESEER, and PUBMED. The results showed improved results on all datasets compared to traditional GCN and GAT methods, reinforcing the idea that edge features are an important source of information regarding node and graph representation learning.

3.4.6 Graphormer

The introduction of the Transformer architecture marked a significant change in sequential data-based learning tasks, and its adaptation to graph data has led to new frontiers beyond traditional message-passing GNNs. Graphormer [Ying et al., 2021] was a foundational work that demonstrated how to apply a Transformer to graphs, achieving state-of-the-art results. Its main contribution lies in augmenting the standard self-attention mechanism with several structural encodings to make the model aware of the underlying graph topology.

Unlike message-passing GNNs that operate on local neighborhoods, the Transformer's self-attention mechanism allows every node to interact with every other node in the graph. To make this global attention mechanism graph-aware, Graphormer incorporates three key structural features directly into the attention formula. For any two nodes i and j in the graph, the attention score is computed as:

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(i,j)} + b_{\text{edge}(i,j)} \tag{3.40}$$

Here, $(h_i W_Q)$ and $(h_j W_k)$ are the standard query and key projections from the Transformer. The innovation lies in the bias terms:

1. **Centrality Encoding:** Each node's initial feature h_i is augmented with a learnable embedding corresponding to its degree centrality, making the model aware of node importance.

2. **Spatial Encoding $b_{\phi(i,j)}$:** A learnable embedding is added based on the shortest path distance between nodes i and j. This informs the model about the spatial relationship between any pair of nodes, even if they are not directly connected.

3. **Edge Encoding $b_{\textbf{edge}(i,j)}$:** An additional learnable bias is added based on the mean of the features of all edges along the shortest path between nodes i and j.

This makes the Graphormer architecture explicitly edge-aware, as the attention between two nodes is directly influenced by the properties of the connection path between them

Graphormer is limited by its computational complexity. Because it computes attention between all pairs of nodes, its complexity is $O(N^2 d)$, where N is the number of nodes and d is the feature dimension. This quadratic scaling makes it computationally expensive and memory-intensive for large graphs, limiting its practical application to smaller-scale graph datasets, particularly in domains like molecular property prediction, where it has shown exceptional performance.

While NENN and EGAT optimize for sparse graph structures, Graphormer's complexity is dominated by the $O(N^2)$ cost of global self-attention, making it less scalable for large graphs but highly effective for smaller, dense structures like molecules. Table 3.4 summarizes computational costs of the discussed attention-based models.

Table 3.4: Comparison of Computational Complexities Between Attention-based models.

Model	**Complexity per Layer**	**Complexity for L Layers**
EGNN(A)	$O(N^2 Pd) + O(N^2 d^2)$	$O(L \cdot (N^2 Pd + N^2 d^2))$
NENN	$O((N+E) \cdot d^2)$	$O(L \cdot (N+E) \cdot d^2)$
EGAT	$O(E \cdot d) + O\left(\sum_{i=1}^{N} d_i^2\right)$	$O(L \cdot (E \cdot d + \sum_{i=1}^{N} d_i^2))$
Graphormer	$O(N^2 \cdot d)$	$O(L \cdot N^2 \cdot d)$

3.4.7 DHT

The previous edge embedding learning methods based on line graphs show some limitations: (i) two different graphs may be transformed in the same line graph, as the transformation is not injective; (ii) the transformation is not scalable; (iii) original node information may be lost during the transformation. To overcome these drawbacks, but still following the idea of performing graph transformations, [Jo et al., 2021] proposes an edge embedding learning framework based on a *Dual Hypergraph Transformation* (DHT). Hypergraphs can model high-order node interactions by grouping multi-node relationships into a single hyperedge. The relation between the original and the transformed hypergraph is also exploited for edge representation learning.

Unlike traditional graphs, hyperedges from hypergraphs may connect any arbitrary number of nodes. A basic graph can be described as an incidence matrix $A \in \{0,1\}^{n \times n}$ which represents the connections between n nodes. On the other hand, hypergraphs have higher order relations that are described as a matrix $M \in \{0,1\}^{n \times m}$. The main difference here is that the interactions between n nodes and m hyperedges

are represented. Formally, a hypergraph G^* with n nodes and m edges is defined by $G^* = (X^*, M^*, E^*)$, where $X^* \in \mathbb{R}^{n \times d}$ is the set of node features, $M \in \{0,1\}^{n \times m}$ is the incidence matrix of the hypergraph and $E^* \in \mathbb{R}^{n \times d'}$ the set of hyperedge features. Traditional graphs can also be represented in the form of a hypergraph $G = (X, M, E)$, where a hyperedge on M is associated only with two nodes.

Given a graph $G = (X, M, E)$, where $X \in \mathbb{R}^{n \times d}$ are the node features, $E \in \mathbb{R}^{n \times d}$ are the edge features and $M \in 0, 1^{n \times m}$ is the incidence matrix representing node and edge connectivity.

During the transformation, the structural role of nodes and edges is interchanged and the incidence matrix for the new hypergraph is obtained by transposing the incidence matrix from the original graph. The edge and node features are also naturally interchanged across G and G^*.

$$DHT : G = (X, M, E) \mapsto G^* = (E, M^T, X), \tag{3.41}$$

where G^* is referred to as the dual hypergraph of the input graph G. Since G^* retains all of the original graph information, G can be reconstructed as follows:

$$DHT : G^* = (E, M^T, X) \mapsto G = (X, M, E), \tag{3.42}$$

The dual hypergraph transformation allows for message-passing schemes to be used on the edges of the original graph G by performing message-passing between the nodes of the hypergraph. Node features are aggregated using an AGGR function and the representation at layer $E_e^{(l+1)}$ is updated accordingly, formally:

$$E_e^{(l+1)} = UPD(E_e^{(l)}, AGGR(\{E_f^{(l)} : \forall f \in \mathcal{N}(e; M^T)\})) \tag{3.43}$$

where the set of features from node $e \in G^*$ at the l-th layer is represented as $E^{(l)}$. The AGGREGATE function combines these features with the messages received from the neighbors of node e, and the value $E_e^{(l+1)}$ is updated by the UPD function to represent the new embedded value of e at layer $l+1$. $\mathcal{N}(e; M^T)$ represents the set of neighbors of node e in G^* and can be obtained using the incidence matrix M^T of G^*. If applied to G^*, any message passing GNN model may be used to learn the edge representations from the original G graph, summarizing:

$$E^{(l+1)} = GNN(E^{(l)}, M^T, X^{(l)}) = EHG(X^{(l)}, M, E^{(l)}) \tag{3.44}$$

where EHG is the proposed framework that performs edge representation learning using DHT.

In addition to the DHT framework, [Jo et al., 2021] proposes two novel edge pooling methods for graph-level embedding. In order to reduce complexity and remove unnecessary information, graph pooling methods can be used to reduce the size of a graph via compression or pooling operations such as mean, sum, or max used over all node representations. Traditional pooling methods, however, completely ignore edge representations, also altering the node set. This leads to an inevitable loss of both node and edge information. To overcome this limitation, *HyperCluster* and *HyperDrop* are proposed.

HyperCluster is an edge clustering method designed to coarsen similar edges into a single edge to obtain a global edge representation. A clustering scheme for nodes can be defined by:

$$X^{pool} = C^T X', M^{pool} = C^T M, \tag{3.45}$$

where the pooled representations are $X^{pool} \in \mathbb{R}^{n_{pool} \times d}$ and $M^{pool} \in \mathbb{R}^{n \times m_{pool}}$, $X' = GNN(X, M, E) \in \mathbb{R}^{n \times d}$ is the set of updated node features and the cluster assignment matrix $C \in \mathbb{R}^{n \times n_{pool}}$ is generated from X'. After the resulting dual hypergraph $E' = EHGNN(X, M, E) \in \mathbb{R}^{m \times d'}$ is obtained, the nodes from that hypergraph are clustered by HyperCluster in the following manner:

$$E^{pool} = C^T E', (M^{pool})^T = C^T M^T, \tag{3.46}$$

where $E^{pool} \in \mathbb{R}^{m_{pool} \times d'}$ is the pooled edge representation, $M^{pool} \in \mathbb{R}^{n \times m_{pool}}$ the incidence matrix of the original input graph, and the cluster assignment matrix $C \in \mathbb{R}^{m \times m_{pool}}$ is generated from the input edge features E'. The fact that instead of dropping the edges, HyperCluster coarsens them, makes this method appropriate for tasks such as graph reconstruction.

On the other hand, **HyperDrop** is a method that drops unnecessary edges to identify the specific task-relevant edges. Traditional node drop methods remove the less relevant nodes based on scores:

$$X^{pool} = X_{idx}, M^{pool} = M_{idx}; idx = top_k(score(X)), \tag{3.47}$$

where idx is the node-wise indexing vector, $score(X)$ computes the score of each node and top_k selects the top k higher scored elements. This approach leads to a loss

of node information, as it drops nodes. **HyperDrop** overcomes this issue by dropping edges obtained from the EHGNN model. The pooling method is as follows:

$$E^{pool} = E_{idx}, (M^{pool})^T = M^T_{idx}; idx = top_k(score(E)). \tag{3.48}$$

The resulting pooled graph $G^{pool} = (X, M^{pool}, E^{pool})$ is obtained by applying DHT to the pooled hypergraph. This method is suitable for graph and node-level classification. As **HyperDrop** learns to remove unnecessary edges, it reduces the message passing scheme to only happen across relevant nodes, alleviating the over-smoothing problem in deep GNNs.

Transforming a graph into a hypergraph involves changing the roles of nodes and edges, but no additional graph construction is necessary beyond the initial transformation. Thus the computational complexity of this step is linear with respect to the number of nodes and edges of the original graph, being:

$$O(N + E) \tag{3.49}$$

Once the graph is transformed into a hypergraph using DHT, the next step involves applying L layers of message-passing algorithms (e.g., GCN or GAT) to the edges of the original graph, which are now treated as nodes in the hypergraph. The message-passing operation is used to update edge embeddings by considering the relationships between edges (via the hyperedges, which correspond to the original nodes). For each layer of message-passing in a GNN, the complexity typically scales with:

$$O(L \cdot E \cdot d) \tag{3.50}$$

Therefore the total computational complexity of DHT avoids the quadratic cost of similar line graph transformation based methods such as CensNet, being defined as:

$$O(N + L \cdot E \cdot d) \tag{3.51}$$

3.4.8 LeL-GNN

Link prediction is an edge-oriented task commonly addressed by GNNs. It is usually divided into three steps: i) subgraph extraction; ii) feature extraction; and iii)

classification-based link prediction. Message passing schemes, in which GNNs are based, aggregate information from a subgraph composed of a node i's neighborhood. If more depth is added to the network, node embeddings tend to contain too much information redundancy, thus leading to oversmoothing, where models can be incapable of differentiating nodes. In order to overcome this limitation, LeL-GNN [Morshed et al., 2023] propose a learnable edge sampling for subgraph extraction combined with a line graph transformation that turns link prediction into a node classification problem.

The process of learnable edge sampling in LeL-GNN involves selecting a subset of edges in an undirected graph $G = (V, E)$. This selective sampling is designed to preserve essential spectral features of the graph. Considering two nodes i and j, where $(i, j) \in E$, the probability p_{ij} of sampling, is obtained considering the total number i and j shared neighbors c_{ij} as follows:

$$p_{ij} = \frac{\frac{2}{c_{ij}+2}}{\sum_{(i,j)\in E} \frac{2}{c_{ij}+2}}. \tag{3.52}$$

A graph sparsifier H, i.e. a subgraph that maintains the connectivity features of the parent graph, is built with m randomly selected edges from G. A learnable parameter α is used to assess the quality of the local network connection as well as determine the number of edges that must be sampled to keep the major spectral feature of the graph. This parameter is defined as:

$$\alpha = \frac{1}{n} \sum_{(i,j)\in E} \frac{2}{c_{ij}+2} \tag{3.53}$$

Given a graph $G^k_{v_1,v_2}$, defined as a k-hop bounding subgraph centered in 2 specific nodes v_1 and v_2 from G, a GNN can be used to predict if a link exists between these nodes. This graph is built as follows:

$$G^k_{v_1,v_2} = \{v | \min(d(v, v_1), d(v, v_2)) \leq k\}, \tag{3.54}$$

where $d(v, v_1)$ is the shortest path between v and v_1.

The nodes from $G^k_{v_1,v_2}$ are then labeled according to their relevance for the graph structure, following the method proposed in [Zhang and Chen, 2018]. The next step is the edge sampling, where $\alpha n \log n$ edges from $G^k_{v_1,v_2}$ are selected, where α is learned

from equation 3.53. Edge features $l_{(v_i,v_j)}$ can be extracted by two methods: i) if the graph is not attributed, the edge features are obtained by concatenating the minimum, maximum and average values of the node labeling function applied to v_i and v_j; ii) if node attributes exist, then they are also concatenated with the previous values. Finally, the sampled subgraph $G^k_{v_1,v_2}$ is transformed into the line graph $L(G^k_{v_1,v_2})$.

In the newly created line graph, the previously extracted features $l_{(v_i,v_j)}$ act as node attributes, and the task of link prediction becomes a problem of node classification. This preprocessing allows for a better representation of the graph structure empowered by an edge-aware mechanism, therefore a regular GNN can be applied to the line graph for feature extraction and node classification.

Considering E edges in a graph, the computational complexity of the learnable edge sampling step is:

$$O\left(\sum_{(i,j)\in E} \min(d_i, d_j)\right), \tag{3.55}$$

where d_i and d_j are the degrees of nodes i and j. This is linear for sparse graphs, however it can approach $O(E \cdot d_{max})$ for dense graphs, where d_{max} is the maximum degree of any node.

Constructing the line graph depends on identifying pairs of edges that share a common node. For each node i in the original graph, this involves comparing its adjacent edges, which takes $O(d_i^2)$. Therefore, the total complexity for line graph construction is:

$$O\left(\sum_{i=1}^{N} d_i^2\right) \tag{3.56}$$

This can be quadratic for nodes with high degrees, making the worst-case complexity $O(N \cdot d_{max}^2)$.

Once the line graph is constructed, a traditional message passing scheme is applied, where for L layers, the overall computational complexity is given by:

$$O(L \cdot E \cdot d_{avg} \cdot d), \tag{3.57}$$

where d_{avg} is the average degree of nodes in the original graph.

In summary, the total complexity of the LeL-GNN model for sparse graphs becomes:

$$O(E) + O(L \cdot E \cdot d_{\text{avg}} \cdot d) \tag{3.58}$$

For dense graphs with high-degree nodes, the total complexity may approach:

$$O(N \cdot d_{\text{max}}^2) + O(L \cdot E \cdot d_{\text{avg}} \cdot d) \tag{3.59}$$

Experiments were conducted on 10 different datasets, and results compared to state-of-the-art models showed that the proposed LeL-GNN outperforms most of the baseline comparison methods, such as Node2Vec and GraphSAGE, also converging significantly faster than any other model. They highlight comparisons with SEAL [Zhang and Chen, 2018], a a popular neural network based method for link prediction that relies heavily on pooling operations, which can lead to information loss. In contrast, LeL-GNN avoids pooling by using line graph transformation and edge sampling, which leads to faster convergence and better performance, especially on larger and denser graphs.

Table 3.5 summarizes key aspects of the transformation based edge-aware GNN methods presented so far.

Table 3.5: Comparison between transformation based edge-aware GNN methods.

	CensNet	**DHT**	**LeL-GNN**
Transformation	Line Graph	Dual Hyperedge	Line Graph/Edge Sampling
Edge Sampling	No	No	Learnable
Complexity	$O\left(\sum_{i=1}^{N} d_i^2 + L \cdot E \cdot d^2\right)$	$O(N + L \cdot E \cdot d)$	$O\left(N \cdot d_{\text{max}}^2 + L \cdot E \cdot d_{\text{avg}} \cdot d\right)$

3.5 Summary

The advent of GNNs marked a paradigm shift for edge-aware learning, moving from sampling-based approaches to an end-to-end deep learning framework. The key lesson from this body of work is the diversity of architectural strategies for integrating edge information, each with distinct trade-offs. A major architectural divergence emerged between methods that transform the graph (e.g., to a line graph or hypergraph) to re-frame the edge task as a node task, and native methods that operate on the original graph by developing novel integration mechanisms. As we have seen, these native mechanisms are primarily based on either Edge-Conditioned Convolutions or Edge-Informed Attention. While transformation-based methods can be powerful, they often introduce significant computational complexity that scales poorly

with node degree. Native GNNs and, more recently, Graph Transformers, offer a more direct and often more scalable approach, though they present their own design challenges. This versatile and rapidly evolving toolkit of GNN architectures has been instrumental in successfully applying edge-aware learning to the wide range of complex, real-world problems discussed in the following section.

To provide a clear overview of the computational trade-offs discussed throughout this section, Table 3.6 explicitly compares the computational complexity of the reviewed edge-aware learning methods.

Table 3.6: Comparison of Computational Complexity

Method	**Computational Complexity (Big-O)**	**Key Scalability Factor**
Random Walk-Based		
Line2Vec	$O(Nd_{max}^2 + Ed_{max}^2 \log N)$	Node Degree (Quadratic)
AttrE2Vec	$O(E(k \cdot L + d))$	Number of Edges (Linear)
edge2vec	$O(N \cdot r \cdot l \cdot T + C \cdot d \cdot k)$	Nodes and RW Parameters (Linear)
GNN-Based		
EGNN(A)	$O(N^2Pd + N^2d^2)$	Number of Nodes (Quadratic)
CensNet	$O(L(Ed + Nd))$	Nodes and Edges (Linear)
NENN	$O(L(N + E)d^2)$	Nodes and Edges (Linear)
EGAT	$O(\sum_{i=1}^{N} d_i^2)$	Node Degree (Quadratic)
DHT	$O(N + L \cdot E \cdot d)$	Number of Edges (Linear)
LeL-GNN	$O(\sum_{i=1}^{N} d_i^2)$	Node Degree (Quadratic)
Graphormer	$O(N^2d)$	Number of Nodes (Quadratic)

Chapter 4

Surveiling Edge-aware Graph Learning Applications

In this chapter, we present recent works that aim to solve real-life problems by using graph-structured data and applying edge-aware learning techniques to enhance previous results. In contrast to the previous section, here we focus on specific scenarios that present real problems for which better solutions can be achieved by using edge-aware GNNs. We first discuss few-shot learning, a class of machine learning problems that aims to solve prediction and classification tasks with only a small set of labeled data. Then we present the results of applying edge-aware graph embedding to solve biomedical-specific problems such as drug discovery and bioactivity between a compound and a protein. Next, we show edge-aware graph embedding applied to point clouds that represent real-life objects scanned on 2D or 3D spaces. We follow with a groundbreaking work that may reduce chip design time from several months to only six hours. We also show an edge-aware GNN applied to enhance the resilience of food supply chains, a critical task for reducing global food insecurity. We conclude with two modern computer network based tasks, namely traffic prediction and radio resource management in wireless networks.

4.1 Few-shot learning

A special class of machine learning tasks that aims to automatically and efficiently make predictions with few labeled data is defined as few-shot learning [Snell et al., 2017]. In these tasks, a classifier must be adapted to recognize new object classes not previously seen in training, given only a few examples of each of the classes. Retraining the model is a naive and often costly approach, which may also cause severe overfitting. Despite being computationally challenging, it has been demonstrated that humans are capable of performing even one-shot classification, where only one example of each new class is provided, with high accuracy [Lake et al., 2011]. Formally, in few-shot learning, a small support set $S = \{(x_1, y_1), ..., (x_n, y_n)\}$ of n labeled example objects where each $x_i \in \mathbb{R}^D$ is the D-dimensional feature vector of an object and

$y_i \in \{1, ..., K\}$ is the corresponding label. S_k is the set of example objects labeled with class k.

Many few-shot learning algorithms [Vinyals et al., 2016; Snell et al., 2017; Mishra et al., 2017; Sung et al., 2018; Oreshkin et al., 2018] require full analysis of the complex relationships between a support set and a query. In that sense, GNNs arise as a natural approach for few-shot learning problems, since they are based on iterative feature aggregation from neighboring nodes by message passing.

Given a few-shot task with its support set, [Garcia and Bruna, 2018] proposed building a graph in which every example object from the support set and a query are densely connected. The input nodes are represented as embeddings generated for example by the output of a convolutional neural neural network and the initially label information, which can be given by a hot encoding. The node features are then iteratively updated via neighborhood aggregation to classify the unlabeled query. [Liu et al., 2018] proposed TPN, a transductive propagation network on the node features obtained from a deep neural network. This network iteratively propagates, at test-time and throughout the whole support and query sets, the one-hot encoded labels combined with a common graph parameter set.

The majority of GNN approaches to few-shot learning are based on frameworks that perform node-labeling by implicitly modeling similarities and differences within and between clusters. Here, clusters represent classification classes. To explicitly perform the clustering with representation and metric learning, an edge-labeling framework must be considered. In graphs for correlation clustering [Kim et al., 2011] and GNNs for citation networks or dynamic systems [Gong and Cheng., 2019; Kipf et al., 2018], the use of edge labels to specify whether two connected nodes are members of the same cluster or class has been proposed. The first edge-labeling GNN (ELGNN) for few-shot learning, specifically for the task of few-shot classification, was proposed by [Kim et al., 2019].

The proposed ELGNN consists of layers composed of a node update block and an edge update block. Therefore, in each layer, not only the node features are updated but the edge features are also explicitly adjusted reflecting the edge labels of the two connected node pairs and exploiting both intra-cluster similarity and inter-cluster dissimilarity.

Each few-shot classification task T contains a labeled support set of input-label pairs set S, and an unlabeled query set Q, which is used in the evaluation of the learned classifier. If for each N unique class, there are K labeled samples in the support set S, the problem is defined as *N-way K-shot* classification.

In the ELGNN approach, first, the feature representations of all samples needed for the given task are obtained from a convolutional neural network, and then a fully connected graph, where nodes are samples and edges represent the relations between

these samples, is built. The model is designed with L layers that process the graph, and the forward propagation of ELGNN is based on alternatively updating node and edge features through each of these layers.

Formally, given task T with a total sample number of $\mid T \mid = N \times K + T$, a graph $G = (V, E; T)$ where V is the set of nodes and E the set of edges, is built. Then if v_i and e_ij are respectively node features of V_i and edge features of E_{ij}, for every edge, its ground truth label y_{ij} will be defined as 1 if $y_i = y_j$ or 0 otherwise.

The two dimensional vector of edge features $e_{ij} = \{e_{ijd}\}_{d=1}^{2} \in [0,1]^2$ contains the normalized weights of the intra- and inter-cluster relations from two connected nodes. In this way, both the intra-cluster similarity and the inter-cluster dissimilarity are separately considered.

Node features are initialized based on the output of the convolutional embedding network $v_i^0 = f_{emb}(X_i; \theta_{emb})$, where θ_{emb} is the corresponding parameter set. Edge features are initialized by edge labels as follows (consider $||$ the concatenation operation):

$$e_{ij}^0 = \begin{cases} \mid 1 \mid\mid 0 \mid, & \text{if } y_{ij} = 1 \text{ and } i, j \leq N \times K, \\ \mid 0 \mid\mid 1 \mid, & \text{if } y_{ij} = 0 \text{ and } i, j \leq N \times K, \\ \mid 0.5 \mid\mid 0.5 \mid, & \text{otherwise.} \end{cases} \tag{4.1}$$

L layers are used by the model for graph processing, and the forward propagation of ELGNN for inference is based on alternatively updating node and edge features through layers. Formally, let v_i^{l-1} and e_{ij}^{l-1} be respectively nodes, and an edge features from the layer $l-1$, at layer l the feature node v_i^l is updated by aggregating the features of other nodes proportionally to their edge features. Then the feature transformation is performed. An attention mechanism that considers the edge feature e_{ij}^{l-1} at the layer $l-1$ is employed as a measure of the corresponding neighbor node's level of contribution. Additionally to the conventional intra-cluster aggregation, which provides information about the similar neighbors of a node, inter-cluster aggregation, which indicates non-similar neighbors, is also considered.

$$v_i^l = f_v^l([\sum_j \tilde{e}_{ij1}^{l-1} v_j^{l-1} \mid\mid \sum_j \tilde{e}_{ij2}^{l-1} v_j^{l-1}]; \theta_v^l), \tag{4.2}$$

where $\tilde{e}_{ijd} = \frac{e_{ijd}}{\sum_k e_{ikd}}$, and f_v^l is the node feature transformation network with the parameter set θ_v^l. Additionally to the conventional intra-cluster aggregation, which provides information about the similar neighbors of a node, inter-cluster aggregation, which indicates non-similar neighbors, is also considered.

Based on the newly updated node features, the edge feature update is performed

by aggregating the previous edge representation value with the recalculated similarities between each pair of nodes:

$$\bar{e}^l_{ij1} = \frac{f^l_e(v^l_i, v^l_j; \theta^l_e) e^{l-1}_{ij1}}{\sum_k f^l_e(v^l_i, v^l_k; \theta^l_e) e^{l-1}_{ik1} / (\sum_k e^{l-1}_{ik1})}, \tag{4.3}$$

$$\bar{e}^l_{ij2} = \frac{(1 - f^l_e(v^l_i, v^l_j; \theta^l_e)) e^{l-1}_{ij2}}{\sum_k (1 - f^l_e(v^l_i, v^l_k; \theta^l_e)) e^{l-1}_{ik2} / (\sum_k e^{l-1}_{ik2})}, \tag{4.4}$$

$$e^l_{ij} = \bar{e}^l_{ij} / \mid\mid \bar{e}^l_{ij} \mid\mid_1, \tag{4.5}$$

where f^l_e is the metric function that computes similarity scores with the parameter set θ^l_e.

After each layer, node features are updated, then the edge features update is performed by aggregating the previous edge representation value with the recalculated similarities between each pair of nodes. More specifically, the node feature is aggregated to edges, and the elements of the edge feature vector are updated separately from each normalized intra-cluster similarity or inter-cluster dissimilarity. Hence, every edge representation is updated based on the relations of its connected pair of nodes as well as on the relations between other pairs of nodes.

The edge-label prediction can be obtained from the final edge feature after L number of alternative node and edge feature updates, i.e. $\hat{y} = e^L_{ij1}$, where the probability that the two nodes V_i and V_j are from the same class is defined by $\hat{y}_{ij} \in [0, 1]$. A simple weighted vote supported by set labels and predicted edge labels can be used to classify each node V_i. The prediction probability of node V_i can be expressed as $P(y_i = C_k \mid T) = p^{(k)}_i$, where:

$$p^{(k)}_i = softmax(\sum_{\{j: j \neq i \wedge (x_j, y_j) \in S\}} \hat{y}_{ij} \delta(y_i = C_k)) \tag{4.6}$$

with $\delta(y_i = C_k)$ being equal to one when $y_i = C_k$ and zero otherwise.

Computational complexity of ELGNN can be analyzed in by its three main steps, node feature update, edge feature update and classification. For each node i, the node feature update step requires summing the contributions of each of its neighbors j weighted by the corresponding edge features e_{ij}. This operation has a cost of:

$$O(N \cdot d^2 + E \cdot d), \tag{4.7}$$

where $N \cdot d^2$ accounts for the feature transformation operation applied to each node, and $E \cdot d$ is the cost of aggregating information from all edges connected to each node.

Edge feature update for each edge (i, j) involves calculating the similarity between the features of nodes i and j, incorporating the similarity metric $f_e(v_i, v_j)$ and previous edge feature values. This process has complexity:

$$O(E \cdot d^2), \tag{4.8}$$

At the classification step, after the final edge update in the last layer, the model assigns labels to edges based on the final edge embeddings. For each edge, this step has a complexity of $O(d)$, as it usually involves a similarity computation or a simple function on the final edge embedding. Therefore for E edges, the edge label prediction has complexity:

$$O(E \cdot d) \tag{4.9}$$

For each query node, a voting mechanism (or softmax aggregation) is applied based on the predicted edge labels between the query and the support nodes. Assuming there are N_q query nodes and each is connected to k support nodes, also considering that N_q is on the order of N this process has a complexity of:

$$O(N \cdot k) \tag{4.10}$$

Thus, the overall computational complexity of ELGNN can be summarized as:

$$O(L \cdot (N \cdot d^2 + E \cdot d^2) + E \cdot d + N \cdot k) \tag{4.11}$$

To evaluate and compare ELGNN with state-of-the-art few-shot learning approaches, two datasets specifically designed as benchmarks were used: miniImageNet [Vinyals et al., 2016] and tieredImageNet [Ren et al., 2018]. The experimental results showed that the proposed ELGNN outperformed other state-of-the-art few-shot learning algorithms on both the supervised and semi-supervised image classification tasks, indicat-

ing that learning and exploiting edge information can increase accuracy on few-shot classification tasks.

4.2 Biomedical Knowledge Discovery

Biomedical sciences are a complex domain of knowledge that represents the biological interactions between entities such as proteins, genes, drugs, diseases, and phenotypes. As such, biomedical knowledge data can often be represented as rich heterogeneous networks. According to [Wilcke et al., 2017], the knowledge graph is a suitable candidate to be the default data model for learning heterogeneous networks knowledge. This special type of data structure can represent nodes and edges of different types and semantics, and knowledge graph methods that consider surrounding node and edge contexts support a rich expanding feature set. In the context of biomedical science, knowledge graphs allow for a variety of statistical knowledge to be inferred, such as probabilistic associations between genes and phenotypic traits.

As opposed to most traditional embedding methods [Grover and Leskovec, 2016; Perozzi et al., 2014; Tang et al., 2015] which were designed to represent homogeneous networks, metapath2vec [Dong et al., 2017] proposed to incorporate meta paths with node semantics to embed nodes from knowledge graphs. However, besides being dependent on previous domain knowledge, metapath2vec considers only node types, disregarding valuable edge information.

To overcome this limitation, edge2vec [Gao et al., 2019] was proposed. To improve node context representation, an Expectation-Maximization model is applied to an edge-type transition matrix. At the maximization step, the edge-type matrix is used to create sequences of nodes based on random walks executed in a knowledge graph. At the expectation step, the edge-type matrix is optimized using the node context generated from node embeddings as feedback. To allow scalability and improve performance, a skip-gram sampling strategy is also used to select partial nodes for the Expectation-Maximization model. The goal is to represent both topologically and semantically similar nodes in similar embeddings.

More specifically, the model processing starts with the creation of an edge-type transition matrix M, that differentiates edge types and influences the probability of transitioning from one node to another based on edge semantics. In the context of biomedical data, an edge type can be "inhibition" between compounds, "activation" between proteins, or "binding" between compounds and proteins, for example. Each entry $M_{(e_i,e_j} \in M$ represents the probability of transitioning from an edge type e_i to an edge type e_j, and the matrix is optimized iteratively to learn effective transition weights for different edge types, aligning the model with real-world relational patterns in the data.

Once M is initialized, edge2vec performs edge-aware random walks on the graph, generating sequences of nodes that respect the edge types encountered during transition. As transition probabilities between nodes are controlled by the entries in M, the paths of the random walks are more likely to follow specific edge-type patterns. Also, each node sequence generated by these random walks captures not only node neighborhoods but also edge semantics, preserving the graph's multi-relational structure in the generated sequences.

The Expectation-Maximization model is used as an optimization tool to refine the edge-type transition matrix. At the expectation step, given the current state of the transition matrix M, edge-aware random walks are conducted to create node sequences that account for edge types. This process builds a "corpus" of paths, where each path implicitly encodes the edge-type dependencies in the transition matrix. Then, at the maximization step, these node sequences are used to update M based on the observed co-occurence of edge types in the paths. Formally:

$$M(e_i, e_j) = \text{Sigmoid}\left(\frac{\mathbb{E}\left[(\vec{v}_i - \mu(\vec{v}_i))(\vec{v}_j - \mu(\vec{v}_j))\right]}{\sigma(\vec{v}_i)\sigma(\vec{v}_j)}\right), \tag{4.12}$$

where $\vec{v}_i$ and $\vec{v}_j$ are the feature vectors of nodes connected by edges of types e_i and e_j, with $\mu(\vec{v}_j)$ and $\sigma(\vec{v}_i)$ representing the mean and standard deviation of node embeddings associated with edge type e_i. A sigmoid function is used to normalize values, helping to constrain the transition probabilities between edge types and ensuring that edge semantics are consistently reinforced across different random walks.

After obtaining a corpus of node sequences that capture edge semantics, edge2vec uses the skip-gram model from word2vec to learn embeddings for the nodes. The skip-gram model is adapted here to maximize the probability of neighboring nodes within the generated paths, taking into account the edge-type context. Here the defined objective is to maximize the probability of observing a neighboring node c given a node v, weighted by the edge-type semantics encoded in the transition matrix M:

$$\text{argmax}_{\theta,M} \prod_{v \in V} \prod_{c \in N(v)} p(c|v;\theta;M), \tag{4.13}$$

where θ are the node embedding parameters and $N_{(v)}$ is the neighborhood of node v according to the random walks.The probability p of a neighboring node c given a node v, considering edge features is:

$$p(c|v;\theta;M) = \frac{\exp(\vec{v} \cdot \vec{c})}{\sum_{c' \in V} \exp(\vec{v} \cdot \vec{c'})}, \tag{4.14}$$

where $\vec{v}$ and $\vec{c}$ are the embeddings of nodes v and c, respectively, and V is the set of all nodes.

During training, a negative sampling strategy is used to improve computational efficiency. This involves sampling node pairs not expected to co-occur, hence negative node pairs, to adjust the embeddings, ensuring that they reflect meaningful similarities based on edge semantics:

$$\log \sigma(\vec{v} \cdot \vec{c}) + \sum_{n=1}^{k} \mathbb{E}_{c_n \sim P_n(c)} \left[\log \sigma(-\vec{v} \cdot \vec{c_n}) \right], \tag{4.15}$$

where σ is the sigmoid function, $P_n(c)$ is the noise distribution from which negative samples c_n are drawn and k is the number of negative samples.

The computational complexity of the edge2vec model can be broken down by examining its primary components: the edge-type transition matrix initialization and update, the edge-aware random walk, and the skip-gram embedding learning process with negative sampling. Each of these contributes to the overall complexity of the model in unique ways, as edge2vec incorporates edge semantics in heterogeneous graphs. Initializing the edge-type transition matrix M for an edge-graph with T edge types involves setting up a $T \times T$ matrix, which has complexity $O(T^2)$. For each edge pair in M, calculating the correlation term has complexity $O(d)$, where d is the dimensionality of node embeddings. Hence for all $T \times T$ entries, the entire process has complexity of:

$$O(T^2 \cdot d) \tag{4.16}$$

Assuming that r random walks of l length are conducted on a graph with N nodes, and that a constant factor related to the matrix lookup must be added, since each walk step depends on the edge type of the current edge and its neighbors, the total complexity for the edge-aware random walks can be determined as:

$$O(N \cdot r \cdot l \cdot T) \tag{4.17}$$

Finally the skip-gram model is applied to learn node embeddings by maximizing

the likelihood of co-occurring nodes in the generated paths. Negative sampling is used to approximate the softmax function during training, improving computational efficiency. Let C be the total number of node pairs across all paths, and k the number of negative samples per node, the complexity of training the skip-gram model with negative sampling is given by:

$$O(C \cdot d \cdot k) \tag{4.18}$$

Therefore, the total complexity of the edge2vec model can be summarized as:

$$O(T^2 \cdot d + N \cdot r \cdot l \cdot T + C \cdot d \cdot k) \tag{4.19}$$

Within the biomedical context, the tasks involved in drug discovery are diverse. For example, the evaluation of drug safety and efficacy depends on the analysis of complex biomolecular relations. Although the possibility of predicting compound-target bioactivity is very challenging, it is also highly valuable for developing new drug leads and hypotheses as well as for clarifying the mechanism of action of existing drugs and compounds. This rich and complex knowledge domain is used as context by [Gao et al., 2019], which applies the edge2vec model on Chem2Bio2RDF [Chen et al., 2009], a highly heterogeneous graph integrating over 25 biomedical and drug discovery datasets. Table 4.1 shows that the classification results obtained of edge2vec on Chem2Bio2RDF outperform node embedding methods that do not consider edge semantics.

The four metrics used to compare classification tasks are precision, recall, F1 score, and hamming loss. Precision reflects the fraction of correct positive results among the returned instances; recall implies the ratio of correct positive results returned; F1 is the harmonic mean of precision and recall. Those metrics range from 0 to 1, and the higher the value the better. On the other hand, the Hamming loss also ranges from 0 to 1, but as it represents the fraction of labels that are incorrectly predicted, we aim for a lower value.

Table 4.1: Classification on node labels in the medical network

Algorithm	**Precision**	**Recall**	**F1**	**H. loss**
Deepwalk [Perozzi et al., 2014]	0.5624	0.5708	0.5650	0.4291
LINE [Tang et al., 2015]	0.6366	0.6390	0.6279	0.3609
node2vec [Grover and Leskovec, 2016]	0.5652	0.5656	0.5622	0.4343
edge2vec [Gao et al., 2019]	**0.7554**	**0.7546**	**0.7544**	**0.2453**

Specifically for biomedical applications, the ability to predict bioactivity between a compound and protein target is of high relevance and can contribute to accelerating early-stage drug discovery [Gao et al., 2019]. This real-world bioactivity prediction task is used as a validation task and compared with another biomedical-specific method [Dong et al., 2017] and also with the three previous baseline models. Compound-gene search ranking is another useful bioactivity prediction task. It consists of predicting the ranking of hits based on a metric that increases the success probability and overall efficiency in potentially costly sub-sequential efforts. We can compare this to a search algorithm, where the number of hits returned is usually less important than the ranking, particularly the top-ranked hits. Hence this can be defined as an information retrieval or search efficiency task. Edge2vec outperforms baseline models in these two applications as well.

4.3 Learning on Point Clouds

Point clouds, i.e. collections of points scattered in a 2D or 3D space representing an object, are a simple yet very important type of data structure [Qi et al., 2017]. 3D scanning and sensing technologies often have point clouds as their data output. Some of the many recent applications of point cloud processing include robotics [Rusu et al., 2008b], indoor navigation [Zhu et al., 2017], self-driving vehicles [Liang et al., 2018; Qi et al., 2018; Wang et al., 2018] and shape synthesis and modeling [Golovinskiy et al., 2009; Guerrero et al., 2018].

These are applications that demand high processing of point cloud data. To perform tasks such as point cloud classification and segmentation, geometrical and semantic features must be identified, and traditional methods used to solve these tasks employ handcrafted features [Lu et al., 2014; Rusu et al., 2008a, 2009]. The growing interest and application of deep learning techniques, however, have motivated a learning-driven approach to identify point cloud features. Methods based on such an approach are rapidly developing and outperform traditional techniques in various tasks [Chang et al., 2015].

As scattered, irregularly structured, and containing semantic and geometry relations between its data, point clouds share many similarities with graphs. It is also not possible to directly apply traditional structure and position-based deep learning models such as CNNs on point clouds. A common approach is to first convert raw point cloud data into a 3D grid [Maturana and Scherer, 2015; Wu et al., 2014] and then use deep learning models to process the data. This inevitably adds increased computational complexity and memory usage to the task. The first approach designed specifically to directly handle raw point cloud data was PointNet [Qi et al., 2017]. The proposed method, however, treats points independently at a local scale and overlooks

geometric relationships between points.

To address this limitation, Dynamic Graph CNN (DGCNN) [Wang et al., 2019] was proposed as an operation that, instead of generating features based directly on point embeddings, considers the edge features describing relationships between points and their neighbors to produce embeddings. DGCNN aims to capture both local and global geometric structures in point clouds, maintaining permutation invariance and translation invariance, essential properties for effectively processing unordered 3D point sets. It can also be integrated into existing deep learning models, including PointNet.

IN DGCNN, a k-NN graph is computed for each point in the feature space at each layer, dynamically updated as the network processes the data. This dynamic update ensures that local neighborhoods are determined based on learned features, allowing the network to capture more abstract relationships as the features evolve in deeper layers. Each point's neighborhood is defined in feature space, meaning that points with similar features (rather than just spatial proximity) form connections. This dynamic neighborhood captures both local geometry and semantic relations across the layers.

The EdgeConv operation is central to the architecture. Unlike standard convolutions on images or fixed graphs, EdgeConv operates on edges of the dynamic graph generated by k-NN in feature space. EdgeConv incorporates both the point and its neighbors to compute a new representation for each point by aggregating edge features, capturing local dependencies and geometric structures. For each edge between point i and its neighbor j, an edge feature e_{ij} is learned using the function:

$$e_{ij} = h_\Theta(x_i, x_j - x_i), \tag{4.20}$$

where x_i are the central point features, x_j are a neighboring point features and h_Θ is an MLP. These features are then aggregated for each point using a symmetric function max-pooling operation:

$$x'_i = \max_{j:(i,j)\in E} h_\Theta(x_i, x_j - x_i), \tag{4.21}$$

where $\max_{j:(i,j)\in E}$) is the max pooling over the neighbors of point i and x'_i are the updated features of point i after the EdgeConv operation. This aggregation ensures permutation invariance, as the result does not depend on the order of neighbors. The resulting feature captures local geometric structure while emphasizing high-response features across all neighboring edges.

After each EdgeConv layer, DGCNN recomputes the graph in feature space using a dynamic k-Nearest Neighbors algorithm. This update allows the receptive field of each point to expand, facilitating non-local information propagation throughout the network. Consequently, the receptive field covers both local details and global semantic structure of the point cloud, important for tasks like segmentation and classification. This layer-wise update contrasts with static graphs used in traditional graph-based methods, enabling DGCNN to learn adaptive and task-specific relationships within the data.

For classification related tasks, DGCNN performs global max pooling on the output of the final EdgeConv layer to create a global descriptor of the entire point cloud, effectively summarizing the object's global geometric and semantic structure. This operation generates a global feature vector for the whole point cloud:

$$x_{\text{global}} = \max_{i=1}^{n} x'_i, \tag{4.22}$$

where x_{global} is the global feature vector representing the entire point cloud and n is the number of points in the cloud.

For segmentation tasks, DGCNN concatenates global features with local EdgeConv outputs, producing a combined representation that preserves both local details and global context for each point, allowing the network to classify points based on both local shape and overall structure. Formally:

$$x_i^{\text{seg}} = \text{concat}(x'_i, x_{\text{global}}), \tag{4.23}$$

where x_i^{seg} are the features of point i and $concat(x'_i, x_{\text{global}})$ concatenates the local features x'_i and global features x_{global} of each point.

As the main component of the DGCNN architecture, the EdgeConv operation is used for feature extraction at each layer L by computing relationships between a point and its neighbors in a dynamic graph using an MLP. Considering a point cloud with n points, where each point has k neighbors and a d-dimensional feature vector, the complexity of the MLP processing is given by:

$$O(L \cdot n \cdot k \cdot d^2) \tag{4.24}$$

DGCNN recomputes the k-NN graph at each layer in feature space, dynamically updating the graph structure as point features evolve through the network. The paper does not explicitly defines the algorithm used to find the k-nearest neighbors, however

complexity can be estimated as $O(n^2 \cdot d)$ using a brute force approach or $O(n \cdot log(n) \cdot d$ with approximate k-NN algorithms such as KD-trees or locality-sensitive hashing.

For both classification and segmentation tasks, features are aggregated globally, with complexity of $O(n \cdot d)$, thus the total complexity of DGCNN can be defined as:

$$O(L \cdot n \cdot k \cdot d^2) + O(n \cdot log(n) \cdot d) + O(n \cdot d) \tag{4.25}$$

Although experimental results show that DGCNN does not always outperform state-of-the-art techniques, they suggest that structural and relational information obtained from edge embeddings can be equally or more valuable than point coordinates. Also, as an easily incorporated module, it can be used for further research and extension.

4.4 Chip Floorplanning

As a fundamental part of most modern technology, computer chips are almost omnipresent in day-to-day life. Chips host dozens of blocks connected by wires, where each block may be a memory subsystem, a computing unit, or a control logic system, for example. The designing of the physical layout of a chip is an engineering task called chip floorplanning.

Since the advent of chips, three broad categories of floorplanning have been proposed: partition-based methods [Breuer, 1977], stochastic approaches [Kirkpatrick et al., 1983], and analytic solvers [Luo and Pan, 2008]. However, a human effort by physical design engineers to produce manufacturable layouts still obtained better accuracy. The major downside is that non-automated floorplanning requires months of human work. To decrease this time consumption, [Mirhoseini et al., 2021] proposes Edge-GNN, a novel method that presents chip floorplanning as a reinforcement learning problem and develops an edge-based GCN approach that is capable of automatically generating chip floorplans comparable to those produced by humans in under six hours.

Each of the blocks that are part of a chip is described by a netlist or hypergraph of circuit components connected by wires. Such components can be defined as macros (memory components) or standard cells (logic gates such as NAND, NOR, and XOR). The goal of chip floorplanning is to accommodate these netlists on chip canvases, which are 2-dimensional grids, in a way that important metrics such as power consumption, area, and wire length are optimized.

The floorplanning approach uses a reinforcement learning (RL) agent, or policy network, to sequentially allocate the macros. At each step, one macro is placed, and

once all macros are placed, the model achieves the final state s_T, where T is equal to the total number of macros, and a force-directed method [Spindler et al., 2008] is used to allocate clusters of standard cells. The policy network starts in state s_t, takes an action a_t, moves to state s_{t+1} and receives a reward r_t. Through repeated iterations, the agent learns to choose actions that will maximize the cumulative reward. Each state s_t is a concatenation of node features, consisting of width, height, and type; the number of connections, which are the edge features; the clustered netlist, represented by its corresponding adjacency matrix; and metadata of the netlist and underlying technology.

To enable a learning domain-adaptive architecture, the main goal of the policy network is to predict, in a supervised manner, the value of reward functions. Therefore, a graph neural network architecture capable of predicting rewards on new netlists is proposed and used as the encoder layer of the policy network. Edge-GNN seeks to represent a node's connection and type into a lower-dimensional feature vector that may be applied to several tasks, including node classification, device placement, and link prediction.

Edge-GNN is, however, specifically tailored for the highly specialized task of chip floorplanning. The model uses a graph representation of the chip's netlist, where nodes represent components (macros or standard cells) and edges represent interconnections between them. The features for each node v_i are initialized using a vector that may include attributes like the type of macro (e.g., logic block, memory), dimensions (width and height), and initial coordinates on the chip layout. Edge features e_{ij} can include the netlist's wire properties or other relevant metadata such as signal type or weight w_{ij}.

After initialization, Edge-GNN layers update node and edge embeddings through message passing. Each edge (i, j) updates its representation based on the current embeddings of its connected nodes v_i and v_j, as well as its edge features w_{ij}. Processing by a fully connected neural network fc allows for each edge to learn a representation that captures the relationship between its connected nodes and any inherent properties defined by w_{ij}:

$$e_{ij}^{(t+1)} = fc_e(concat(v_i^t, v_j^t, w_{ij})) \tag{4.26}$$

Each node v_i also aggregates the updated embeddings of its incident edges to update its own embedding. This aggregation is performed using a symmetric function such as mean or sum:

$$v_i^{(t+1)} = AGGREGATE_{j \in N(i)}(e_{ij}^{(t+1)}), \tag{4.27}$$

where $N(i)$ are the neighbors of node i. The updated node embeddings $v_i^{(t+1)}$ capture information not only from the node's direct features but also from its interactions with connected nodes via the updated edge features. These edge and node update operations are repeated for T iterations, allowing the information to propagate across the graph and enhance the embeddings to include multi-hop neighborhood information.

Next, a policy network takes the learned node embeddings as input and outputs a probability distribution over potential placement locations for the current macro. This distribution is used to sample the next action (i.e., choosing a grid cell for placement). The policy network is typically designed as a fully connected network with non-linear activations that transforms the aggregated node features into a set of probabilities:

$$\pi(a|s) = softmax(fc_\pi(v_{global})), \tag{4.28}$$

where v_{global} is the global node representation aggregated from the current state s of the placement. Expected reward for s is estimated by a value network that shares the same feature space as the policy network, but outputs a scalar value:

$$V(s) = fc_V(v_{global}) \tag{4.29}$$

The entire architecture operates within an RL framework, specifically employing Proximal Policy Optimization (PPO) to update the policy and value networks. PPO ensures stable training by clipping policy updates and maintaining a balance between exploration and exploitation.

In summary, the complete supervised model consists of (i) Edge-GNN, which embeds both node type and netlist adjacency matrix, (ii) a fully connected feed-forward neural network responsible for embedding netlist metadata, and (iii) a layer used for predictions, where the input is a concatenation of netlist adjacency matrix and metadata embeddings, and the output is the prediction reward. This supervised task, which takes into account edge information, allows for a way to find the features and architecture necessary to generalize reward prediction across different netlists.

Computational complexity of Edge-GNN is determined by its main steps, namely edge and node updates, and the reinforcement learning framework. Assuming E edges with d-dimensional features are processed by an MLP, the complexity of the

edge update step is:

$$O(E \cdot d^2) \tag{4.30}$$

Given N nodes and k average neighbors, node aggregation and update complexity can be summarized as:

$$O(N \cdot k \cdot d + N \cdot d^2) \tag{4.31}$$

For dense operations across all nodes, the reinforcement learning policy can achieve complexity of $O(N \cdot d^2)$. Therefore, considering L Edge-GNN layers and t steps of reinforcement learning training, the overall complexity of the model can be estimated as:

$$O(L \cdot (E \cdot d^2 + N \cdot k \cdot d + N \cdot d^2)) + O(t \cdot N \cdot d^2) \tag{4.32}$$

By using and edge-aware method, Edge-GNN was able to significantly reduce the time required for chip placement compared to traditional manual design, achieving optimized layouts in under six hours versus months of human effort. The model also generalized well across different chip designs due to its ability to learn transferable representations from various training examples.

4.5 Analyzing Multi-commodity Food Flows

Assessing and enhancing the resilience of food supply chains are critical goals for tackling increasing global food insecurity. In this context, geospatial resilience refers to the ability of the food supply network to withstand and recover from various disruptions, such as natural disasters, economic instability, or logistic failures. Traditional methods for analyzing supply chain resilience often rely on centralized static models that do not account for the dynamic, interconnected nature of modern supply networks, leading to sub-optimal strategies for mitigating risks.

Federated learning is a decentralized machine learning approach that has been used in many supply chain, agricultural, and medical systems [Rieke et al., 2020]. Also, the natural network shape and consequential graph representation of food supply chains make them suitable candidates for GNN analyses. Aggregating these two ideas, [Qu et al., 2023] introduce FLEE-GNN, a model that combines federated learning with an edge-enhanced GNN to analyze the resilience of food supply networks.

In their graph representation, nodes are U.S. states and edges represent the flow of food commodities, with edge features capturing the monetary value, tonnage, and distance of shipments. By using an edge-aware GNN within a federated learning framework, the model can be trained on decentralized data from different regions to predict network resilience without requiring sensitive data to be shared.

FLEE-GNN considers a $G = (V, E)$ graph, where V is the set of nodes, here representing each U.S. state, and E the set of edges, representing all commodity food transported from a source state s to a destination state d. Each commodity transport has a monetary value v, weight tonnage t, and average transportation distance a. These values make for the edge features in the graph. Latitude and longitude are used as node features.

A message, composed of a combination of node and edge features is passed through a fully connected neural network layer and merged into a single latent vector. The message-passing scheme uses a summation function to aggregate neighborhood information. The aggregated values are passed through another fully connected layer followed by a sigmoid activation function that outputs a value between 0 and 1, representing the predicted resilience score. At each processing stage, edge and node features are leveraged to enrich data representation. Formally, for each node i and its connected edges e_{ij}, fc_e is a learnable function that processes concatenated node features v_i and v_j and edge features w_{ij}. Then, each node i updates its embedding based on the aggregated features of its neighboring edges.

The federated learning approach is used to evaluate the geospatial resilience of multi-commodity food flows. It uses decentralized training, where a central server initializes and distributes a model to four predetermined region nodes. Each node then uses its local dataset to compute its model, and a summary containing differences between central and local models is generated. These summaries are sent periodically by every node to the central server that aggregates information, refines its model, and sends it back to the regional nodes. This happens until a pre-defined stop criterion is met. Due to data scarcity, they also employ an adjustable data generator that introduces noise into the data by adding and removing transport records and changing specific features.

In FLEE-GNN, aside from node and edge updates and graph processing, computational complexity is also increased by the federated scheme. Considering that E edges and N nodes are processed by MLPs with complexity $O(d^2)$, we can estimate a total computational complexity per edge as $O(E \cdot d^2)$.

At the node update step, there is the added process of aggregating the features from its connected edges and updating its representation. Therefore, assuming each node aggregates k neighboring edge features the computational complexity can be described as $O(N \cdot k \cdot d + N \cdot d^2)$. Considering L layers of FLEE-GNN, where each

layer processes both edge and node updates the total complexity is $O(L \cdot (E \cdot d^2 + N \cdot k \cdot d + N \cdot d^2))$.

For sparse graphs, where $E \approx O(N \cdot k)$, this can be approximated as $O(L \cdot (N \cdot k \cdot d^2))$.

Lastly, the federated learning process adds complexity related to model updates and communication between the central server and local silos. Each silo trains the model on its local data, which has the same complexity as the graph processing complexity. The server must also aggregate local updates from S silos with the cost depending on the model size M (e.g., total number of parameters). Considering R aggregation rounds, complexity can be given by $O(R \cdot S \cdot M)$.

In summary, the edge and node update complexity makes FLEE-GNN feasible for sparse graphs, but challenging for dense graphs or large-scale data without optimization. However, the main bottleneck of the model is the federated approach, especially when S (number of silos) and/or M (model size) is large. Techniques like mini-batching, parallel processing, and communication-efficient FL algorithms (e.g., federated averaging) can help mitigate these computational burdens.

Experimental results show that FLEE-GNN can obtain predictions with a strong correlation to ground truth resilience data. For instance, when trained on data with a moderate noise ratio (0.3) and evaluated on 2017 food flow data, the model achieved a Spearman's rank correlation coefficient of 0.8215 and a Pearson correlation of 0.7886, demonstrating a high degree of concordance with the ground truth resilience rankings. The federated learning approach also demonstrated performance robustness, and the use of an edge-aware GNN method proved to be necessary in a dataset that consists predominantly of rich edge features representing commodity flows.

4.6 Predicting Traffic Matrices

A network traffic matrix (TM) is a key component of computer network management and traffic engineering [Svigelj et al., 2015]. In a network with N nodes, the TM is an $N \times N$ matrix that represents the traffic between all possible origin-destination (OD) pairs. Due to the dynamic nature of network traffic, direct prediction of future TMs is a very complex problem. A common approach is to try to estimate future traffic based on historical data, relying on the self-similarity and commonly cyclical nature of such networks. To this end, traditional machine learning models such as ARIMA and SARIMA [Harvey, 1990] are commonly used simple solutions that often lead to low prediction accuracy.

More recently, neural network-based models such as LSTM [Wan et al., 2022] and GRU [Yang et al., 2020], which introduce long-term data dependency in time, have shown stronger performance and better accuracy in temporal data prediction and

became the most popular general network traffic prediction models. As a natural evolution of deep neural networks applied to network data, GNNs arise as potentially powerful tools for network traffic matrix predictions.

In this context, [Tao et al., 2023] propose a cross-aggregate GNN (CAGNN) model. The key innovation is to model the network as two separate graphs: a standard node graph and a dual edge graph where the original edges become nodes. The model then uses a cross-aggregation scheme to pass information between these two graph representations, allowing it to learn a richer representation of traffic dynamics by considering both device-level and link-level interactions.

Two CAGNN blocks are used to cross-aggregate information from the node graph G and the edge graph H, enhancing both feature vectors through iterative updates. Node and edge features are processed separately but interdependently, and their representations are updated according to both local connectivity and cross-graph interactions. This iterative dual block processing is shown to be crucial for developing a more comprehensive representation capable of representing the multifaceted dynamic nature of network traffic. CAGNN uses Graph Isomorphism Networks [Xu et al., 2018a] (GIN) as its backbone, applying convolutional blocks for message passing.

For each node i, its representation $h_i^{(t)}$ is updated at iteration t based on information aggregated from its neighbors and its connected edge representations. Similarly, the representation h_{ij}^t of an edge (i, j) is updated by aggregating information from adjacent edges and connected nodes.

CAGNN is trained using historical traffic data from multiple time intervals, which allows the model to capture the temporal nature of traffic flows. The training objective is to minimize the mean squared error between the predicted and actual traffic values across all OD pairs.

Since there are no transformations or additional layers such as attention mechanisms, computational complexity of CAGNN is defined only by node and edge aggregation steps. Assuming that the aggregation function (AGG) and combination function (COM) involve operations that scale with the feature dimensionality d, each node or edge update has complexity $O(k \cdot d^2)$, where k is the average number of node or edge neighbors. Therefore, considering L layers, the total complexity can be estimated as $O(L \cdot (N \cdot k \cdot d^2 + E \cdot k_e \cdot d^2))$.

Experimental results show the efficacy of the CAGNN model, particularly when tested against traditional models like ARIMA, SVR, and LSTM. Real-world data from the Abilene network was used for testing, and CAGNN demonstrated better performance across multiple metrics. For short-term (15-minute) traffic prediction, the CAGNN model achieved an RMSE of approximately 0.02, an improvement over the LSTM-OD model's score of around 0.03. This enhanced precision, attributed to its cross-aggregation mechanism, highlights its potential to improve network traffic pre-

dictions through its edge-enhanced network representation significantly.

4.7 Radio Resource Management in Wireless Networks

Managing radio resources efficiently is key to achieving high data rates and ubiquitous connectivity in future wireless networks. Power allocation and beamforming design are crucial for enhancing the spectrum and energy efficiency in various types of networks including ultra-dense networks, cloud radio access networks, and cell-free massive MIMO systems. These management tasks are vital for supporting the large-scale deployment of next-generation wireless technologies.

Traditional optimization-based algorithms used for radio resource management are iterative and often computationally costly, which does not meet the low-latency requirements essential for real-time applications in 5G networks. To achieve more real-time performance, many deep learning methods such as MLPs and CNN, that can learn a mapping function from problem features to corresponding solutions are being deployed. However, these methods usually require a large volume of training data and can still result in poor performance.

Due to their capability of leveraging network topology information, GNNs arise as a better-suited deep learning method for radio resource management tasks. Many wireless interactions, however, are more easily mapped as edge features, which makes traditional GNN methods inefficient in handling problems such as beamforming design and power allocation. [Peng et al., 2024] compares node-focused GNNs (vertex-GNNs) and edge-focused GNNs (edge-GNNs) applied to resource management in wireless networks, concluding that edge-GNNs can perform at least as well as vertex-GNNs with much lower training time.

Combining node and edge-centered methods, [Wang et al., 2023] propose ENGNN, an edge-update empowered GNN architecture for radio resource management in wireless networks. The ENGNN architecture integrates both node and edge feature updates, aiming to leverage the relationships between transmitter nodes (TX-nodes) and receiver nodes (RX-nodes). The proposed model is divided into a processing layer, L update layers, and a postprocessing layer. Initially, in the preprocessing layer, node and edge features are transformed by a one-layer MLP to reduce data dimensionality. A ReLU is used as an activation function. After the preprocessing stage, node and edge representations are updated in different manners.

Compared to state-of-the-art methods, ENGNN obtained better performance with much less computational time. In cooperative beamforming design simulations with 5 Base Stations and 8 User Equipments, ENGNN achieved a sum-rate of over 32 bps/Hz, outperforming both the WMMSE algorithm (approx. 22 bps/Hz) and the GP method (approx. 21 bps/Hz). Most importantly, ENGNN accomplished this with

an average computation time of less than 0.01 seconds, over 1000 times faster than WMMSE. The model was also able to generalize well for different network sizes and topologies.

4.8 Summary

In this chapter, we explored the practical uses of edge-aware graph learning across diverse real-world domains. The reviewed literature shows that edges are rarely simple binary connectors; rather, they encode critical information that node-centric models are prone to discard.

Our analysis reveals three distinct roles that edge features can play in modern applications:

1. Edges as Metric Indicators: In tasks like Few-shot Learning (Section 4.1) and Point Cloud analysis (Section 4.3), edges represent similarity or geometric distance. Models like ELGNN and DGCNN demonstrate that explicitly learning these metric relationships allows for better clustering and segmentation than node features alone.

2. Edges as Semantic Relations: In Biomedical Knowledge Graphs (Section 4.2), edges carry rich semantic types (e.g., "inhibits," "binds to"). Methods like edge2vec show that embedding this semantic context is essential for accurate link prediction in heterogeneous networks.

3. Edges as Physical and Flow Constraints: In Chip Floorplanning (Section 4.4), Supply Chains (Section 4.5), and Wireless Networks (Section 4.7), edges represent physical connections (wires) or capacity constraints (traffic/commodity flow). Here, the edge features are not just attributes but the defining constraints of the system's optimization landscape.

Despite these advancements, a recurring limitation in the state-of-the-art is the trade-off between expressiveness and computational efficiency. Methods that rely on complex graph transformations (like dual graphs in CAGNN) or heavy pre-processing often suffer from scalability issues. This highlights a clear research gap: the need for a streamlined, native edge-aware architecture capable of efficiently weighting edge importance.

The following chapter addresses this gap. We introduce AttEAGNN, a novel architecture that leverages an edge-specific attention mechanism to capture these complex dependencies without the computational overhead of full graph transformations.

Chapter 5

AttEAGNN - An Attention Based Edge-Aware GNN

To further investigate the potential of edge-aware GNNs, we implemented AttEAGNN, our model. Here, we propose to process node and edge in different convolutional layers and also use a specific edge attention layer. With this hybrid architecture, we are able to capture topological structure, node neighborhood information, and relationships defined by edge features. The attention layer further enhances edge representation by defining weights that indicate the most critical edges in the network.

In this chapter, we formally define the AttEAGNN framework. We begin by detailing the dual-stream architecture that processes node and edge features in parallel (Sections 5.2 and 5.3), followed by the derivation of our magnitude-based edge attention mechanism (Section 5.4). We then present our strategy for augmenting edge features with topological metrics (Section 5.5), which allows the model to extract rich structural information even from sparse datasets. Finally, we analyze the computational complexity of the model (Section 5.6 and present the open-source implementation framework developed to ensure reproducibility (Section 5.7).

5.1 Data Flow Overview

As Fig. 5.1 shows, first, we extract node and edge features from the input graph dataset. Two GNN layers process these features and further aggregate them according to edge weights. The final step involves concatenating the processed node and edge features and a final linear layer for node load prediction.

AttEAGNN was designed to map a network graph G to a vector of predicted node loads. The model inputs are:

1. A node feature matrix $X \in \mathbb{R}^{|V| \times |V|}$, where each node is represented as a one-hot encoding.

2. An edge feature vector $EF \in \mathbb{R}^{|E| \times D}$ that combines inherent and augmented features.

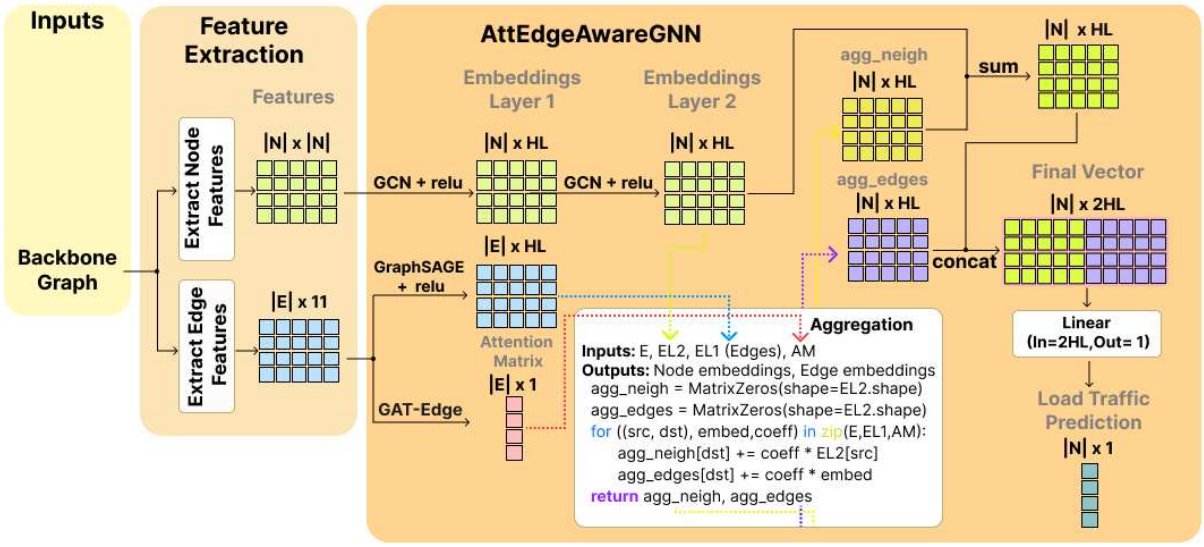


Figure 5.1: AttEAGNN model overview.

3. The edge indices that indicate which nodes are connected to each other.

The output is a vector of scalar values representing each normalized node load. We detail these steps in the following sections.

5.2 Node Feature Processing

The one-hot encoded features are processed by two Graph Convolutional Network (GCN) layers [Kipf and Welling, 2016], which aggregate and transform the features based on the graph structure. The propagation rule for the first layer is defined as:

$$X^{(1)} = \text{ReLU}(\hat{A}X^{(0)}W_1) \tag{5.1}$$

where $X^{(0)}$ is the input feature matrix, W_1 is the learnable weight matrix, and $\hat{A}$ is the normalized adjacency matrix defined as $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ (with $\tilde{A} = A + I$). This ensures that node representations are updated by aggregating information from their local neighborhoods.

The second GCN layer further refines these embeddings:

$$X^{(2)} = \text{Dropout}(\text{ReLU}(\hat{A}X^{(1)}W_2)) \tag{5.2}$$

5.3 Edge Feature Processing

To leverage edge information, we used datasets of real-world networks (detailed in Section 6.1) containing explicit edge features. We also benchmark our method using

datasets without explicit edge features. In both cases, we calculated edge betweenness centrality, edge degree (as the sum of connected nodes degrees), and edge clustering coefficient (as the average of connected nodes coefficients). We use these attributes as implicit edge features and detail the process of computing them in Section 5.5.

To process edge features, we employ a GraphSAGE (Graph Sample and Aggre-GatE) layer [Hamilton et al., 2017a]. Unlike simple linear projections that treat each edge in isolation, GraphSAGE allows edge representations to incorporate structural information. We define the neighborhood of an edge based on shared endpoints. The update rule is:

$$e'_i = \sigma\left(W_1 \cdot e_i + W_2 \cdot \text{mean}_{j \in \mathcal{N}(i)}\{e_j\}\right) \tag{5.3}$$

where e_i is the feature vector of edge i, $\mathcal{N}(i)$ denotes the set of adjacent edges (i.e., edges sharing a common node with i), W_1 and W_2 are learnable weight matrices, and σ is the ReLU activation function.

5.4 Edge Attention Mechanism

Edge features are also processed by an attention-based mechanism in the *Graph Attention Networks* (GATs) model proposed by [Veličković et al., 2018]. Initially designed for node attribute processing, we adapt and apply this mechanism to edge features.

To compute the attention coefficients, we implement a learned magnitude-based saliency mechanism. First, the edge features are projected into a hidden space via a learnable linear transformation:

$$e'_i = W_e e_i + b_e \tag{5.4}$$

where W_e is the weight matrix and b_e is the bias vector of the linear layer. We then compute the attention score a_i as the squared L_2-norm (magnitude) of this projected vector. Instead of comparing edge pairs, this mechanism encourages the model to learn a projection W_e that scales up the magnitude of topologically critical edges while suppressing irrelevant ones towards zero. The score is calculated as:

$$a_i = \sum_{k=1}^{D} (e'_{i,k} \cdot e'_{i,k}) \tag{5.5}$$

where k iterates over the feature dimensions. This scalar score represents the learned "energy" or importance of the edge. To introduce non-linearity and normalize these scores into a probability distribution, we apply LeakyReLU followed by a Softmax operation across all edges:

$$\alpha_i = \frac{\exp(\text{LeakyReLU}(a_i))}{\sum_{j \in \mathcal{E}} \exp(\text{LeakyReLU}(a_j))} \tag{5.6}$$

This last step results in a single coefficient for each edge, proportional to its relevance in the network.

The resulting attention coefficients are then applied in the aggregation step. We perform a dual-weighted aggregation, filtering both the structural information from neighboring nodes and the physical attributes of the connecting edges. The aggregated contexts are defined as:

$$e_i^{agg} = \sum_{j \in \mathcal{N}(i)} \alpha_{ji} \cdot e'_{ji} \tag{5.7}$$

$$h_i^{agg} = \sum_{j \in \mathcal{N}(i)} \alpha_{ji} \cdot x_j^{(2)} \tag{5.8}$$

where $\mathcal{N}(i)$ is the set of neighbors of node i, α_{ji} is the attention coefficient of the edge connecting j and i, and $x_j^{(2)}$ is the node embedding from the second GCN layer.

To preserve the original node features while incorporating the new edge-aware context, we employ a residual connection. The aggregated node context h_i^{agg} is added to the current node embedding $x_i^{(2)}$, and the result is concatenated with the aggregated edge context e_i^{agg}. The final node representation is given by:

$$x_i^{final} = [(x_i^{(2)} + h_i^{agg}) \parallel e_i^{agg}] \tag{5.9}$$

After the concatentaion of node and edge flows, AttEAGNN generates a vector of dimension $2 \times HL$, where HL is the hidden layer dimension. This vector is then passed through a fully connected linear neural network layer to generate the node load prediction output y_i:

$$y_i = W_f x_i^{final} + b_f \tag{5.10}$$

The predicted output represents the normalized node lode obtained from the original TMs. This is calculated by summing in and out traffic for each node, then normalizing the result considering the total network load. This ensures that the model learns proportional and not absolute traffic values.

5.5 Augmented Edge Features

To properly explore the power of edge-aware GNNs, we need datasets with rich and relevant edge features. However, many existing datasets do not have such explicit

features, so we used calculated implicit edge information as features. We also found that combining these augmented data with existing edge features on the backbone network datasets led to better results.

To compute the augmented edge features, considering a graph $G = (V, E)$, where V is the set of nodes and E the set of edges, for each edge $e = (u, v) \in E$, the following steps are performed.

First, each edge is initialized with a feature vector that represents the original explicit edge features of the graph, if available:

$$f_e = \begin{cases} \text{Original feature}, & \text{if } f_e \text{ exists in edge attributes,} \\ \mathbf{0}_8, & \text{otherwise.} \end{cases} \tag{5.11}$$

Here, 0_8 represents an 8-dimensional zero vector used if there are no explicit edge features.

Then, we proceed to compute edge betweenness centrality, which is a measure that quantifies the importance of an edge within the overall structure of a graph. It counts the shortest paths between pairs of nodes that pass through the edge. Intuitively, it identifies edges that serve as critical connectors or bridges between different parts of the graph. The edge betweenness centrality of an edge $e = (u, v)$ is given by:

$$C_B(e) = \sum_{s \neq t \in V} \frac{\sigma(s, t|e)}{\sigma(s, t)}, \tag{5.12}$$

where:

- $\sigma(s, t)$ is the total number of shortest paths between nodes s and t,
- $\sigma(s, t|e)$ is the number of those paths that traverse e.

Next, we compute the edge degree. Though there are different approaches to this calculation, here we define the degree of an edge as the sum of the degrees of its connected nodes:

$$\text{Degree}(e) = \text{Degree}(u) + \text{Degree}(v), \tag{5.13}$$

where

$$\text{Degree}(u) = \sum_{v \in \text{Neighbors}(u)} 1. \tag{5.14}$$

Another feature considered is the clustering coefficient, which quantifies the tendency of nodes to form triangles. For each node u of $e = (u, v)$ the local clustering coefficient is computed as:

$$C(u) = \frac{2 \cdot |\{(v, w) \in E : (u, v) \in E \wedge (u, w) \in E\}|}{\text{Degree}(u) \cdot (\text{Degree}(u) - 1)}. \tag{5.15}$$

We then average the coefficients of connected nodes (u, v) to determine the clustering coefficient of e:

$$C_{\text{avg}}(e) = \frac{C(u) + C(v)}{2}. \tag{5.16}$$

Each computed value is appended to f_e and the feature vectors $\{f_e : e \in E\}$ are converted into a tensor format for downstream processing:

$$\mathbf{F}_E = \begin{bmatrix} f_{e_1} \\ f_{e_2} \\ \vdots \\ f_{e_m} \end{bmatrix}, \tag{5.17}$$

where $m = |E|$ is the number of edges in the graph.

5.6 Computational Complexity Analysis

To address the scalability of AttEAGNN, we provide a theoretical analysis of its computational time complexity. We analyze the computational time requirements in relation to the number of nodes $(N = |V|)$, edges $(E = |E|)$, and the hidden layer dimension (HL).

The forward pass of AttEAGNN consists of three main stages: node feature processing, edge feature processing, and aggregation.

1. Node Feature Processing: This stage uses two GCN layers. The computational complexity of a single GCN layer is linear in the number of edges, approximately $O(E \cdot F_{in} \cdot F_{out})$, where F_{in} represents the input feature dimension of a node before the layer, and F_{out} is the output feature dimension after the layer's transformation. For our first GCN layer, the input is a one-hot vector, making $F_{in} = N$, and the output is the hidden layer, making $F_{out} = HL$. For the second layer, both the input and output dimensions are HL. Thus, the complexity for this two-layer module is $O(E \cdot N \cdot HL + E \cdot HL^2)$.

2. Edge Feature Processing: This parallel stream involves a GraphSAGE layer and an attention mechanism. The complexity of both operations is linear in the number of edges, resulting in a combined complexity of approximately $O(E \cdot F_{edge} \cdot HL)$.

3. Aggregation and Final Layers: The final aggregation step iterates through the edges to update node representations, costing $O(E \cdot HL)$. The subsequent linear layers operate on the final node embeddings, adding a complexity of $O(N \cdot HL)$.

Combining these stages, the dominant factor is the first graph convolution on the nodes. Therefore, the overall time complexity per epoch for AttEAGNN is approximately $O(E \cdot N \cdot HL)$, which simplifies to $O(E \cdot HL^2)$ if we consider the hidden dimension (HL) as the leading factor in feature size.

The baseline models, GCN and GSAGE, also have a time complexity that is approximately linear in the number of edges, typically $O(E \cdot HL^2)$ for a similar two-layer architecture. While AttEAGNN has a larger constant factor due to its parallel processing streams for nodes and edges, its asymptotic scaling behavior is comparable to standard GNNs. This analysis suggests that our model, despite its more complex architecture, is computationally feasible for larger graphs and scales efficiently with network size.

5.7 Implementation Framework and Reproducibility

To address Research Question 4 (RQ4) regarding the promotion and accessibility of edge-aware research, we developed and released the AttEAGNN implementation as an open-source, modular framework. Existing literature often suffers from a lack of standardized implementations, making it difficult to benchmark new edge-aware methods against node-centric baselines.

Our framework is designed not merely as a repository for the model code, but as a flexible testbed that ensures reproducibility and extensibility. The implementation is available at[1] and includes:

1. Modular Architecture: The code decouples the node processing (GCN), edge processing (GraphSAGE), and attention mechanisms. This allows future researchers to easily swap components (e.g., replacing the GCN backbone with GAT) to test different inductive biases without rewriting the entire pipeline.

2. Automated Feature Augmentation: As detailed in Section 5.5.1, the framework includes an automated pre-processing module that computes implicit edge features (Betweenness Centrality, Clustering Coefficients) on the fly. This lowers the barrier to entry for experimenting with datasets that lack explicit edge attributes.

3. Standardized Benchmarking: The repository includes the specific training scripts, hyperparameter configurations, and random seeds used to generate the results in Chapter 6, ensuring that the experiments are fully reproducible.

By providing this open framework, we aim to standardize the evaluation of edge-aware GNNs, allowing the community to validate this paradigm across new domains beyond those presented in this thesis.

[1] https://github.com/wagneraljr/AttEAGNN

5.8 Summary

In this chapter, we detailed the architecture of AttEAGNN, a novel graph neural network designed to exploit edges as primary informational entities on graph data. We addressed the limitations of node-centric models by implementing a dual-stream architecture that processes node and edge features in parallel. We established a methodology for Edge Feature Processing that utilizes GraphSAGE convolution to capture structural context within edge representations, treating edges as evolving entities rather than static attributes. Furthermore, we introduced a Magnitude-Based Edge Attention Mechanism. This mechanism diverges from traditional pair-wise attention by learning to encode edge importance directly into the magnitude of the feature vectors. This allows the model to function as a learned saliency filter, dynamically amplifying critical connections (like bottlenecks in a backbone network) while suppressing irrelevant ones. We also outlined the Computational Complexity, demonstrating that our approach scales linearly with the number of edges, $O(E)$, maintaining efficiency comparable to standard GNN baselines. Finally, we described our Feature Augmentation Strategy, which enriches standard graph benchmarks with implicit topological metrics (such as betweenness centrality and clustering coefficients) to fully leverage the proposed edge-aware capabilities. With the theoretical and architectural foundations established, the following chapter will detail the experimental setup and present the results, validating AttEAGNN's performance on both real-world backbone networks and standard classification benchmarks.

Chapter 6

Experiments and Results

In this chapter, we detail our experiments and present our results. Two different sets of experiments were conducted. First, we aimed to predict node loads in real-world backbone networks, which were detailed in Sections 6.1.1 and 6.1.2. Second, we tested our model on three commonly used datasets that do not contain explicit edge features.

For both cases, we compare our model with two traditional well-established non-edge-aware GNN models, namely GCN [Kipf and Welling, 2016] and GraphSAGE [Hamilton et al., 2017a]. We also implement CAGNN, a different edge-aware GNN model based on the description of [Tao et al., 2023]. We fixed a random initialization seed for each model and optimized hyperparameters using a Bayesian sequential model-based optimization algorithm [Akiba et al., 2019].

6.1 Datasets

In this section, we detail the datasets used in this work. We explore two backbone network datasets that contain explicit edge features (Abilene and RNP). We also compare AttEAGNN's performance on CORA, CITESEER, and PUBMED, which are widely used benchmarking datasets without explicit edge features. We used the feature augmentation strategy defined in Section 5.5, enriching the raw topological data with calculated Betweenness Centrality and Edge Degree metrics to maximize the information available to the edge attention mechanism.

6.1.1 Abilene Network Dataset

As an application case to explore the use of edge-aware GNNs, we propose solving the task of predicting node loads in backbone networks. We used the Abilene dataset due to its well-documented structure, real-world traffic data, and historical significance as a benchmark for testing optimization algorithms and network performance models. Its explicit edge features make it an ideal choice for demonstrating the advantages of edge-aware models in capturing and utilizing edge information.

The Abilene Network was a high-performance backbone network in the United

States that operated as part of the Internet2 project [Teitelbaum et al., 1999]. This project was a collaborative initiative among universities, industry, and government to develop advanced networking technologies and applications for research and education. Established in 1999 and decommissioned in 2007, the Abilene Network comprised 11 nodes and 14 links, primarily connecting academic and research institutions. Over its operational lifetime, its capacity was regularly upgraded, achieving speeds of up to 10 Gbps in its final phase.

In their work on network tomography, [Fang et al., 2007] published data on the structure and traffic of the Abilene Network. Since then, this network has been widely used as a benchmark for testing algorithms and optimization solutions. Figure 6.1 shows the structure of the Abilene Network. Although networking technologies and infrastructures have significantly evolved since its inception, the historical data from the Abilene Network remains a valuable resource for developing or testing models aimed at performance enhancement or routing optimization in computer networks [Hope, 2020; Tao et al., 2023; Qiu et al., 2023; Yang et al., 2023; Zheng et al., 2024].

The dataset used in this study comprises six months of traffic data between the nodes of the Abilene Network, measured at five-minute intervals. The data is represented as $N \times N$ traffic matrices, where N is the number of network nodes. Since we aim to predict node loads, each node's incoming and outgoing traffic in the traffic matrix was aggregated and normalized. The network structure includes edges with eight real-valued attributes each. Furthermore, the model was enhanced by computing implicit augmented edge features as previously described.

6.1.2 RNP Network

Although the Abilene dataset is widely used in research, it represents a relatively small network decommissioned in 2007, when network traffic patterns and volume were significantly different than nowadays. With that in mind, we also evaluate our model node load prediction potential using the Brazilian Rede Nacional de Pesquisa (RNP)[1] dataset. RNP is an academic network connecting multiple educational institutions throughout the country, with 27 points of presence (PoPs) and 46 links, represented by Fig. 6.2.

Based on this information, we create a graph where the nodes are the PoPs and edges are the links between them. Compared to Abilene, RNP is much more recent, still active, and capable of considerably higher traffic volume and speed.

The RNP network provides various information about traffic, including real-time traffic load. However, traffic matrices are not explicitly made available. We imple-

[1] https://www.rnp.br/sistema-rnp/rede-ipe

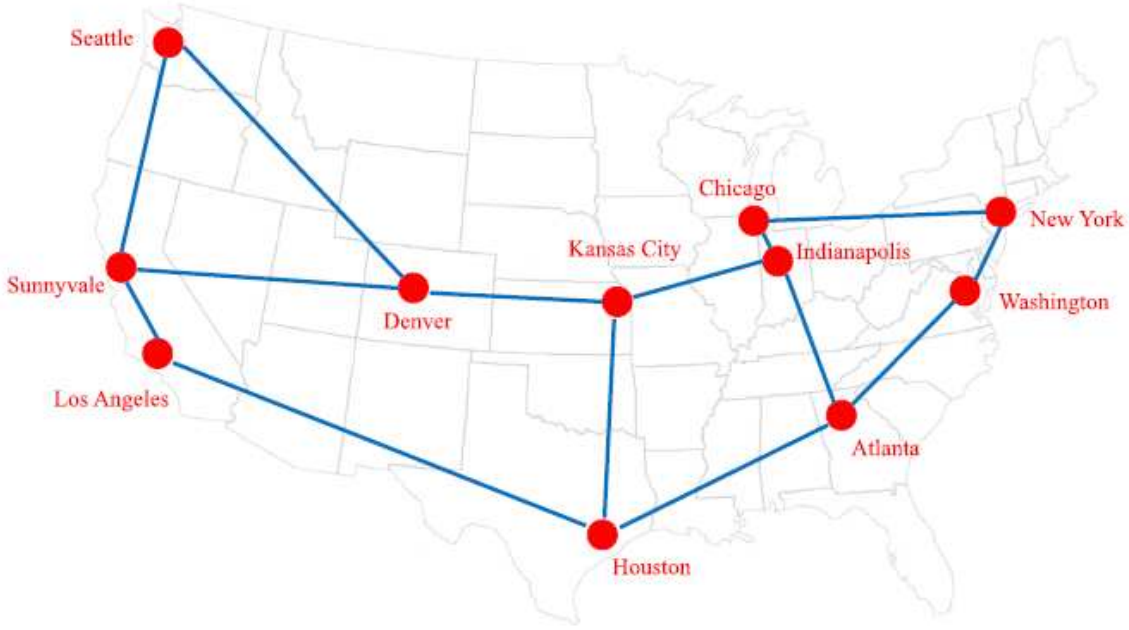


Figure 6.1: Abilene Network Structure.

mented an artificial traffic generator using a sparsified gravity cyclical model to overcome that.

Gravity models are widely used in network traffic simulation to estimate the volume of traffic T_{ij} between two nodes [Qian and Han, 2009]. Traffic volume is proportional to the product of the capacities (masses) of the nodes and inversely proportional to the square of the distance between them. Formally expressed as:

$$T_{ij} = K \cdot \frac{M_i \cdot M_j}{D_{ij}^2} \tag{6.1}$$

where K is a proportionality constant set as 1, M_i and M_j are respectively the masses of nodes i and j, which we obtained by summing the capacities of all the links connected to these nodes, and D_{ij}^2 is the distance between nodes i and j.

To simulate realistic temporal patterns, the model generates a set of q base traffic matrices that repeat cyclically. This cyclical pattern can represent daily or weekly traffic fluctuations. For a given cycle length q, the traffic matrix for interval t is defined as:

$$T(t) = T_{t \mod q} \tag{6.2}$$

where $T_{t \mod q}$ is one of the q base traffic matrices.

Finally, the sparsification process introduces randomness into the model, simulating reduced activity or inactive connection periods. This is achieved by setting some

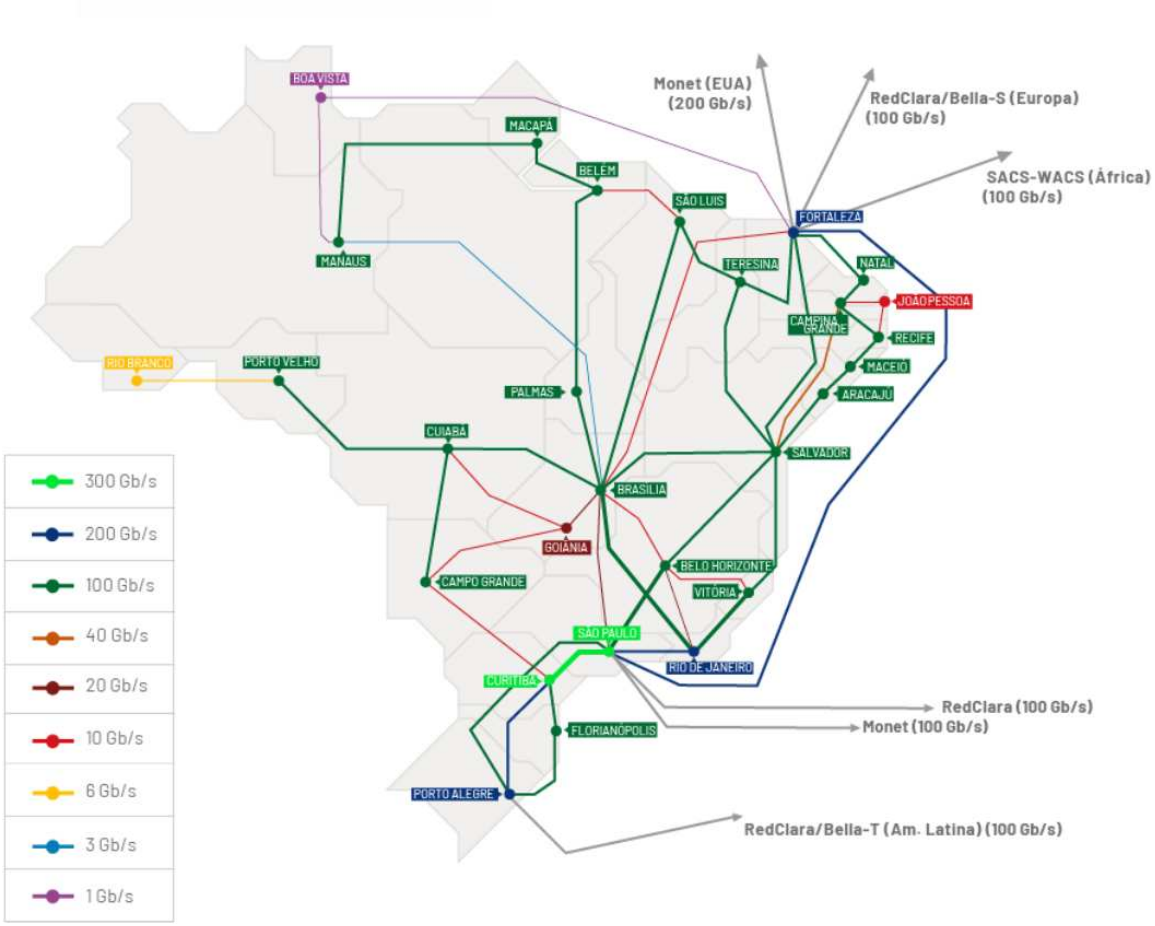


Figure 6.2: RNP Network Structure.

traffic volumes to zero based on a sparsification probability p we set as 0.5. For each element T_{ij} in the traffic matrix, a uniform random variable s is generated, and:

$$T_{ij} = \begin{cases} 0 & \text{if } s > p \\ T_{ij} & \text{otherwise} \end{cases} \tag{6.3}$$

where s is a uniformly distributed random variable.

Incorporating cyclical patterns and sparsification into this model allows for more realistic simulations of network traffic. The cyclical aspect accounts for periodic fluctuations in traffic volumes, such as daily or weekly patterns, while sparsification introduces randomness, simulating periods of low activity or inactive connections. With this enhanced model, we seek to reflect better real-world traffic conditions and variability, making it a more suitable tool for network performance evaluation and forecasting.

Using the sparsified gravity cyclical model, we generate an entire month's worth of traffic, with traffic matrices representing a 5-minute time window of measurements.

6.1.3 Citation Benchmark Datasets

To evaluate the generalization capabilities of AttEAGNN beyond physical backbone networks, we utilized three widely recognized benchmark datasets: CORA, CITESEER, and PUBMED. These datasets represent citation networks where nodes correspond to scientific documents and edges denote citation links between them. The learning task for these datasets is semi-supervised node classification, where the goal is to predict the research topic (label) of a document based on its content and connections.

- CORA: A citation network of machine learning papers. It consists of 2,708 nodes and 5,429 edges, with node features represented by a sparse bag-of-words vector indicating the presence of specific words in the document.
- CITESEER: A similar citation network containing 3,312 nodes and 4,732 edges, characterized by a relatively sparse connectivity structure compared to CORA.
- PUBMED: A larger dataset related to diabetes research, containing 19,717 nodes and 44,338 edges. This dataset challenges the model's scalability and ability to handle larger, denser structures.

We chose to experiment on these datasets because they are widely used as benchmarks for node and graph-centric GNNs. They do, however, lack explicit edge features. In order to leverage the edge-aware capabilities of AttEAGNN, we applied the Feature Augmentation Strategy detailed in Section 5.5. By calculating metrics such as edge betweenness centrality and clustering coefficients, we artificially enriched these graphs, transforming simple binary links into attributed edges that the attention mechanism could weigh effectively.

6.2 Baselines

To demonstrate the effectiveness of our proposed edge-aware architecture, we compare AttEAGNN against three established Graph Neural Network models. These baselines were selected to represent both traditional node-centric approaches and competing edge-aware strategies:

- GCN (Graph Convolutional Network) [Kipf and Welling, 2016]: The standard semi-supervised baseline for graph learning. GCN operates on the spectral domain and aggregates neighborhood information using a fixed Laplacian normalization. It serves as our primary node-centric baseline.

- GraphSAGE [Hamilton et al., 2017a]: An inductive framework that generates node embeddings by sampling and aggregating features from a node's local neighborhood. As AttEAGNN incorporates SAGE-style convolution for its edge processing stream, comparing against a pure node-based GraphSAGE isolates the impact of our specific edge-attention mechanism.

- CAGNN (Cross-Aggregate GNN) [Tao et al., 2023]: A state-of-the-art edge-aware model designed for traffic prediction. CAGNN utilizes a dual-graph approach (transforming edges to nodes) and a cross-aggregation mechanism. Comparing against CAGNN allows us to evaluate whether our native attention mechanism offers advantages over graph-transformation approaches.

6.3 Hyperparameter Optimization and Reproducibility

To ensure the optimal performance of the proposed AttEAGNN model and provide a fair comparison against baselines, we implemented a rigorous hyperparameter optimization process using the Optuna framework [Akiba et al., 2019]. The optimization targeted key training parameters, specifically the number of training epochs, the dropout rate, and the configuration parameters for both the optimizer and the learning rate scheduler.

Optuna employs an iterative process wherein a set of hyperparameters is selected at each step based on a specific optimization strategy. Our implementation utilizes algorithms such as the Tree-structured Parzen Estimator (TPE) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES). These algorithms leverage results from previous iterations to guide the selection of subsequent hyperparameters, effectively balancing exploration (testing new areas of the hyperparameter space) and exploitation (refining promising configurations).

For every selected set of hyperparameters, an objective function is executed, which involves training and validating the machine learning model. The model's performance is then evaluated using specific metrics, such as the coefficient of determination (R^2) or Mean Squared Error (MSE). This performance metric is returned to the Optuna framework, which uses it to adjust its selection strategy for future iterations. Upon completion of the defined number of iterations, the set of hyperparameters yielding the best model performance is selected.

Furthermore, to maintain consistency across results and ensure the reproducibility of our experiments, we fixed the random seeds used for model weight initialization. We tested over 20 different random seeds for each model configuration, and the results presented in this work correspond to the best optimizations obtained. Our open-source implementation framework allows for the modification of these seeds,

as well as the integration of different GNN models and traffic matrices for extended benchmarking.

6.4 Environment

All experiments were conducted on a standardized hardware setup to ensure fair comparison of training times and computational resources. The models were trained and evaluated on a workstation equipped with an Intel Core i7-11800H processor @ 2.30GHz and 32 GB of RAM. The implementation was built using the PyTorch Geometric library, within the open-source framework detailed in Section 5.7.

6.5 Backbone Network Node Load Prediction

In this section, we evaluate the performance of AttEAGNN on the task of predicting node traffic loads. This regression task assesses the model's ability to capture the physical constraints and flow dynamics of backbone networks. We compare our model against the baselines (GCN, GraphSAGE, CAGNN) across two training scenarios:

- Scenario 1 (Short-term): Training with 24 hours of traffic data from the day immediately preceding the target window.
- Scenario 2 (Long-term): Training with 24 hours of data from the same day of the previous week, testing the model's ability to capture cyclical weekly patterns.

6.5.1 Evaluation Metrics

To rigorously assess prediction accuracy, we utilize three complementary metrics:

- Mean Squared Error (MSE): Used as the loss function during training, it penalizes larger errors more severely.
- Root Mean Squared Error (RMSE): Provides an error metric in the same units as the traffic load.
- Mean Absolute Error (MAE): Measures the average magnitude of errors without focusing on outliers.
- Coefficient of Determination (R^2): Indicates the proportion of variance in the dependent variable that is predictable from the independent variables.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{6.4}$$

6.5.2 Results on Abilene Network

The Abilene network serves as our primary real-world benchmark. Table 6.1 presents the R^2 scores for both scenarios.

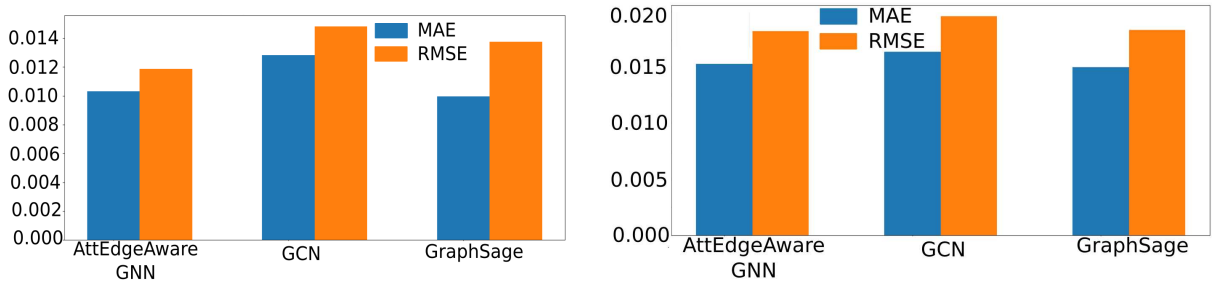
In Scenario 1, AttEAGNN achieved an R^2 score of 0.92, outperforming both the node-centric baselines (GCN: 0.88, GraphSAGE: 0.89) and the edge-aware CAGNN (0.91). This indicates that our specific attention mechanism effectively filters noise in the traffic data.

In Scenario 2, we observe a general performance drop across all models due to the weaker temporal correlation of weekly data. However, AttEAGNN maintained the highest robustness with an R^2 of 0.84, significantly higher than CAGNN (0.71). This suggests that the magnitude-based saliency of our model successfully encodes structural traffic patterns that persist over time, whereas other models rely more heavily on immediate temporal proximity.

Table 6.1: Abilene Dataset Node Load Prediction R^2 Scores.

	Scenario 1	**Scenario 2**
AttEAGNN	0.92	0.84
CAGNN	0.91	0.71
GSAGE	0.89	0.68
GCN	0.88	0.67

To understand the distribution of errors, we analyze the MAE and RMSE metrics presented in Figure 6.3. In Scenario 1, AttEAGNN achieved an RMSE of 0.011, which is superior to GraphSAGE (0.013) and GCN (0.014). However, regarding MAE, our model (0.010) performed slightly worse than GraphSAGE (0.009).



(a) Scenario 1

(b) Scenario 2

Figure 6.3: MAE and RMSE comparison in the Abilene Network.

This discrepancy highlights a specific strength of our architecture. RMSE applies a heavier penalty to large errors (outliers), while MAE treats all deviations equally. The fact that AttEAGNN achieves the lowest RMSE while maintaining a comparable MAE suggests that our model is significantly better at predicting extreme values or

traffic bursts. In backbone networks, accurately predicting these peaks is often more critical for congestion control than minimizing the average error of low-traffic states.

We also compared our GNN approach against traditional statistical baselines (ARIMA, SARIMA, LSTM) in Table 6.2. All GNN-based methods significantly outperformed the statistical models, with AttEAGNN reducing the MAPE (Mean Absolute Percentage Error) to 0.140 compared to 1.096 for ARIMA.

Table 6.2: Metrics Comparison.

Scenario 1						
	AttEAGNN	GraphSAGE	GCN	ARIMA	SARIMA	LSTM
MAE	0,010	**0,009**	0,012	0,025	0,020	0,025
RMSE	**0,011**	0,013	0,014	0,029	0,024	0,030
MAPE	**0,140**	0,197	0,146	1,096	0,932	0,829
Scenario 2						
	AttEAGNN	GraphSAGE	GCN	ARIMA	SARIMA	LSTM
MAE	**0,015**	**0,015**	0,016	0,025	0,025	0,029
RMSE	**0,018**	**0,018**	0,019	0,029	0,031	0,034
MAPE	0,177	**0,171**	0,191	1,096	0,885	3,438

6.5.3 Results on RNP Network

Experiments on the RNP network (Table 6.3) corroborate the findings from Abilene. AttEAGNN achieved an R^2 of 0.92 in Scenario 1 and 0.91 in Scenario 2.

Unlike Abilene, the RNP traffic was generated using a gravity model. The minimal performance drop between Scenario 1 and 2 (0.92 to 0.91) reflects the highly regular nature of the synthetic traffic generated by the gravity model, which lacks the unpredictable anomalies found in real-world trace data like Abilene. Also, even in this controlled environment, AttEAGNN consistently outperformed the baselines, demonstrating that its advantage is not an artifact of irregular real-world noise but a result of superior structural encoding.

Table 6.3: RNP Dataset Node Load Prediction R² Scores.

	Scenario 1	**Scenario 2**
AttEAGNN	0.92	0.91
CAGNN	0.9	0.89
GSAGE	0.89	0.88
GCN	0.89	0.87

The error metrics for RNP, summarized in Table 6.4 and Figure 6.4, mirror the behavior seen in Abilene. AttEAGNN achieved the lowest RMSE (0.0062) compared to CAGNN (0.0072) and GCN (0.0075). Similar to the Abilene experiments, while our MAE (0.0049) was marginally higher than CAGNN (0.0046), the superior RMSE score confirms that the magnitude-based attention mechanism is particularly effective at capturing the structural constraints that govern high-traffic links, preventing large prediction errors.

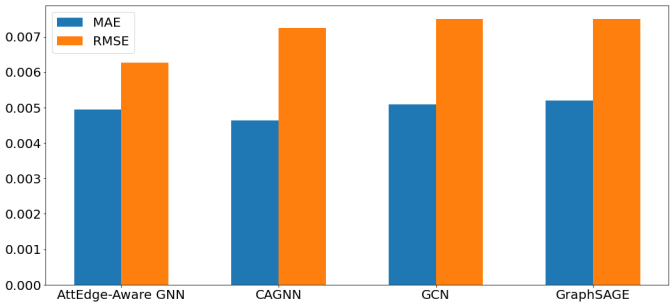


Figure 6.4: Model Comparison: MAE and RMSE values.

Table 6.4: MAE and RMSE comparison in the RNP Network.

	AttEAGNN	**CAGNN**	**GraphSAGE**	**GCN**
MAE	0.0049	0.0046	0.0051	0.0050
RMSE	0.0062	0.0072	0.0074	0.0075

Training Dynamics and Computational Cost

Further analyzing the RNP Dataset Load Prediction in Scenario 1, Figure 6.5 illustrates the training loss (MSE) evolution. AttEAGNN demonstrates faster convergence and greater stability compared to CAGNN, which exhibited significant volatility in early epochs. This stability is likely attributed to the normalization effect of our magnitude-based attention, which prevents exploding gradients from high-traffic edges.

As shown in Table 6.5, the trade-off for this performance is computational cost. AttEAGNN required approximately 17 seconds per training cycle, compared to 2 seconds for GCN/GraphSAGE. However, it was notably more efficient than the competing edge-aware model, CAGNN (25 seconds). This confirms our complexity analysis in Chapter 5: while edge-aware processing adds overhead, our streamlined attention mechanism is more efficient than dual-graph transformations.

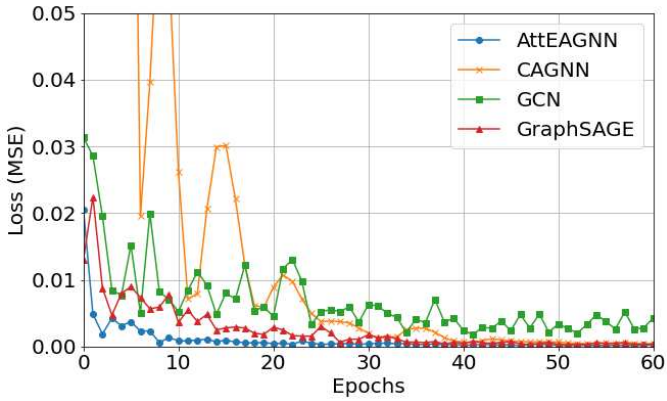


Figure 6.5: Model Comparison: MSE evolution during training

Table 6.5: Training Time Comparison

	AttEAGNN	**CAGNN**	**GraphSAGE**	**GCN**
Time (s)	17	25	2	2

6.6 Node Classification on Benchmark Datasets

CORA, CITESEER, and PUBMED are widely recognized benchmark datasets commonly used in graph-based machine learning for evaluating classification tasks. These datasets represent citation networks where nodes correspond to documents (e.g., research papers), and edges represent citation relationships between them. Each dataset includes node features, labels for classification, and the network structure. They do not contain, however, explicit edge features. We used an edge feature augmentation scheme in our experiments, detailed in Section 5.5.

To answer RQ1, we extended our evaluation to the CORA, CITESEER, and PUBMED citation networks (Table 6.6). On CORA, AttEAGNN (0.78) outperformed CAGNN (0.68) and GCN (0.77) but slightly trailed GraphSAGE (0.80), on PUBMED, a larger dataset, our model (0.77) matched GraphSAGE and outperformed GCN.

The results on these benchmarks are mixed compared to the backbone networks. This is expected: citation networks have binary edges without intrinsic physical attributes. Our augmented features (Section 5.5) provided enough signal to make the model competitive, but the true power of AttEAGNN is unlocked in domains like backbone networks, where edges possess rich, physically meaningful attributes (bandwidth, latency) that inherently drive the system's behavior.

Our method outperformed GCN and matched GraphSAGE results on the PUBMED dataset. As Table 6.7 shows, PUBMED contains significantly more nodes and edges. This outcome may indicate that edge-aware models can obtain state-of-the-art results on larger, denser datasets.

Table 6.6: Classification Accuracy on Benchmark Datasets

	CORA	CITESEER	PUBMED
AttEAGNN	0.78	0.66	0.77
CAGNN	0.68	0.56	0.60
GSAGE	0.8	0.71	0.77
GCN	0.77	0.67	0.76

Table 6.7: Datasets Description

	CORA	CITESEER	PUBMED
Nodes	2,708	3,312	19,717
Edges	5,429	4,732	44,338

The results of the first set of experiments indicate that edge-aware models can be valuable assets for working with datasets that contain rich edge information, outperforming traditional methods on every test. However, the second set of experiments' results indicate that the added complexity and processing time of edge-aware models may not compensate for the results.

6.7 Summary

In this chapter, we evaluated the performance of AttEAGNN across two distinct domains: node load prediction in backbone networks and node classification in citation graphs.

Our experiments on the Abilene and RNP backbone networks demonstrated that AttEAGNN consistently outperforms state-of-the-art baselines, achieving R^2 scores of up to 0.92. Our error analysis revealed that the proposed attention mechanism is particularly effective at minimizing Root Mean Squared Error (RMSE), indicating a greater capability to predict traffic peaks and bottlenecks compared to node-centric models like GraphSAGE and GCN.

We also observed that while edge-aware processing introduces a computational overhead, increasing training time from 2 seconds to 17 seconds per epoch compared to simple baselines—it remains significantly more efficient than competing edge-aware methods like CAGNN (25 seconds).

Finally, our results on benchmark citation datasets confirmed that while AttEAGNN is competitive in general graph tasks, its true advantage is realized in domains with rich, meaningful edge attributes. This validates our main hypothesis: explicit edge modeling is not merely an enhancement but can contribute for accurately capturing the dynamics of complex network systems.

Chapter 7

Conclusions and Future Directions

In this chapter we summarize the main contributions and discuss the lessions we learned on this work. Although we present edge-aware learning as a promising field of research, we also acknowledge that there are limitations to the proposed AttEAGNN model and the edge-aware paradigm itself, while presenting possible new avenues of research.

7.1 Summary of Contributions

This thesis addressed the underutilization of edge information in Graph Neural Networks. We hypothesized that considering graph edges as primary informational entities rather than auxiliary structures would enhance overall performance on ML taks in complex networks. Through the development of a novel taxonomy and the proposal of AttEAGNN, we have provided both theoretical and empirical evidence to support this hypothesis.

Our main contributions are threefold:

- Theoretical Unification: We define and establish a novel comprehensive taxonomy of edge-aware learning methods, categorizing multiple approaches ranging from random walks to modern Graph Transformers.
- Architectural Innovation: We introduced AttEAGNN, an attention based edge-aware GNN basend on a dual-stream architecture that utilizes a magnitude-based saliency mechanism to dynamically weight edge importance.
- Empirical Validation: We demonstrated that explicit edge modeling can lead to improved results on real-world ML tasks, such as backbone network load prediction, where our model outperformed state-of-the-art baselines by up to 4% in R^2 scores.

7.2 Addressing the Research Questions

- RQ1 (Benchmark Performance): Can edge-aware methods outperform traditional GNN approaches?
  - Yes, but with some constraints. On standard citation benchmarks (CORA, CITESEER, PUBMED), AttEAGNN matched or slightly outperformed node-centric baselines. This confirms that while edge-awareness is universally applicable, it is more impactful when edges carry rich semantic or physical attributes.
- RQ2 (Real-World Efficiency): Are edge-aware methods effective for real-world problems?
  - Yes. In backbone network load prediction (Abilene, RNP), AttEAGNN significantly outperformed all baselines. The error analysis (RMSE vs. MAE) revealed that our edge-attention mechanism is particularly adept at capturing peak traffic loads and bottlenecks.
- RQ3 (Computational Trade-offs): What are the costs?
  - Edge-aware learning introduces a linear computational overhead ($O(E)$). While AttEAGNN requires more training time than simple GCNs (17s vs 2s), it is notably more efficient than edge-aware transformation-based methods like CAGNN (25s), achieving a sactisfactory balance between accuracy and scalability.
- RQ4 (Accessibility): How can we promote research?
  - We addressed this by publicly releasing a modular, open-source framework that implements our model, facilitating reproducibility and providing open acess to existing models, tools and datasets.

7.3 AttEAGNN Limitations

While AttEAGNN successfully showed the value of edge-aware learning, our experimental evaluation revealed specific limitations inherent to its architectural design:

1. Dependency on Explicit Edge Features: The magnitude-based attention mechanism operates on the premise that edge importance correlates with the value of its feature vector. This proved highly effective for backbone networks where

edges represent physical capacity (bandwidth). However, in domains like citation networks (CORA, CITESEER, PUBMED), where edges are binary and lack physical semantics, the model relied heavily on augmented topological features. This dependency explains why AttEAGNN matched but did not significantly surpass node-centric baselines in those tasks, suggesting that our current attention mechanism is optimized for systems with explicit flow constraints rather than purely semantic associations.

2. Computational Overhead vs. Simple Baselines: Although AttEAGNN is more efficient than competing edge-aware models like CAGNN (17s vs 25s training time), it still incurs a computational cost approximately $8\times$ higher than simple node-centric models like GCN (2s). While this trade-off is justifiable for the significant accuracy gains in critical infrastructure prediction, it may be prohibitive for real-time applications or resource-constrained devices without further optimization.

3. Static Topology Assumption: The current implementation assumes a fixed graph structure during the message-passing phase. While the attention weights are dynamic, the underlying connectivity (the set of edges E) remains static. This limits the model's immediate applicability to highly volatile dynamic graphs where edges are continuously created and destroyed, a scenario we identified as a key direction for future work.

7.4 Limitations and Future Directions for Edge-Aware Learning

While edge-aware learning methods have demonstrated significant performance improvements across various tasks, several challenges and open research questions still remain. Current approaches often struggle with trade-offs between model complexity, computational efficiency, and applicability to dynamic, real-world scenarios. This section discusses the main limitations of existing methods and outlines future directions that could further advance the field.

7.4.1 Scalability and Computational Cost

A primary challenge for edge-aware GNNs is achieving scalability without compromising performance, particularly when applied to large, dense graphs. A significant computational bottleneck arises from methods that rely on graph transformations. Techniques employing a line graph transformation, such as CensNet and EGAT, face

a high computational cost, as the number of nodes and edges in the transformed graph can grow quadratically with the maximum node degree of the original graph.

This can render them computationally infeasible for networks with high-degree nodes. In contrast, native edge-aware GNNs like EGNN and NENN avoid this explicit transformation but introduce their own complexities. For instance, EGNN's use of a multi-dimensional edge tensor can be memory costly, especially for dense graphs. At the same time, attention-based mechanisms in both EGNN(A) and NENN introduce quadratic complexity concerning the number of nodes and edges per layer. More recent approaches, such as the Dual Hypergraph Transformation (DHT), have been proposed to mitigate some of these issues by offering a more scalable transformation that maintains a linear relationship with the original graph's size, thus providing a more efficient alternative for learning edge representations.

7.4.2 Applicability to Temporal Graphs and Evolving Taxonomies

The majority of existing edge-aware methods, as covered by our taxonomy and including AttEAGNN, are designed for static graphs, where the topology and features are assumed to be fixed. However, many real-world systems, such as social networks, financial transaction graphs, and biological interaction networks, are inherently dynamic, with nodes and edges appearing, disappearing, and changing their attributes over time. Applying static models to such temporal graphs is suboptimal as it fails to capture the evolving nature of relationships. Extending edge-aware learning to the temporal domain presents unique challenges, including the need to model dynamic edge features and develop aggregation mechanisms that are sensitive to time Barros et al. [2023]. Future edge-aware frameworks could incorporate recurrent neural networks (RNNs) or temporal attention mechanisms to capture dependencies across different graph snapshots. Consequently, our proposed taxonomy could be extended with a temporal dimension, classifying methods based on their ability to handle discrete-time dynamic graphs (sequences of static graphs) versus continuous-time dynamic graphs (event-based data).

7.4.3 Graph Neural Networks Explainability

A significant challenge in graph learning in general that naturally can be extended to edge-aware methods is to understand the workings of GNNs and the underlying reasons behind their predictions.

Since most existing graph datasets have no ground-truth explanations, GNN explainability Agarwal et al. [2023] seeks to identify what are the most relevant features for a graph analysis ML model.

In the context of our work, while we empirically shown that edge-awareness can significantly improve performance on graph learning tasks, it is often unclear why and when edge features provide the most substantial gains.

Future research should focus on developing more interpretable models that can explicitly quantify the contribution of edge features to a given prediction. Techniques from explainable AI (XAI) Ying et al. [2019], such as feature attribution or counterfactual explanations, could be adapted to measure the impact of specific edge properties. This would not only enhance model transparency but also provide domain-specific insights into which types of relationships are most critical for a task, moving beyond performance metrics to a deeper, more causal understanding of the underlying graph structure.

7.4.4 Heterogeneous Graphs and Standardized Benchmarking

Many real-world graphs are heterogeneous, containing diverse types of nodes and edges, each with distinct semantics. While some methods like edge2vec are designed for such graphs, many edge-aware GNNs implicitly assume homogeneous edge features. Developing more generalized frameworks that can effectively model and differentiate between multiple edge types simultaneously remains a key research direction. Furthermore, the field currently lacks standardized benchmarks and evaluation metrics designed explicitly for edge-centric tasks beyond link prediction. The development of standard datasets for tasks like edge classification and edge-aware graph regression would facilitate more direct and rigorous comparison between models, helping to unify and advance the field.

7.4.5 Self-Supervised and Pre-training Paradigms

Exploring advanced training paradigms like self-supervision, where models learn from the data's intrinsic structure without explicit labels. A dominant trend is the use of masked autoencoders, as seen in GraphMAE Hou et al. [2022], where models are pre-trained by reconstructing masked node features. A promising future direction for edge-aware learning is the adaptation of this paradigm to explicitly mask and reconstruct edge features, which could lead to powerful pre-trained models that are highly sensitive to relational information.

7.4.6 Beyond Graphs to Higher-Order Structures

Moving beyond traditional graphs, which only model pairwise interactions, to more expressive, higher-order topological structures. A significant frontier is the application of deep learning to simplicial and cellular complexes, as demonstrated by early

work on Message Passing Simplicial Networks Bodnar et al. [2021]. These models can associate features not only with nodes (0-cells) and edges (1-cells) but also with higher-order structures like triangles (2-cells) and tetrahedra. This allows for a much richer representation of complex systems where multi-way interactions, not just pairwise ones, are fundamental.

7.5 Final Conclusions

As the field of Graph Representation Learning grows, the initial focus on node embeddings has solved many fundamental classification problems. However, the next frontier of complex systems analysis, whether in biological interactions, supply chains, or critical infrastructure—requires a change in perspective. This thesis has argued that edges are not merely the connections between notes, but important parts of graph data representation.

By establishing a comprehensive taxonomy and proposing the AttEAGNN architecture, we have demonstrated that treating connections as relevant parts of the graph structure with learnable, dynamic representations is essential for capturing the true structure of interconnected systems. We showed that when edges are modeled explicitly, we can predict the behavior of backbone networks with improved accuracy, capturing bottlenecks and flow dynamics that node-centric models could miss.

Ultimately, this work serves as a first step toward a more holistic graph learning paradigm. As data becomes increasingly relational, the ability to learn how entities interact, rather than just what they are also grows in relevance. We hope that the open frameworks and methodologies presented here will empower the research community to explore this rich, edge-centric landscape further.

Bibliography

Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

Chirag Agarwal, Owen Queen, Himabindu Lakkaraju, and Marinka Zitnik. Evaluating explainability for graph neural networks. *Scientific Data*, 10(1):144, 2023.

Charu Aggarwal, Gewen He, and Peixiang Zhao. Edge classification in networks. In *IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1038–1049, Helsinki, Finland, 2016.

Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48, 2013.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

Wagner Almeida, Fábio Ramos, Alex V Borges, José Augusto M Nacif, and Ricardo F dos Santos. Explorando gnns sensíveis a arestas para previsão de carga em uma rede backbone. In *Workshop de Gerência e Operação de Redes e Serviços (WGRS)*, pages 84–97. SBC, 2024a.

Wagner Almeida, Fábio Ramos, Ricardo dos Santos, José Augusto Nacif, and Alex Borges. Atteagnn: Attention based edge-aware gnn applied to network load prediction. In *2024 XIV Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 1–6. IEEE, 2024b.

Wagner Almeida, Fabio Ramos, José Augusto Nacif, Ricardo Ferreira, and Alex Borges. Leveraging edge-aware graph neural networks to predict node load in backbone networks. *Journal of Complex Networks*, 13(6):cnaf050, 12 2025. ISSN 2051-1329. doi: 10.1093/comnet/cnaf050. URL https://doi.org/10.1093/comnet/cnaf050.

Wagner Almeida, Fábio Ramos, Alex B. Vieira, José Augusto M. Nacif, and Ricardo S. Ferreira. A survey on edge-aware graph learning methods. *IEEE Transactions on Network Science and Engineering*, pages 1–18, 2026. doi: 10.1109/TNSE.2025.3649386.

Sambaran Bandyopadhyay, Anirban Biswas, M. Narasimha Murty, and Ramasuri Narayanam. Beyond node embedding: A direct unsupervised edge representation framework for homogeneous networks. *[Online]*, 2019. [Online], Available: http://arxiv.org/abs/1912.05140.

Albert-László Barabási and Márton Pósfai. *Network science*. Cambridge University Press, Cambridge, 2016. ISBN 9781107076266 1107076269. URL http://barabasi.com/networksciencebook/.

Claudio D. T. Barros, Matheus R. F. Mendonça, Alex B. Vieira, and Artur Ziviani. A survey on embedding dynamic graphs. *ACM Computing Surveys*, 2023.

Piotr Bielak, Tomasz Kajdanowicz, and Nitesh V. Chawla. Attre2vec: Unsupervised attributed edge representation learning. *[Online]*, 2012. [Online], Available: https://arxiv.org/abs/2012.14727.

Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Liò, Guido F Montufar, and Michael Bronstein. Weisfeiler and lehman go cellular: Cw networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 2625–2640. Curran Associates, Inc., 2021.

Melvin A. Breuer. A class of min-cut placement algorithms. In *Proceedings of the 14th Design Automation Conference*, pages 284–290, 1977.

Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336, 1998.

Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE transactions on knowledge and data engineering*, 30(9):1616–1637, 2018.

Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *Journal of Machine Learning Research*, 23:1–64, 2022.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository, 2015.

Bin Chen, Xiao Dong, Dazhi Jiao, Huijun Wang, Qian Zhu, Ying Ding, and David J. Wild. Chem2bio2rdf: a semantic framework for linking and data mining chemogenomic and systems chemical biology data. *BMC Bioinformatics*, 11, 2009.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. Available at http://arxiv.org/abs/1412.3555.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. Metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 135–144, 2017.

Jiangang Fang, Yehuda Vardi, and Cun-Hui Zhang. *An iterative tomogravity algorithm for the estimation of network traffic*, page 12–23. Institute of Mathematical Statistics, 2007. doi: 10.1214/074921707000000030. URL http://dx.doi.org/10.1214/074921707000000030.

Vincent Franç ois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018. doi: 10.1561/2200000071. URL https://doi.org/10.1561%2F2200000071.

Z. Gao, G. Fu, and C. et. al. Ouyang. edge2vec: Representation learning using edge semantics for biomedical knowledge discovery. *BMC Bioinformatics 20*, 2019. BMC Bioinformatics 20, 2019.

Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. In *International Conference on Learning Representations*, 2018.

Yoav Goldberg. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

Aleksey Golovinskiy, Vladimir G. Kim, and Thomas Funkhouser. Shape-based recognition of 3d point clouds in urban environments. In *2009 IEEE 12th International Conference on Computer Vision*, pages 2154–2164, 2009.

Liyu Gong and Qiang Cheng. Exploiting edge features in graph neural networks. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019.

Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 9211–9219.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

Aditya Grover and Jure. Leskovec. node2vec: Scalable Feature Learning for Networks. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 855–864.

Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. Pcpnet: Learning local shape properties from raw point clouds. *Computer Graphics Forum*, 37(2):75–85, 2018.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017a.

William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017b.

Andrew C. Harvey. *ARIMA Models*. Springer, 1990.

Oliver D. N. Hope. Generalisable data-driven routing using deep rl with gnns. Master's thesis, University of Cambridge, Cambridge, 2020.

Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 594–604, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393850. doi: 10.1145/3534678.3539321. URL https://doi.org/10.1145/3534678.3539321.

Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11313–11320, 2019a.

Xiaodong Jiang, Pengsheng Ji, and Sheng Li. Censnet: Convolution with edge-node switching in graph neural networks. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019b. Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, 2019, pp. 2656–2662.

Jaehyeong Jo, Jinheon Baek, and Seul Lee et. al. Edge representation learning with hypergraphs. *Advances in Neural Information Processing Systems*, 2021. Advances in Neural Information Processing Systems, 2021, vol. 34, pp. 7534–7546.

Wei Ju, Zheng Fang, Yiyang Gu, Zequn Liu, Qingqing Long, Ziyue Qiao, Yifang Qin, Jianhao Shen, Fang Sun, Zhiping Xiao, Junwei Yang, Jingyang Yuan, Yusheng Zhao, Yifan Wang, Xiao Luo, and Ming Zhang. A comprehensive survey on deep graph representation learning. *Neural Networks*, 173, 2024.

Bharti Khemani, Shruti Patil, Ketan Kotecha, and Sudeep Tanwar. A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. *Journal of Big Data*, 11(1), 2024.

Shima Khoshraftar and Aijun An. A survey on graph representation learning methods. *ACM Transactions on Intelligent Systems and Technology*, 15(1):1–55, 2024.

Jongmin Kim, Taesup Kim, Sungwoong Kim, and Chang D. Yoo. Edge-labeling graph neural network for few-shot learning. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 11–20.

Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang Yoo. Higher-order correlation clustering for image segmentation. In *Advances in Neural Information Processing Systems*, volume 24, 2011.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems, 2018.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

Ajay Kumar, Shashank Sheshar Singh, Kuldeep Singh, and Bhaskar Biswas. Link prediction techniques, applications, and performance: A survey. *Physica A: Statistical Mechanics and its Applications*, 553, 2020.

Brenden M. Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B. Tenenbaum. One shot learning of simple visual concepts. *Cognitive Science*, 33, 2011.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 641–656, 2018.

Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, and Yi Yang. Transductive propagation network for few-shot learning, 2018. Available at http://arxiv.org/abs/1805.10002.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019.

Min Lu, Yulan Guo, Jun Zhang, Yanxin Ma, and Yinjie Lei. Recognizing objects in 3d point clouds with multi-scale local features. *Sensors*, 14:24156–24173, 2014.

Tao Luo and David Z. Pan. Dplace2.0: A stable and efficient analytical placement based on diffusion. In *2008 Asia and South Pacific Design Automation Conference*, pages 346–351, 2008.

Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, 2015.

Tasneem Mewa. Birds of a feather: Homophily in social networks, 2020. Identifying Gender and Sexuality of Data Subjects.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nature*, pages 207–212, 2021.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. Meta-learning with temporal convolutions, 2017. Available at http://arxiv.org/abs/1707.03141.

Md Golam Morshed, Tangina Sultana, and Young-Koo. Lee. Lel-gnn: Learnable edge sampling and line based graph neural network for link prediction. *IEEE Access*, 2023. IEEE Access, 2023, vol. 11, pp. 56083–56097.

Mark Newman. *Networks: An Introduction*. Oxford University Press, 03 2010. ISBN 9780199206650. doi: 10.1093/acprof:oso/9780199206650.001.0001. URL https://doi.org/10.1093/acprof:oso/9780199206650.001.0001.

Boris N. Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114, 2016.

Yao Peng, Jia Guo, and Chenyang Yang. Learning resource allocation policy: Vertex-gnn or edge-gnn? *IEEE Transactions on Machine Learning in Communications and Networking*, pages 190–209, 2024.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online Learning of Social Representations. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*, 2014. Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining., 2014, pp. 701–710.

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

Charles Ruizhongtai Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018.

Jiang-Hai Qian and Ding-Ding Han. Gravity model for transportation network based on optimal expected traffic. In *International Conference on Complex Sciences*, pages 514–524. Springer, 2009.

Lingang Qiu, Lizuo Jin, and Lin Chai. Network traffic prediction based on spatio-temporal graph convolutional network. In *2023 42nd Chinese Control Conference (CCC)*, pages 8426–8431. IEEE, 2023.

Yuxiao Qu, Jinmeng Rao, Song Gao, Qianheng Zhang, Wei-Lun Chao, Yu Su, Michelle Miller, Alfonso Morales, and Patrick R. Huber. Flee-gnn: A federated learning system for edge-enhanced graph neural network in analyzing geospatial resilience of multicommodity food flows. *Proceedings of the 6th ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery*, 2023. Proceedings of the 6th ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery, 2023, pp. 63–72.

Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B. Tenenbaum, Hugo Larochelle, and Richard S. Zemel. Meta-learning for semi-supervised few-shot classification. In *International Conference on Learning Representations*, 2018.

Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, and others. The future of digital health with federated learning. *NPJ digital medicine*, 2020. NPJ digital medicine, 2020, vol. 3, number 1, pp. 1–7.

Everett M. Rogers. *Diffusion of innovations*. Free Press, New York, NY [u.a.], 5th edition, 08 2003. ISBN 0-7432-2209-1, 978-0-7432-2209-9.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. ISSN 0033-295X. doi: 10.1037/h0042519. URL http://dx.doi.org/10.1037/h0042519.

Radu Rusu, Nico Blodow, Zoltan Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3384–3391, 2008a.

Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3d point cloud based object maps for household environments. *Robotics and Autonomous Systems*, pages 927–941, 2008b.

Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3212–3217, 2009.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29: 93–106, 2008.

Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Peter Spindler, Ulf Schlichtmann, and Frank M. Johannes. Kraftwerk2—a fast force-directed quadratic placement approach using an accurate net model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27:1398–1411, 2008.

Claudio Stamile, Aldo Marzullo, and Enrico Deusebio. *Graph Machine Learning: Take graph data to the next level by applying machine learning techniques and algorithms*. Packt Publishing Ltd, 2021.

Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H. S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1199–1208, 2018.

Aleš Svigelj, Radovan Sernec, and Kemal Alic. Network traffic modeling for load prediction: A user-centric approach. *IEEE Network*, 29(4):88–96, 2015.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077, 2015.

Jing Tao, Ken Cao, and Teng Liu. Traffic matrix prediction based on cross aggregate gnn. *2023 IEEE Intl Conf on Dependable*, 2023. 2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, 2023.

Benjamin Teitelbaum, Susan Hares, Larry Dunn, Robert Neilson, Vishy Narayan, and Francis Reichmeyer. Internet2 qbone: Building a testbed for differentiated services. *IEEE network*, 13(5):8–16, 1999.

Petar Veličković. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology*, 79:102538, 2023.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Xianbin Wan, Hui Liu, Hao Xu, and Xinchang Zhang. Network traffic prediction based on lstm and transfer learning. *IEEE Access*, 10:86181–86190, 2022.

Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018.

Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and S. Yu Philip. A survey on heterogeneous graph embedding: Methods, techniques, applications and sources. *IEEE Transactions on Big Data*, 9(2):415–436, 2022.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics*, 2019. ACM Transactions on Graphics, 2019 vol. 38, number 5, pp. 1–12.

Yunqi Wang, Yang Li, Qingjiang Shi, and Yik-Chung Wu. Engnn: A general edge-update empowered gnn architecture for radio resource management in wireless networks. *IEEE Transactions on Wireless Communications*, 2023.

Ziming Wang, Jun Chen, and Haopeng Chen. Egat: Edge-featured graph attention network. In *Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks*, pages 253–264, 2021.

Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, September 2000. ISBN 0130144002.

Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54:158–168, 1932.

Xander Wilcke, Peter Bloem, and Viktor de Boer. The knowledge graph as the default data model for learning on heterogeneous knowledge. *Data Sci.*, 1:39–57, 2017.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, may 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: A benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530, 2018.

Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets for 2.5d object recognition and next-best-view prediction, 2014. Available at http://arxiv.org/abs/1406.5670.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

Feng Xia, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan, and Huan Liu. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence*, 2(2):109–127, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *[Online]*, 2018a. [Online], Available: http://arxiv.org/abs/1810.00826.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462. PMLR, 2018b.

Shuicheng Yan, Dong Xu, Benyu Zhang, Hong-Jiang Zhang, Qiang Yang, and Stephen Lin. Graph embedding and extensions: A general framework for dimensionality reduction. *IEEE transactions on pattern analysis and machine intelligence*, 29(1):40–51, 2006.

Li Yang, Xiangxiang Gu, and Huaifeng Shi. A novel satellite network traffic prediction method based on gcn-gru, 2020.

Yulei Yang and Dongsheng Li. Nenn: Incorporate node and edge features in graph neural networks. *Proceedings of The 12th Asian Conference on Machine Learning*, 2020. Proceedings of The 12th Asian Conference on Machine Learning, 2020, pp. 593–608.

Zecan Yang, Laurence T. Yang, Huaimin Wang, Bocheng Ren, and Xiangli Yang. Bayesian tensor completion for network traffic data prediction. *IEEE Network*, 37 (4):74–80, 2023. doi: 10.1109/MNET.005.2200650.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation? In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.

Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5782–5799, 2022.

Muhan Zhang and Yixin. Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 2018. Advances in neural information processing systems, 2018, vol. 31.

Weiping Zheng, Yiyong Li, Minli Hong, Gansen Zhao, and Xiaomao Fan. Network traffic matrix prediction with incomplete data via masked matrix modeling. *Information Sciences*, 657:119835, 2024. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2023.119835. URL https://www.sciencedirect.com/science/article/pii/S0020025523014202.

Yu Zhou, Haixia Zheng, Xin Huang, Shufeng Hao, Dengao Li, and Jumin Zhao. Graph neural networks: Taxonomy, advances, and trends. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(1):1–54, 2022.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364, 2017.