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

Master's Dissertation

Resource Profiling in the Training of Graph Neural Networks

Lucas de Angelo Martins Ribeiro

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

Advisor: Prof. Dr. Nelson Duarte Lopes Filho
Co-advisor: Prof. Dr. Marcelo de Rita Pias

Rio Grande, 2023

Ficha Catalográfica

R484r Ribeiro, Lucas de Angelo Martins.
Resource profiling in the training of graph neural networks / Lucas de Angelo Martins Ribeiro. – 2023.
80 f.

Dissertação (mestrado) – Universidade Federal do Rio Grande – FURG, Programa de Pós-Graduação em Computação, Rio Grande/RS, 2023.
Orientador: Dr. Nelson Duarte Lopes Filho.
Coorientador: Dr. Marcelo de Rita Pias.

1. Redes Neurais de Grafos 2. Processamento de Grafos 3. Gerenciamento de Dados 4. Processamento Paralelo 5. Análise de Performance de Sistema I. Lopes Filho, Nelson Duarte II. Pias, Marcelo de Rita III. Título.

CDU 004

Catalogação na Fonte: Bibliotecário José Paulo dos Santos CRB 10/2344

Universidade Federal do Rio Grande
Centro de Ciências Computacionais
Programa de Pós-Graduação em Computação
Curso de Mestrado em Engenharia de Computação

DISSERTAÇÃO DE MESTRADO

Resource Profiling in the Training of Graph Neural Networks

Lucas de Angelo Martins Ribeiro

Banca examinadora:

Documento assinado digitalmente
GERSON GERALDO HOMRICH CAVALHEIRO
Data: 22/11/2023 13:49:11-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Gerson Geraldo Homrich Cavalheiro (UFPEL)

Documento assinado digitalmente
LEONARDO RAMOS EMMENDORFER
Data: 22/11/2023 10:46:28-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Leonardo Ramos Emmendorfer (UFSM)

Documento assinado digitalmente
MARCELO DE GOMENSORO MALHEIROS
Data: 21/11/2023 10:20:14-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Marcelo de Gomensoro Malheiros (FURG)

Documento assinado digitalmente
MARCELO RITA PIAS
Data: 20/11/2023 20:14:04-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Marcelo Rita Pias (FURG)
Coorientador

Documento assinado digitalmente
NELSON LOPES DUARTE FILHO
Data: 18/11/2023 13:26:31-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Nelson Lopes Duarte Filho (FURG)
Orientador

ATA DE SESSÃO DE DEFESA DE DISSERTAÇÃO DE MESTRADO

Ata nº 13/2023

Aos vinte e três dias do mês de outubro de dois mil e vinte e três, às dezessete horas, ocorreu a sessão de Defesa de Dissertação de Mestrado de Lucas de Angelo Martins Ribeiro, que apresentou a dissertação intitulada "**Resource Profiling in the Training of Graph Neural Networks**", realizada sob a orientação do Prof. Dr. Nelson Lopes Duarte Filho e coorientação do Prof. Dr. Marcelo Rita Pias. A banca examinadora foi constituída pelos Prof. Dr. Gerson Geraldo Homrich Cavalheiro (UFPEL), Prof. Dr. Leonardo Ramos Emmendorfer (UFSM) e Prof. Dr. Marcelo de Gomensoro Malheiros (FURG), sob a presidência do orientador. Após a apresentação do trabalho, a banca arguiu a candidato e, a seguir, deliberou pela

(X) aprovação da Dissertação
( ) aprovação da Dissertação, sugerindo modificações no texto
( ) reprovação da Dissertação

Rio Grande, 23 de outubro de 2023.

Documento assinado digitalmente
GERSON GERALDO HOMRICH CAVALHEIRO
Data: 22/11/2023 13:49:11-0300
Verifique em https://validar.iti.gov.br

_____
Prof. Dr. Gerson Geraldo Homrich Cavalheiro (participação remota)

Documento assinado digitalmente
LEONARDO RAMOS EMMENDORFER
Data: 22/11/2023 10:46:28-0300
Verifique em https://validar.iti.gov.br

_____
Prof. Dr. Leonardo Ramos Emmendorfer (participação remota)

Documento assinado digitalmente
MARCELO DE GOMENSORO MALHEIROS
Data: 21/11/2023 10:20:14-0300
Verifique em https://validar.iti.gov.br

_____
Prof. Dr. Marcelo de Gomensoro Malheiros (participação remota)

Documento assinado digitalmente
MARCELO RITA PIAS
Data: 20/11/2023 20:12:39-0300
Verifique em https://validar.iti.gov.br

_____
Prof. Dr. Marcelo Rita Pias (participação remota) - Coorientador

Documento assinado digitalmente
NELSON LOPES DUARTE FILHO
Data: 18/11/2023 13:13:23-0300
Verifique em https://validar.iti.gov.br

_____
Prof. Dr. Nelson Lopes Duarte Filho (participação remota) - Orientador

ACKNOWLEDGEMENTS

First of all, I thank God for the strength and courage granted to me every day during this long journey, which is just beginning. I thank my parents Marize and Matusalem for all the values, teachings, support, and sacrifices they have dedicated to me throughout my life. None of this would have been possible without you.

Next, I express my gratitude to my family members, Juliana, Nil, Fabiano, Camila, Cecília, Otávio and Matteo, for their unconditional love and support, and for being my home even from afar.

It has been many years away from you all, and yet you have always been present.

I also want to express my gratitude to the friends, colleagues, and teachers who have been part of this journey and have made the days brighter, making the journey worthwhile. Furthermore, I extend a special thanks to the professors who are part of the examining committee for their willingness to contribute and engage in this study. Finally, I would also like to thank the participating members and colleagues of the *Text of Things (ToT)* project, who have been instrumental in my intellectual and professional growth over these past years.

“But they that wait upon the LORD shall renew their strength;
they shall mount up with wings as eagles;
they shall run, and not be weary;
and they shall walk, and not faint.”
ISAIAH 40:31

ABSTRACT

RIBEIRO, Lucas de Angelo Martins. **Resource Profiling in the Training of Graph Neural Networks**. 2023. 81 p. Thesis (Master) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

Graph processing can be applied in various areas of society, technology, industry, and science to extract knowledge from real-world data. With the growth in the understanding and application of Artificial Intelligence, derived fields have emerged to explore the application of graphs using intelligent mechanisms, such as recommendation systems, social networks, among others. In this context, numerous models and frameworks for Graph Neural Networks have emerged, expanding the capabilities of these mechanisms. Despite significant advances in the study of Graph Neural Networks aimed at achieving better accuracy in models, the analysis of resources for processing these models is still an area that can be further explored to gain a deeper understanding of these systems from an architectural and execution environment perspective. In this work, a quantitative analysis of hardware resource consumption during the training phase of graph neural networks was conducted, taking into account potential benefits from using GPU accelerators when evaluating datasets of different compositions. The GCN and GraphSAGE models were used as the subject of study in this work, with implementations in the PyTorch Geometric framework. As a result, a significant improvement in the processing time of these models was observed when GPU accelerators were used, along with significant variations in the points of high resource utilization. It was also noted that datasets with different structural compositions (e.g., the number of edges and average node degree) can also exhibit significant variations in results.

Keywords: Graph Neural Networks, Graph Processing, Data management, Parallel Processing, System performance analysis.

RESUMO

RIBEIRO, Lucas de Angelo Martins. **Análise do Consumo de Recursos no Treinamento de Redes Neurais de Grafos**. 2023. 81 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

O processamento de grafos podem ser aplicados em diversas áreas da sociedade, tecnologia, indústria e ciência de modos a extrair conhecimento de dados do mundo real. Com o crescimento do entendimento e da aplicação de Inteligência Artifical, áreas derivadas surgiram com o intuito de explorar a aplicação de grafos a partir de mecanismos inteligentes, como sistemas de recomendação, redes sociais, entre outros. Neste contexto, inúmeros modelos e frameworks para uso de redes neurais de grafos emergiram, expandindo a atuação destes mecanismos. Apesar do grande avanço nos estudos das redes neurais de grafos visando obter melhores valores de precisão nos modelos, a análise de recursos para o processamento destes modelos ainda é um campo que pode ser mais explorado, de modo a se obter maior entendimento destes sistemas à partir de uma perspectiva de arquitetura e ambiente de execução. Neste trabalho, foi realizada uma análise quantitativa do consumo de recursos de hardware durante a fase de treinamento de redes neurais para grafos, levando-se em conta possíveis ganhos com o uso de aceleradores em GPU, quando avaliados datasets de constituições diferentes. Os modelos GCN e GraphSAGE foram utilizados como objeto de estudo neste trabalho, com implementações no framework PyTorch Geometric. Como resultado, observou-se um ganho significativo no tempo de processamento destes modelos quando utilizados aceleradores em GPU, além de variações significativas nos pontos de alta utilização dos recursos base. Notou-se também que, datasets de diferentes constituições estruturais (por exemplo número de arestas e grau médio dos nós) podem apresentar também grandes variações nos resultados.

Palavras-chave: Redes Neurais de Grafos, Processamento de Grafos, Gerenciamento de Dados, Processamento Paralelo, Análise de Performance de Sistema.

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
C&A	Construction and Assembly
ConvGNN	Convolutional Graph Neural Network
CPU	Central Processing Unit
ETL	Extract, Transform and Load
GAE	Graph Autoencoder
GNNs	Graph Neural Network
GPU	Graphic Processing Unit
HUP	High Utilisation Point
ISO	International Organization for Standardization
LED	Linked Engineering Documents
O&G	Oil and Gas
OLAP	Online Analytical Processing
OLTP	Online Transactional Processing
RecGNN	Recurrent Graph Neural Network
SAGE	Sample and Aggregate
SQL	Structured Query Language
STGNN	Spatial-Temporal Graph Neural Network

TABLE OF CONTENTS

1 Introduction

In the last few decades, there has been an exponential increase in real-world data being generated and made available on the World Wide Web [22]. With this sharp growth, there has also been an explosion in the number of areas and sub-areas in computing dedicated to data processing. These areas extensively explore different techniques [44], algorithms, and mechanisms capable of extracting, processing, acquiring knowledge, and generating insights to cause a new industrial, technological, and social revolution. Although known in mathematical fields for centuries, **graph processing** in computer science now refers to analysing, manipulating, and processing large-scale graphs (which can reach billions of nodes and edges). Furthermore, with the advent of Artificial Intelligence (AI), this field of study can be divided into two categories: classical graph processing, which involves calculating inherent graph properties using traditional mathematical and computational tools, and intelligent graph processing, which employs tools and methodologies from AI to explore and make predictions.

Among those techniques, graph processing can be applied in several areas of society, technology, industry, and science to extract knowledge from real-world data. For example, in the field of social sciences, graph processing can be used in social network analysis, involving the study of the structure and dynamics of social relationships and their influence on individual and collective behaviour. By using algorithms in existing graphs (graph algorithms), researchers around the world can analyse large social networks to identify key influencers [2], community detection [58], and rumours spread [63]. This information can inform and predict social media's political, cultural, or understanding impact on society.

Other areas have embraced graphs to extract knowledge using real-world data. For example, in AI applications, graph processing can be used to make personalised recommendations, such as friend suggestions on social networks or product suggestions in online stores, and countless other applications that have been applied in recent years. Graph processing is also used to detect fraud in financial systems, identify anomalous behaviour patterns, synthesise network meshes in large corporations, and improve cyber security. In the industry, graph processing has also been explored to improve the efficiency and effectiveness of its business processes. For example, in logistics industries, graph pro-

cessing can optimise delivery routes, ensuring that deliveries arrive at their destinations as quickly as possible. Additionally, in telecommunications industries, graph processing can improve service quality, quickly identifying problems in complex networks and resolving them before affecting customer service quality. However, despite the great potential in the field and the numerous studies developed recently, there are still many challenges and bottlenecks in the graph processing environments.

1.1 Graph Neural Networks

Artificial Intelligence is the field of study of intelligent agents, that is, machines capable of taking actions that maximise the chances of success in a given environment by "imitating" human cognitive activities (such as learning a new language or playing chess) [57]. Deep Learning (or Deep Machine Learning) can be considered a sub-field of Artificial Intelligence dedicated to study artificial neural networks, which can be superficially regarded as abstractions of human neurons and their interconnections [57]. Neural networks can be trained to perform numerous tasks, such as pattern detection and relationship identification in data. In this context, Graph Neural Networks [29, 61, 46] (or GNN) are present as a sub-set of Neural Networks, specialising in the processing and learning from structured data in the form of graphs.

Among the most commonly explored tasks in neural network research, natural language processing, image processing, video processing, and audio processing (data represented in grid-like formats such as sequences, matrices, or vectors) are mentioned. Graph Neural Networks can be applied to arbitrary graph-structured data, allowing the networks to capture complex dependencies and patterns within discrete entities and the relation between them [8]. Despite being a subject of study initiated in the late 20th century in computer science, it was only after 2017 that many studies and applications (for example, [34, 16, 40]) for Graph Neural Networks emerged, solving complex real-world problems. However, classical computing systems (outside the scope of AI) had been exploring the concept and applicability of graphs long before that.

1.2 Graph Pipeline

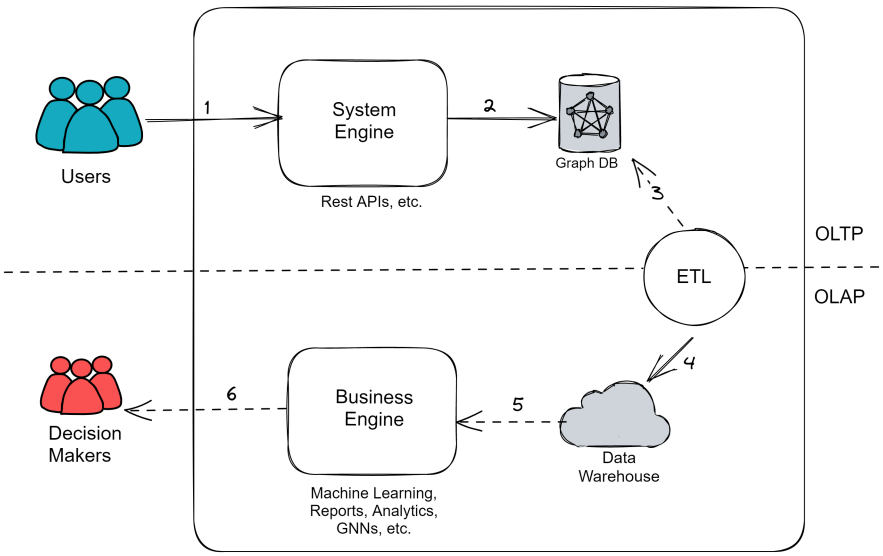
Graphs can be used in numerous different ways in an information system pipeline. However, these ways are generally grouped into two distinct stages in the architecture of an application: Online Transaction Processing (or OLTP) and Online Analytical Processing (or OLAP) [15]. The first stage consists of using the data during the real-time processing of the application, supporting the user as the end of the entire architecture (OLTP). The second stage involves using the data to obtain insights, reports, and business analytics focused on a particular area (OLAP). Although the concept is applied in Artificial Intelli-

gence applications, Business Analytics, and large-scale systems and is generally applied to microservices as a whole, the same concept is applied to graph-oriented systems.

For example, in a social network (represented as a graph, G), we can define a user as a node in this graph and each relationship or connection between users as edges. In this system, whenever a user A "accepts" the connection requested by a user B, a new edge is formed (e.g., $(A) \rightarrow (B)$), and new subgraphs can connect at this moment, resulting in new recommendations for connections between users surrounding this graph. Thus, the processing needs to be done in real time, and as a consequence, the entire social network needs an architecture focused on processing these new connections and their impacts on the network as a whole; this is the OLTP stage. On the other hand, this system could also be used by a team of researchers to study, for example, key people in the system, such as those most connected or more likely to convert a sale from an advertisement on the system. This processing can be done in real-time, and this calculus could be done using algorithms or techniques that are optional for the base functioning of the application; this is the OLAP stage.

Each of these system stages has its challenges and bottlenecks. For example, the OLTP stage needs to support a large load of creating nodes and edges in the system so that the application continues to run smoothly for the user. However, the OLAP stage needs to be able to apply algorithms (such as centrality calculations) on a very large subgraph or to be able to predict or anticipate behaviours in this network from previous information about the network.

Figure 1: Pipeline for a graph application with both OLTP and OLAP.



Source: The author(s)

For this scenario, we modelled in Figure 1 an architecture capable of representing

an application for both OLTP and OLAP stages. In this architecture, steps (1), (2), and (3) are defined as part of the OLTP phase, while steps (4), (5), and (6) are part of the OLAP phase. This example formulates a fictional scenario of a complete pipeline for using graphs in a corporate or scientific environment:

1. **Source Application**: This is the input of real-world data into the application. Based on user interaction, such as a "connection" in a social network or uploading an operational document file. This stage can be presented as a Web application or system that provides the user with an interactive environment for performing a desired task.

2. **System Modelling** This data is processed in this stage. This stage may consist of a dozen backend services for tasks such as pre-processing and data modelling through queries. In this stage, the most common transactions are of the Write type and should have the lowest latency possible to sustain the entire application. The services implemented in this scope (System Engine) can be written through Restful APIs using Python, Java, C, or other languages. A database management system capable of storing the input data that has been processed and modelled is also usually present in this stage. In our example system, Neo4j or Neptune can be good alternatives.

3. **Data Migration I (ETL)**: This stage consists of the system's ability to read the previously stored data so that this data can be used to feed the other part of the system through extraction, transformation, and loading operations. In other words, this module must be able to read a large amount of data from a source and take it to another source according to business needs.

4. **Data Migration II (ETL)**: After reading the data modelled in the previous stage, the system must be able to store this data in an environment suitable for analysis. Since these sources are usually diverse, as well as the data format, this process can be stored in either a Data Warehouse or Data Lake.

5. **Data Processing and Analytics**: A new processing module is inserted in this stage. In this module, the system should be able to provide data analysts with the appropriate tools for performing their analyses and/or obtaining insights that will feed strategic decisions. Artificial Intelligence systems (Machine Learning, Deep Learning, and others) are common in this module. In the scenario where the data is focused on using graphs, it is common to use Graph Neural Networks to predict behaviours and use tools for visual analysis and graph algorithms.

6. **Presentation**: This is the phase of using the insights and analysis obtained through graphs. They are usually presented with reports in graphs, tables, or any other way, or even with a second Web application.

The hypothetical architecture presented in Figure 1 can represent an application for both OLTP and OLAP. However, these steps shown in each of the stages can be expanded into sub-modules depending on numerous factors such as complexity of the main application (OLTP), types of analyses and insights extracted (OLAP), quantity and quality of input data (both OLTP and OLAP) among other numerous factors. Each of these modules has its own set of inherent challenges and bottlenecks, which can increase the complexity of the application in various ways or even make it unfeasible.

1.2.1 Bottlenecks

The state-of-the-art concerning these challenges and bottlenecks constantly evolves, with new developments and innovations being made regularly. Some common challenges found in the literature are:

1. **Scalability**: Graph-modeled databases can reach the scale of billions of nodes and edges; however, computer systems still have limitations such as limited bandwidth, processing power, and memory. This challenge can be addressed by developing distributed graph processing systems that efficiently handle large-scale graphs (horizontal scalability). Other techniques can be addressed by partitioning the graphs into small pieces, parallelising the processing of the structures or using more performant data structures (vertical scalability). Both horizontal and vertical scalabilities are subjects of study at this point.

2. **Data quality and integration**: Depending on the data quality, inconsistencies in the data can make it difficult to use graph processing on any part of the pipeline. Addressing this challenge involves improving data cleaning, normalisation, and integration methods to improve data quality and consistency. This can help prevent inaccuracies and errors in graph processing and analysis.

3. **Real-time processing**: To deal with this challenge, researchers are developing algorithms and systems that can process graph data in real-time, with low latency. This requires efficient data structures, algorithms, and hardware acceleration technologies like GPUs.

4. **Model interpretability**: Related in helping in better understanding the decisions made by the graph model, improving its transparency and accountability. This challenge involves developing interpretable graph neural network models that can provide insights into how the model makes predictions.

5. **Model generalisation**: This challenge is being addressed by developing graph neural networks that can generalise to unseen graph data and handle noisy or incomplete data. This is achieved through Graph Convolutional Networks, Graph Attention Networks and other types of GNN.

6. **Visualisation**: Visualisation is a big challenge that can be encountered in graph data analysis applications. Depending on the purpose of the study, the graph structure can become large enough that pure visualisation of these structures is not feasible, requiring techniques to improve this aspect, such as partitioning the graph into subgraph structures or using other data formats derived from graphs.

The OLTP stage constantly faces the challenges (1), (2), and (3) in the development of a graph application, while the OLAP stage is often facing (4), (5), and (6). Besides that, items (1) and (2) will often be present in both stages.

1.2.2 Frameworks

In response to these challenges and the emergence of Big Data concepts, large-scale parallel graph processing frameworks have arisen – for example, Pregel [48], GraphX [80] and PowerGraph [28] – to simplify the design, implementation, and adaptation of graph algorithms at scale, as well as to explore failure tolerance strategies in large systems [80]. According to Xin et al. [80], data-parallel systems, such as MapReduce [20], and Spark [85], are highly recommended for building graphs and processing extensive data. However, representing graph algorithms and computation becomes challenging in these systems, causing excessive data operations.

In his work, Guo et al. [33] presented a benchmarking of the leading graph processing platforms, more popular at the time of the study: Hadoop, YARN [71], Giraph [35], GraphLab, Stratosphere and Neo4j [60]. In Guo et al. [32], the authors report the main challenges encountered in the study [33], and raise important questions regarding the methodology for evaluating and validating graph processing platforms. These include assessing processing from the perspectives of a diversity of input datasets, the complexity of algorithms, and the peculiarities of each platform, as well as expanding the scope and importance of metrics applied to the results obtained. In the same work, four major challenges in result measurement methodologies for graph processing are presented:

- Evaluation process: the rules of the game must be well defined for a fair evaluation. This means that consideration must be given to the format of data supported by platforms, and processing flows should also be taken into account, as multi-user processing has become common in current applications, resulting in well-specific sub-atomic operations.

- Selection and design of performance metrics: execution time, resource utilisation, scalability, system overhead, cost, and energy consumption should be considered.

- Selection of the dataset: the work prioritised datasets with a significant amount of data with great internal variability that are easily transformed into different scales in similar formats, reducing the discrepancy of results.

- Algorithms evaluated: another challenge found in this work was the choice of algorithms that would be considered, where the authors grouped these algorithms by classes based on the purpose of the algorithm and its context.

In addition, the authors present three other practical challenges: (i) scalability in data selection and evaluation processes, (ii) portability between platforms, and (iii) presentation of results, pointing out the difficulties encountered in the benchmarking process of data processing in these platforms or tools that have their intrinsic differences. However, most of the frameworks and tools presented in such studies are only related to the ETL (Extraction, Transform and Load) processes in the graph pipeline and hence need the capability of performing training and modelling of GNN.

1.3 The Case Study: Graphs for Processing Linked Engineering Documents

The crucial role that information systems play in managing large companies is undeniable [49]. Documents, which have always been essential in communication, management, and contracting processes [6], are now undergoing perhaps their biggest revolution, following the technological transformations that have a significant impact on organizations [21]. Among the challenges found, one can cite the difficulty in indexing and searching for data from unstructured and semi-structured documents, data redundancy [68], the choice of the technique to be used to locate and relate data from one or more documents [59] [36], and also the difficulty of processing data at scale.

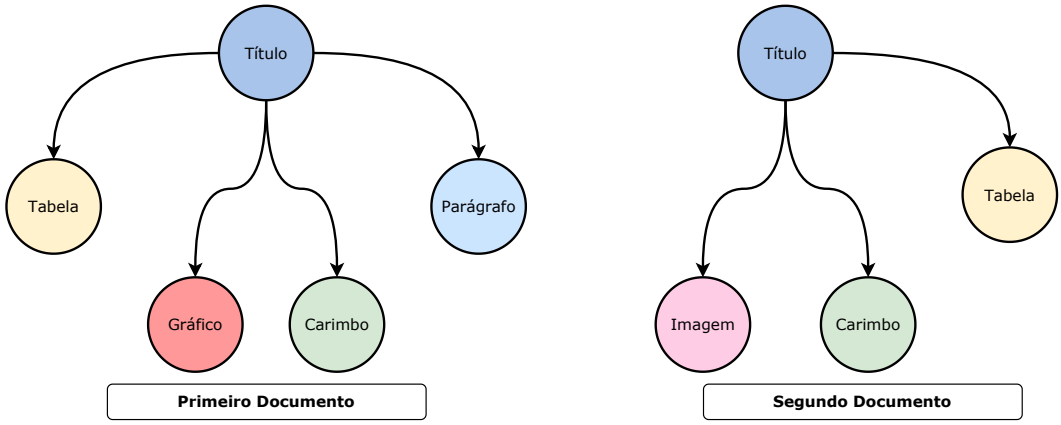
Lafetá et al. [43] pointed out the vulnerabilities faced by organizations in the Oil and Gas (O&G) sector, such as market, culture and technology issues. So, organizations need to adapt quickly to remain competitive in this market. This also highlights the need for technological solutions that ease project management processes and complexity. The study by Aragao [7] presented the use of graphs as a promising approach in the search for relationships between different documents and data insights with a high degree of relationships. Figure 2 shows an example of how the same document can be represented in graph format despite differences in structural configuration.

The work described in da Silva et al. [19] aimed to develop a computer system capable of visualizing, evaluating and manipulating Construction & Assembly, Manufacturing, and Supply data in the O&G industry, known as Databooks.

1.3.1 Previous Results

The first results at da Silva et al. [19] led us to a prototype tool for graph visualization of Databooks (named *GraphLED*). The results obtained and the validation and feedback from stakeholders were positive and showed great potential for expanding the scope and exploring new features. Currently, the proposed application is undergoing continuous

Figure 2: Representing documents in a graph model.



Source: The author(s)

testing and implementing improvement processes, and a prototype version is in use by stakeholders of the system who will provide data and evaluations regarding its behaviour in a corporate environment. Fueled by iteration cycles, the application evolves through the process of separating responsibilities – with the implementation of microservice-oriented components – giving rise to a more robust and scalable version, which has the potential for extending its functionalities. Despite being in a controlled stage, it has been possible to observe the benefits of using this architecture as reported in Newman [54] and Martin [50], such as ease of resource management (enabling high scalability in the future) and easy maintainability, since components have a reduced scope.

The final business objective is to provide analysis and insights into a highly competitive sector and, in a current scenario, an incentive for technological solutions to support administrative, executive and decision-making areas. However, some pipeline steps have been simplified in this tool to reduce the application's time-to-market. In this context, the application can perform the entire OLTP stage, but the present analysis was carried out with a Neo4j database plugin, capable of running algorithms on graphs and extracting analyses (OLAP).

1.3.2 Problems Encountered

Despite delivering a prototype version of the tool in previous works, some difficulties were found in the architecture. We grouped both the OLTP and OLAP difficulties in the application development.

1. The graph model used was developed after a pre-analysis of the engineering team (co-authors in da Silva et al. [19]) and also by technical professionals with prior knowledge of the documents, which in some way assisted in the data modelling. This means that new types of documents or documents not previously identified could not be automatically identified and processed by the system;

2. Related to the previous item, the graph modelling and node detection are rule-based (*SQL-like* based) and not self-detected, which means that out-of-pattern documents may present some processing failures. It is enough for an OLTP system, but it could be improved;

3. The processing of graph algorithms for analysis (see item 5) is implemented with a Neo4j Plugin and is highly dependent on the processing capacity of the Neo4j [60] database and plugins;

4. With the increase in the number of nodes and edges in the graphs, processing bottlenecks and memory usage are noticeably present in both frontend visualization and backend processing;

5. The Neo4j processing module demonstrates not keeping up with the scalability capabilities of the rest of the system since it cannot scale horizontally.

Even though some of these problems are related to the absence of a complete OLTP and OLAP architecture for the segmentation of each of the necessary stages for building a graph-based application, scalability is the common point that can interfere with the progress of the entire graph project.

Problems (1) and (2) can be addressed with the implementation of a Business Engine capable of making predictions and, in some way, complementing the System Engine. Problems (3), (4) and (5) show indications of how the absence of a dedicated ETL module and separation of the responsibility of a Business Engine can impact a graph system. However, despite many studies on the scalability of frameworks dedicated to the ETL phase, more is needed to address the scalability problem in a Business Engine module capable of making predictions using Graph Neural Networks. Many studies relate and evaluate the performance of systems in executing graph algorithms on large datasets (see Section 1.2.2), but there is little in the literature about how the choice of the correct framework, as well as the variability of its execution environment, can impact the application's performance.

1.4 Research Questions and Objectives

To a better understand of graph applications and address problems (3), (4), and (5) (Section 1.3.2), a study of these tools, as well as their techniques and related algorithms in a distributed processing scenario, is proposed in the Chapter 2. Some questions need to be addressed:

- (RQ1) What is the resource footprint in running Graph Neural Networks in an accelerated environment?

- (RQ2) What is the impact of choosing different well-known Models and Datasets from the literature, considering variable node degrees and graph structures?

The present work explores these two questions in more detail in the following chapters.

1.4.1 Specific Objectives

- **S01 - Study the different types of frameworks used for graph analysis, with support for the prediction.**

  Different frameworks approach scalability in graphs in different ways. Some may focus on vertical scalability, optimising memory access and structures representing the data. Others may specialise in distributed processing to achieve greater horizontal parallelism. This study will consider tools capable of training Graph Neural Networks in a relevant environment and classify them based on their scalability approach characteristics [RQ1].

- **S02 - Investigate possible performance gains in GNN model training according to changes in the execution environment.**

  This work seeks to evaluate possible performance gains from changes in their execution environment. For example, what could be the system impact if we rely on GPUs instead of CPUs [RQ1]?

- **S03 - Investigate possible performance gains according to Datasets and GNN Models changes.**

  In the same context, we seek to establish a relationship between the structures in the system (e.g., more dense datasets or complex models) that could change the scalability for training GNN [RQ2].

2 Background and Related Work

This chapter first introduces the main definitions of graph processing. Then, we also present some relevant concepts and studies in Graph Processing and its related topics. At least we mention the design concept considerations found in these studies and how they may relate.

2.1 Definitions

In this study, a graph is defined as a mathematical structure consisting of a set of nodes (or vertices), V, and a set of edges, E, connecting pairs of nodes. The mathematical definition of a graph can be represented as a tuple $G = (V, E)$, where:

- V is a finite non-empty set of nodes;
- E is a set of pairs of nodes from V, representing the edges of the graph;
- $X = \{x_1, x_2, \ldots, x_n\}$ is the node features, and;
- A is the adjacency matrix of the graph.

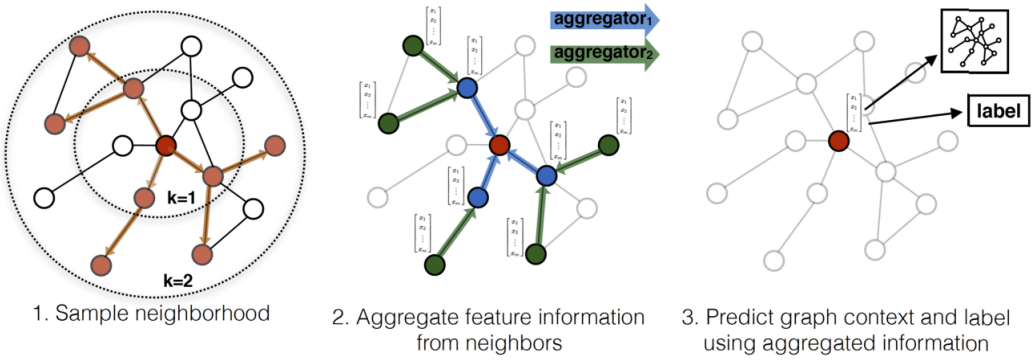
2.1.1 Graph Neural Networks

As mentioned, GNN are Neural Networks applied to arbitrary graph-structured data, allowing the networks to capture complex dependencies and patterns within discrete entities and the relation between them [8]. They are commonly mentioned in the literature for performing tasks such as Node Classification, Graph Classification, Link Prediction, Anomaly Detection and Community Detection. The literature presents a large set of GNN, often grouped inconsistently and confusingly. In this work, the taxonomy of Wu et al. [79] was used, where GNN are grouped into four categories: Recurrent Graph Neural Networks (RecGNN), Graph Convolutional Neural Networks (ConvGNN), Graph Autoencoders (GAEs), and Spatial-Temporal Graph Neural Networks (STGNN). In the study by Wu et al. [79], some applications of graph neural networks were presented; in addition to a thorough review of the terms, some considerations were made about future research directions in this area.

2.1.2 Node Classification

Node classification involves assigning a ground truth category to a node from a set of categories. In other words, it aims to find the correct label for a node based on the labels of its neighbours and nearby structures; some examples of models used for this task are GraphSAGE [34] and Graph Attention Networks [72]. In the Figure 3, the example presented by Hamilton et al. [34] shows how it is possible to use the local neighbourhood data to predict the node label.

Figure 3: Node Classification with GraphSAGE.



Source: Hamilton et al. [34].

2.1.3 Graph Classification

Graph Classification involves classifying the entire graph based on some structural property. An example of this task is the classification of molecular arrangements, where it is possible to predict the effectiveness of a molecule in combating a harmful agent [67].

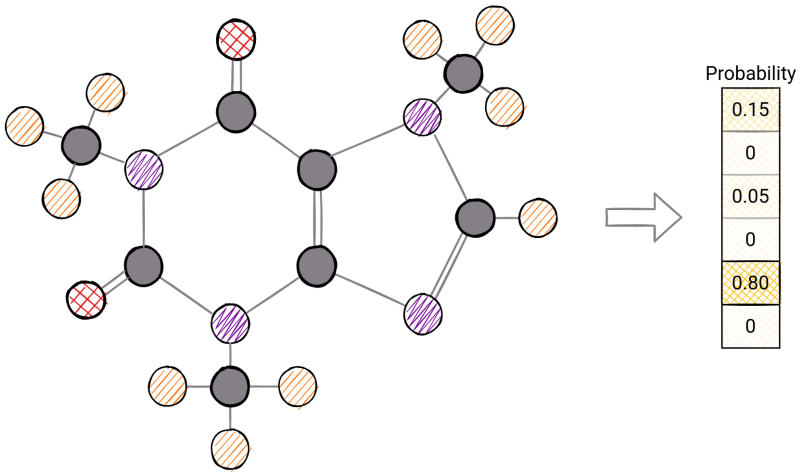
2.1.4 Link Prediction

Link prediction involves estimating a relationship between two nodes based on existing attributes and edges. Examples mentioned in the state-of-art include predicting relationships between two users in social networks, actors in an event (such as an e-mail), and more [26]. Figure 5 shows us how the task of link prediction (or edge prediction) can be used to predict relationships between entities, objects, or even people.

2.1.5 Recurrent Graph Neural Networks

Recurrent Graph Neural Networks (RecGNN) are the first proposed type of GNN that uses recurrence as a fundamental mechanism for processing structured data in graphs [61]; they were first proposed at Biochemical domain studies [61] [25] [46]. The first characteristic of RecGNN is maintaining their state throughout iterations in graph processing, allowing the model to capture temporal dependencies in the graph structure. In

Figure 4: Graph Classification Algorithm.



Source: Memgraph [53].

RecGNN, each node in the graph has its hidden state updated at each iteration based on its features and the features of its neighbouring nodes. This allows RecGNN to capture both local and global dependencies in the graph structure, making them suitable for tasks that require analysis of evolving graph data, such as node classification and link prediction. In short, RecGNN provide a flexible and powerful framework for processing applications that require analysis of temporal dependencies. The graph encoding function is defined as:

$$h_v^0 = f_{enc}(x_v)$$

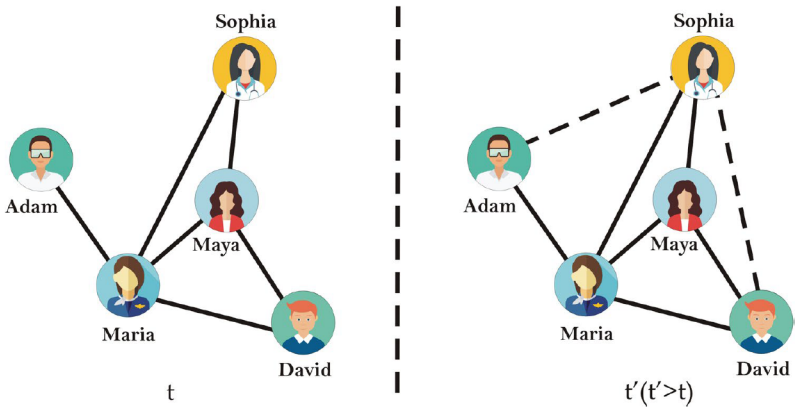
- f_{enc} is a non-linear function that maps the node features x_v to a hidden representation h_v^0.

The RecGNN take the graph representation as input and produce a new representation for each node at each time step. The computation at each time step t can be defined as:

$$h_v^t = \sigma\left(\sum_{u \in \mathcal{N}(v)} W_{uu'} h_u^{t-1} + W_{vv'} h_v^{t-1} + b\right)$$

- $\mathcal{N}(v)$ is the set of neighbors of node v;
- $W_{uu'}$ and $W_{vv'}$ are the weight matrices for the edges;
- σ is an activation function (e.g., Sigmoid, Maxout or ReLU);
- b is the bias term;

Figure 5: Graphical representation of missing link prediction. Dashed lines depict possible edges



Source: Ahmad et al. [3].

Finally, the output of the RecGNN can be computed as:

$$y_v = f_{out}(h_v^T)$$

- f_{out} is another non-linear function that maps the final node representation h_v^T to the output y_v.

2.1.6 Graph Convolutional Neural Networks

Graph Convolutional Neural Networks (ConvGNN) are GNN that apply convolutional filters to the graph structure [12]. They are designed to process graph-structured data, aggregating information from neighbouring nodes to make predictions or perform node and graph classification tasks. They have several advantages compared to other graph neural networks. For example, compared to RecGNN, ConvGNN have a simpler architecture. They do not require a recurrence to process graph-structured data, which makes them easier to train and less computationally expensive [77].

However, one of the main limitations of ConvGNN is the difficulty of capturing long-range dependencies in large graphs. ConvGNN aggregate information from neighbouring nodes, which may not be sufficient to capture complex relationships in the graph. In contrast, RecGNN and STGNN can maintain information from previous iterations, making them more suitable for capturing deeper dependencies. Another limitation of ConvGNN is the lack of ability to capture non-linear relationships in the graph structure, which is explained by the use of linear convolution filters to the graph structure. The graph convolution operation at each node v can be defined as:

$$h'_v = \sigma\left(\sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{d_u d_v}} W h_u\right)$$

- $\mathcal{N}(v)$ is the set of neighbors of node v;
- W is the weight matrix;
- σ is an activation function (e.g., Sigmoid, Maxout or ReLU);
- d_u and d_v are the degrees of nodes u and v, respectively, and;
- h'_v is the updated node representation.

The ConvGNN can be trained end-to-end using supervised learning to minimise the prediction error between the true labels and the ConvGNN predictions. Wu et al. [79] presents as examples of this group of GNN, the models: GCN [42], GraphSage [34], FastGCN [17], StoGCN [16], and Clust-GCN [18]. The same work reports differences between Spectral and Spatial Convolutional models. Additionally, it is mentioned that some types of Graph Attention Networks can fit into the ConvGNN model, while others cannot due to variations in the capturing weights (attention) from neighbouring nodes.

2.1.7 Graph Autoencoders

Graph Autoencoders (GAEs) are graph neural networks designed to learn a low-dimensional representation of graph-structured data. They consist of an encoder and a decoder, where the encoder compresses the graph data into a lower dimensional representation as $z = f_{\text{enc}}(X, A)$, and the decoder decompresses the lower dimensional representation back into the original graph data as $X' = f_{\text{dec}}(z)$.

Compared to ConvGNN, GAEs have a more flexible and robust architecture that can capture non-linear relationships in the graph structure, making them better suited for capturing complex relationships. The goal of most GAEs is to learn an encoding function f_{enc} and a decoding function f_{dec} such that $X' \approx X$, where the encoding function can be defined as:

$$z = f_{\text{enc}}(X, A) = \sigma\left(\sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{d_u d_v}} W_1 x_u + W_2 x_v\right)$$

- $\mathcal{N}(v)$ is the set of neighbors of node v;
- W_1 and W_2 are weight matrices, and;
- σ is an activation function (e.g., Sigmoid, Maxout or ReLU).

The decoding function can be defined as:

$$X' = f_{\text{dec}}(z) = \sigma\left(\sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{d_u d_v}} W_3 z_u + W_4 z_v\right)$$

- W_3 and W_4 are weight matrices

The Graph Autoencoder is trained by minimising the reconstruction loss, which measures the difference between the original graph X and the reconstructed graph X'. The reconstruction loss can be defined as:

$$\mathcal{L} = \sum_{v \in V} |X_v - X'_v|^2$$

Where $|\cdot|$ is the Euclidean norm.

Graph Autoencoders are commonly used for learning network embeddings or generating new graphs. When it comes to network embeddings, GAEs use the encoder function to extract the networks (through a ConvGNN, RecGNN, or other networks), while the decoder function is used to preserve the topology of the graph (through a multi-layer perceptron, similarity, or other technique) [79]. Wu et al. [79] presents DNGR [13] and SDNE [75] as examples of such networks.

2.1.8 Spatial-Temporal Graph Neural Networks

Spatial-temporal graph Neural Networks (STGNN) [84] is a graph neural network designed to process graph-structured data with spatial and temporal dependencies. An STGNN aims to learn a mapping from the sequence of graphs and node features to a set of node representations $Z = z_1, z_2, \ldots, z_n$ that capture the critical spatial-temporal information in the data, where the mapping can be defined as:

$$z_v^{(t)} = f_{\text{ST-GNN}}(x_v^{(t)}, \mathcal{N}_v^{(t)})$$

- $f_{\text{ST-GNN}}$ is the ST-GNN function;
- $x_v^{(t)}$ is the node feature of node v at time step t;
- $\mathcal{N}_v^{(t)}$ is the set of neighbors of node v in the graph G_t.

One of the significant advantages of STGNN compared to other graph neural networks is the ability to capture both spatial and temporal dependencies in the data explicitly. This makes STGNN well-suited for processing evolving graph data where the relationships between nodes change over time. On the other hand, one main disadvantage of STGNN is that they tend to be more computationally expensive than ConvGNN and GAEs due to the explicit incorporation of temporal dependencies in their processing of graph-structured

data. Some examples of STGNN can be found in the works of Yu et al. [84] Yan et al. [81] and Wu et al. [78].

GNN Type	Advantages	Disadvantages
RecGNN	Capturing temporal dependencies Capturing local and global dependencies	Need for intense recurrency
ConvGNN	No need for recurrency Easier to train Less computationally expensive	Capturing long-range dependencies Capturing complex relations Capturing non-linear relations
GAEs	Capturing non-linear relations Capturing complex relations No need for recurrency Less computationally expensive Model generalization	May lose small and much relevant information
STGNN	Capturing spatial and temporal dependencies Process relations over time Capturing complex relations	Classification nodes Computationally expensive

Table 1: GNN advantages and disadvantages.

2.1.9 Data Layout

In graph processing, data layout refers to how the nodes and edges of a graph are stored in memory and how they are organised to enable efficient graph algorithm processing. The choice of data layout can significantly impact the performance of graph processing tasks, such as graph traversal, search, and computation. This section introduces and explores the main types of data layouts used for graph processing.

2.1.9.1 Adjacency List

Each node is represented as an entry in an array, and the edges connecting to that node are stored as a list of neighbours. This is a simple and flexible representation, but it can result in sparse data structures and may not be as efficient as other layouts for some graph algorithms, and can be expressed as:

$$adj_{list}(v) = u \mid (u, v) \in E \text{ for each vertex } v \in V$$

- $adj_{list}(v)$ is the list of nodes adjacent to vertex v, and
- (u, v) is an edge connecting nodes u and v in graph G.

2.1.9.2 Adjacency Matrix

The graph is a two-dimensional matrix, with rows and columns representing nodes and the cells representing edges. This layout is well-suited to dense graphs but can be very memory-inefficient for sparse graphs, as in large-scale graphs, much memory space is wasted.

$$adj_{matrix}[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

- (i, j) is an edge connecting nodes i and j in graph G.
- If the entry $adj_{matrix}[i, j]$ is 1, it means that there is an edge between nodes i and j, and if it is 0, it means no edge between them.

2.1.9.3 Edge List

In this layout, the graph is represented as a list of edges, where each edge is defined as a tuple of the form (source node, target node). This layout is simple to implement but can be inefficient for some graph algorithms, as it requires a linear search to find the edges of a particular node.

$$E = (u, v) \mid u, v \in V$$

Each pair (u, v) in the edge list represents an edge connecting nodes u and v in graph G. In the case of a directed graph, the order of the nodes in the pair (u, v) is important and indicates the direction of the edge

2.1.9.4 CSR (Compressed Sparse Row)

Is a variation of the adjacency matrix layout designed for sparse graphs. It uses three lists to represent the graph: one for the node indices, one for the edges, and one for the offsets into the edge array for each node. This layout is more memory-efficient than a full adjacency matrix and can be faster for some algorithms, as it provides a more compact representation of the graph structure.

The three lists that represent the sparse matrix of the graph are:

- val is an list of size m, which stores the non-zero values of the matrix.
- col_ind is a list of size m, which stores the column indices of the non-zero values in the matrix.
- row_ptr is a list of size $n + 1$, which stores each row's starting and ending indexes in the col_ind array.

$$A_{i,j} = val_k$$

With $k \in [row_{ptr}[i], row_{ptr}[i + 1] - 1]$, and $col_{ind}[k] = j$

Here, $A_{i,j}$ is the entry in the ith row and jth column of the sparse matrix representation of graph G.

2.1.9.5 Others

There are also other forms of graph representation in memory, e.g. V-Graphs [11]. Still, the choice of data layout will depend on the characteristics of the graph and the specific graph processing tasks to be performed. Some graph processing libraries allow you to dynamically switch between different data layouts, making it easy to experiment with and find the most efficient representation for your needs. According to Shi et al. [65], when considering an execution environment using GPU accelerators, the tradeoff is in the search for minimising the PCIe bandwidth consumption while maximising parallelism and memory access. A possible solution to this problem is to divide large graphs into smaller structures. However, this approach may result in significant complexity in "reorganising" the data after processing.

2.1.10 Memory Access Pattern

Due to the focus on the SIMD architecture, the use of shared memory in blocks, and other features, GPUs are more performant in parallel data processing environments.

However, compared to traditional CPUs, GPUs are equipped with reduced-sized main memories, which can result in intrinsic limitations to processing, such as memory access conflicts and making it impossible to cache massive graphs entirely in the same memory [65].

Other points can be mentioned as potential limitations in using GPUs for graph processing, such as load unbalancing [41] (also known as *workload mapping*), branch divergence, kernel calls, and kernel configuration. Algorithms like BFS cause significant performance losses in the performance of GPUs due to branch divergence among the system threads [69].

2.1.11 Graph Algorithms

Graphs are just the means to an end. After modelling data in graph format and going through all necessary pipeline steps (depending on the chosen approach), it is essential to extract knowledge from this modelling to explore the benefits of graph representation. To do this, graph algorithms are used. The term "graph algorithms" can currently represent sets of algorithms employed for graph processing from different perspectives. From the traditional graph processing standpoint, these algorithms can be divided into (i) traversal and (ii) iterative algorithms.

Transversal algorithms are algorithms that visit (or traverse) all nodes in a graph in a systematic and pre-defined way. Their objectives usually focus on finding paths between nodes, identifying connections between nodes or subgraphs, or calculating inherent attributes of nodes and edges, such as Degree Centrality, Betweenness Centrality, and others. Among the most used are Breadth-First Search (BFS), Single-Source Shortest Path (SSSP), and Depth-First Search (DFS). Iterative algorithms perform multiple iterations on a graph to solve a problem until a stopping condition is found. They are usually implemented using multiple loop structures (such as *do..while* and *for..in*) and can update the node values during their iterations; the most popular examples are PageRank, Sparse Matrix-vector Multiplication, and Minimum Spanning Tree (MST).

Furthermore, the term "graph algorithms" can also refer to a large set of other algorithms and concepts, such as $A*$, the Floyd-Warshall algorithm, Kruskal's algorithm, Prim's algorithm, and, more recently, algorithms used in modelling GNN, regardless of the type.

2.1.12 Graph Processing

Graph processing refers to the consumption of computational resources for applying techniques and methodologies to analyse and extract insights from data in graph format. This processing can be primarily employed for (i) preprocessing of input data, (ii) processing of graph algorithms on top of preprocessed data, (iii) processing of neural network models used to predict information based on processed data, (iv) processing of output data

and visualisation of graphs. As mentioned earlier (subsection 1.2.2), each stage can be carried out using specialised frameworks and environments, and they come with specific challenges. Although used for all these stages in this work, graph processing focuses explicitly on the computational cost in the GNN model processing training phase.

2.2 The Scalability Challenge

In 1967, Amdahl [5] presented in his work a thesis on the limitation of single processing machine computing, which is popularly known and cited to this day as the Amdahl Law – which mathematically describes the limitation of a sequential algorithm processing when "compared" to parallel processing. Despite there being works in the literature that report on parallel and/or distributed processing [52] [39] [38] [4] relating computer systems from neuroscience to biology elements, John von Neumann went further, founding many of the techniques that today form the basis of computing [74] [73].

Currently, applications such as social networks, search engines, and network monitors are the most common and can require a high degree of processing and storage of these structures containing up to trillions of nodes and edges. However, only from the beginning of the 2000s did studies presenting robust systems for processing these structures emerge [66] [31] [83] [9]. Possible reasons for this are the growth of indexing systems such as Google, Wikipedia, and others.

Most studies present some points about scalability in graph neural networks, such as the cost associated with the incompleteness of the generated graphs. Among the techniques mentioned are (i) the use of data sampling, causing a node to miss its influential neighbours, and (ii) the use of data clustering, with the tradeoff being the loss of distinct structural patterns among sub-graphs. However, in this study, the authors do not consider the possibility of other types of scalability, such as the use of vertical or horizontal parallelism, as well as typical problems of large-scale systems growth, such as real-time processing, increase in the number of users and GPUs memory resource limitations.

Regarding addressing the scalability problem, two approaches are usually possible: horizontal scalability and vertical scalability. Horizontal scalability is a system's ability to handle a growing load by adding more nodes to a distributed system. This means that the system can scale proportionally to the number of resources added, such as servers, to deal with the increase in traffic or demand. This technique is usually accompanied by other methods for managing and balancing these nodes, such as load balancing, which distributes incoming requests among multiple resources, allowing the system to handle more load without reaching its limit. In contrast, vertical scalability is the ability of a system to handle a growing load by adding more power to an existing node in the system. This is usually achieved by upgrading the hardware components of the node, such as adding more RAM or a faster CPU. It can also be achieved by increasing the parallelism

of the code executed on a node. In other words, vertical scalability is about scaling a single node in the system instead of adding more nodes, as in horizontal scalability.

2.2.1 Graph Processing on GPUs

In recent years, numerous advances have been implemented in Graphics Processing Units (GPUs), such as intense parallel processing and increased memory access bandwidth, causing researchers and professionals worldwide to migrate from traditional CPUs to GPUs for specific tasks. Among these tasks, numerous studies using GPUs for graph processing have emerged [65]. Unlike CPUs that use the MIMD (Multiple Instruction Multiple Data) instruction architectures, GPUs use the SIMD (Single Instruction Multiple Data) patterns, allowing gains in parallel processing at the cost of a loss in memory space.

2.2.2 Scalability on GPUs

When it comes to GPUs, we can achieve both types of parallelism, combining the concepts of horizontal and vertical scalability with the power and performance of GPUs, depending on the scenario the application is in. Horizontal scalability can be achieved on GPUs through GPU clusters, where multiple nodes with GPUs are combined to form a single, more extensive computing system, a technique commonly used for cryptocurrency mining, for example. This allows you to expand your computing capacity by adding more nodes to the cluster. In addition, each node in the cluster can be responsible for processing a portion of the workload, and the cluster can be balanced to ensure efficient work distribution. On the other side, achieving vertical scalability with GPUs is possible through the following:

- Use of GPUs with a high number of cores: GPUs can reach thousands of cores that can work in parallel, so choosing a GPU with a higher number of cores will give you more parallel processing power.

- Take advantage of GPU parallelism: Make sure your algorithms are designed to exploit the GPU's parallel processing capabilities. For example, parallel algorithms, such as CUDA or OpenCL, perform complex operations in parallel on the GPU.

- Use batch processing: Divide your data into smaller pieces and process each element in parallel on the GPU. This can help you maximize the GPU's parallel processing capabilities and reduce the time required to complete a computation.

- Optimize memory access patterns: GPUs can process data in parallel, but their memory bandwidth can limit them. Optimizing how data is stored and accessed can reduce memory overhead and increase parallel processing performance.

2.2.3 Scaling Neural Networks

When it comes to training neural networks, some techniques using both vertical and horizontal scalability are mentioned in the literature, such as the use of pipeline parallelism [37]. In the GPipe technique, Huang et al. [37] introduces the concept of *micro-batch* to approach some of the main advantages found in the use of SPMD (*Single Program Multiple Data*), without introducing additional bottlenecks. As a result, the approach reaches linear scalability based on the number of machines used. Another possible method is partitioning a graph into smaller subgraphs and extracting samples from these subgraphs, a task initially presented by Chiang et al. [18]. However, there are indications that both approaches can break important graph structures and do not scale well with the growth of the number of hidden layers [45].

Another way to optimize graph neural network (GNN) processing is by optimizing the data structures and operations used to represent the data in memory. The Message-passing step can be very computationally expensive, and numerous studies have been conducted to optimize these processes. Yang et al. [82] used sparse-matrix dense-matrix (SpMM) multiplication on GPU, resulting in an average speed-up of 31.7% up to 4.1x speed-up when compared to other classic implementations of algorithms for message passing. The authors used Compressed Sparse Rows (CSR) structures in this implementation.

2.3 Related Work

Fey and Lenssen [24] introduced the *PyTorch Geometric* framework. Their work evaluated the performance of the framework against *Deep Graph Library* (DGL) [76], regarding the training time of ConvGNN, GAT (*Graph Attention Networks*), and RGCN (*Relational Graph Convolution Network*) models. However, the results presented for most experiments when both frameworks used the **gather and scatter** optimisation method achieved similar results (except for GAT). The study also does not explore other forms of parallelism that could optimise training results, such as using multiple GPUs or a cluster of CPU nodes. It is mentioned that possible improvements could be implemented in the framework. In December 2022, a version (*2.2.0*) was released with a series of parallelism-related implementations and support for multiple GPUs (which did not exist until then). The study also does not mention the average resource consumption during the training epochs.

Capotă et al. [14] conducted a benchmark study on the powerful graph processing platforms at the time of the study, evaluating the performance of algorithms such as breadth-first search, connected components, community detection, graph evolution, and more for a set of large-scale datasets, assessing performance in terms of processing time in both single node and cluster environments. However, the tools mentioned (Giraph, GraphX, Neo4j and MapReduce) in the study are dedicated to the ETL stage of graph processing,

and there needs to be a comparison regarding the training of any GNN.

Dwivedi et al. [23] conducted a benchmarking of a large set of different graph neural networks, mainly evaluating their performance in terms of model accuracy. The evaluation demonstrates robust and extensive results, covering various datasets and GNN. The study also presents the processing time required for each training epoch in each model. However, despite the tests being conducted in a well-equipped processing environment (GPU clusters), there needs to be more indication that parallelism and scalability methods were explored, and all models were implemented using the same framework (DGL based on PyTorch).

Abadal et al. [1] presents a survey on the main techniques, studies, and frameworks for accelerating Graph Neural Networks. The study observes the main stages of training a graph neural network, the main models used (algorithms in work), and a straightforward overview of the different accelerators. The study and classification of frameworks for GNN acceleration demonstrate a significant academic contribution to the field. However, it could be more explicit regarding the scope of use of each framework (examples of ETL in conjunction with accelerators and model creation frameworks are cited), and the taxonomy of these frameworks needs to follow their purpose. In the search for the accelerators and frameworks cited in the work, some do not have any easy-to-use API for programmers or researchers and do not have documentation supporting their use, relying only on their introduction paper. Additionally, no comparative study with benchmarking is presented between the types of models or between the different accelerators.

In Table 2, the studies were grouped according to their research aims and open gaps.

Work	Year	Research Aims	Open Gaps
Fey and Lenssen [24]	2019	Framework Definition	Don't explore multiple forms of parallelism Don't measure resource consumption during training epochs
Capotă et al. [14]	2015	Graph Tools Benchmarking	Only explore ETL frameworks
Dwivedi et al. [23]	2020	GNN Tools Benchmarking	Don't explore multiple forms of scalability Only use DGL framework
Abadal et al. [1]	2021	Graph Accelerators Survey	Don't show any comparative benchmark Confused taxonomy and comparison

Table 2: Benchmark Studies and Open Gaps.

3 Evaluation Methodology

In the development of a project or application focused on the creation, analysis, visualisation and knowledge extraction in graphs, some decisions need to be made based on the following:

- The amount and type of the input data;
- The number of end users;
- The need for GPUs or CPUs;
- The data is received in real time;
- The application will be run on-premises or in a cloud environment;
- Project budget;
- The need for training or using some GNN, among others.

Depending on the answers extracted for the above points, some tools (libraries, frameworks or toolkits) will be necessary to achieve the goal. However, the state-of-the-art on the subject could be more explicit regarding this set of tools and their best applicability is given the scenario. In most cases, the input data type will guide the project in choosing this set of tools, but this will only sometimes be the case if, for example, the project budget is a limiting factor.

Graph tools are presented in large quantities. Sometimes, they are dedicated to dealing with specific problems, such as specialised datasets or applying some algorithms. Other times, they are generic enough to be distributed as a large graph working ecosystem. To achieve the goals proposed in Chapter 1, some boundaries were set in the methodology of this study. This work aims to conduct a comparative study of the resource footprint related to frameworks used for training neural networks on graphs (GNN) and their main features related to scalability; for this, the following steps were taken:

1. Selection of the frameworks;

2. Classification of the frameworks according to some features of interest, including intuitive API for programmers, scalability methods, and relevance in the community;

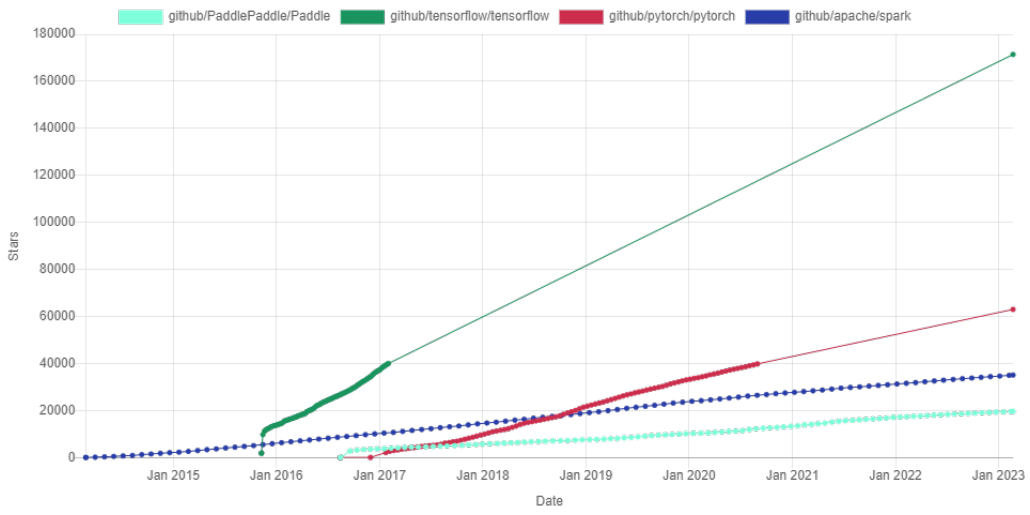
3. Elaboration of black-box scenarios with different GNN models;

4. Execution of the test scenarios;

5. Extraction of results.

Initially, the most well-known and widely used graph processing frameworks in the industry were separated into three classes: (i) *General Purpose Analytical Engines*; (ii) *ETL Graph Engines* and (iii) *GNN Engines*. Based in Nguyen et al. [55] were considered public open-source data (like *GitHub* stars and mentions) to select these tools. Some of the frameworks presented were identified using the terms **gnn, graph, tools, frameworks, deep learning** through *GitHub's* "Advanced Search" tool. Figures 6, 7, and 8 presented the evolution of GitHub stars for each of them (extracted from *CodeTabs*). Only the tools with more than 1.000 stars were extracted for this study. For this reason, some newly arisen tools (e.g., *TF-GNN*) have been removed from the set, even though they may have a considerable impact on the community in the future. Note that *GraphX* and *Giraph* (Figure 7) are both part of the Apache project as well as their codebase.

Figure 6: General Purpose Analytical Engines GitHub Stars.

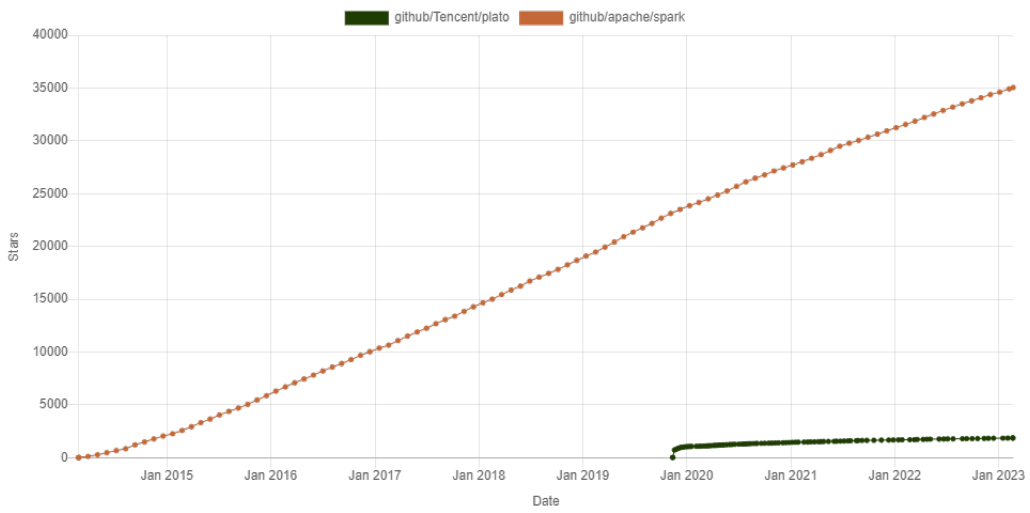


Source: *CodeTabs*.

3.1 Frameworks Selection and Classification

The group of *General Purpose Analytical Engines* is mentioned only core frameworks, where the significant examples are PyTorch, TensorFlow, PaddlePaddle and

Figure 7: ETL Graph Engines GitHub Stars.



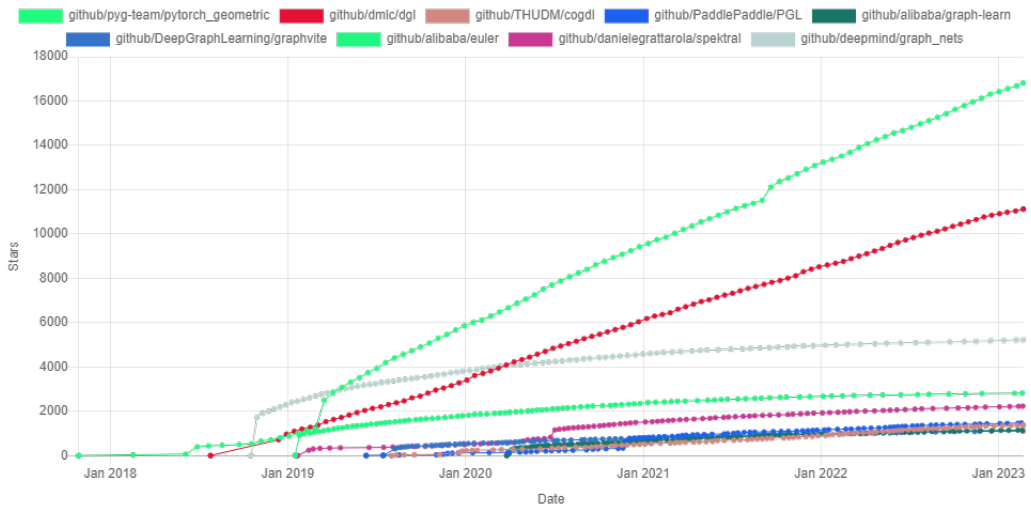
Source: *CodeTabs*.

Apache Spark. Those frameworks are the base for dozens or thousands of other toolkits and libraries. In the *ETL Engines* are grouped the frameworks that, for the most part, have been used for this purpose for decades or are branches of larger projects that have been specialised for the use in graph domains in the ETL stage, meaning that they can perform some activities as calculating Centralities, running the most used graph algorithms as PageRank, and more. As examples, GraphX e Apache Giraph and Plato (or TGraph) are mentioned in this group.

For the *GNN Engines*, only the frameworks used for Machine Learning prediction models were selected, and most of them have emerged (or are plugins) from one or more General Purpose Engines. The "GNN Engines" nomenclature refers to their ability to train Graph Neural Networks and related Machine Learning models. Although some of them can perform graph algorithm calculations, this is not their primary focus, for this group is mentioned PyTorch Geometric, DGL, CogDL, PGL, GraphLearn (or AliGraph), GraphVite, Euler, Spektral and GraphNets.

Following the extraction of these tools, they were selected for evaluation based on the following criteria: the tool (1) is dedicated to the Business Engine with a focus on intelligent systems analysis, (2) it has an API that is easy to use and understand, (3) it has easily understandable and accessible documentation, (4) the documentation is written primarily in English, (5) the main types of GNN can be implemented using the same tool, (6) the tool has indications of horizontal and vertical scalability methods, and (7) the use of these methods is straightforward to implement. Of course, this is still a work in progress (see Table 4). However, the first impression is that implementing some of these frameworks is not easy, probably because of technical immaturity (most were released only in 2020).

Figure 8: GNN Engines GitHub Stars.



Source: *CodeTabs*.

Based on the analysis of the frameworks conducted, it was possible to observe a significant advantage of the PyTorch Geometric framework compared to others, specifically when focusing on processing Graph Neural Networks. The difference in the number of stars obtained provides evidence of greater community support, which consequently offers users and researchers more accurate answers to their questions besides the ability to optimise the features over time. Utilising General-Purpose tools may require substantial initial effort to implement methods and basic interfaces for graph operations, potentially significantly extending the time necessary for GNN model analysis. In addition to the support shown in Table 4, the chosen framework also offers a wide range of functionalities dedicated to parallel and distributed processing, including Batch Processing support, GPU acceleration, processing on CPU clusters, and GPU clusters (with the latter being released during the development of this work).

It is worth mentioning that the features and analysis of the frameworks presented here may undergo variations over time. Some may experience significant improvements, become more specialised in a specific task, or even be discontinued. Furthermore, new tools may emerge, surpassing all current ones.

3.2 Profiling Resources

When evaluating the performance of applications and systems, numerous existing alternatives and methodologies are available, with context being the most significant differential among cases. Two of the most popular, USE and RED, are complementary metrics that assess Usage, Saturation, and Errors (USE acronym), or Rate, Errors and Duration (RED acronym). The USE methodology (an acronym for Utilisation, Saturation, and

Framework	Stage	Based On	Stars
PyTorch	General Purpose	-	+63k
TensorFlow	General Purpose	-	+171k
PaddlePaddle	General Purpose	-	+19.6k
Apache Spark	Mostly ETL	-	+35k
Apache Hadoop	Mostly ETL	-	+13.3k
GraphX	ETL	Spark	Unknown
Giraph	ETL	Hadoop	Unknown
Plato	ETL	Inconclusive	+1.9k
PyG	Analytical	PyTorch	+16.8k
DGL	Analytical	PyTorch/TensorFlow/MXNet	+11.1k
CogDL	Analytical	PyTorch	+1.4k
PGL	Analytical	PaddlePaddle	+1.5k
GraphLearn	Analytical	PyTorch/TensorFlow	+1.1k
GraphVite	Analytical	PyTorch	+1.1k
Euler	Analytical	TF/X-DeepLearning	+2.8k
Spektral	Analytical	Keras/TensorFlow	+2.2k
GraphNets	Analytical	TensorFlow/Sonnet	+5.2k

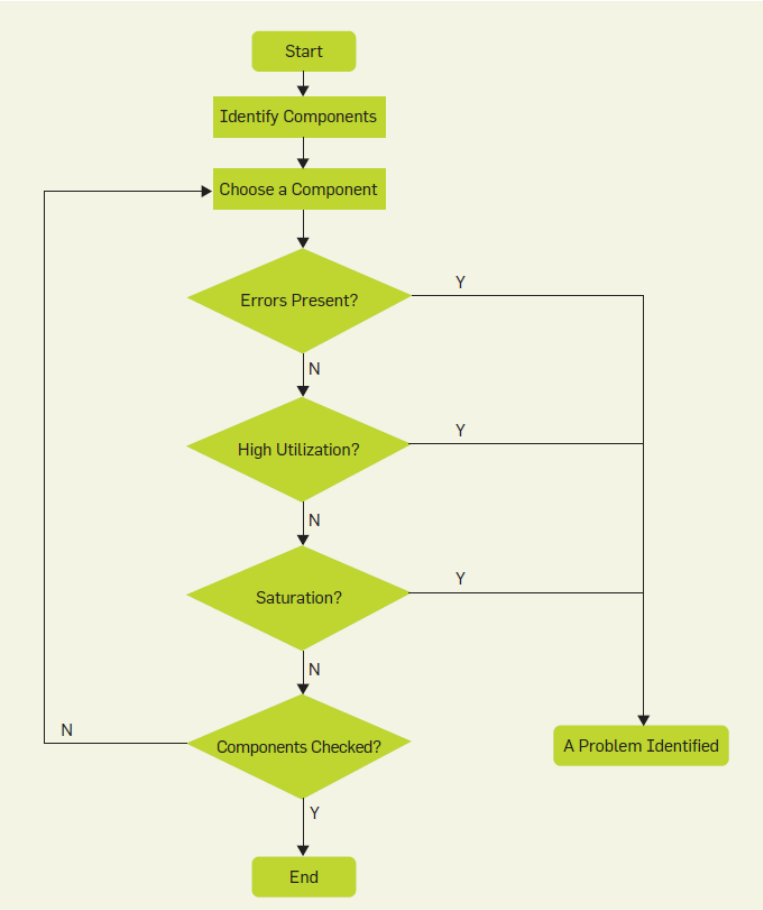
Table 3: Frameworks Stage Support and Base Tools.

Framework	Easy API	Easy Docs	GNN Features
PyG	Yes	Yes	RecGNN, ConvGNN, GAE, STGNN
DGL	Yes	Yes	RecGNN, ConvGNN, GAE, STGNN
CogDL	Yes	Yes	Inherit PyG
PGL	Inconclusive	Inconclusive	ConvGNN, STGNN, GATs variations
GraphLearn	Yes	Inconclusive	ConvGNN, STGNN
GraphVite	Inconclusive	Inconclusive	Inconclusive
Euler	No	No	ConvGNN, GAE
Spektral	Inconclusive	Yes	ConvGNN, GATs variations
GraphNets	No	No	Graph Networks

Table 4: GNN Frameworks Features Support

Errors) provides a simple and effective method for analysing system performance [30]. In this representation, Utilisation is related to the percentage of time a resource is busy performing a specific task, and saturation is when a resource cannot accept any new workload because it has reached 100% Utilisation. At the same time, Errors are the number of errors observed during the execution of this workload. The flow defined in the USE methodology can be seen in Figure 9. It is important to note that each type of resource in a computer system has its Utilisation and saturation calculated relative to its inherent unit of measurement (such as bytes for memory). In this work, two metrics referenced by these techniques were primarily utilised. From the USE methodology, consider the metric high utilization point (HUP) as a reference for peaks of 100% resource utilisation (CPU, GPU, and memory) and execution time as a reference for the duration (time to execute the task), based in the duration of the RED methodology.

Figure 9: The USE Method.



Source: Gregg [30].

3.3 Creating the Test Cases

In the cases used for testing and evaluating the frameworks, the black-box validation method was used, where the focus of the evaluation is on the results obtained as the output of the assessment in contrast to a set of inputs of the system, and the intermediate steps are seen as a large black box. In this scenario, it is essential to note that the values considered as test inputs are (1) the execution environment (CPU or GPU), (2) the type of GNN to be evaluated, (3) the dataset, and (4) the absence or presence of techniques that influence scalability. The set of output considered was: (1) the training execution time (in seconds) of the model under the specified conditions, (2) the CPU or GPU consumption (in percentage) during each epoch of training, (3) the CPU or GPU memory consumption (in GB) during each epoch of training and (4) the existence or absence of high utilisation points. The experiments were carried out as a black-box approach for all selected tools, using their specific APIs.

3.3.1 Execution Environment

For the single CPU tests, was used a machine with 2x *Intel(R) Xeon(R) CPU @ 2.20GHz* and *12GB* of RAM running in a *Google Colaboraty* Cloud environment (Colab Pro Plan). A machine with the same host CPU configuration was used for the single GPU tests, and a *NVIDIA Tesla T4* with *15GB* of dedicated memory. The execution environment is completely restarted between test suites to prevent any potential remnants from the previous execution from influencing the subsequent execution (such as Garbage Collector, Caching, etc.). Furthermore, the resource consumption before and after the training executions has been extracted.

3.3.2 Models

For the experiments, was used GCN [42] and GraphSAGE [34] models with the following hyperparameters: (i) learning rate of 0.01, (ii) optimization using the Adam gradient, (iii) dropout rate of 0.5, (iv) 16 hidden channels, (v) 2 hidden layers and (vi) ReLu as activation function. These values were established in a significant portion of the experiments presented by Kipf and Welling [42] and Hamilton et al. [34]. In the original GCN experiments, up to 200 epochs were used [42], while in the GraphSAGE experiments, originally, 10 epochs were utilised [34]. In this work, 100 epochs were used as a reference in both experiments, aiming to obtain a substantial dataset for resource consumption analysis.

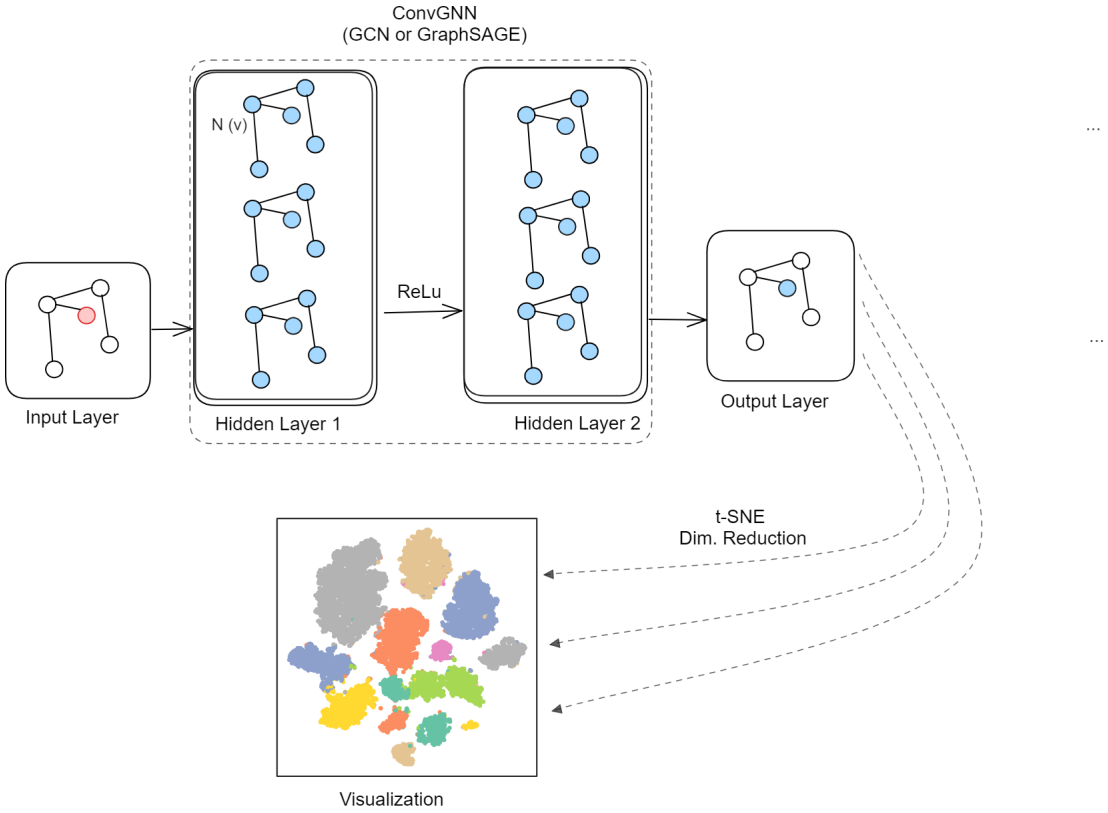
The Figure 10 represents the architecture of the models used for profiling. In the input layer, we receive the node(s) to be classified by the node classification task, then pass through the two hidden layers (GCN or GraphSAGE) that are activated by the ReLu function until they are predicted in the output layer. Finally, the graph dimension is reduced using t-SNE for model visualization. Except for the last step, hardware consumption values during model training as well as total processing time are extracted, both of which will be discussed in the following chapters.

The utilized implementation of the GraphSAGE model presents nearly double the number of parameters utilized in the GCN. As a point of comparison, the GCN model using the Amazon Computers dataset has a total of 12,458 parameters, whereas the GraphSAGE implementation presents 24,890 parameters. Examining the parameters with the CiteSeer dataset, one can observe 9,750 parameters in GCN compared to 19,478 parameters in GraphSAGE. These values were obtained using the "torch_geometric.nn.summary" function of PyTorch Geometric.

3.3.2.1 GCN

For the first set of experiments, a ConvGNN model was set with two convolutional layers, where the output of the first layer is used as input to the second convolutional layer. The model is defined by Kipf and Welling [42] with a Convolutional Operator

Figure 10: Representational Diagram of the used models (GCN and GraphSage).



Source: The author(s).

popularly known as *GCN* and were used the implementation of Fey and Lenssen [24] in PyTorch Geometric. The time and space complexity of the GCN model was analyzed in the work of Blakely et al. [10]. At each epoch iteration, were extracted the values of CPU and GPU consumption and memory usage, and at the end of the execution, were extracted the total execution time.

3.3.2.2 GraphSAGE

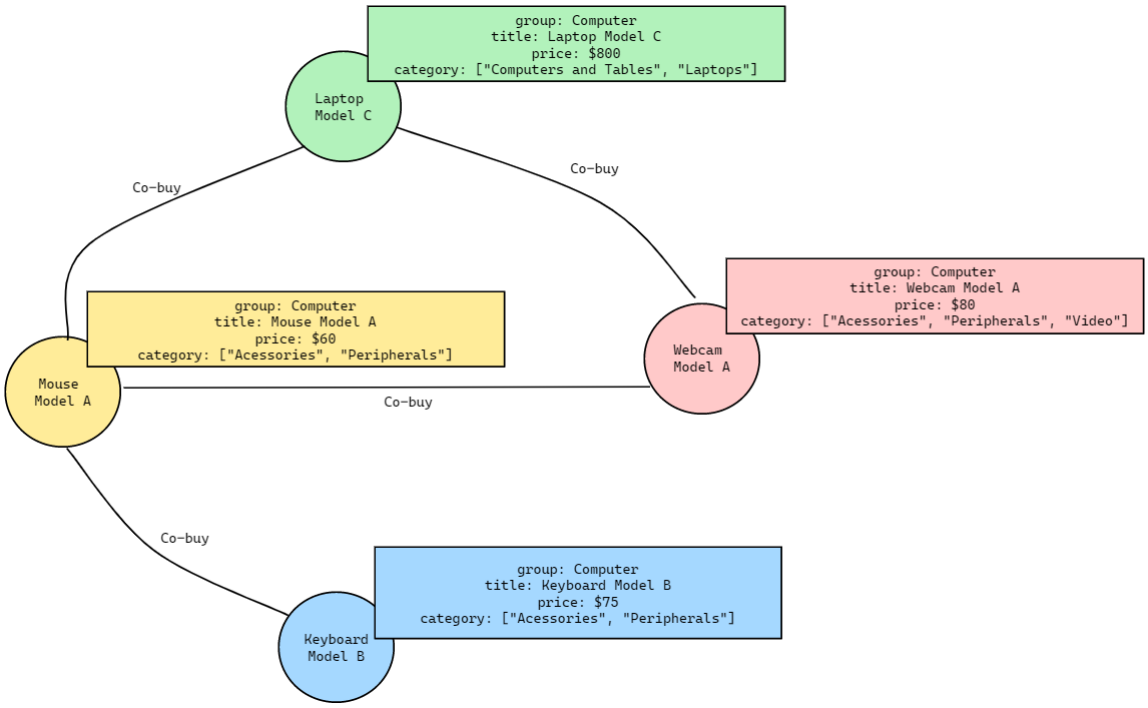
The second chosen model, also known as GraphSAGE, was presented in the work by Hamilton et al. [34]. The model uses convolutional operators, employing the same parameters given in the first experiment. GraphSAGE [34] has an aggregation mechanism for neighbouring node information, allowing it to learn a node's neighbourhood's topology and feature distribution. This way, it allows the models to process previously unknown nodes without the need for model retraining, thereby increasing the model's flexibility compared to GCN.

3.3.3 Datasets

3.3.3.1 Amazon Computers

Introduced by Shchur et al. [64], the Amazon Computers dataset is a subset of the dataset defined by McAuley et al. [51], where nodes represent goods (mainly electronics), edges represent two goods that are frequently purchased together (*Customers Who Bought This Item Also Bought*), features are a set of keywords extracted from product reviews, and labels represent the category of each product [64]. This dataset is commonly used in the literature for node classification and graph classification tasks; in Figure 11, a hypothetical and simplified example of the structures found in this dataset is presented, where different types of computer-related products that a customer commonly purchases together are depicted, forming a graph of relationships between these products. The graph in question is undirected, with unweighted edges and nodes with 767 features (although only a few are represented in the example).

Figure 11: Amazon Computers (or AmazonCoBuy) Representation.

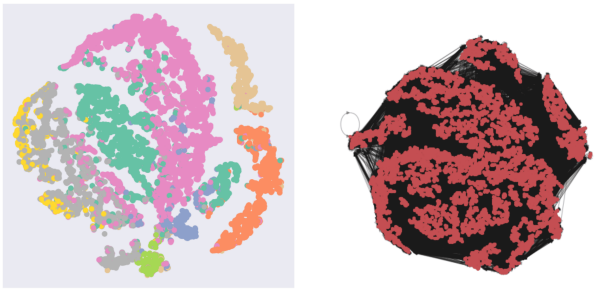


Source: The author(s).

For the experiments, this dataset has represented 767 features, along with ten classes, approximately 13700 nodes and 490000 edges, partitioned in 128 parts, with "*batch_size = 32*" resulting in 4 subgraphs with around 3500 nodes; where the nodes have an average degree of 35.76. The graph is undirected, with isolated nodes and without self-loops. Up to a point, the same partitioning strategy was used for every test case.

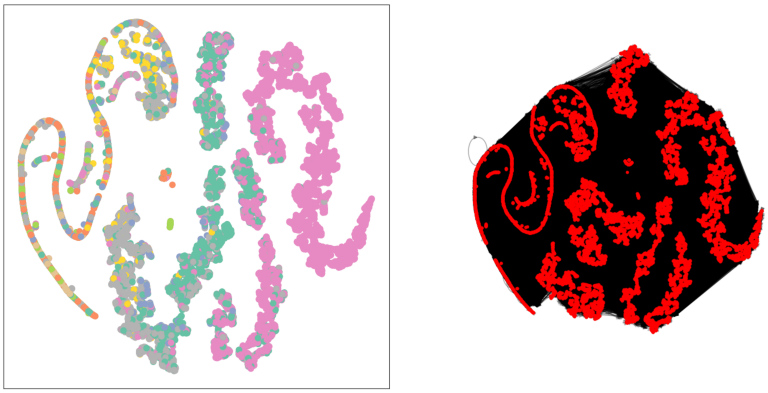
The visualisation of nodes and their respective classes with *GCN* shown in Figure 12

Figure 12: Visualisation of the Amazon Computers graph in GCN.



Source: The author(s).

Figure 13: Visualisation of the Amazon Computers graph in GraphSAGE.



Source: The author(s).

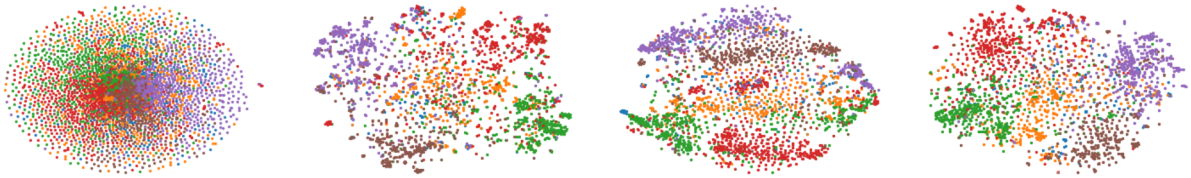
(left) was performed using t-Distributed Stochastic Neighbor Embedding (t-SNE), which is used for data with relatively low dimensions, taking around 2 minutes and 45 seconds to render. In the right of the figure, the same embedding technique was applied to reduce the graph's dimension, allowing visualisation of both nodes and edges using the draw_networkx_nodes and draw_networkx_edges methods from NetworkX. This process took approximately 40 minutes to complete rendering.

The same node visualisation presented by the GraphSAGE model (see Figure 13) took 3 minutes and 13 seconds for rendering, while the complete graph visualisation took approximately 47 minutes and 20 seconds.

3.3.3.2 CiteSeer

CiteSeer was introduced as an electronic system for indexing citations in academic literature [27]. Nowadays, it comprises a dataset used in numerous works in computer science, particularly for node classification tasks in graphs [62].

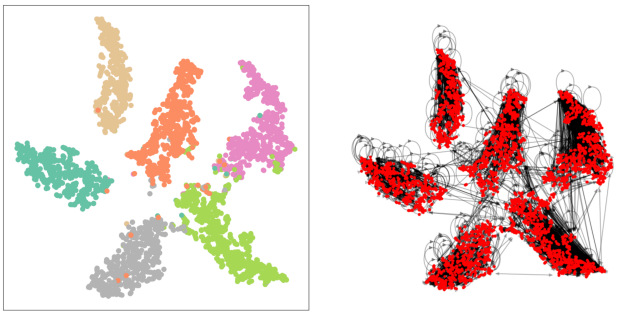
Figure 14: Visualisation of the Citeseer dataset.



(a) Citeseer(raw) (b) Citeseer(GAE) (c) Citeseer(DGI) (d) Citeseer(DBGAN)

Source: Zheng et al. [86].

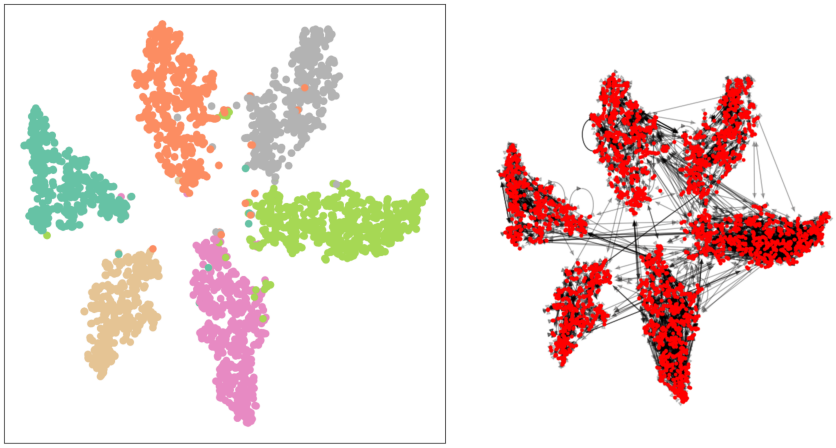
Figure 15: Visualisation of the CiteSeer dataset in GCN.



Source: The author(s).

The CiteSeer dataset was used for the experiments, which cite papers in the literature. In this dataset, 602 features are found, along with six classes, approximately 4230 nodes and 10600 edges, where the nodes have an average degree of 2.52. The graph is undirected, with no isolated nodes or self-loops. The visualisation of nodes in their respective classes shown in Figure 15 (left) was performed using t-Distributed Stochastic Neighbor Embedding (t-SNE), which is used for data with relatively low dimensions, taking around 45 seconds to render. In the right of the figure, the same embedding technique was applied to reduce the graph's dimension, allowing visualisation of both nodes and edges using the draw_networkx_nodes and draw_networkx_edges methods from NetworkX. This process took approximately 17 minutes to complete rendering.

Figure 16: Visualisation of the CiteSeer dataset in GraphSAGE.

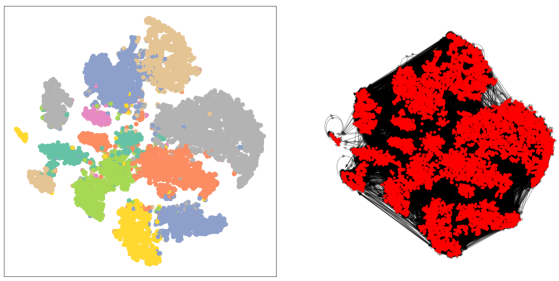


Source: The author(s).

3.3.3.3 CoAuthors (CS)

CoAuthor was introduced by Shchur et al. [64] paper. The graph represents authors (as nodes) connected by an edge if they co-authored a literature paper, where the task is to map authors to their respective fields of study by paper keywords. In the experiments, the "CS" sub-dataset was used, representing papers related to Computer Science. In this dataset, 6805 features are found, along with 15 classes, approximately 18000 nodes and 160000 edges, where the nodes have an average degree of 8.93. The graph is undirected, with no isolated nodes or self-loops.

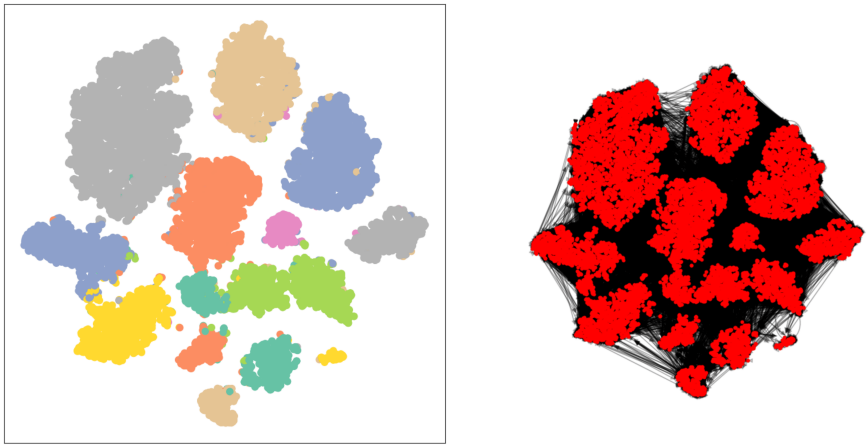
Figure 17: Visualisation of the CoAuthors (CS) dataset in GCN.



Source: The author(s).

The visualisation of nodes in their respective classes shown in Figure 17 (left) was performed using t-Distributed Stochastic Neighbor Embedding (t-SNE), which is used for data with relatively low dimensions, taking around 4 minutes to render. In the right of the figure, the same embedding technique was applied to reduce the graph's dimension, allowing visualisation of both nodes and edges using the draw_networkx_nodes and

Figure 18: Visualisation of the CoAuthors (CS) dataset in GraphSAGE.



Source: The author(s).

draw_networkx_edges methods from NetworkX. It took approximately 17 minutes and 30 seconds to complete rendering.

Although the graphs used inherently do not possess self-loops, this phenomenon can be observed in some cases after reduction through t-SNE. In certain instances, this can be attributed to anomalies resulting from the dimension reduction process, while in other cases, it may stem from the high density of nodes and edges within a limited visualization space.

3.3.3.4 Synthetic Datasets

Tests and parallel experiments were conducted using synthetic datasets, particularly with the FakeDataset and GeometricShapes methods from PyTorch Geometric. However, the accuracy values obtained with these methods – ranging from 10000, 100000, and 500000 nodes and with graph configurations similar to the previous datasets – were extremely low (between 0.10 and 0.12), making them less relevant when compared to others, considering their fidelity to the real world data. Therefore, the results obtained with these datasets were not explored in-depth in this study, although they appear to offer a significant opportunity for future research.

4 Results

The experiments were conducted in both scenarios: training with CPU only and training with CPU accelerated by GPU, using the PyG framework API for the GCN and GraphSAGE models, applied to the Amazon Computers, CiteSeer and CoAuthors (CS) datasets. Table 5 summarises the main results obtained during the training and validation of the models, presenting the environment configurations first, followed by the number of high utilisation points (HUP in units) for CPU and GPU, and finally, the maximum accuracy achieved in the validation phase. Although only one execution of resource profile behaviours is presented, it can be observed through the repetition of executions that, despite minor variations, these are normal behaviours. Changes in the execution environment, such as the local time of the execution, may interfere with the results; however, these cases are inherent to the use of Cloud environments. All the extracted results are presented on GitHub[1].

4.1 GCN

Training the GCN with the **Amazon Computers** dataset was possible to observe 33 high utilisation points (see Figure 19), resulting in 33.3% of the trained epochs when evaluating the CPU's average consumption (all cores). The total training time using only CPUs was approximately 1 minute. On the other hand, using GPU acceleration, it was impossible to observe any HUP on the host or the device. The maximum GPU consumption did not reach 20% in any training epoch (see Figure 20). The total GPU acceleration training time was approximately 7 seconds – resulting in an 8.5x gain. The memory consumption rate remained stable in both scenarios, with slight variation when using CPUs. However, it is essential to note that the memory read and write rate during the data loading stage in memory was not calculated but only during training.

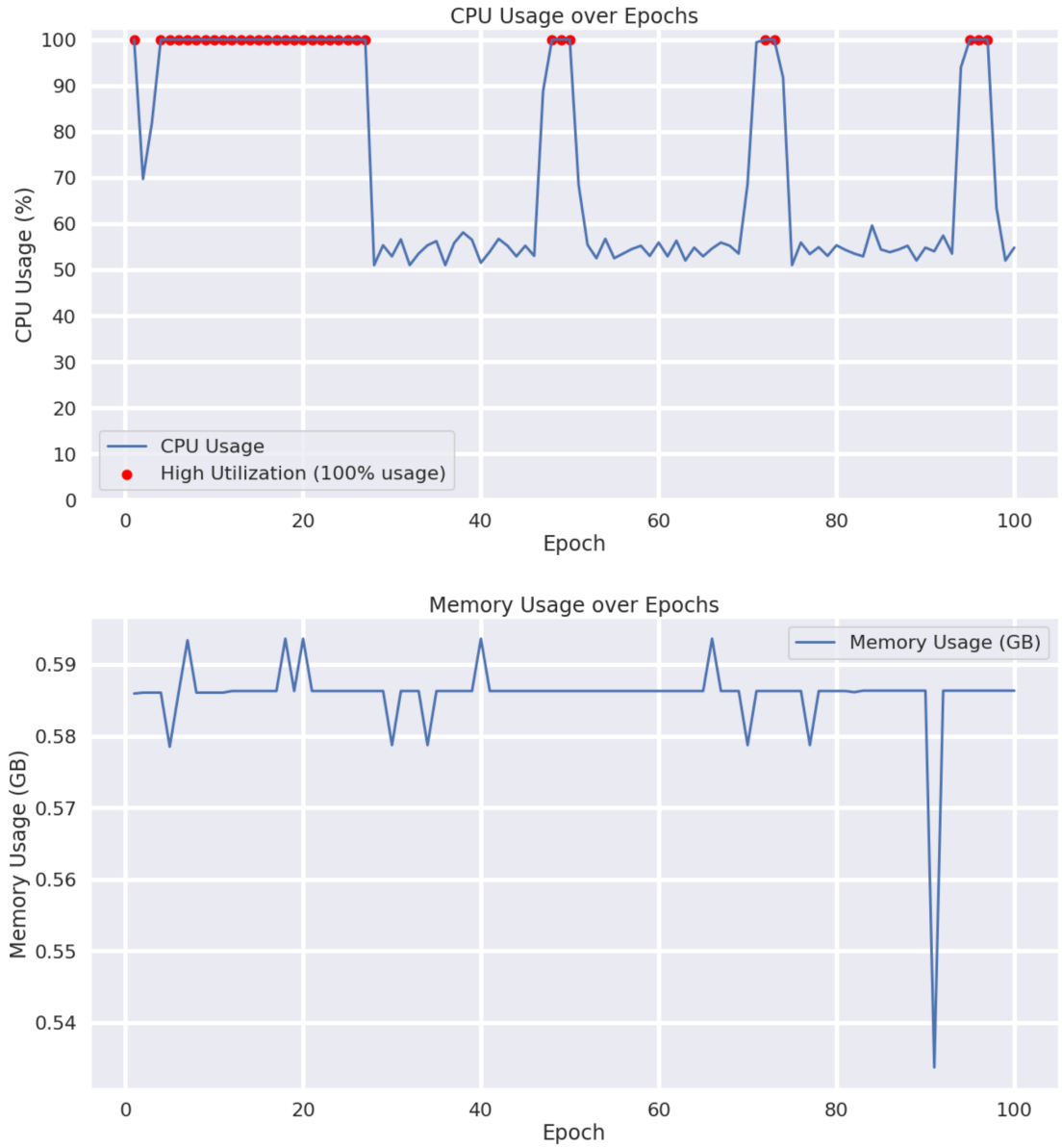
With the **CiteSeer** dataset, it was possible to observe 15 high utilisation points (see Figure 21), resulting in 15.0% of the trained epochs when evaluating the CPU's average consumption (all cores). The total training time using only CPUs was approximately 23

[1] `https://github.com/luucasrb/MasterResults`

seconds. Using GPU acceleration, it was possible to observe 64 (64%) peaks of high utilisation on the host machine but none on the device. The maximum GPU consumption is about 3% (see Figure 22). The total GPU acceleration training time was approximately 22.8 seconds – resulting in 1,009x gain. The memory consumption rate remained stable in both scenarios, with slight variation when using CPUs.

With the **CoAuthors** dataset, it was possible to observe 14 high utilisation points (see Figure 23), resulting in 14.0% of the trained epochs when evaluating the CPU's average consumption (all cores). The total training time using only CPUs was approximately 55.7 seconds. Using GPU acceleration, it was possible to observe 16 peaks of utilisation on the host (16.0%), but the maximum GPU consumption reached 20% (see Figure 24) with no point of high utilisation. The total GPU acceleration training time was approximately 16.4 seconds – resulting in a 3.3x gain. The memory consumption rate remained stable in both scenarios, with slight variation when using only the CPU.

Figure 19: Training GCN with Amazon Computers - CPU only.

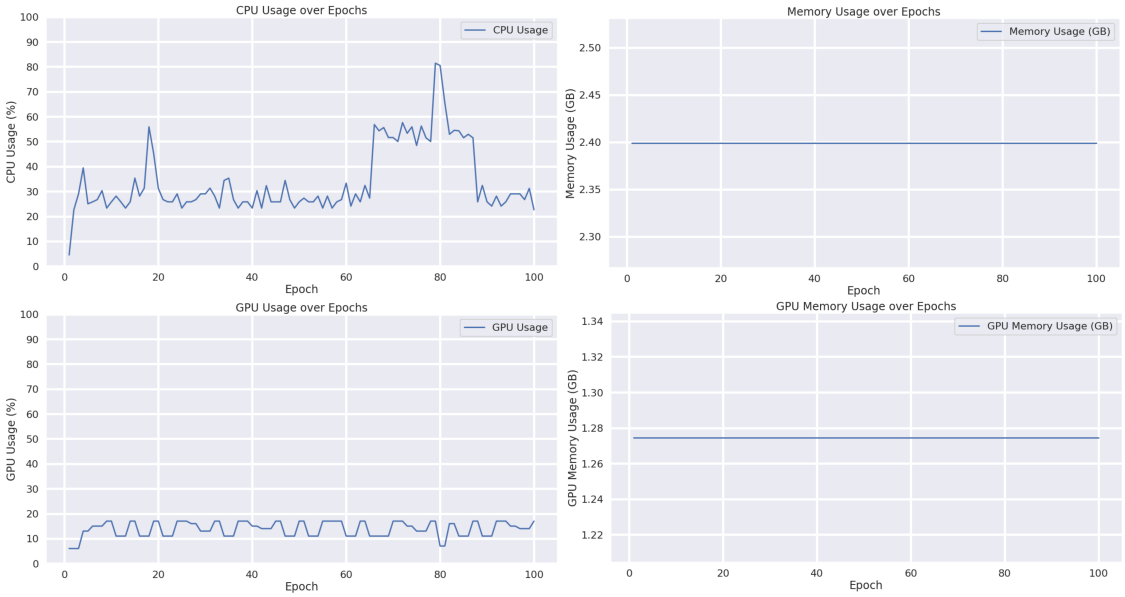


Source: The author(s).

In Figure 19, the training of the GCN model with the Amazon Computers dataset is presented[2], where it is possible to observe the 33 points of high CPU utilisation, meaning periods where the average consumption of all CPU cores is at 100% utilisation. The critical period occurs in the first quarter of execution (between epochs 0 and 25). The x-axis represents epochs, while the y-axis represents the percentage of utilisation. In the second part of the figure, the consumption of main memory (in GB) during epoch execution is shown, where a low change-over in memory consumption can be noticed (between 0.54 and 0.6 GB).

[2]Amazon Computers represents the related electronic products dataset, containing approximately 13700 nodes, 490000 edges, 767 features, 10 classes, and an average node degree of 35.76, as defined in Section 3.3.3.1. Using the GCN model defined in Section 3.3.2.1 for the node classification task.

Figure 20: Training GCN with Amazon Computers - GPU Accelerated.



Source: The author(s).

In Figure 20, the execution of the same (GCN-Amazon) scenario using both CPU (host) and GPU (device) is observed, where no points of high utilisation are visible on both the host and the device. The x-axis represents epochs, while the y-axis represents the percentage of utilisation. The consumption of both memories remained stable. In the right side of the figure, the consumption of both main and device memories (in GB) during epoch execution is shown, where a constant memory consumption can be noticed (2.4 GB for the main memory and 1.25 GB for GPU memory).

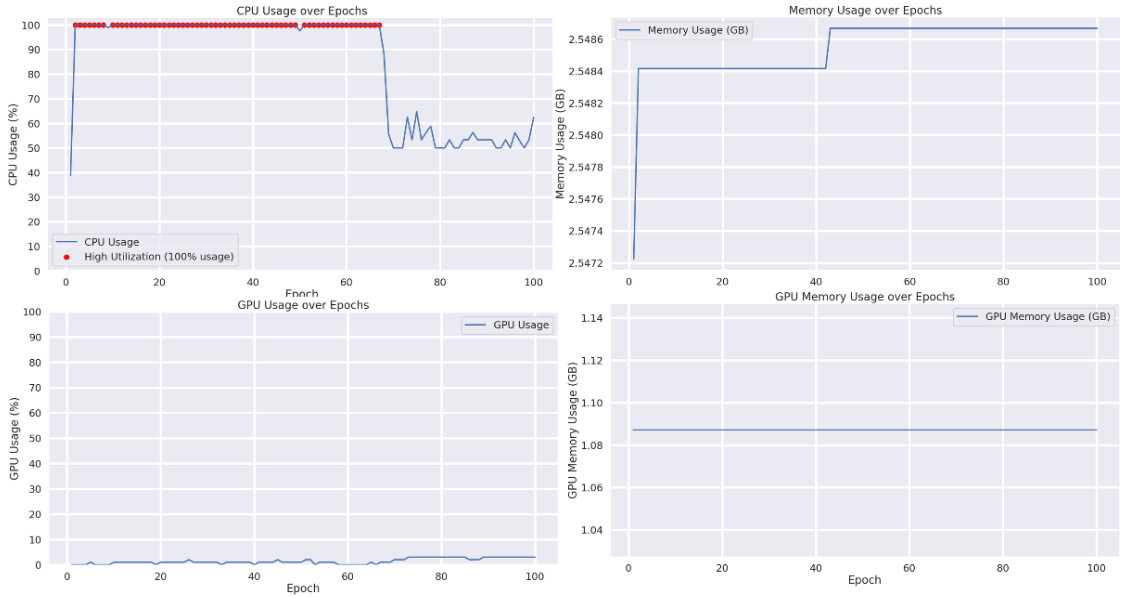
Figure 21: Training GCN with CiteSeer - CPU only.

Source: The author(s).

In Figure 21, the training of the GCN model with the CiteSeer dataset is presented[3], where it is possible to observe the 15 points of high CPU utilisation. The critical period occurs between epochs 60 and 80. The x-axis represents epochs, while the y-axis represents the percentage of utilisation. In the second part of the figure, the consumption of main memory (in GB) during epoch execution is shown, where a low change-over in memory consumption can be noticed (0.511 to 0.514 GB).

[3]CiteSeer represents the academic citation dataset, containing approximately 4230 nodes, 10600 edges, 602 features, 6 classes, and an average node degree of 2.52, as defined in Section 3.3.3.2. Using the GCN model defined in Section 3.3.2.1 for the node classification task.

Figure 22: Training GCN with CiteSeer - GPU Accelerated.

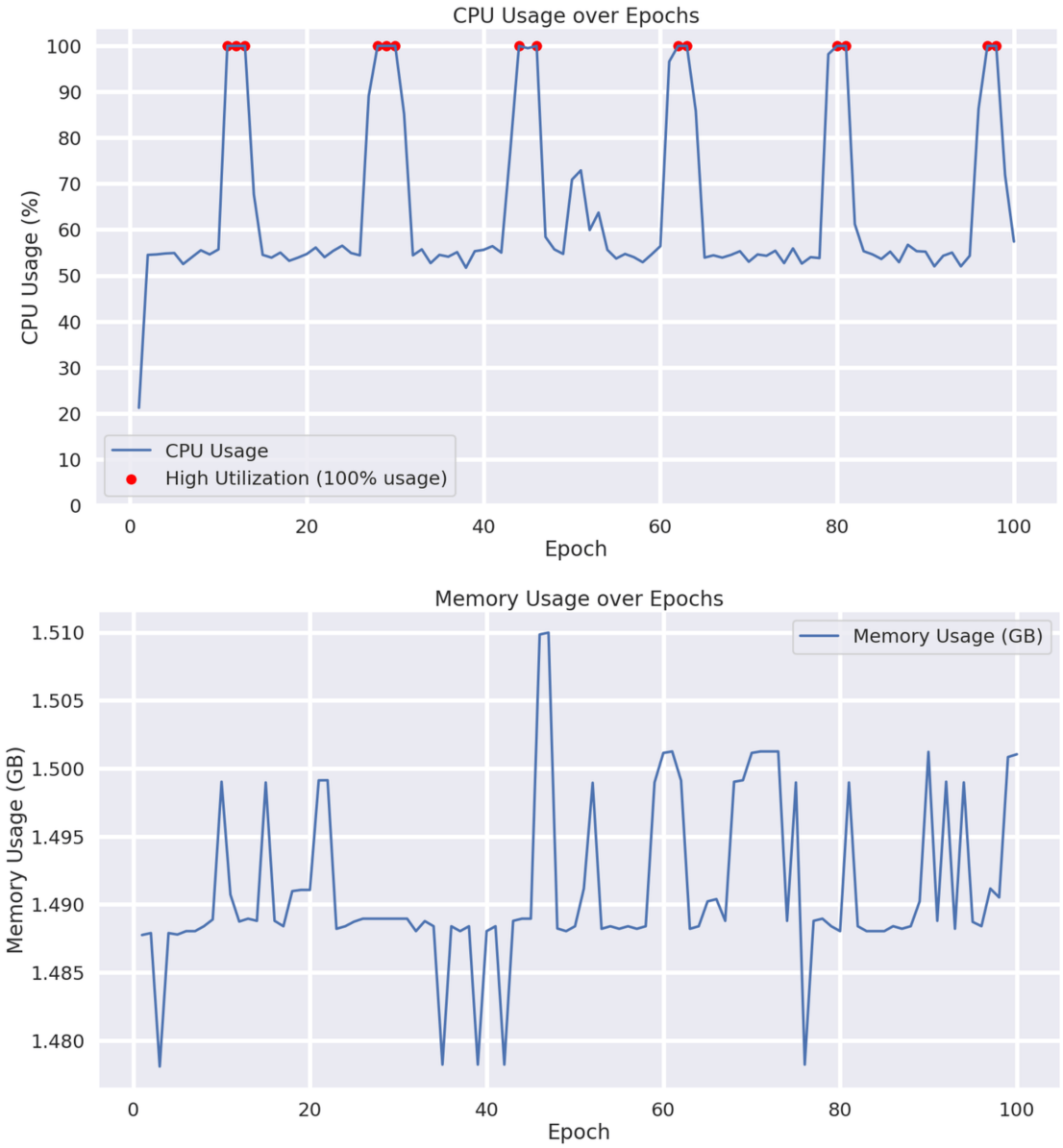


Source: The author(s).

In Figure 22, the execution of the same (GCN-CiteSeer) scenario using both CPU (host) and GPU (device) is observed, where 64 points of high utilisation are visible on the host but no HUP on the device.

From the first to the 67th epoch, the host exhibited high utilisation while the device demonstrated low consumption. The x-axis denotes epochs, while the y-axis represents the percentage of utilisation. On the right side of the figure, the consumption of the main and device memories (in GB) during epoch execution is depicted, with both memories showing almost stable consumption and minimal fluctuation.

Figure 23: Training GCN with CoAuthors CS - CPU only.

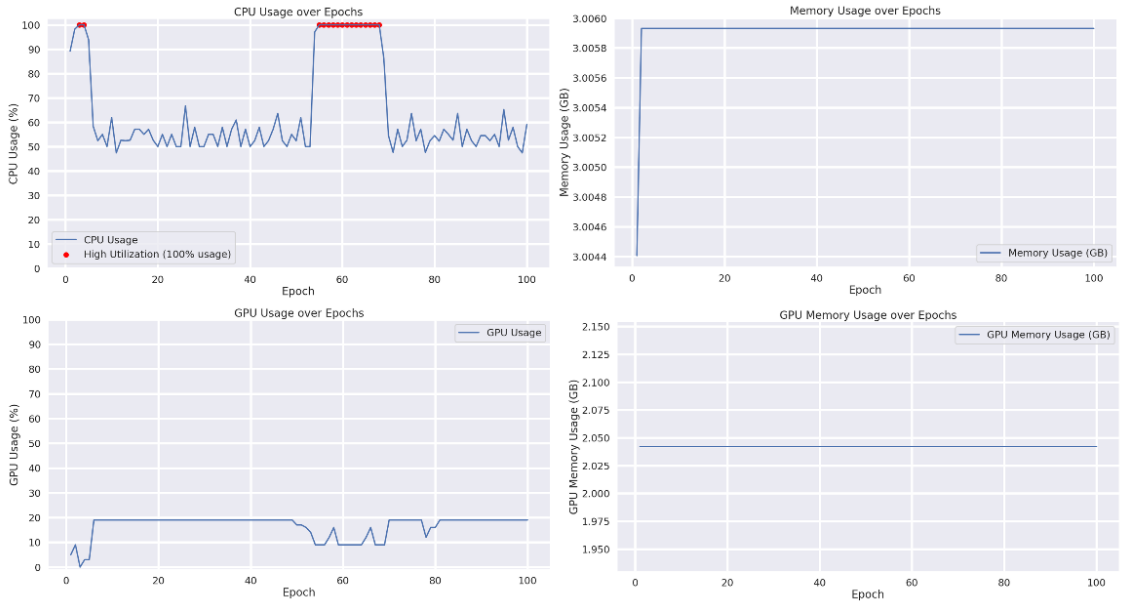


Source: The author(s).

In Figure 23, the training of the GCN model with the CoAuthors (CS) dataset is presented[4], where it is possible to observe the 14 points of high CPU utilisation. Unlike previous executions, here the machine's behaviour shows crest and troughs that alternate approximately every 10 epochs, forming a pattern similar to a sinusoidal wave. The x-axis represents epochs, while the y-axis represents the percentage of utilisation. In the second part of the figure, the consumption of main memory (in GB) during epoch execution is shown, where a low change-over in memory consumption can be noticed in a pattern similar to the consumption graph.

[4]CoAuthors (CS) represents the literature co-authored authors dataset, containing approximately 18000 nodes, 160000 edges, 6805 features, 15 classes, and an average node degree of 8.93, as defined in Section 3.3.3.3. Using the GCN model defined in Section 3.3.2.1 for the node classification task.

Figure 24: Training GCN with CoAuthors CS - GPU Accelerated.



Source: The author(s).

In Figure 24, the execution of the same (GCN-CoAuthors) scenario using both CPU (host) and GPU (device) is observed, where 16 points of high utilisation are visible on the host but no HUP on the device. Between the epochs 50 and 70 the host machine reaches its peaks. The x-axis represents epochs, while the y-axis represents the percentage of utilisation. In the right side of the figure, the consumption of the main and device memories (in GB) during epoch execution is shown, the consumption of both memories remained almost stable with a very low change-over.

4.2 GraphSAGE

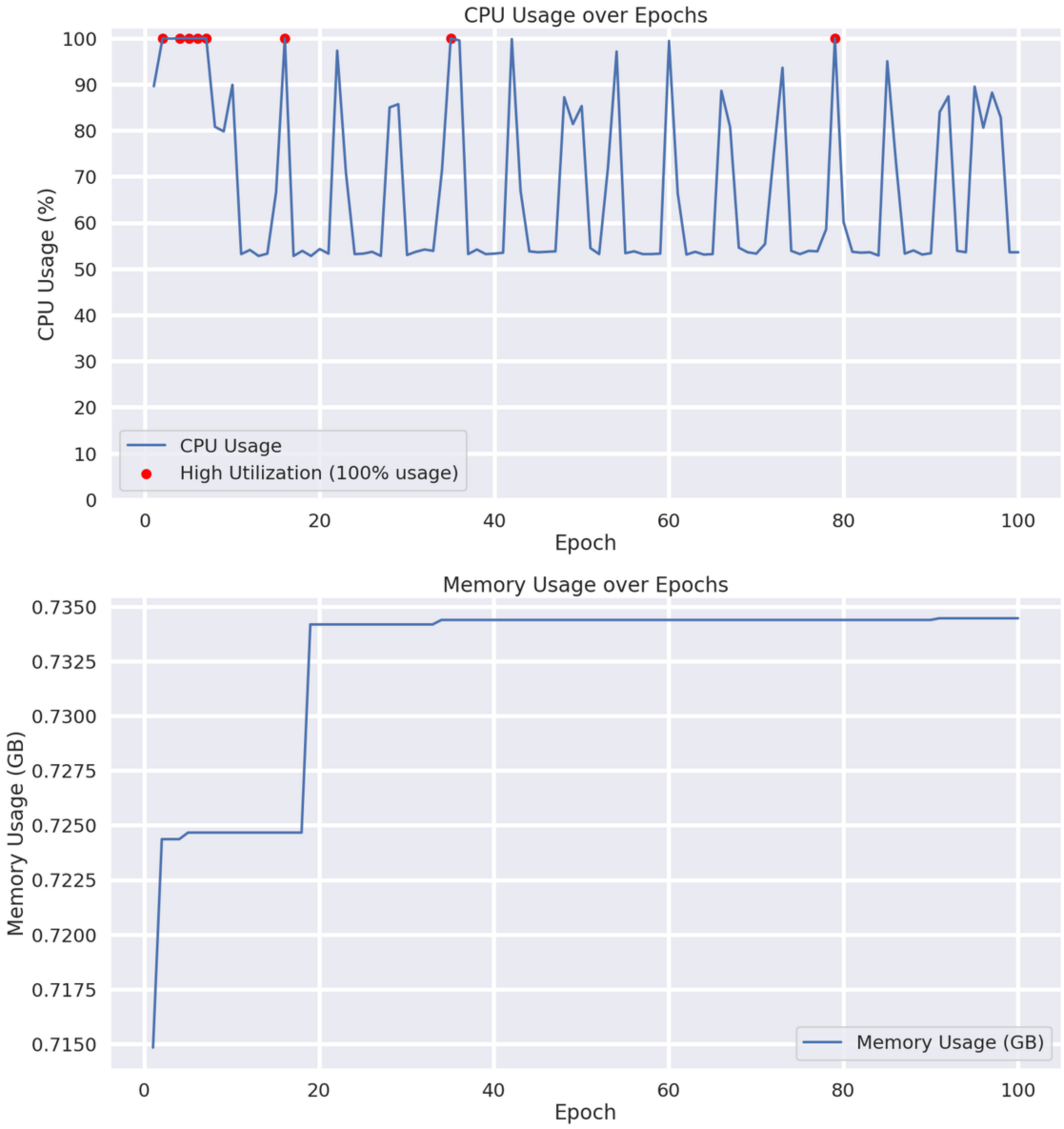
Training the **Amazon Computers** with GraphSAGE, it was possible to observe 8 points of high utilisation (see Figure 25), resulting in 8% of training epochs when evaluated using only CPU execution. The total training time was 3 minutes and 48 seconds, and the memory consumption was slightly higher when compared with the GCN profile. Using the GPU-accelerated environment for training the GraphSAGE network, it was possible to observe a total training time of 24.8 seconds – resulting in a 9.1x gain. However, there was an increase to 65 (65%) HUP on the Host machine, in addition to obtaining a peak GPU device utilisation of 60%. However, it has still not been possible to observe any points of high utilisation (see Figure 26).

Training with the **CiteSeer** with CPU only took approximately 32 seconds, with 78 points of high utilisation (78% of the training epochs) observed (see Figure 27). Meanwhile, training the model using GPU acceleration took around 18 seconds – resulting in 1.7x gain – where 43 HUP (43%) were observed on the Host machine, and no points of

high utilisation were observed on the GPU device, with a peak usage of only 6% (see Figure 28).

The **CoAuthors** dataset with CPU only took approximately 17 minutes and 32 seconds, with 0 points of high utilisation observed (see Figure 29). Meanwhile, training the model using GPU acceleration took around 38 seconds – resulting in 27.6x gain – where 21 points of high utilisation (21%) were observed on the Host machine and 9 points of high utilisation on the GPU device (see Figure 30). So far, it's the only experiment where we've observed a point of high utilisation on the GPU device.

Figure 25: Training GraphSAGE with Amazon Computers (CPU only) resources.

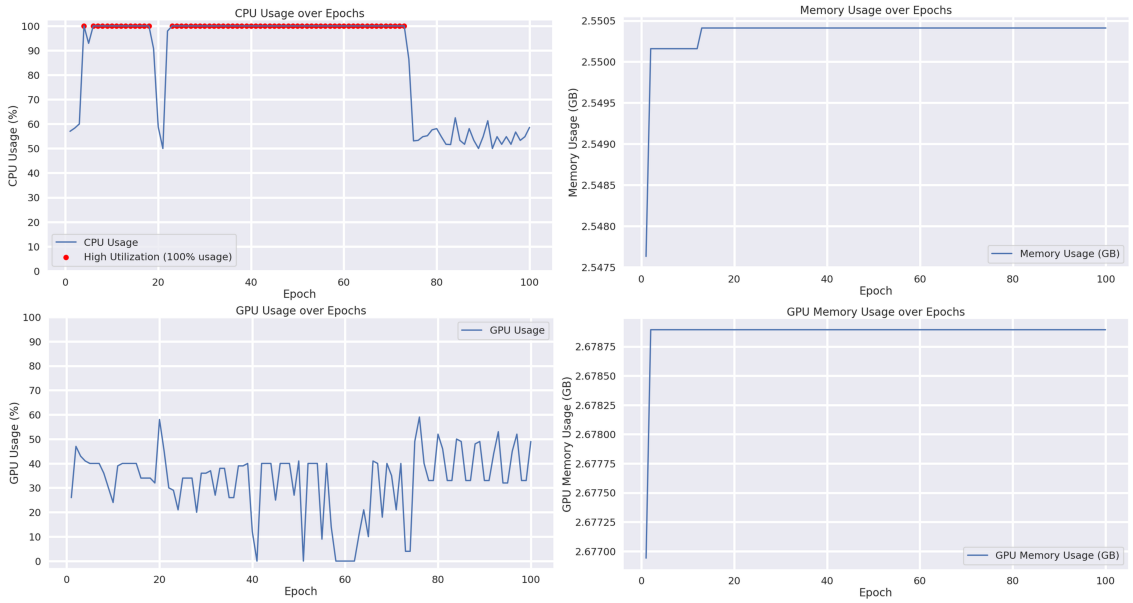


Source: The author(s).

In Figure 25, the training of the GraphSAGE model with the Amazon Computers dataset is presented[5], where it is possible to observe the 8 points of high CPU utilisation. Here the machine's behaviour shows crest and troughs that alternate approximately every 5 epochs. The x-axis represents epochs, while the y-axis represents the percentage of utilisation. In the second part of the figure, the consumption of main memory (in GB) during epoch execution is shown, where a low change-over in memory consumption can be noticed (between 0.715 and 0.735 GB).

[5] Amazon Computers represents the related electronic products dataset, containing approximately 13700 nodes, 490000 edges, 767 features, 10 classes, and an average node degree of 35.76, as defined in Section 3.3.3.1. Using the GraphSAGE model defined in Section 3.3.2.2 for the node classification task.

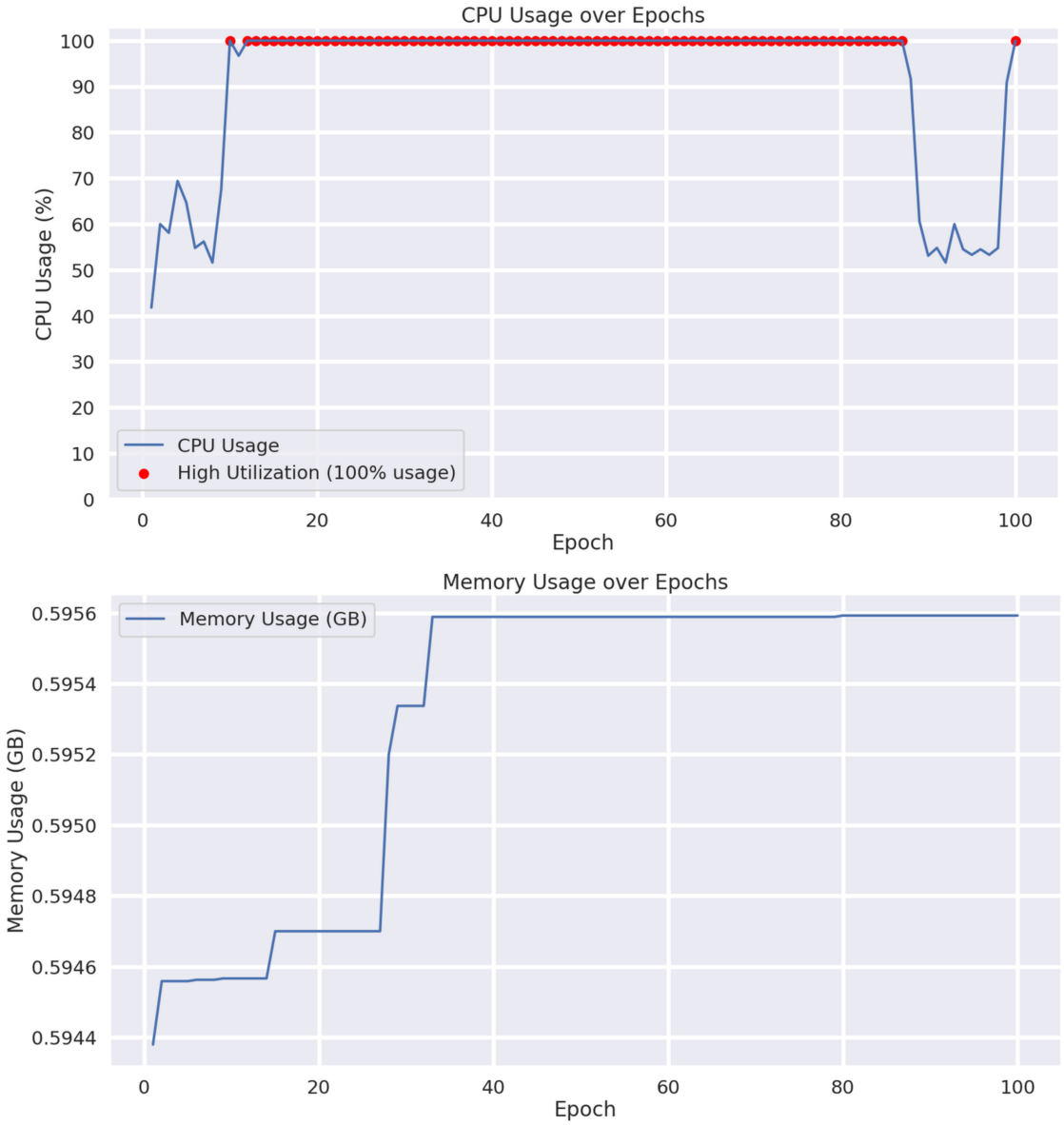
Figure 26: Training GraphSAGE with Amazon Computers - GPU Accelerated.



Source: The author(s).

In Figure 26, the execution of the same (SAGE-Amazon) scenario using both CPU (host) and GPU (device) is observed, where it is possible to observe 65 HUP are visible on the host and no HUP on the device. The x-axis represents epochs, while the y-axis represents the percentage of utilisation. The critical period for the host occurs between epochs 0 and 70, while the GPU's peak usage reaches 60% at some points. The consumption of both memories remained stable. In the right side of the figure, the consumption of both main and device memories (in GB) during epoch execution is shown, where an almost constant memory consumption can be noticed (2.55 GB for the main memory and 2.678 GB for GPU memory).

Figure 27: Training GraphSAGE with CiteSeer - CPU only.

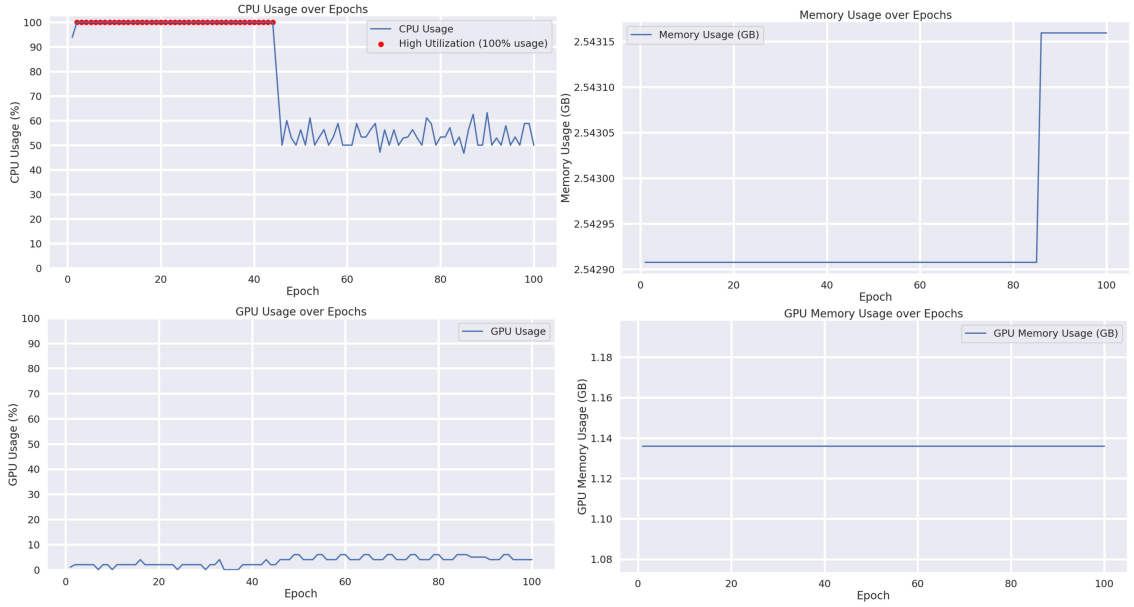


Source: The author(s).

In Figure 27, the training of the GraphSAGE model with the CiteSeer dataset is presented[6], where it is possible to observe the 78 points of high CPU utilisation. The critical execution periods are concentrated in the central part of the chart, excluding the first and last tenth of the period. The x-axis represents epochs, while the y-axis represents the percentage of utilisation. In the second part of the figure, the consumption of main memory (in GB) during epoch execution is shown, where a low change-over in memory consumption can be noticed (between 0.594 and 0.596 GB).

[6]CiteSeer represents the academic citation dataset, containing approximately 4230 nodes, 10600 edges, 602 features, 6 classes, and an average node degree of 2.52, as defined in Section 3.3.3.2. Using the GraphSAGE model defined in Section 3.3.2.2 for the node classification task.

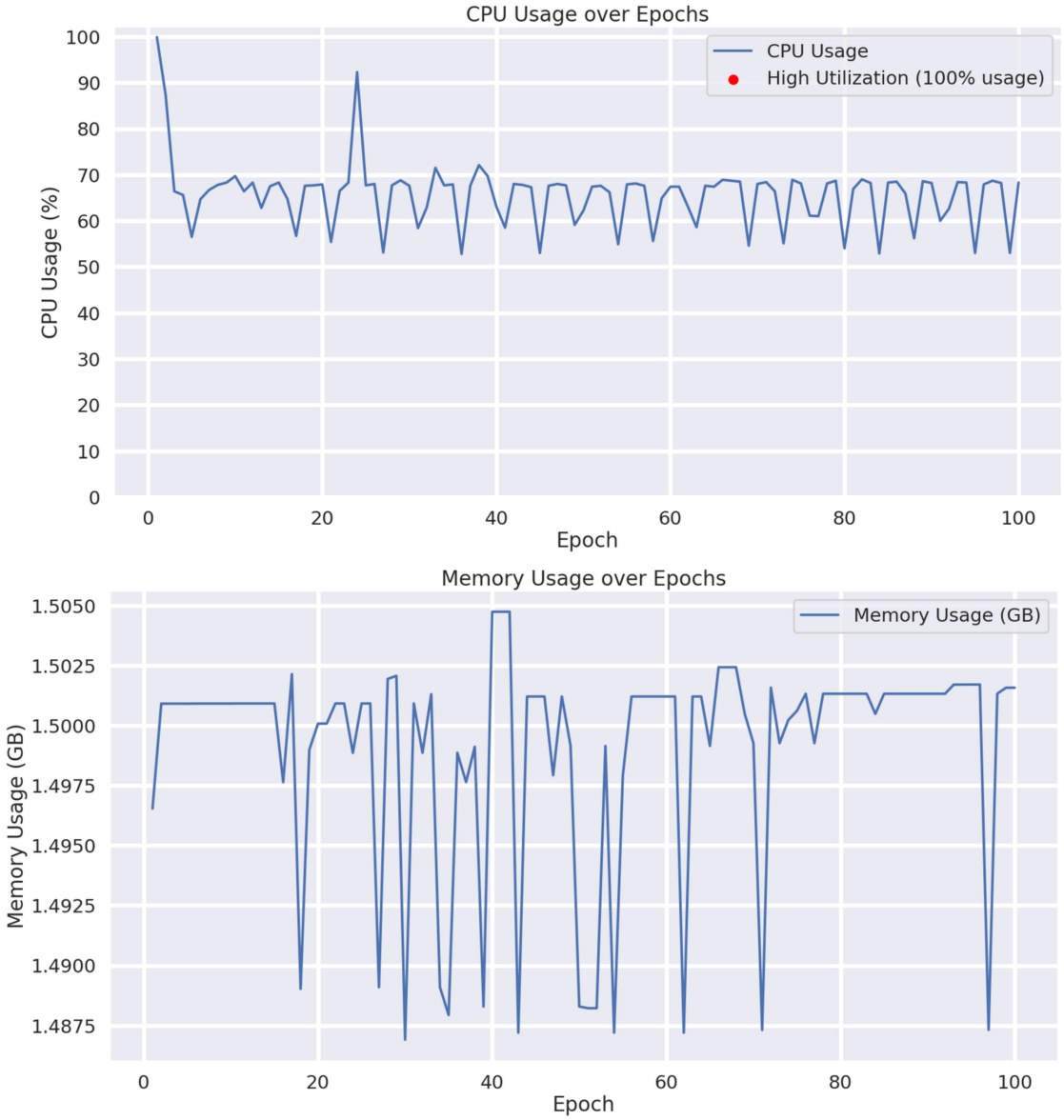
Figure 28: Training GraphSAGE with CiteSeer - GPU Accelerated.



Source: The author(s).

In Figure 28, the execution of the same (SAGE-CiteSeer) scenario using both CPU (host) and GPU (device) is observed, where it is possible to observe 43 HUP are visible on the host and no HUP on the device. The critical execution period for the host are concentrated in the first half of the chart, while the device did no reach 10% of usage. The x-axis represents epochs, while the y-axis represents the percentage of utilisation. The consumption of both memories remained stable. In the right side of the figure, the consumption of both main and device memories (in GB) during epoch execution is shown, where a low change-over in the memory consumption can be noticed and a constant consumption on the GPU's memory (2.54 GB peak for the main memory and 1.14 GB for GPU memory).

Figure 29: Training GraphSAGE with CoAuthors CS - CPU only.

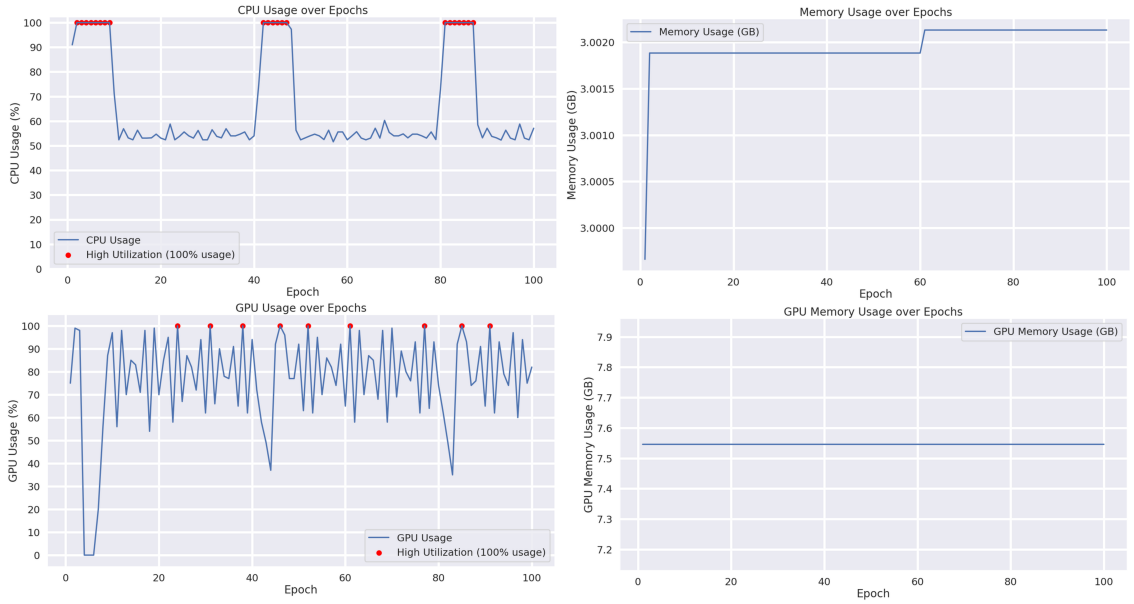


Source: The author(s).

In Figure 29, the training of the GraphSAGE model with the CoAuthors dataset is presented[7], where it is possible to observe no points of high CPU utilisation. However, it is important to emphasize that despite not reaching HUP, this test exhibited the longest training time. In the second part of the figure, the consumption of main memory (in GB) during epoch execution is shown, where a low change-over in memory consumption can be noticed (between 1.487 and 1.505 GB) considering absolute values.

[7]CoAuthors (CS) represents the literature co-authored authors dataset, containing approximately 18000 nodes, 160000 edges, 6805 features, 15 classes, and an average node degree of 8.93, as defined in Section 3.3.3.3. Using the GraphSAGE model defined in Section 3.3.2.1 for the node classification task.

Figure 30: Training GraphSAGE with CoAuthors CS - GPU Accelerated.



Source: The author(s).

In Figure 30, the execution of the same (SAGE-CoAuthors) scenario using both CPU (host) and GPU (device) is observed, where it is possible to observe **21 HUP on the host and 9 HUP on the device**, the only test case to present HUP on both CPU and GPU. It is possible to observe that while the host machine are at a HUP, the device consumption are at a trough. In the right side of the figure, the consumption of both main and device memories (in GB) during epoch execution is shown, where a constant memory consumption can be noticed (3 GB for the main memory and 7.5 GB for GPU memory) and reaching the highest absolute values for the whole benchmarking.

4.3 Discussion

It was noted that inherent differences in models and datasets greatly influenced the results. Training with the GraphSAGE model usually required more processing time (except for the GPU-accelerated CiteSeer execution). However, it was possible to observe that despite the longer processing time, training with the GraphSAGE model reduced high utilisation points in 4 out of 6 cases. However, concerning GPU-accelerated execution, the GraphSAGE model was the only one that exhibited high GPU utilisation points (in the CoAuthors CS dataset). This can be explained by the greater complexity of the GraphSAGE network, which is capable of storing and utilising information from neighbouring nodes to extract insights about the topology of a node. As presented in Section 3.3.2, the GraphSage network has nearly twice as many parameters as the GCN network, which is one of the inherent factors for the results presented, particularly regarding the increased computational demands of this model. Indeed, this fact is also reflected in the study's

higher values of main memory consumption.

In general terms, based on the result values (Table 5), it can be inferred that factors contributing to high utilisation values include (1) the number of edges in the graph, (2) the average degree of connectivity of nodes in the graph, and (3) the complexity of the model used. Regarding training time, the overall graph size can also be mentioned in addition to the abovementioned points. In all cases, GPU acceleration reduced the training processing time; however, it also increased the processing load on the host machine in many situations.

Dataset	Environment	Model	Training Time (sec)	CPU HUP	GPU HUP	Max Val. Acc.
Amazon Computers	CPU	GCN	60	33	-	0.773
	GPU Accelerated		7	0	0	0.847
	CPU	GraphSAGE	228	8	-	0.544
	GPU Accelerated		24.8	65	0	0.353
CiteSeer	CPU	GCN	23	15	-	0.954
	GPU Accelerated		22.8	64	0	0.959
	CPU	GraphSAGE	32	78	-	0.954
	GPU Accelerated		18	43	0	0.953
CoAuthors CS	CPU	GCN	55.7	14	-	0.947
	GPU Accelerated		25.3	16	0	0.942
	CPU	GraphSAGE	1052	0	-	0.942
	GPU Accelerated		38.1	21	9	0.947

Table 5: Aggregated of Extracted Results - Resource Consumption in Training.

5 Conclusion and Future Work

This work reported the journey and challenges towards building a tool for creating, processing, and visualising graphs. The ultimate goal of this study is to answer the following questions:

- (RQ1) What is the resource footprint in running Graph Neural Networks in a accelerated environment?
- (RQ2) What is the impact of choosing different well-known Models and Datasets from the literature, considering variable node degrees and graph structures?

Chapter 1, was introduced the topic by presenting state-of-the-art examples that aimed to build a user-end graph application, showing how a graph application can be represented in both OLTP and OLAP pipelines. The work presented at da Silva et al. [19], reported the step-by-step designing of a prototype that can extract, process, visualise, and analyse graphs from Databooks in a web interface named GraphLED. Despite the true character of the data used as motivation for the tool, the steps and techniques described here can be applied in numerous other applications regardless of the area (related to RQ1).

Still, Chapter 1 mentions the main challenges encountered in developing GraphLED, emphasising the scalability challenges. Were also reported the main limitations of the tool and the challenges faced in overcoming the limitations of GraphLED, such as the difficulty in scaling the tool, the lack of automatic graph detection, and others. In addition, the main techniques for scaling graph neural network training for graphs were also mentioned, using horizontal and vertical scalability techniques as examples.

In Chapter 2, the main concepts capable of handling the problems encountered were grouped and organised in a concise overview to clarify the next steps for the GraphLED tool. In addition, in this chapter, we provide a complete introduction to Graph Neural Networks, their types, and how they can be explored to perform numerous activities related to graph usage. Finally, we summarise the main advantages and disadvantages of using these networks using robust state-of-art examples.

Chapter 3 defined a method for analysing GNN from a system performance perspective (related to RQ1), contrary to traditional evaluations focusing primarily on accuracy

analysis. Initially, the main frameworks capable of providing different techniques to optimise GNN training were selected and evaluated. Next, these frameworks were classified according to their applicability, relevance in the community, and the existence of intuitive APIs and documentation, among other parameters. Finally, the use of the USE methodology was defined to evaluate the performance of these frameworks in different test scenarios, parameterising attributes such as execution environment, model used, and datasets. To obtain the results in their entirety, it is expected to extract values of Saturation and Errors for the defined scenarios. Initially, only the PyG (PyTorch Geometric) framework was used for evaluation. However, it was identified that different frameworks have different ways of approaching scalability in their implementation.

Finishing with the selection of a single framework (PyTorch Geometric) due to the arguments presented regarding its advantages over the others. Next, two convolutional graph neural network models, GCN and GraphSAGE, were chosen. These models were used for training the Amazon Computers, CiteSeer, and CoAuthors (CS) datasets (related to RQ2) in different execution environments (CPU only and GPU-accelerated). From this execution, hardware resource consumption values concerning each training epoch were extracted (limited to 100). A significant processing time improvement was observed when the GPU-accelerated environment was utilised (related to RQ1 and RQ2).

In da Silva et al. [19], were identified two possible points of evaluation for adding value to the study and reducing bias: (1) a deeper performance analysis (benchmarking) to identify all the bottlenecks in the tool, (2) a usability analysis with end users, to extract the evaluation of the target audience regarding the usefulness of the tool, (3) thorough analysis of Databooks using GraphLED to extract insights from the data to bring competitive or operational advantage to the users. For example, one question to be answered would be: "Out of a set of *n* Databooks processed by GraphLED, how many of them have anomalous structural patterns?". To address point (2), we aim to use one of the most common scales for analysing system usability: the Likert Scale [47] or the *ISO 9241-11* [56]. To address the issue (3), some of our colleagues and co-authors of GraphLED are conducting parallel work to extract knowledge from these data, from centrality calculations, anomaly detection techniques and GNN models. None of the planned improvements for the work in da Silva et al. [19] will be continued upon the completion of this thesis but will instead be continued in subsequent jobs related to enhancing GraphLED.

As show in the results (Chapter 4), it can be observed that changing the execution environment of a ConvGNN, using GPU acceleration, results in a significant gain in training time, as well as the elimination of processing saturation in a low-stress test environment (related to RQ1).

During the development of this thesis, it was possible to observe that there are still unlimited possibilities when it comes to benchmarking GNN. Due to their analytical nature, most studies in this field are focused on achieving better accuracy and F1 values in

models. However, other areas of science can contribute to the evaluation of these neural networks from different perspectives, such as resource consumption, parallelism techniques, scalability, and more. It becomes feasible to study other types of neural networks, such as RecGNN, STGNN, etc., as well as conduct more in-depth studies of existing techniques, such as error data extraction, saturation, and others. Although not deeply explored in this thesis, the study of different frameworks can also be promising because it is known that frameworks use different implementations for their GNN usage APIs, thus potentially directly impacting the performance of systems that rely on these implementations.

BIBLIOGRAPHY

[1] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)*, 54(9):1–38, 2021.

[2] N. Agarwal, H. Liu, L. Tang, and P. S. Yu. Identifying the influential bloggers in a community. In *Proceedings of the 2008 international conference on web search and data mining*, pages 207–218, 2008.

[3] I. Ahmad, M. U. Akhtar, S. Noor, and A. Shahnaz. Missing link prediction using common neighbor and centrality based parameterized algorithm. *Scientific reports*, 10(1):1–9, 2020.

[4] G. S. Almasi and A. Gottlieb. *Highly parallel computing*. Benjamin-Cummings Publishing Co., Inc., 1994.

[5] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, 1967.

[6] R. Amor and M. Clift. Documents as an enabling mechanism for concurrent engineering in the construction industry. In *1st international conference on Concurrent Engineering in Construction, CEC*, volume 97, 1997.

[7] R. R. Aragao. *Using Network Theory to Manage Knowledge From Unstructured Data in Construction Projects: Application to a Collaborative Analysis of the Energy Consumption in the Construction of Oil and Gas Facilities*. PhD thesis, University of Toronto (Canada), 2018.

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

[9] J. W. Berry, B. Hendrickson, S. Kahan, and P. Konecny. Graph software development and performance on the mta-2 and eldorado. In *48th Cray Users Group Meeting, Switzerland*, 2006.

[10] D. Blakely, J. Lanchantin, and Y. Qi. Time and space complexity of graph convolutional networks. *Accessed on: Dec*, 31, 2021.

[11] G. E. Blelloch. *Vector models for data-parallel computing*, volume 2. Citeseer, 1990.

[12] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[13] S. Cao, W. Lu, and Q. Xu. Deep neural networks for learning graph representations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[14] M. Capotă, T. Hegeman, A. Iosup, A. Prat-Pérez, O. Erling, and P. Boncz. Graphalytics: A big data benchmark for graph-processing platforms. In *Proceedings of the GRADES'15*, pages 1–6. ACM New York, NY, USA, 2015.

[15] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[16] J. Chen, J. Zhu, and L. Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.

[17] J. Chen, T. Ma, and C. Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[18] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019.

[19] V. T. da Silva, L. d. A. M. Ribeiro, W. B. de Lemos, S. S. d. C. Botelho, M. R. Pias, et al. Graphled: A graph-based approach to process and visualise linked engineering documents. *arXiv preprint arXiv:2302.08905*, 2023.

[20] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[21] E. A. Dudziak. Arquivos e documentos empresariais: da organização cotidiana à gestão eficiente. *Revista de Gestão e Secretariado*, 1(1):90–110, 2010.

[22] E. Dumbill. A revolution that will transform how we live, work, and think: An interview with the authors of big data. *Big Data*, 1(2), 2013.

[23] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[24] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[25] C. Gallicchio and A. Micheli. Graph echo state networks. In *The 2010 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2010.

[26] L. Getoor and C. P. Diehl. Link mining: a survey. *Acm Sigkdd Explorations Newsletter*, 7(2):3–12, 2005.

[27] C. L. Giles, K. D. Bollacker, and S. Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98, 1998.

[28] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 17–30, 2012.

[29] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[30] B. Gregg. Thinking methodically about performance. *Communications of the ACM*, 56(2):45–51, 2013.

[31] D. Gregor and A. Lumsdaine. The parallel bgl: A generic library for distributed graph computations. *Parallel Object-Oriented Scientific Computing (POOSC)*, 2: 1–18, 2005.

[32] Y. Guo, M. Biczak, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke. How well do graph-processing platforms perform? an empirical performance evaluation and analysis. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 395–404. IEEE, 2014.

[33] Y. Guo, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke. Benchmarking graph-processing platforms: A vision. In *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*, pages 289–292, 2014.

[34] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[35] M. Han and K. Daudjee. Giraph unchained: Barrierless asynchronous parallel execution in pregel-like graph processing systems. *Proceedings of the VLDB Endowment*, 8(9):950–961, 2015.

[36] E. Hovy, R. Navigli, and S. P. Ponzetto. Collaboratively built semi-structured content and artificial intelligence: The story so far. *Artificial Intelligence*, 194:2–27, 2013.

[37] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[38] K. Hwang and N. Jotwani. *Advanced computer architecture: parallelism, scalability, programmability*, volume 199. McGraw-Hill New York, 1993.

[39] J. JéJé. An introduction to parallel algorithms. *Reading, MA: Addison-Wesley*, 10: 133889, 1992.

[40] M. S. Kettouch. *A new approach for interlinking and integrating semi-structured and linked data*. PhD thesis, Anglia Ruskin University, 2017.

[41] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis. Mizan: a system for dynamic load balancing in large-scale graph processing. In *Proceedings of the 8th ACM European conference on computer systems*, pages 169–182, 2013.

[42] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[43] F. G. Lafetá, C. F. de Oliveira Barros, and P. d. O. C. D. Leal. Projetos complexos: estudo de caso sobre a complexidade dos projetos de engenharia de telecomunicações em uma empresa do setor de óleo e gás. *Gestão e Projetos: GeP*, 7(1):41–55, 2016.

[44] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on computers*, 100(1): 24–35, 1987.

[45] G. Li, M. Müller, B. Ghanem, and V. Koltun. Training graph neural networks with 1000 layers. In *International conference on machine learning*, pages 6437–6449. PMLR, 2021.

[46] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[47] R. Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[48] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146, 2010.

[49] C. H. Marcondes. Tecnologias da informação e impacto na formação do profissional da informação. *Transinformação*, 11(3), 2012.

[50] R. C. Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[51] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 43–52, 2015.

[52] J. L. McClelland, D. E. Rumelhart, P. R. Group, et al. *Parallel distributed processing*, volume 2. MIT press Cambridge, MA, 1986.

[53] Memgraph. Graph classification algorithm. `https://memgraph.com/docs/mage/algorithms/machine-learning-graph-analytics/graph-classification-algorithm`, 2023. Accessed: 2023-06-05.

[54] S. Newman. *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

[55] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, Á. López García, I. Heredia, P. Malík, and L. Hluchỳ. Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review*, 52: 77–124, 2019.

[56] I. S. . E. of Human-System Interaction (Subcommittee). *Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs).: Guidance on Usability*. International Organization for Standardization, 1998.

[57] P. Ongsulee. Artificial intelligence, machine learning and deep learning. In *2017 15th international conference on ICT and knowledge engineering (ICT&KE)*, pages 1–6. IEEE, 2017.

[58] C. Pizzuti. Ga-net: A genetic algorithm for community detection in social networks. In *Parallel Problem Solving from Nature–PPSN X: 10th International Conference,*

Dortmund, Germany, September 13-17, 2008. Proceedings 10, pages 1081–1090. Springer, 2008.

[59] A. Rajaraman and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[60] I. Robinson, J. Webber, and E. Eifrem. *Graph databases*. " O'Reilly Media, Inc.", 2013.

[61] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[62] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[63] E. Seo, P. Mohapatra, and T. Abdelzaher. Identifying rumors and their sources in social networks. In *Ground/air multisensor interoperability, integration, and networking for persistent ISR III*, volume 8389, pages 417–429. SPIE, 2012.

[64] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[65] X. Shi, Z. Zheng, Y. Zhou, H. Jin, L. He, B. Liu, and Q.-S. Hua. Graph processing on GPUs: a survey. *ACM Computing Surveys*, 50(6):81:1–81:??, Jan. 2018. ISSN 0360-0300 (print), 1557-7341 (electronic). doi: https://doi.org/10.1145/3128571.

[66] J. Siek, L.-Q. Lee, A. Lumsdaine, et al. *The boost graph library*. Pearson India, 2002.

[67] F.-Y. Sun, J. Hoffmann, V. Verma, and J. Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*, 2019.

[68] I. Szulc, K. Stencel, and P. Wiśniewski. Using genetic algorithms to optimize redundant data. In *International Conference: Beyond Databases, Architectures and Structures*, pages 165–176. Springer, 2017.

[69] J. Tan and X. Fu. Addressing hardware reliability challenges in general-purpose gpus. *Adv. GPU Res. Pract*, pages 649–705, 2017.

[70] V. Telles da Silva, L. d. A. Martins Ribeiro, W. Borges de Lemos, S. Silva da Costa Botelho, N. Lopes Duarte Filho, and M. R. Pias. Graphled: A graph-based approach to process and visualise linked engineering documents. *arXiv e-prints*, pages arXiv–2302, 2023.

[71] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, pages 1–16, 2013.

[72] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[73] J. Von Neumann. The principles of large-scale computing machines. *Annals of the History of Computing*, 3(3):263–273, 1981.

[74] J. Von Neumann, A. W. Burks, et al. Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1966.

[75] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234, 2016.

[76] M. Y. Wang. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*, 2019.

[77] O. Wieder, S. Kohlbacher, M. Kuenemann, A. Garon, P. Ducrot, T. Seidel, and T. Langer. A compact review of molecular property prediction with graph neural networks. *Drug Discovery Today: Technologies*, 37:1–12, 2020.

[78] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang. Graph wavenet for deep spatial-temporal graph modeling. *arXiv preprint arXiv:1906.00121*, 2019.

[79] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[80] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica. Graphx: A resilient distributed graph system on spark. In *First international workshop on graph data management experiences and systems*, pages 1–6, 2013.

[81] S. Yan, Y. Xiong, and D. Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[82] C. Yang, A. Buluç, and J. D. Owens. Design principles for sparse matrix multiplication on the gpu. In *Euro-Par 2018: Parallel Processing: 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings*, pages 672–687. Springer, 2018.

[83] A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, and U. Catalyurek. A scalable distributed parallel breadth-first search algorithm on bluegene/l. In *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pages 25–25. IEEE, 2005.

[84] B. Yu, H. Yin, and Z. Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

[85] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, et al. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[86] S. Zheng, Z. Zhu, X. Zhang, Z. Liu, J. Cheng, and Y. Zhao. Distribution-induced bidirectional generative adversarial network for graph representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7224–7233, 2020.

A GraphLED: A graph-based approach to process and visualise linked engineering documents

In the work presented by da Silva et al. [19] we introduced *GraphLED*, a system tasked with data processing, graph-based modeling, and colorful visualization of related documents. It was submitted to the *ArXiv* Repository in february 2023 (*DOI: https://doi.org/10.48550/arXiv.2302.08905*) [70].

The graph-based approach ensures improved understanding of linked information, as the graph structure offers a promising tool for modeling the underlying data properties of engineering documents. This work has the potential to benefit the industry by improving the reliability and resilience of industrial production systems through automated summaries of large quantities of documents and their linkages.

Roles: Lucas de A. M. Ribeiro: Conceptualization, Programming, Simulations, Writing - original draft, Writing - review and editing; Vanessa T. da Silva: Conceptualization, Programming, Analysis, Writing - original draft, Writing - review and editing; Conceptualization, Programming, Simulations, Writing - original draft, Writing - review and editing; Marcelo R. Pias: Conceptualization, Programming, Analysis, Writing - original draft, Writing - review and editing; Sílvia S. da C. Botelho: Writing - review and editing; Nelson L. D. Filho: Writing - review and editing.