SANTA CATARINA STATE UNIVERSITY – UDESC
COLLEGE OF TECHNOLOGICAL SCIENCE – CCT
GRADUATE PROGRAM IN APPLIED COMPUTING – PPGCAP

KLEITON PEREIRA

SCHEDULING HPC JOBS WITH GRAPH NEURAL NETWORKS

JOINVILLE
2024

KLEITON PEREIRA

SCHEDULING HPC JOBS WITH GRAPH NEURAL NETWORKS

Master thesis presented to the Graduate Program in Applied Computing of the College of Technological Science from the Santa Catarina State University, as a partial requisite for receiving the Master's degree in Applied Computing.

Supervisor: Guilherme Piêgas Koslovski

JOINVILLE

2024

Ficha catalográfica elaborada pelo programa de geração automática da Biblioteca Universitária Udesc,

com os dados fornecidos pelo(a) autor(a)

Pereira, Kleiton
Scheduling HPC jobs with Graph Neural Networks / Kleiton
Pereira. -- 2024.
83 p.

Orientador: Guilherme Piêgas Koslovski
Dissertação (mestrado) -- Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada, Joinville, 2024.

1. High Performance Computing. 2. Graph Neural Networks. 3. Scheduling. 4. Supervised Learning. I. Koslovski, Guilherme Piêgas. II. Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada. III. Título.

KLEITON PEREIRA

SCHEDULING HPC JOBS WITH GRAPH NEURAL NETWORKS

Master thesis presented to the Graduate Program in Applied Computing of the College of Technological Science from the Santa Catarina State University, as a partial requisite for receiving the Master's degree in Applied Computing.

Supervisor: Guilherme Piêgas Koslovski

EVALUATION COMMITTEE:

Guilherme Piêgas Koslovski, Ph.D.
Santa Catarina State University

Committee:

Ricardo Pfitscher, Ph.D.
Federal University of Santa Catarina

Maurício Aronne Pillon, Ph.D.
Santa Catarina State University

Joinville, February, 27, 2024

In loving memory of Gilberto Eugênio Pereira, whose life taught me the value of perseverance and kindness. This achievement is a testament to his enduring inspiration.

ACKNOWLEDGEMENTS

In completing this journey, my heart is filled with sincere gratitude for those who have walked alongside me, offering their support, guidance, and encouragement. Their contributions have been invaluable, and it is with a humble spirit that I extend my thanks to each of them.

I am deeply grateful to my advisor, whose guidance has been a beacon during this endeavor. His patience, motivation, and camaraderie have not only guided me through challenges but also inspired me to strive for excellence. His belief in my abilities has been a source of strength and for that I cannot thank him enough.

To my friends, whose support has been a lifeline in times of difficulty, I owe a debt of gratitude. Your presence, encouragement, and unwavering belief in me have made a profound difference in my journey. The generosity and kindness you have shown me are treasures I hold dear and I hope to mirror that same support and positivity in your lives.

My family deserves my deepest appreciation for their inspiration. Their love and belief in me have been an anchor, providing me with the courage to persevere through the challenges. I carry the bonds that we share with me in every step I take.

I also wish to thank the professors at Santa Catarina State University for their invaluable guidance and wisdom. The knowledge they have shared has not only enriched my academic experience, but also offered insights that extend beyond the classroom. Their dedication to teaching and nurturing curiosity has been a critical part of my growth.

This thesis, while a symbol of academic endeavor, is also a testament to the collective support of all mentioned. Your roles in this journey have been a source of strength and I am truly grateful for each of you. I sincerely thank you for being part of my journey.

“Learning is the only thing the mind never
exhausts, never fears, and never regrets.”
Leonardo Da Vinci

ABSTRACT

Neste estudo, a otimização da programação de tarefas em Computação de Alto Desempenho (HPC) é explorada utilizando Redes Neurais em Grafos (GNNs). O foco está em comparar diferentes variantes de GNNs. O objetivo é abordar os desafios da programação de tarefas com dependências entre elas em ambientes HPC, aproveitando o conhecimento criado por algoritmos de programação determinísticos para treinar modelos de GNN. Por meio de experimentação extensiva e avaliação, a pesquisa revela as diferenças de desempenho entre as diversas variantes de GNN, demonstrando seu potencial para superar as abordagens tradicionais de programação em termos de eficiência e utilização de recursos. Esta análise não apenas destaca os desempenhos comparativos entre cada variante de GNN, mas também aprimora nosso entendimento das aplicações de GNN na programação de sistemas complexos usando Aprendizado Supervisionado.

Palavras-chave: computação de alto desempenho, redes neurais em grafos, escalonamento, aprendizado supervisionado

ABSTRACT

In this study, the optimization of job scheduling in High-Performance Computing (HPC) is explored using Graph Neural Networks (GNNs). The focus is on comparing different variants of GNNs. The goal is to address the challenges of scheduling jobs with task dependencies in HPC environments by tapping into the knowledge created by deterministic scheduling algorithms to train GNN models. Through extensive experimentation and evaluation, the research reveals the performance differences between various GNN variants, demonstrating their potential to outperform traditional scheduling approaches in terms of efficiency and resource utilization. This analysis not only highlights the comparative performances between each GNN variant and enhances our understanding of GNN applications in complex system scheduling using Supervised Learning.

Keywords: high performance computing, graph neural networks, scheduling, supervised learning

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
DC	Data Center
DAG	Directed Acyclic Graph
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
HEFT	Heterogenous Earliest Finish Time
HPC	High Performance Computing
KAIROS	k-Graphs Artificial Intelligence-based Request Optimization Scheduler
ML	Machine Learning
PEFT	Predict Earliest Finish Time
WaaS	Workflow as a Service

CONTENTS

1 INTRODUCTION

Scheduling is a classical problem present in all systems whose resources are finite and receive requests over time, as it is precisely the process of decision-making regarding the distribution of resources to such requests, respecting a set of constraints imposed both by the architecture of the resources and by the demands. Being a problem that is notoriously NP-hard given its combinatorial nature (HOOGEVEEN; LENSTRA; VELTMAN, 1996), for small scales, deterministic optimization algorithms exist that find the optimal solution and can be applied, but as the scale increases, the deterministic search for the optimal solution becomes computationally infeasible.

In the context of this study, a job is a collection of tasks, each of which being a piece of computational effort that comprises the job completion. These tasks are often connected in a specific way that determines the order of execution of the tasks, these connections are called dependencies. Each task requires one or more resources to be executed, and it needs a minimum amount of time to complete its operations. These resources may vary from system to system, but for a context such as High Performance Computing (HPC) the resources are usually provided in nodes, which are a collection of resources (*e.g.*, CPU, RAM, etc.) available to execute tasks. A job is said to be completed when all its tasks have been successfully completed.

Since scheduling is a combinatorial problem that deals with resources, which are connected in some way and distributed to requests that have some kind of dependency, it is natural to represent these entities as graphs (KOCOT; CZARNUL; PROFICZ, 2023), non-Euclidean mathematical structures for data representation through a set of vertices that are connected to each other by a set of edges. These representations are used to map the resource nodes and also the tasks that compose the jobs, representing the relationships (*i.e.*, communication and dependency) between the elements. The graph representation has its many advantages, usually being able to reduce the problem, in small scales, to other solved sub-problems (BRUCKER, 1999). More specifically, a particular type of graph known as a Directed Acyclic Graph (DAG) can be used to represent a job (or workflow of tasks). A DAG is a graph that has no cycles in its connections and its edges are directed, meaning that all edges are one-sided and once a vertex is visited there is no way to arrive back to it by following the graphs edges. This representation provides a powerful tool for better understanding workflows and their possible compositions.

Even with useful representations, the task of scheduling workflows is still a complex activity, specially in HPC, where the scale of jobs and infrastructure is exacerbated. Not only the scale is a factor here, but the heterogeneity of computing resources, the dynamic nature of workflows in this context and the competing demands of multiple tenants add even more complexity to the problem (RODRIGUEZ; BUYYA, 2018). In light of this challenge, the acronym Workflow as a Service (WaaS) was coined (RODRIGUEZ; BUYYA, 2018), which are services and platforms dedicated to provide an effective way for HPC users and providers to design, execute and manage workflows. These services are highly impacted by their ability to effectively schedule the dynamic jobs that arrive as demand, which is an incentive for the development of better and more adaptive scheduling solutions.

Optimized scheduling has undeniable impacts on any kind of system with finite resources, affecting metrics such as fragmentation, availability, and response time, among others (FEITELSON; RUDOLPH, 1998a). However, as the scale of the system grows, the complexity of finding the optimal scheduling grows disproportionately, which means that for large HPC systems, the search for a deterministic optimal algorithm is impossible, so researchers have to rely on heuristics and stochastic algorithms that use a policy that approximates an optimal scheduling. However, what constitutes an optimized scheduling may vary from system to system, specially in HPC where a system's purpose heavily depends on the specific workload and requirements of the applications being executed. For example, in a scientific processing-oriented HPC system, the optimized scheduling may prioritize minimizing the total execution time of scientific simulations, maximizing the utilization of computing resources, and ensuring high throughput for a large number of parallel jobs (CASANOVA et al., 2020). On the other hand, in a big data-oriented HPC system, the optimized scheduling may focus on efficiently handling large-scale data processing tasks, prioritizing data locality and minimizing data movement across the distributed system, and ensuring efficient resource allocation for data-intensive workloads (HASSAN et al., 2014).

In another example, in an HPC system used for machine learning or AI workloads, optimized scheduling may prioritize accelerating training and inference tasks, optimizing GPU allocation, and managing the interplay between CPU and GPU resources (AMARAL et al., 2017). Furthermore, the security requirements of the HPC system, such as data privacy or confidentiality concerns, may also influence scheduling policies, such as ensuring that sensitive data jobs are scheduled on secure nodes or taking into account data locality to minimize data exposure risks. Even highly specific HPC scenarios, such as quantum computing, scheduling plays a role in the system's operation and is a critical aspect of research (NGUYEN; USMAN; BUYYA, 2023). This means that not only is the field of research scheduling relevant today, but its relevance may increase as HPC research expands the field.

There are many variations for heuristics, meta-heuristics and stochastic scheduling algorithms, each having their strengths and limitations. Some limitations that may be found in one or more of these algorithms include the lack of optimality guarantees, a high computational time for an acceptable answer, the need for a significant amount of historical data either for training or decision-making, and a lack of consistency in results, among others. In this context, one type of scheduling algorithms used in large scales are machine learning-based algorithms, which use the adaptative properties of machine learning solutions to train a model on large datasets seeking to produce a scheduler with a policy capable of efficiently scheduling new jobs. Specifically, neural networks have gained notoriety recently, specially deep neural networks used in the field of Deep Learning (LECUN; BENGIO; HINTON, 2015).

In this context, the Graph Neural Networks (GNNs) (SCARSELLI et al., 2008; MICHELI, 2009) represent a type of neural network, developed independently in the fields of Chemistry and

Computer Science to solve different problems, but operating on graph-structured data. The GNNs work by propagating information through the edges and nodes of the graph using a set of trainable functions (ZHOU et al., 2020). This propagation process usually involves iteratively aggregating information from neighboring nodes and edges and updating the hidden representations of each node, learning and modifying the flow of information to fit a set of training data. Although there are variants of this technique, all of them work on this same principle. Given the gain that the graph representation of the scheduling problem elements brings and the prevalence of the application of probabilistic techniques in it, it is natural that a portion of the literature uses GNNs and their variants for scheduling.

1.1 OBJECTIVES

The general objective of this study is to investigate the efficiency of Graph Neural Networks to improve the overall performance of HPC schedulers. In addition to the general objective, a list of specific objectives that may complement or even compose the general objective is as follows:

1. Compare the efficiency of different GNN variants applied to the job scheduling problem in HPC.
2. Investigate the behavior of GNN models in response to changes in the set of target metrics to be optimized.
3. Develop a prototype of a GNN-based scheduler.
4. Determine a protocol for the evaluation of the prototype developed.
5. Evaluate the performance of the GNN-based scheduler on a set of metrics and improve the scheduler's performance based on these parameters.

1.2 METHODOLOGY

The purpose of this study is to investigate the feasibility of using GNNs under a specific framework to improve the scheduling of HPC jobs. To achieve this goal, a quantitative and experimental research methodology was used. The research used an experimental approach to logical reasoning. Specifically, the GNNs was trained under the KAIROS framework (PEREIRA; KOSLOVSKI, 2020) when applied to the job scheduling problem in the context of HPC. To begin with, a systematic mapping was conducted to understand the role of GNNs in scheduling HPC jobs comparing many factors, such as performance, learning paradigm, etc. The systematic mapping will include a categorical classification of existing research on GNNs and their applications in the scheduling of HPC jobs and other contexts. This analysis helped to identify research gaps, trends, and common fields of applicability.

The study also involved the development of a GNN-based scheduling algorithm that was evaluated through simulations. Simulation setups were designed based on real-world scenarios and involved HPC jobs with varying degrees of complexity. The experiments used data collected from or based on real-world HPC job scheduling scenarios. Data were collected from experiments and real-world traces and analyzed using statistical techniques to evaluate the performance of the GNN-based scheduling algorithm under a set of metrics. The results were compared with those obtained from existing scheduling algorithms to determine the effectiveness of the GNN-based algorithm, describing the benefits and drawbacks.

In conclusion, this study aims to contribute to the existing body of knowledge on scheduling HPC jobs by exploring the potential of GNNs in this field and the applicability of the framework used to train the GNN-based model. The quantitative and experimental research methodology provided a rigorous design for analyzing the performance of the GNN-based scheduling algorithm. The findings of this research may have practical implications for HPC job scheduling in real-world applications.

1.3 ORGANIZATION

The remainder of this study was broken down into five chapters. In Chapter 2 the definitions and details of scheduling jobs in the HPC context were explained. Chapter 3 discussed the usage of machine learning solutions in HPC job scheduling and related work was presented. In Chapter 4, the proposed solution was presented together with the experiments for performance analysis. In Chapter 5 the results for the proposed solutions were presented and discussed. Finally, in Chapter 6, the conclusions were presented along with opportunities for future work.

2 SCHEDULING HPC JOBS

HPC systems provide an extensive scale of computational resources to its users. However, efficiently managing and scheduling HPC workloads can be a challenging task. In this chapter, we explore the fundamental concepts and attributes of HPC jobs and tasks, including resource requirements, runtime limits, and dependencies. Additionally, we will discuss how graphs can be used to represent these tasks and their dependencies, and also how they can be optimized for performance and utilization of HPC resources.

2.1 HPC JOBS

HPC jobs involve the execution of computationally intensive tasks on large-scale computing systems. These systems provide significant computational power and resources, enabling users to solve large and complex scientific and engineering problems. In this section, we will discuss how jobs and tasks may be defined and represented in the context of scheduling in HPC.

2.1.1 Tasks and Jobs

In the field of HPC, tasks and jobs are fundamental concepts that play a crucial role in the efficient execution of computational workloads on HPC systems. Tasks are units of computational work that can be independently executed and scheduled on compute resources. They represent individual computational steps or processes that are part of a larger computational workflow, such as simulations, data analyses, or model calculations. In turn, jobs in the HPC environment refer to the collection of tasks that users submit to a HPC system for execution. Jobs are typically composed of one or more tasks that are grouped together, typically in a queue, and submitted to the scheduler along with any additional attributes about the jobs.

There are various attributes associated with HPC jobs and tasks, which provide additional information to the scheduler on how tasks should be executed. These attributes include, but are not limited to, resource requirements, runtime limits, priority levels, dependencies, and job accounting. Resource requirements specify the number of CPU cores, memory, storage, and/or any other computational resource needed for each task in the job. Runtime limits specify the maximum amount of time that a job is allowed to run, managing the overall workload on the HPC system. Priority levels are used to prioritize jobs based on their importance, urgency, or user-defined preferences. Dependencies can be specified to ensure that tasks are executed in the correct order and with the necessary data available. Job accounting data, such as CPU time, memory usage, and I/O activity, can be tracked for reporting, billing, and performance analysis purposes.

In the context of this research the following attributes will be further discussed and expanded upon: resource requirements, runtime limits, and dependencies. This is mainly because these are core attributes and/or are more relevantly represented through the structures of a graph.

Table 1 presents the definitions for these attributes. Understanding these concepts and attributes is essential for effective management and scheduling of HPC workloads to optimize performance and utilization of HPC resources.

Attribute	**Definition**
Resource Requirements	The specific computational resources that a task requires to execute successfully. This may include CPU cores, memory allocation, disk space, etc.
Runtime Limits	The maximum amount of time that a task is allowed to run before it is terminated. This is crucial for preventing tasks from monopolizing resources and delaying other jobs in the queue.
Dependencies	The relationships between tasks that dictate their execution order. A task is said to have dependencies when one or more tasks need to be completed before its execution can start.

Table 1 – Task attribute definitions.

2.1.2 Graph Representation

Graphs are mathematical structures composed of a set of vertices (also known as nodes) and a set of edges that connect these vertices. The vertices represent entities or objects in a system, while the edges represent the relationships or connections between them. Graphs can be used to represent a variety of systems, such as social networks, transportation networks, and computer networks. Figure 1 represents the basic structure of a graph. Each node or vertex is represented as a point having a label or value (A, B and C in the context of Figure 1) that are connected to each other by the edges, represented as arrows that demonstrate which nodes are connected and their direction.

An edge's direction may be omitted, which by definition implies that the edge is bidirectional, or in other words, that the nodes connected by this edge have a mutual connection to each other. Figure 2 exemplifies the differences between a directed graph (*i.e.*, a graph where edges have specific directions) and an undirected graph (*i.e.*, a graph where edges are bidirectional). Specifically directed graphs may or may not have one or more cycles in them, where a cycle is defined as a path where you traverse the graph starting from a node and arriving on that same node only by using the edges of the graph. It is then said that graphs with no cycles in them are Acyclic and graphs with one or more cycles are Cyclic. In Figure 3, its possible to see an example.

DAGs are a type of graph that has a specific set of properties. As the name suggests, a DAG is a graph with directed edges without any cycles in it. These properties are important because they allow for a natural representation of cause-and-effect relationships, as well as a way

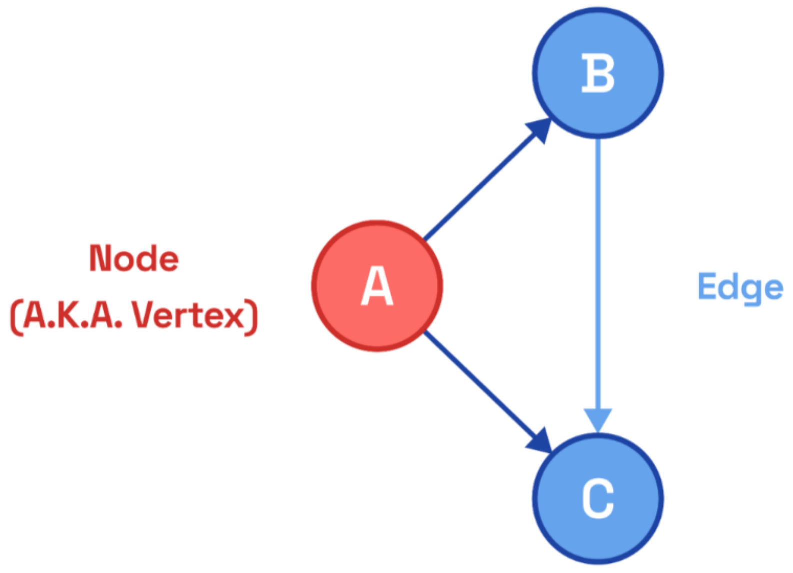


Figure 1 – General structure of a graph. Source: author.

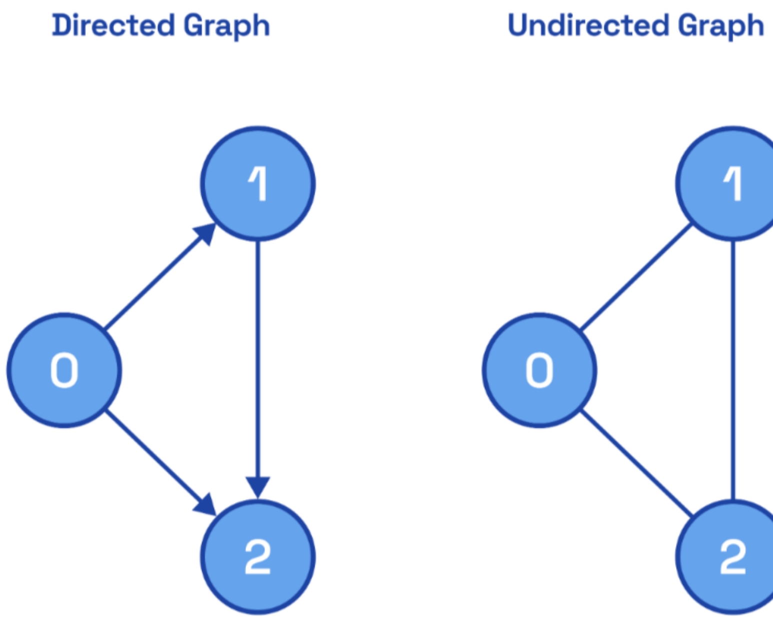


Figure 2 – A comparison between a directed graph (left side) and an undirected one (right side). Source: author.

to represent processes that have a clear order of steps, such as a set of tasks to be executed in a HPC job. In the context of HPC, the inner dependencies of the jobs can be represented as a DAG, where each task corresponds to a vertex, and the dependencies between them are represented as directed edges. For example, if task A must be completed before task B can start, there will be a

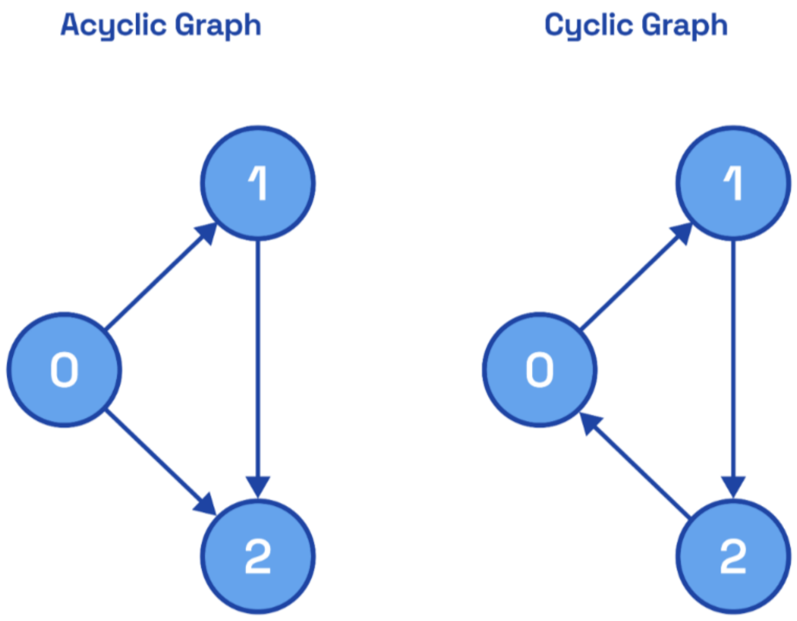


Figure 3 – A comparison between an acyclic graph (left side) and a cyclic one (right side). Source: author.

directed edge from task A to task B in the DAG representation.

Representing HPC job dependencies as DAGs has several advantages. First, it provides a clear and intuitive representation of the dependencies between tasks, which can help in the design and analysis of scheduling algorithms. Furthermore, DAGs can be used to identify potential bottlenecks or inefficiencies in a scheduling scheme, as well as to optimize the allocation of resources to jobs. Finally, the use of DAGs can facilitate the development of automated scheduling algorithms, as the graph structure can be used to guide the scheduling process, since it can represent important information about the inner connections of a job, which can lead to a better allocation that is sensitive to this type of information.

2.2 SCHEDULING DAGS

DAGs are an useful way to portray some aspects of the scheduling problem, providing the representational power and theoretical background needed to perform a practical analysis of these elements. In this section, we will discuss scheduling in HPC using DAGs as a representational tool along with metric definitions for the performance assessment of the scheduling process.

2.2.1 Concepts and Definitions

In HPC, jobs typically represent computationally intensive tasks that require significant amounts of processing power and memory. Examples of HPC applications include scientific simulations, data analytics, and machine learning. The goal of job scheduling is to allocate compute resources to jobs in an efficient and effective manner, in order to minimize the overall execution time and maximize the utilization of available resources. However, this is a complex problem due to the large number of jobs and the limited availability of compute resources. In addition, jobs may have inner dependencies, which means that their tasks need to be executed in a specific order.

The job scheduling problem in HPC can be divided into several sub-problems, such as resource allocation, task mapping, and job prioritization (SHAH; MAHMOOD; OXLEY, 2010). Resource allocation involves determining which compute resources (such as processors or memory) should be allocated to each job. Task mapping refers to the assignment of individual tasks within a job to specific compute resources, while job prioritization involves determining the order in which tasks and jobs should be executed, taking into account their dependencies and other factors such as their expected execution time and impact on the availability of the system. Addressing these sub-problems requires the development of scheduling algorithms that can effectively balance the competing demands of resource utilization and job execution time. In parallel, HPC datacenters are typically represented as a collection of compute nodes or servers interconnected by a high-speed network (BRUCKER, 1999). These compute nodes are equipped with processors, memory and other resources that can be allocated to execute jobs.

In HPC environments, jobs are often modeled as DAGs to represent task dependencies and other relationships. Each task in a job is a node, and dependencies are depicted as directed edges. DAG modeling enables a fine-grained representation of job requirements, however, the presence of task dependencies complicates scheduling. Tasks cannot run until their dependencies are complete, requiring a correct execution order. As the number of tasks and dependencies increases, the scheduling problem becomes computationally demanding (HOOGEVEEN; LENSTRA; VELTMAN, 1996). In addition, resource constraints further complicate DAG-based job scheduling, requiring sophisticated algorithms to optimize resource utilization while satisfying dependencies.

Along with these inevitable challenges that arise from the introduction of complex relationships between the tasks of a job comes the representational power of graphs when represented as DAGs. This means that a mature field of mathematics comes along with this representation, bringing concepts, techniques, algorithms and more (WEST et al., 2001) that have been defined, studied and improved over time. Some of these concepts that may be useful for scheduling DAGs include, but are not limited to, topological sorting, critical path, node coloring, node degree, etc.

In this context, scheduling DAGs involves determining the order in which tasks should be executed and which of the available compute resources should be allocated to process them. This requires careful consideration of task dependencies, resource availability, and other factors. Various scheduling algorithms have been developed to address this problem, such as list scheduling, priority-based scheduling, and backfilling (CASANOVA et al., 2023). These algorithms aim to minimize or maximize certain indicators of a datacenter's performance, often called metrics.

2.2.2 Metrics and objectives

In the context of HPC scheduling, the choice of appropriate metrics and objectives is crucial to evaluate and compare the performance of different schedulers (FEITELSON; RUDOLPH, 1998b). Metrics provide quantitative measures of the performance of a scheduling algorithm, while objectives represent the goals or targets that a scheduler aims to achieve.

A commonly used metric in HPC scheduling is slowdown, which measures the ratio between the actual execution time of a job and its ideal execution time, which is the time it would take to complete the job alone without contention for resources (FEITELSON; RUDOLPH, 1998b). The slowdown metric is a normalized value that reflects the efficiency of resource allocation, as higher slowdown values indicate longer waiting times and resource contention, leading to decreased performance. Additionally, the slowdown metric may be bounded by setting a minimum value for the ideal execution time, preventing smaller and faster jobs from having massive slowdown values from small delays that, compared to the size of the job, appear to be more impactful than they actually are (FEITELSON; RUDOLPH, 1998a).

Another commonly used metric is the node's usage percentage, which represents the proportion of resources that a server node is actively using to perform computational tasks compared to the total resources it is available for use. This metric reflects the overall utilization of resources and the ability of the scheduler to efficiently use the system resources.

There are several other common metrics used in HPC scheduling to guide and evaluate the performance of scheduling algorithms. These include makespan, which measures the total time taken to complete all jobs in the system; resource utilization, which quantifies the extent to which available resources are utilized during the scheduling process; fairness, which assesses the equitable distribution of resources among jobs; energy efficiency, which evaluates the energy consumption of the system during scheduling; and deadline adherence, which measures the ability of a scheduling algorithm to meet specified deadlines for completing jobs.

The set of metrics that are most relevant depends on the type of system, with a classification being the distinctions between online and offline systems combined with the difference between open and closed online systems (FEITELSON; RUDOLPH, 1998a). In this classification, an offline system assumes that all the jobs have already arrived, leaving no space for additional jobs to be included after the scheduler's policy execution. In online systems there are jobs arriving over time, with closed systems having a fixed upper bound on the number of jobs that are capable of being in the system at any single point in time, which does not exist in open systems where the jobs arrive in streams of indeterminate size.

As seen in Figure 4, the common metric for open online systems is slowdown; however, for closed online systems it is the throughput metric, which measures the number of tasks completed per unit of time, indicating the system's processing capacity. Finally, for offline systems, it is the makespan metric, representing the total time taken to complete all jobs. As the systems addressed in this work vary between offline, online open or online closed, the metrics selected are justified by being representative metrics used for scheduling policy performance analysis.

In this sense, we selected metrics that are related to slowdown, throughput, and makespan as main metrics to represent both the user's and infrastructure's perspectives.

These metrics will be further discussed in Chapter 5.

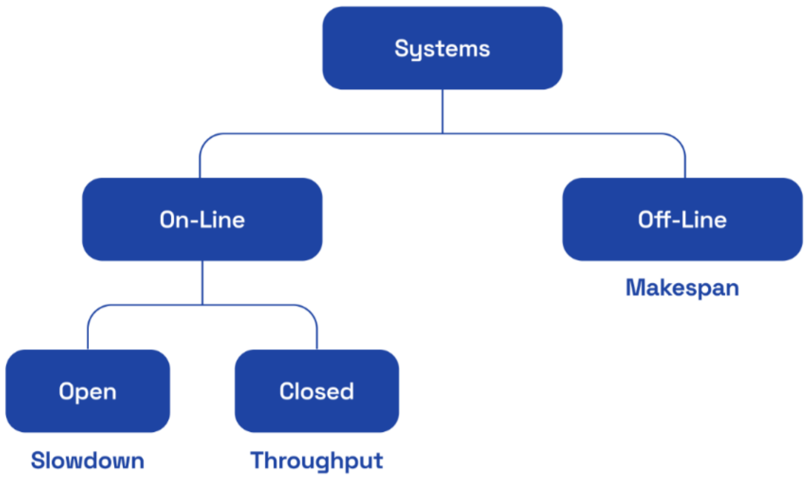


Figure 4 – Common metrics used in each classification. Source: adapted from (FEITELSON; RUDOLPH, 1998a).

2.3 CONSIDERATIONS

In this chapter, we introduced the fundamental concepts of HPC jobs and tasks, and their associated attributes, which are essential for effective management and scheduling of HPC workloads. We also discussed the graph representation of dependencies between tasks, and how DAGs are used to represent dependencies between tasks in HPC workflows.

When managing HPC workloads, it is important to consider the resource requirements and runtime limits of each job, as well as any dependencies between tasks, in order to optimize the performance and utilization of HPC resources. In the following chapters, we will discuss how machine learning techniques can be applied to the scheduling problem in HPC environments.

3 MACHINE LEARNING APPLIED IN HPC SCHEDULING

Machine Learning (ML) is a subset of Artificial Intelligence techniques that learn from data by recognizing patterns in it and extrapolating information from the probabilistic distributions identified in the data. ML algorithms can be categorized into four main learning paradigms: supervised learning, semi-supervised learning, unsupervised learning, and reinforcement learning.

In HPC scheduling, ML can be applied in different forms and subprocesses. In addition, these models may be used at an upper level of abstraction. The application of ML in HPC scheduling is only constrained by the data, and as long as there is the possibility of acquiring structured data with significant probabilistic correlation and in sufficient quantity for a given ML-based technique, a model can be constructed as a solution to extrapolate the data and predict or classify new examples to solve one or more parts of the HPC scheduling problem.

3.1 CONCEPTS AND DEFINITIONS

Techniques based on ML heavily depend on the structure, volume and quality of the data that is fed to the algorithm (MAHESH, 2020). This means that ML-based algorithms try to recognize patterns in the data, extrapolating information from the probabilistic distribution of examples given to them without being explicitly informed of the data's correlation to each other. The way in which these algorithms arrive at the final solution may vary from technique to technique, however, they are all founded on the same idea of learning from data.

Given that the application of ML is only constrained by the data, there are multiple steps in the scheduling process in which ML-based solutions can be used (ZARANDI et al., 2020). As long as there is the possibility of acquiring structured data with significant probabilistic correlation and in sufficient quantity for a given ML-based technique, a model can be constructed as a solution to extrapolate the data and predict or classify new examples to solve one or more parts of the HPC scheduling problem.

More specifically, these models can be used as end-to-end schedulers or as additional information extractors for other systems or models (ZARANDI et al., 2020). Also, these models may be used in an upper level of abstraction, such as classifiers, to choose another scheduler solution that the model has been trained to identify as the best fit for the job load and/or environment state. As the history of the research field indicates, it is possible that other applications for ML-based models will be presented as the exploration and discovery of new techniques develop, along with the application and combination of new and existing techniques at different levels of the scheduling problem.

The techniques in ML can be categorized in different ways, a common classification is through the learning paradigm (MAHESH, 2020), which is the method in which data is used to train the algorithms. In Figure 5 there is a visual representation of the division of techniques by the learning paradigm, along with examples of sub-techniques that are derived from each paradigm. The four main learning paradigms in ML are supervised learning, semi-supervised learning, unsupervised learning, and reinforcement learning (MAHESH, 2020).

Supervised learning is a type of learning in which an algorithm learns from labeled data. The algorithm is trained on a dataset that includes input data and the corresponding output or target values. The goal of supervised learning is to create a model that can accurately predict the output for new unseen data. Examples of supervised learning algorithms include: Decision Trees, Linear Regression, Support Vector Machines, among others.

Semi-supervised learning is a paradigm in which an algorithm learns from both labeled and unlabeled data. The algorithm is trained on a dataset that includes both labeled data and a larger set of unlabeled data. The goal of semi-supervised learning is to improve the accuracy of the model by leveraging the additional unlabeled data. Some examples of semi-supervised learning include: Label Propagation, Self-Training, etc.

In turn, unsupervised learning algorithms learn from unlabeled data. The algorithm is trained on a dataset that includes only input data, without any corresponding output or target values. The goal of unsupervised learning is to identify patterns and relationships in the data. Examples of unsupervised learning algorithms include: K-Means Clustering, Principal Component Analysis, etc.

Finally, reinforcement learning is a type of learning in which an algorithm learns through trial and error. The algorithm is trained in an environment in which it can take actions and receive rewards or penalties based on its actions. The goal of reinforcement learning is to learn a policy that maximizes cumulative rewards over time. Examples of reinforcement learning algorithms are as follows: Q-Learning, Markov Decision Process, among others.

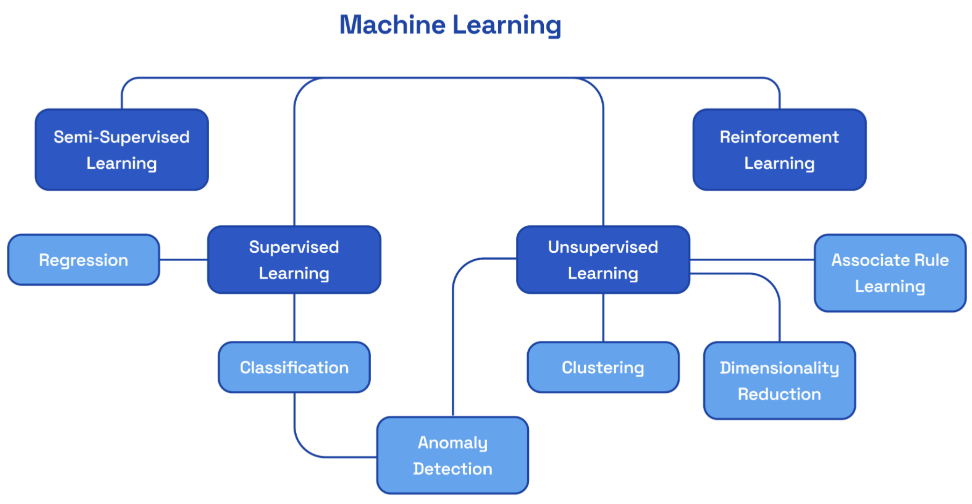


Figure 5 – Classification of Machine Learning Techniques. Source: the author.

Some methods are more prolific than others; for example, artificial neural networks are a ML technique that has seen a considerable increase in research and development recently, especially with the application of Deep Learning to build models through supervised and reinforcement learning (LECUN; BENGIO; HINTON, 2015). These models use large-sized neural networks and are trained with large datasets using HPC systems with dedicated hardware to find complex patterns in the datasets, which has been successful in a growing number of fields such as image recognition, natural language processing, and even HPC scheduling (LECUN; BENGIO; HINTON, 2015; FAN et al., 2021).

3.2 NEURAL NETWORKS AND GRAPHS

As Neural Networks developed and became a prominent technique in Machine Learning along with the aforementioned usefulness of graphs for representing and analyzing systems with complex relationships, their intersection would be inevitable. However, despite the progress made in both of these fields of research, there were limitations of classical Neural Networks when working with graph-structured data, since these networks operate with fixed-sized tensor-shaped inputs, which do not provide a natural way of handling graphs of varying sizes and structures. This initially required the transformation of the graph-represented data to another format, one accepted by the constraints of the network's input that carried with it problems with scalability and the partial destruction of the graph's dynamic representational power.

Integrating graph structures into classical neural networks requires workarounds due to their inherent differences. Several techniques have been proposed, such as utilizing adjacency matrices to capture pairwise relationships, employing graph coloring algorithms to represent a graph as a vector of colorization, or considering node degrees as features. While these techniques enable the incorporation of graph information into traditional Neural Networks, they often result in a loss of the inherent representational power of pure graph structures. The simplified representations obtained through these workarounds fail to capture the rich topological and structural characteristics of graphs, limiting the ability to effectively model intricate dependencies and interactions present in real-world scenarios. Consequently, while these techniques offer valuable insights into integrating graphs with neural networks, a more graph-centered model of neural networks was needed to not leave the domain of graph representation, consequently not losing its representational strengths.

Even with the advent of Deep Learning, which proved to be better suited for some class of problems that represent hierarchical data, the dense networks still struggled with the same problems of disparity between the input's constraints and the dynamism of graph-structured data. With this in mind, researchers working with graph data and Neural Networks worked in the development of a type of Neural Network that could better handle graph-structured data in a way that preserves the information of the graph.

3.3 GRAPH NEURAL NETWORKS

Graph Neural Networks are a type of neural network designed to operate on graph-structured data (SCARSELLI et al., 2008; MICHELI, 2009). In a GNN, each node in the graph is represented by a feature vector and the edges between nodes indicate the relationships between them. The goal of GNNs is to learn a representation of the entire graph that can be used for various tasks, such as node classification, graph classification, and link prediction. A GNN typically consists of several layers, each of which updates the feature vectors of the nodes based on their current values and those of their neighbors in the graph. The basic operation of a GNN layer can be described as follows:

1. Compute the message vector for each edge of the graph, which is a function of the feature vectors of the nodes at the endpoints of the edge.
2. Aggregate the message vectors for each node, typically by taking their sum or average.
3. Apply a transformation to the aggregated message vector for each node, typically a neural network layer.

The basic building block of a GNN layer is the message passing operation. In this operation, each node in the graph aggregates the features of its neighboring nodes and computes a new feature vector based on this aggregation. In this way, the goal of the network is to better manipulate and aggregate these messages, preserving the graph structure and learning from the connections and values in the graph, which is the core of a graph's representational power. The message passing operation can be formalized as follows:

$$m_{i\to j} = M(h_i, h_j)$$

where h_i and h_j are the feature vectors of nodes i and j respectively, and M is a function that computes the message sent from node i to node j. The message $m_{i\to j}$ represents the information that node i sends to node j. After the messages have been passed between the nodes, a node updates its feature vector by aggregating the messages it has received, generating h_i'. This operation can be formalized as follows:

$$h_i' = U(h_i, \sum_{j\in\mathcal{N}(i)} m_{j\to i})$$

where $\mathcal{N}(i)$ is the set of neighboring nodes of node i, and U is a function that updates the feature vector of node i based on the messages it has received. There are many different choices for the message passing and node update functions. Using the Graph Convolutional Network (GCN) layer (KIPF; WELLING, 2016) as an example, it uses the following message passing and node update functions:

$$m_{i\to j} = \frac{1}{|\mathcal{N}(i)|} W^l h_i$$

$$\tilde{h}_i = \sum_{j\in\mathcal{N}(i)} m_{j\to i} + W^l h_i$$

$$h_i' = \sigma(\tilde{h}_i + b^l)$$

where W^l is a weight matrix for the l-th layer of the network, b^l is a bias vector, and σ is a non-linear activation function.

The output of a GNN layer is a new set of feature vectors for the nodes, which can be used as input to the next layer of the network. The process is repeated for several layers until a final representation of the graph is obtained.

Several variations of the basic GNN architecture have been proposed in the literature, including the Graph Convolutional Network (GCN), Graph Attention Network (GAT), and GraphSAGE (ZHANG et al., 2019; VELIČKOVIĆ et al., 2017; HAMILTON; YING; LESKOVEC, 2017). These models differ in their specific implementation of the message passing and aggregation steps, as well as in the choice of activation functions and regularization techniques. Different GNN variants have varying performance across problem domains, depending on their architectural characteristics. Choosing the most appropriate variation of GNN for a particular problem set enhances the effectiveness of graph-based learning and analysis.

For example, the Graph Attention Network (GAT) is a variant of the GNN that is designed to capture the importance of different nodes in a graph by assigning attention weights to them. These attention weights are learned during the training process and used to compute a weighted sum of the node features. Unlike the Graph Convolutional Network (GCN), which aggregates information from neighboring nodes using a fixed graph convolution operation, the GAT uses a self-attention mechanism to dynamically compute the attention weights for each node. This allows the model to focus on the most relevant nodes for a given task, rather than treating all nodes equally, by assigning weights right before the same message-passing cycle explained above takes place.

GNNs have been successfully applied to a variety of graph-based tasks, including social network analysis, protein structure prediction, and recommendation systems (WU et al., 2020). In the context of HPC scheduling, GNNs can be used to learn a representation of the DAG that captures the dependencies between tasks and the available computational resources. This representation can then be used to predict a scheduling policy fit for a given set of tasks and resources, potentially leading to improved performance and efficiency of the HPC system, depending on the metrics chosen to be optimized.

3.4 RELATED WORK

To investigate and highlight some aspects in the body-of-work portraying scheduling with GNNs, this section presents a review and systematic mapping of papers presenting scheduling solutions modeled with GNNs. The listed articles will be discussed and reviewed, primarily seeking relevant patterns for the research questions.

To conduct this review, we employed specific search terms to identify relevant literature. The search words used to find the papers are *"('schedule' OR 'scheduling') AND ('graph neural network' OR 'graph convolutional network' OR 'graph attention network')"*. We searched for papers in the following research databases: ACM Digital Library[1], IEEE Xplore[2], Web of Science[3], ScienceDirect[4], Scopus[5], and Engineering Village[6].

The main objective of this systematic mapping is to identify how GNNs are being used to solve scheduling problems. Specifically, our objective is to identify the variants of GNNs being used, the performance of the solutions compared to classic algorithms, and the learning paradigm used in the development of these solutions. To achieve this objective, we have defined the following selection criteria:

Objective Criteria: *(i)* only papers published after 2009 will be included; *(ii)* only papers from journals and events will be included; and *(iii)* only papers written in English will be included.

Inclusion Criteria: *(i)* only articles that solve scheduling problems using graph neural networks will be included; and *(ii)* only primary works will be included, *i.e.*, no secondary studies or reviews.

Exclusion Criteria: *(i)* articles that are not available for download will be excluded; and *(ii)* papers that do not include GNNs as an integral part of a solution to the scheduling problem will be excluded.

We believe that these criteria enable us to identify the most relevant works in this field and ensure that our review is comprehensive and relevant. Following these selection criteria, our aim is to identify the most recent and relevant research studies related to the use of GNNs to solve scheduling problems. In Table 2, it is possible to see a list of all selected works, together with the type of learning technique, the type of GNN-derived technique used, an indication of whether the scheduler achieved better performance than classical algorithms from the literature, and the year of publication.

The following sections provide a summary, chronologically, of the chosen research studies, classified into two distinct classes: *HPC Environment* and *Non-HPC Environment*. For each class, an overview is offered for the proposals, detailing which specific problem each study is solving, what variant of GNNs it is using, how the model's training is being conducted, and the model's relative performance. This organization helps establish a historical context, identifying gaps and potential avenues for further exploration.

[1] https://dl.acm.org/
[2] https://ieeexplore.ieee.org/Xplore/home.jsp
[3] https://www.webofscience.com/wos
[4] https://www.sciencedirect.com/
[5] https://www.scopus.com/home.uri
[6] https://www.engineeringvillage.com/home.url

Work	**Learning**	**Technique**	**Performance**	**HPC Related**	**Year**
(HU et al., 2020)	Reinforcement	GCN	Better	No	2020
(WANG; GOMBOLAY, 2020)	Hybrid	GAT	Better	No	2020
(GRINSZTAJN et al., 2020)	Reinforcement	GNN	Better	Yes	2020
(KIAMARI; KRISHNAMACHARI, 2021)	Supervised	GCN	Better	Yes	2021
(NI et al., 2021)	Reinforcement	GCN	Better	No	2021
(ZHAO et al., 2021)	Supervised	GCN	Better	No	2021
(PARK et al., 2021)	Reinforcement	GNN	Better	No	2021
(JING et al., 2022)	Reinforcement	GCN	Better	No	2022
(LEI et al., 2022)	Reinforcement	GNN	Better	No	2022
(LIN et al., 2022)	Reinforcement	GAT	Better	Yes	2022
(LIU et al., 2022)	Reinforcement	GNN	Undefined	No	2022
(PENG et al., 2022)	Reinforcement	GCN	Better	Yes	2022
(WANG et al., 2022)	Reinforcement	GCN	Undefined	Yes	2022
(ZHANG et al., 2022)	Reinforcement	GCN	Better	No	2022
(ZHOU et al., 2022)	Reinforcement	GNN	Better	Yes	2022
(DING et al., 2022)	Supervised	GCN	Better	No	2022
(WANG; GOMBOLAY, 2022)	Reinforcement	GNN	Better	No	2022
(SONG et al., 2023)	Reinforcement	GCN	Better	Yes	2023
(XING et al., 2023)	Reinforcement	GAT	Better	No	2023

Table 2 – Related work review. Source: the author.

3.4.1 HPC Environment

In this subsection, we focus on papers that apply GNNs HPC environments. These works typically address the challenges and complexities inherent in HPC systems. Initially, in the study by (GRINSZTAJN et al., 2020), the authors addressed the challenge of dynamic DAG scheduling in HPC environments. The solution employs a GNN leveraging a geometric deep reinforcement learning approach, specifically an actor-critic algorithm known as A2C. The GNN is designed to build an adaptive representation of the problem, dynamically adjusting to runtime system states and unexpected events, thereby offering a flexible scheduling mechanism. This GNN-based scheduler demonstrates competitive performance against established HPC scheduling heuristics. The adaptability of the methodology to incorporate additional knowledge for performance improvement further highlights its potential for broader applications in complex scheduling scenarios.

In the research done by (KIAMARI; KRISHNAMACHARI, 2021), the authors used a GCN, a variant of GNNs, to produce a scheduler for applications in large-scale distributed systems. The work uses HEFT and TP-HEFT algorithms (GALLET; MARCHAL; VIVIEN, 2009) to train a GCN and expand problem solving capabilities on a larger scale than deterministic algorithms can handle. The algorithms generate labels for Supervised Learning, which are later evaluated in scenarios with increasing workloads and different optimization objectives. The solution developed in the paper achieved performance similar to deterministic algorithms in smaller-scale scenarios and superior performance in larger-scale scenarios, maintaining an order of magnitude lower decision-making time.

In the research of (PENG et al., 2022), the authors also used a GCN to develop a scheduler, this time using Reinforcement Learning, in which the model interacts with an environment and receives rewards according to its performance. In this article, the GCN is used both as a decision-making model and for inference of certain aspects about the element being scheduled, which are workflows to be executed in a Cloud Computing environment, modeled in the form of DAGs. The produced model presented superior performance when compared to classical algorithms (*i.e.*, SJF and Random) and also other Reinforcement Learning models (*i.e.*, AC, Tetris, and PPO), achieving gains of 2-10% in reduction of makespan, as well as gains of up to 20% in resource utilization.

The authors of (WANG et al., 2022) presented MAGCIS, a multi-agent model based on Reinforcement Learning that uses a combination of Graph Convolutional Networks with Recurrent Neural Networks to produce a task scheduler in a Cloud Manufacturing environment, a recent concept in which consumers submit requests for manufacturing processes online that are distributed, using Cloud Computing and Edge Computing, to geographically distributed industrial manufacturing resources. In the work, a case study is analyzed for the manufacturing processes of structural aircraft parts, in which MAGCIS outperforms other Reinforcement Learning methods, but there is no information on performance compared to classical deterministic algorithms in the literature.

In the paper by (LIN et al., 2022), a scheduler for jobs represented in the form of DAGs is proposed using Graph Neural Networks with Attention, which is a variant of GNNs proposed by (VELIČKOVIĆ et al., 2017) that uses attention mechanisms (VASWANI et al., 2017) to focus on specific edges of the graph. The scheduler model was trained using Reinforcement Learning with heterogeneous loads. After training, an extensive comparison with various classical deterministic, heuristic, and metaheuristic algorithms was conducted, including algorithms such as the aforementioned HEFT (GALLET; MARCHAL; VIVIEN, 2009), HTDG (SULAIMAN et al., 2021), HDPSO (SHIRVANI, 2020), among others. The produced model showed superior performance in all different scenarios, with an average reduction of 12. 38% in the slowdown and average completion time metrics.

The authors of (ZHOU et al., 2022) presented LACHESIS, a job scheduler that uses a GNN model trained using Reinforcement Learning techniques on data from a well-known datacenter benchmark called TPC-H[7]. This benchmark is notorious for being used by companies such as Microsoft, IBM, Oracle, Nvidia, among others, for almost three decades. After training through interaction with a simulated environment, the scheduler was extensively compared with classical literature algorithms (*e.g.*, FIFO, SJF, among others), as well as heuristics that include the aforementioned HEFT (GALLET; MARCHAL; VIVIEN, 2009) and others. In addition, other probabilistic machine learning-based algorithms were also included in the comparisons. The presented scheduler achieved the best performance in all scenarios, sometimes achieving an improvement of 26.7% in the makespan metric and 35.2% in Speedup.

[7] https://www.tpc.org/tpch/

In (SONG et al., 2023), the authors presented a GCN-based model for the scheduling of job DAGs in heterogeneous computing environments. The solution was modeled in such a way that the network was used to extract features from the DAGs that were fed to a policy network for computing node selection. The model was trained used Reinforcement Learning in a simulated environment. When compared to classical job scheduling algorithms, the proposed solution had superior performance, getting better results in the average job cost, parallel performance, stability and other evaluation metrics.

3.4.2 Non-HPC Environment

In this subsection, we explore the papers where GNNs are utilized for scheduling in environments other than HPC. This includes a variety of domains that may be adjacent to HPC but aren't defined by the body of work as such.

In the article by (HU et al., 2020), the authors proposed a Reinforcement Learning model based on a GCN using layers modified by the authors to better fit problems modeled in graphs representing Petri Nets. The model aims to schedule Flexible Manufacturing Systems for production line optimization and does so through the interaction of the model with a virtual representation of the real production line. The model presented superior results compared to deterministic algorithms, such as FCFS+ (YOU; WANG; ZHOU, 2015) and D^{2}WS (LUO et al., 2014), as well as showing greater resilience to changes in the structure of the environment compared to other Reinforcement Learning methods.

In the work of (WANG; GOMBOLAY, 2020), a task scheduler was presented for coordinating movements and interactions between robots. The presented scheduler uses a GAT network trained through a combination of Supervised Learning, along with Reinforcement Learning in a simulated environment. After training, the scheduler called RoboGNN was compared with three other algorithms from the literature: the heuristic algorithm EDF (HELLERMAN, 1969), the state-of-the-art hybrid algorithm Tercio (GOMBOLAY; WILCOX; SHAH, 2018), and finally, the linear programming solver software Gurobi that finds the optimal result through a computationally intense process. The comparison between the schedulers was made with increasing numbers of robots and task complexity. RoboGNN achieved better makespan than EDF in all scenarios and results close to or even better than Tercio. Furthermore, RoboGNN was still able to solve more problems than EDF and Tercio, maintaining between 89.3% and 91.5% resolution rate, while EDF and Tercio remained between 0% and 62%. Additionally, RoboGNN maintains this performance with 100 times speedup compared to the exact method using Gurobi.

In the work by (ZHAO et al., 2021), a request scheduler for wireless networks was presented, which combines a GCN with greedy algorithms. The GCN is used to generate an approximate representation of the optimal embedding of the current network state and then uses greedy algorithms for scheduling decision-making. The GCN is trained through Supervised Learning using random and heterogeneous graphs provided by (ERDŐS; RÉNYI et al., 1960) and (ALBERT; BARABÁSI, 2002). Finally, the trained model was compared to greedy heuristic algorithms LoGreedy (JOO; SHROFF, 2011) and MP (PASCHALIDIS; HUANG; LAI, 2014), achieving results between 3% and 5% closer to the optimal result than its competitors.

In (NI et al., 2021), a authors proposed a scheduler based on a GCN combined with a neural network with attention mechanisms for scheduling jobs in the context of industrial automation in warehouses. Reinforcement Learning techniques were used for training, and two distinct databases were used, one of them being a standard test database and the other a real database captured during automation execution. After training, the model was compared with a classic metaheuristic algorithm known as IteratedGreedy (IG) (RUIZ; STÜTZLE, 2007). Results in different scenarios showed the presented scheduler as superior, achieving on average a 53% better performance compared to IG, with no statistically significant changes in decision-making time.

In (PARK et al., 2021), the authors introduced the GNN-RL scheduler model, which uses a GNN trained through Reinforcement Learning to predict a parameterized scheduling policy, using the prediction of probability density functions for job scheduling decision making. To test the model, a simulation environment of Job-shop Scheduling Problem (JSSP) was proposed, in which the proposed model and other classic job scheduling algorithms would interact with synthetic data loads. The results showed that the GNN-RL model obtained consistently lower error rates in all proposed load scenarios, demonstrating superior performance compared to classical algorithms.

In the research done by (LIU et al., 2022), a framework was presented for modeling the problem of flexible scheduling of jobs (i.e., a set of tasks with some type of dependency) for traffic control, using GNNs in the form of Reinforcement Learning. The authors use Graph Neural Networks competing with each other within an environment that provides rewards according to performance in scheduling jobs, which are also represented in the form of DAGs, similar to the work by (PENG et al., 2022) but using other techniques of GNNs and Reinforcement Learning. The model presented similar results to other Machine Learning techniques, but there were no comparisons with deterministic algorithms in the literature.

On the other hand, the work by (LEI et al., 2022) also addressed the problem of flexible scheduling of jobs by defining a framework for developing Reinforcement Learning-based models. The problem is modeled through graphs representing the structure of Multiple Markov Decision Processes, which serves as input to a GNN that will make the scheduling decision. Several stochastically generated scenarios were assembled, and a model was generated and compared to several other classical algorithms and state-of-the-art meta-heuristic algorithms. The generated model was extensively tested and consistently showed results that outperformed deterministic algorithms and approached state-of-the-art meta-heuristic results, while maintaining a consistently better decision-making time in all scenarios.

In the work of (ZHANG et al., 2022), the authors addressed the scheduling of DAGs in a Vehicular Edge Computing context, taking into account scenarios with significant network delays. Using Reinforcement Learning techniques, the authors used a GCN trained on both stochastic data and real communication scenarios. The model interacts with an environment that provides a reward based on the makespan metric, which is the target metric of the work. After training, the generated scheduler was compared with the classical algorithms HEFT (GALLET; MARCHAL; VIVIEN, 2009), LC (KIM, 1988), and CPOP (TOPCUOGLU; HARIRI; WU, 2002). In addition, Machine Learning-based algorithms were also used in the comparison. When compared to Machine Learning algorithms, the proposed scheduler achieved performance improvement (*i.e.*, reduction of makespan) between 8% and 15%, while when compared to classical algorithms, the improvement in performance ranges from 15% to 25%.

In (JING et al., 2022), the authors presented another job scheduling algorithm based on GCNs with Reinforcement Learning, this time in an Internet of Things in Manufacturing scenario. The scheduler, called GMAS, was trained on multiple consolidated datasets, aiming to optimize multiple metrics such as makespan, CPU time, among others, depending on the scenario. The generated algorithm was extensively compared with classic algorithms as well as state-of-the-art meta-heuristic algorithms, consistently obtaining better results in both comparisons, maintaining scalability through a technique of transposing the scheduling problem into a probability distribution prediction problem.

In (DING et al., 2022), the authors presented a model for predicting communication load and energy usage in Road Side Units (RSUs) integrated with vehicular communication networks, using multiple GCNs trained with Supervised Learning to assist other Machine Learning models, this time trained with Reinforcement Learning, in the task of scheduling energy harvesting and signal transmission. The algorithm is compared with other predictive models and also with traditional algorithms, the latter being surpassed by the model presented in all measured efficiency metrics, reaching up to 50% better performance in some scenarios.

In the article by (WANG; GOMBOLAY, 2022), the authors presented a maintenance scheduling model for a fleet of aircraft, using a technique they introduced for Reinforcement Learning called HetGPO. The technique involves the interaction of the scheduler with a virtual benchmark environment called AirME, to adjust the weights of a GNN that tries to maximize the performance of aircraft maintenance teams through efficient scheduling of maintenance tasks. The trained model was then compared with classical and heuristic methods, showing superior performance in small, medium, and large-scale scenarios, achieving results between 16.5 and 17.7% better in the availability metric and between 27.5 and 29.1% superior in the total profit metric when compared to classical schedulers and heuristics.

Finally, in the work of (XING et al., 2023), a GAT-based model is presented as a scheduling assistant solution for scheduling power supply in Active Distribution Networks, *i.e.*, a modern, technology-enabled electric power distribution system that allows for bidirectional power flow and advanced grid management. The GAT model was trained using Reinforcement Learning and performed better than classical approaches in metrics such as power loss per MWh, voltage deviation per power unit, operation cost and others. The solution was specially remarkable under special electrical fault scenarios where it surpassed other approaches by being more dynamic under topology variations.

3.4.3 Discussion and challenges

The increase in the number of articles in each year shows a growing trend in the use of GNNs for developing solutions to the scheduling problem, which may be related to recent developments in the technique. Although GNNs have been introduced more than a decade ago, they have recently been explored with the creation of more powerful and suitable variants for large-scale problems. Furthermore, the growth in data availability and computational resources is a factor for recent development in the field of Machine Learning (LECUN; BENGIO; HINTON, 2015), which may also be a contributing factor to the increase in the number of articles. It is also interesting to note that although the objective criteria include articles from 2009 onward, which was the year of publication of the first GNN, only recent articles that met all the requirements imposed in this work were found.

Despite extensive research, there is a notable scarcity of papers at the intersection of GNNs, scheduling, and HPC in the sources used for this related work section. Even after extensive search through popular scientific repositories, the combination of these three elements remains underexplored compared to the larger field. Consequently, this work seeks not only to further explore the intersection of GNNs, scheduling and HPC, but also to extend the investigation to the application of GNNs in scheduling problems outside of HPC environments. This broader approach aims to underline the value that GNNs bring to the domain of scheduling problems, regardless of the specific environment.

The surveyed papers predominantly present positive outcomes when compared to classical algorithms, suggesting that GNNs can be a useful tool for solving scheduling problems. However, challenges emerge, particularly when applying GNNs using Supervised Learning, which represents a significant gap in the existing literature. This gap is of particular interest to the current work, which seeks to explore and potentially address the hurdles associated with implementing Supervised Learning methods in GNN-based scheduling solutions.

Table 3 shows the distribution of published papers from the related work in relation to the techniques derived from GNNs, with a clear majority for GCNs, followed by similar proportions of GNNs and GATs. This indicates a greater popularity of GCNs, which may be due to the short exploration time of GATs that were introduced in the literature in 2017. Finally, the use of pure GNNs seems to be trending towards being replaced by their variants.

Technique	**Published Papers**	**Percentage**
GCN	10	52.6%
GNN	6	31.6%
GAT	3	15.8%

Table 3 – Published papers by technique. Source: the author.

Table 4 illustrates the distribution of works by the type of learning used in the networks, with a majority for Reinforcement Learning and a minority divided between Supervised and Hybrid Learning, the latter being just a combination of Supervised and Reinforcement Learning during training. The predominance of the use of Reinforcement Learning can be attributed to the difficulty of obtaining labeled data for combinatorial problems, since obtaining the optimal result becomes computationally infeasible depending on the desired scale.

Learning Paradigm	**Published Papers**	**Percentage**
Reinforcement	15	78.9%
Supervised	3	15.8%
Hybrid	1	5.3%

Table 4 – Published papers by learning paradigm. Source: the author.

Finally, Figure 6 shows the Systematic Mapping of the literature indicating the density of works in the intersection between GNN-derived technique and type of learning. Notably, the learning types were reduced to Supervised and Reinforcement, with Hybrid counted as a record for both, since it was nothing more than an application of both techniques to different data. Figure 6 demonstrates possible research gaps, especially in Supervised Learning, a class that obtained little representation in the articles found for reasons discussed previously.

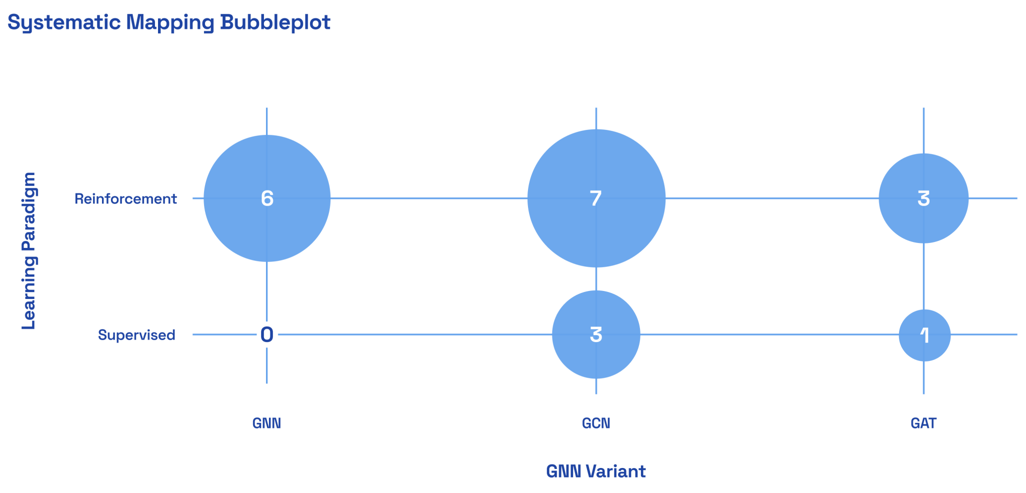


Figure 6 – Bubble Plot of the Systematic Mapping

Thus, it is possible to conclude that the literature, for the most part, uses the GCN variant through Reinforcement Learning for the production of schedulers that have performance above the classical schedulers in the literature. The literature seems to specially lack solutions using Supervised Learning, possible reasons for it include the challenge of acquiring reliable and clean representative data for the learning process, the difficulty of modeling such data due to the complex nature of scheduling problems, etc. Another aspect lacking is the amount of solutions using GAT, which utilizes a technique that demonstrated to be specially suitable for combinatorial problems (VELIČKOVIĆ et al., 2017), *i.e.*, the attention mechanism. The scheduling problem being a combinatorial one may benefit from this mechanism the same way sequence-to-sequence, a combinatorial problem in nature, has benefited from the attention mechanism (VASWANI et al., 2017).

3.5 CONSIDERATIONS

In this chapter, we have discussed how Machine Learning (ML) is applied as a solution or as an assistance to the solution to the job scheduling problem inside and outside of the context of HPC. We also discussed how the usage of ML affects the modeling of the scheduling problem, along with all the constraints the techniques usually bring with them. Furthermore, we have discussed how the representation of the scheduling problem using DAGs interacts with the usage of ML-based solutions, the main challenges of the combination of these concepts and how Graph Neural Networks (GNNs) solve some of these issues. Finally, the related work has been presented in the form of a systematic mapping of the research area with respect to GNNs applied to the solution of the job scheduling problem.

The systematic mapping carried out in this study demonstrates the growing trend in the use of GNNs to develop scheduling solutions. The popularity of GCNs can be attributed to its long exploration time compared to other variants such as GATs. Reinforcement Learning remains the most utilized learning type due to the difficulty in obtaining labeled data for combinatorial problems. However, there is a clear gap in the literature on the use of Supervised Learning for scheduling problems. This could be due to challenges in acquiring representative data and modeling them due to the complex nature of the scheduling problem.

The application of ML-based techniques in HPC scheduling has been a research field for many years; however, as the ML field is further pushed to new grounds and techniques are improved, we can expect new state-of-the-art algorithms to be derived from the intersection of these fields, as has happened with many other fields in recent years (LECUN; BENGIO; HINTON, 2015). But since the field branches into many different approaches, each new technique arises with strong and weak points, which means that further research is needed to see in which scenario each technique inside of ML can be better used. The motivation for this research arises from the lack of studies comparing variants of GNNs trained with reliable Supervised Learning to solve the scheduling of jobs modeled as DAGs in the context of HPC. As GNNs are suited for problems modeled with graphs, there is a natural interest in applying it to problems similar to job scheduling for its natural representation using graphs. However, as representative data is not so easily acquireable for such a problem, due to its computational complexity at higher scales, many researchers opt to use Reinforcement Learning, which is a valid solution since as the model interacts with an environment it may pick up on the nature of scheduling; nevertheless, this modeling is highly dependable on the reward function, and so it has a high bias introduced by said function which is usually a difficult task to ascertain its validity and consistency, especially in a multifaceted problem such as HPC job scheduling. In conclusion, this motivates this research to develop a solution using Supervised Learning, which may bring a more reliable and dynamic behavior of the model, but also brings the aforementioned challenges.

In Chapter 4, a solution proposal is presented, along with a framework that will be used to work through the challenges that arise from the chosen specific research context.

4 GNN-BASED HPC JOB SCHEDULER

In this chapter, we detail the methodology for implementing multiple GNN-based models trained using the KAIROS framework with the objective of producing job schedulers for HPC systems. We introduce the proposal by detailing how the KAIROS framework operates, which improvements were made to it for this research, how would a GNN model be used inside the KAIROS framework and how the model will be trained and compared.

Additionally, this chapter presents how the objectives of this research are fulfilled. That means that it mainly drives to investigate the efficiency of GNNs as models for HPC schedulers. Also, it aims to compare the GNN variants in the job scheduling problem in HPC, also to investigate the sensitivity of GNN-based models to change in scenario, to develop a scheduler based on GNN and to evaluate this scheduler inside a strict protocol.

4.1 KAIROS FRAMEWORK: CONCEPTS AND IMPROVEMENTS

To achieve the objectives of the research and to develop the proposal model, the KAIROS framework (PEREIRA; KOSLOVSKI, 2020) was chosen, both because it is based on Supervised Learning and supports a range of different models, including GNNs and their variants. Another main reason for the adoption of this framework is its natural usage of DAGs for the representation of jobs and environmental entities. However, it is worthwhile to mention that KAIROS just offers the baseline support for implementing the present work. After training and evaluating the model, it can be attached and used by any resource management system or simulator.

KAIROS is a framework for the development of models based on Supervised Learning for end-to-end job scheduling, inspired by the ML method called stacking (WOLPERT, 1992). In the KAIROS framework, the model learns through Supervised Learning from data gathered by the interaction of classical deterministic scheduling algorithms, called tutor algorithms in the framework, with a simulated HPC environment. Knowledge from the weaker schedulers is aggregated and extrapolated by the ML-based model, which may be able to surpass its tutors.

The framework was developed and improved by the authors in previous works (PEREIRA; KOSLOVSKI, 2020; ALBUQUERQUE, 2022), and the improvements made in this work enhance its capabilities. Essentially, the framework consists of three main components: tutor algorithms, the simulated environment, and the ML-based model. The tutor algorithms are the classical deterministic scheduling algorithms that interact with the simulated environment. The simulated environment is a virtual environment that emulates a real-world HPC scheduling scenario. The simulated environment generates data that is used to train the ML-based model. Finally, the ML-based model is a supervised learning model that learns from the data generated by the simulated environment.

One of the strengths of the KAIROS framework is its ability to generate reliable high quality data in large quantities via interaction with tutor algorithms. The tutor algorithms are considered the weaker schedulers, and the knowledge derived from their interaction with the environment is extrapolated by the ML-based model. The data generated by the simulated environment can be used to train the ML-based model to predict the optimal schedule for a given set of jobs.

4.1.1 Data Extraction and Model Training

The process of training an AI-based end-to-end scheduler using the KAIROS framework can be divided into two major stages, the Data Extraction stage and the Model Training stage. Both of these stages have sub-steps that need to be completed, which are explored in detail in this section. The Data Extraction stage of the KAIROS framework is represented in Figure 7 where a four-step process is depicted. Step 1 presents one of the inputs of the system, which consists of a collection of jobs structured as DAGs. Each DAG represents a job to be scheduled within the system. Moving to Step 2, these jobs are decomposed into individual tasks using the topological sort algorithm (PEARCE; KELLY, 2007), forming a queue of tasks. Step 3 illustrates another input of the system, namely, a graph representing the computational infrastructure for the simulation. Finally, in Step 4, a system called the Aggregator System executes each tutor algorithm in an event-based simulation, meaning that each tutor can see a snapshot of the system and the queue at each timestep to make its decisions, with these timesteps representing an arbitrary amount of time, depending on the problem's modeling. Each tutor aims to schedule the task queue, defined in Step 2, in the infrastructure, defined in Step 3, with every decision being monitored and logged. This process generates the output of the Data Extraction stage, which is the raw data to be refined and used as training data in the subsequent stage.

The Model Training stage of the KAIROS framework is represented in Figure 8, which depicts the stage as a five-step process. Step 1 represents one of the inputs to this stage, which is the raw data collected during the previous Data Extraction stage. The raw data, now referred to as the Tutors' Knowledge, contains every interaction of each tutor with the system's simulation. Moving to Step 2, the Tutor's Knowledge is filtered by selecting a set of metrics. These metrics determine which knowledge should be preserved and which should be discarded based on the following score function, derived from (YEN; ABBASLOO; CHAO, 2023): $(T-\alpha \cdot L)^k/\bar{D}$. Here, T represents the HPC system throughput in terms of task execution; L the loss rate that in this work was measured in window steps that separate tasks from the same job; α a weighting factor; k a scaling constant and $\bar{D}$ is the average delay (bounded slowdown) of scheduled tasks for each of the aggregated jobs. This score is applied for the scheduling of each tutor at each timestep, this score determines the probability of each interaction to be chosen in relation to all others, and an interaction is chosen stocastically based proportionally on these score functions, with better interactions having a higher chance to be chosen at each timestep. This process aims to leverage exploration over exploitation, avoiding issues that arise from greedily choosing only the highest average of metrics at each timestep.

After the filtering process in Step 2, we move to Step 3, where the filtered data is amalgamated into the training data. The training data is divided into two subsets: the training

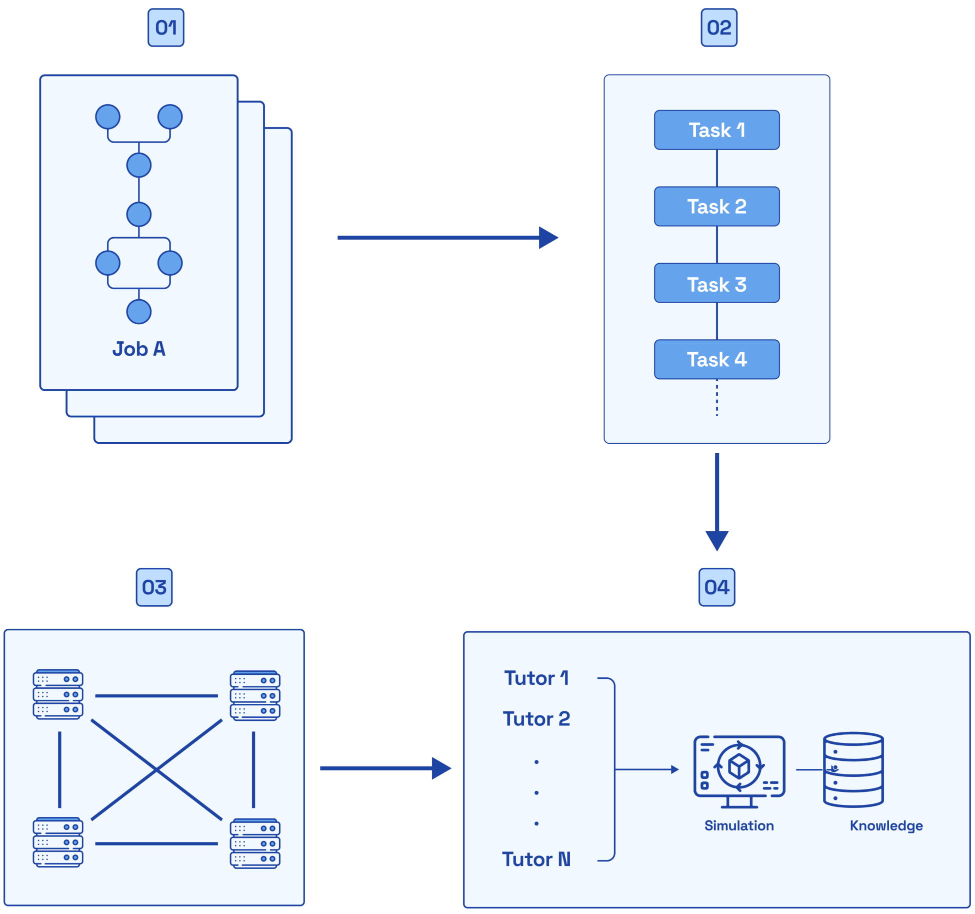


Figure 7 – KAIROS framework training data extraction architecture. Source: the author.

dataset and the validation dataset. The training dataset is used to train the model, while the validation dataset is used to assess the model's performance. This separation ensures that the model is tested on data points it has never seen before, which helps evaluate its generalization capabilities. Step 4 involves training and testing the model, which is another input into this stage. Training and testing are performed using the datasets defined in Step 3. The inputs to the model are the graphs representing the tasks in the queue, while the labels are the definitive scheduling of each task. The model is trained using various algorithms and techniques to learn patterns, usually depending on the model architecture. Finally, in Step 5, the output of the Model Training stage is an end-to-end job scheduler trained with Supervised Learning in a dataset generated through the interaction of classic scheduling algorithms with an event-based simulation.

4.1.2 Improvements

In this research, the KAIROS framework has been improved in several ways. First, two sliding windows have been included in the framework: the hindsight window and the decision window. The hindsight window is the number of timesteps in the future where the decomposed tasks can be allocated, while the decision window is the number of decomposed tasks from the queue that the model is able to see. The inclusion of these two windows improves the flexibility of the KAIROS framework, making it better suited to a wider range of scheduling scenarios. In

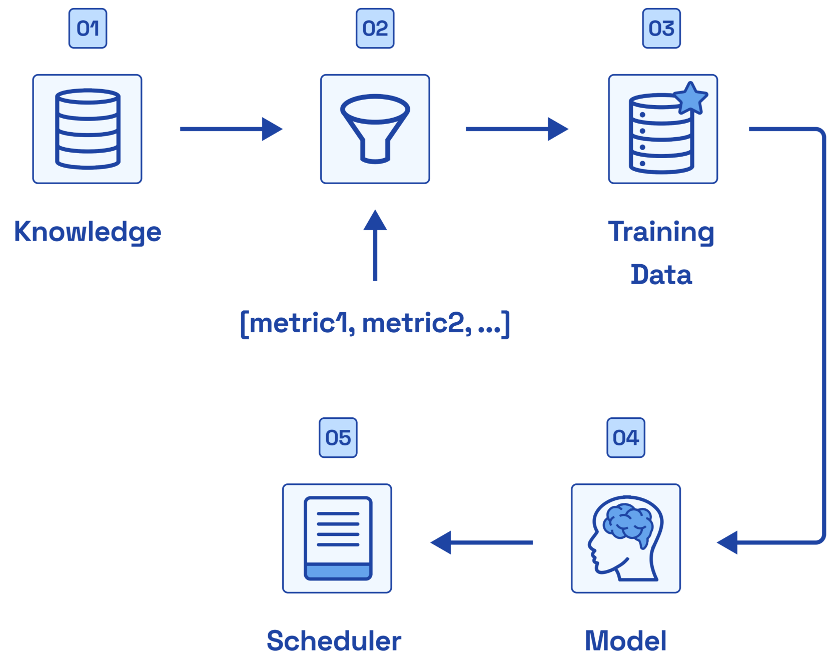


Figure 8 – KAIROS framework model training architecture. Source: the author.

addition to that, the framework had its graph integration capabilities enhanced, being able to receive the job graphs, decompose them into tasks that hold references to the original job's graph representation, allowing models to access the full graph structure and still work with individual tasks on each timestep.

These improvements were made to address the challenges associated with input and output dimensionality when modeling solutions to the scheduling problem. The sliding windows serve the purpose of placing upper limits on the dimensions of input and output for the neural network model, ensuring its compatibility with open and closed system simulations. Moreover, the increased graph integration allows for the easier inclusion of certain models, notably Graph Neural Networks, facilitating more effective and comprehensive scheduling solutions.

Figure 9 illustrates the job decomposition process into a queue of tasks and the two sliding windows (*i.e.*, decision window and hindsight window). Step 1 illustrates a collection of jobs, each of which has a finite number of tasks to be executed in an order constrained by the dependencies. In Step 2 the jobs have been decomposed into a group of tasks, each task having the specific details about its expect processing time, its resources demand and its dependencies inside the original job. The Decision Window determines how many tasks the scheduler will be able to see at any given time, and it slides forward as the scheduler further allocates the tasks. Finally, in Step 3, the tasks are allocated to some specific node, in one of the spots inside the Hindsight Window, which dictates how much in the future the scheduler can allocate a task, meaning how much it can delay any given task.

Furthermore, the KAIROS framework has been enhanced to efficiently receive DAGs and

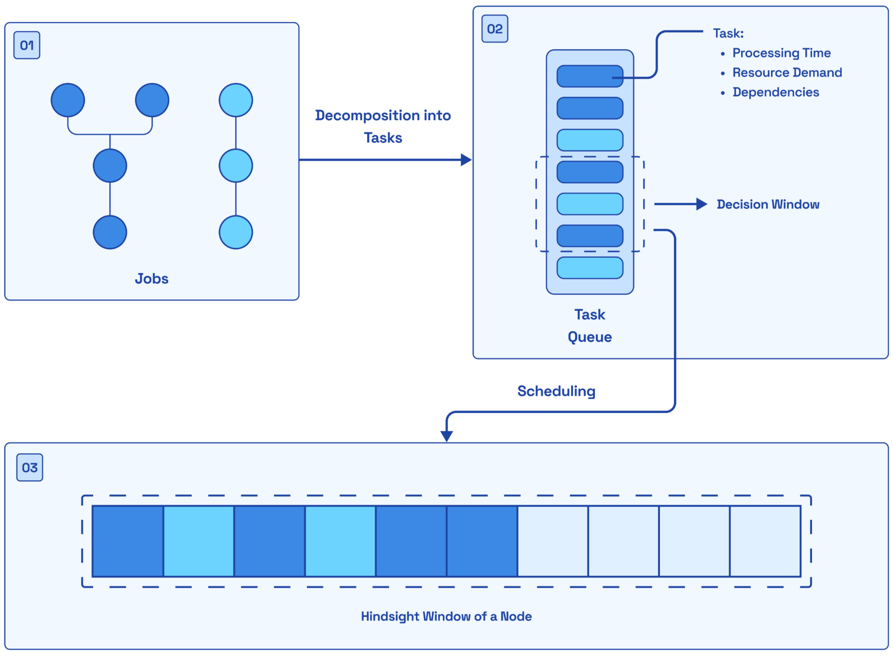


Figure 9 – KAIROS framework sliding windows. Source: the author.

decompose them into tasks organized in a queue. DAGs are commonly used to represent complex workflows in job scheduling, and the ability to efficiently decompose them into tasks while maintaining relationship information from the DAG is an useful improvement for the KAIROS framework, since the original graph and its subgraphs can be recovered from the task at any given point. The enhanced KAIROS framework can now handle DAGs with greater dynamism, making it more suitable for complex scheduling scenarios that utilize GNN models as schedulers.

In general, the KAIROS framework is a tool for the development of ML-based Supervised Learning models for job scheduling. Improvements made in this work enhance the capabilities of the framework and make it more suitable for a wider range of scheduling scenarios. The ability of the KAIROS framework to generate quality data in large quantities through the interaction of tutor algorithms is a key strength that can lead to significant improvements in job scheduling performance.

4.1.3 Tutors

In the KAIROS framework, the tutor algorithms are usually classical deterministic scheduling algorithms that interact with the simulated environment to generate data that is used to train the ML-based model. The tutors are weaker schedulers, and the ML-based model is designed to learn from their behavior arriving at an aggregated decision process with better scheduling performance. We have selected six classical deterministic scheduling algorithms as tutors:

1. First-Come-First-Served (FCFS). The FCFS scheduling algorithm is a simple scheduling

algorithm that schedules tasks based on their arrival time. In this algorithm, the task that arrives first is scheduled first (BRUCKER, 1999).

2. Shortest Area First (SAF). The SAF algorithm schedules tasks based on the amount of resources they occupy (BRUCKER, 1999), taking into account both the computational requirements and the time it is expected to take for the task to be executed. It selects the task that requires the smallest area, which is defined as the product of the computational requirement and time required for execution (*i.e.*, walltime).

3. F1, F2, F3, and F4. The four priority-based schedulers denoted by F1, F2, F3, and F4 are based on different priority functions (CARASTAN-SANTOS; CAMARGO, 2017). Each function assigns a different priority and they all take into consideration the processing resource demand of each task and the time needed for the execution of each task. The priority-based schedulers assign priorities to tasks based on these priority functions and then schedule tasks in descending order of priority.

The Table 5 further defines the tutor algorithms, detailing their policies definitions and the mathematical formula for finding the task to be scheduled in the nearest available processing node. The mathematical formulas take into consideration p_t, which is the total processing time needed to complete task t; q_t, which is the amount of computational resources required to process task t and r_t, which is the submission time of task t. For each policy decisions, the queue is sorted by the formulas, from the lowest to the highest.

Algorithm	**Definition**	**Formula**
FCFS	The task with the earliest submission time is selected.	$FCFS = r_t$
SAF	The task with lowest total value of resource and time to completion is selected.	$SAF = p_t \cdot q_t$
F1	Nonlinear function derived from statistical analysis.	$\log_{10}(p_t) \cdot q_t + 8.70 \cdot 10^2 \cdot \log_{10}(r_t)$
F2	Nonlinear function derived from statistical analysis.	$\sqrt{p_t} \cdot q_t + 2.56 \cdot 10^4 \cdot \log_{10}(r_t)$
F3	Nonlinear function derived from statistical analysis.	$p_t \cdot q_t + 6.86 \cdot 10^6 \cdot \log_{10}(r_t)$
F4	Nonlinear function derived from statistical analysis.	$p_t \cdot \sqrt{q_t} + 5.30 \cdot 10^5 \cdot \log_{10}(r_t)$

Table 5 – Tutor schedulers definitions. Source: the author.

These classical deterministic scheduling algorithms are not specifically designed to handle tasks with dependencies. To address this limitation and ensure valid scheduling, an algorithm will be introduced that modifies each classical algorithm by enforcing constraints. This algorithm guarantees that each task is allocated no sooner than the latest end of the processing of its dependencies, preventing invalid schedulings. By uniformly applying this dependency-aware approach to all classical algorithms, task dependencies are effectively considered, leading to correct and consistent scheduling results. This integrated solution combines the strengths of each deterministic algorithm with dependency awareness, making them viable tutors for the model to be trained.

In addition to these classical deterministic algorithms, two heuristics were selected as tutors, these being algorithms specifically tailored for scheduling DAGs with dependencies. The chosen heuristics are:

1. Heterogenous Earliest Finish Time (HEFT) (GALLET; MARCHAL; VIVIEN, 2009). The HEFT algorithm aims to minimize the total execution time of a set of tasks by calculating the earliest finish time for each task on each available resource based on the task's computation and communication costs. The algorithm then assigns tasks to resources that have the earliest finish time, taking into account the data dependencies between tasks.
2. Improved Predict Earliest Finish Time (IPEFT) (ZHOU et al., 2017). The IPEFT predicts the earliest finish time of tasks by estimating the execution time of each task on each available resource based on historical data. Then it can make more informed scheduling decisions and allocate tasks to resources that are likely to provide the best performance.

Both of these heuristics can serve as viable tutors. HEFT's focus on minimizing execution time by considering computation and communication costs makes it suitable for generating training data and establishing a baseline for scheduling performance. IPEFT, on the other hand, enhances scheduling by predicting the earliest finish time of tasks using historical data. Integrating both heuristics as tutors allows the generated data to be enriched with traces of more complex dependency handling and predictive capabilities.

In summary, each of the tutor algorithms uses a different scheduling strategy to determine the order in which tasks are scheduled. The data generated by the interaction of these tutor algorithms was used to train the GNN-based models, to produce the end-to-end schedulers.

4.2 SCHEDULING GNNS

In the KAIROS framework, the scheduling using GNNs is achieved through the integration of these models into the ML-based model component. The ML-based model learns from the data generated by the tutor algorithms that interact with the simulated environment, and is responsible for making scheduling decisions based on the learned patterns and features.

As mentioned previously, the framework can now freely access graph information, even when dealing with a subset of tasks from a job, allowing it to gather graph information and transmit it to the model component. Having this in mind, for the GNN integration, the framework is set up with a GNN variant (*e.g.*, GCN, GAT, etc.) as the model to be trained, which will then be provided with the relevant graph information by the framework, as it stores each of the tutor algorithms interaction with the environment, including each task scheduling result and the associated job graph, even if the tutor itself does not take that information into consideration, which is usually the case with classical algorithms (BRUCKER, 1999).

As the framework controls the simulation, including the system resource status, queue and sliding windows, a snapshot of the scheduling process can be constructed at each timestep.

After the tutors finish interacting with the environment, the data is collected by the aggregator module, which then trains the GNN model with the experiences of the tutors, choosing the best scheduling tactic at each timestep based on the score function that takes into account the target metrics previously set up in the framework. The model is then trained and it is ready to be evaluated and reiterated if necessary.

4.3 SIMULATION AND MODEL TRAINING

A snapshot of the simulation process used in KAIROS is shown in Figure 10. The simulation operates on a queue of tasks, where the number of visible tasks to any scheduler is restricted by a sliding window known as the Decision Window. Within this window, a tutor algorithm is employed to schedule the tasks and allocate them to specific nodes in the system. The execution start time for each task is determined by another sliding window, known as the Hindsight Window. The allocation process takes place on a virtual matrix representation, where each row corresponds to a node in the system and each column represents a timestep within the Hindsight Window; this is called the Allocation Matrix. The system assigns each task in the virtual matrix by sampling a subset of the queue defined by the Decision Window. This approach approximates real-world HPC scenarios, where tasks may experience varying levels of delay and systems can range from offline to online. After all tasks from the Decision Window are allocated, the system simulates the passage of time by sliding the Hindsight Window to the end of the latest allocation, finishing a cycle of the simulation. A new cycle is then started by moving the Decision Window to the end of the last task allocated, sampling a new subset of tasks to be allocated in a now clean Allocation Matrix. The simulation cycle continues until all tasks in the queue have been successfully allocated.

The design of the simulation system, as described above, is driven by the need to accurately model real-world HPC environments while allowing flexibility and scalability. The inclusion of the Decision Window addresses the requirement to simulate offline and online systems. By limiting the number of visible tasks to the scheduler, the system mimics the dynamic nature of HPC workloads. This enables the evaluation of scheduling algorithms under realistic conditions, even when working with a predetermined set of jobs.

The presence of the Hindsight Window serves another crucial purpose in the simulation. It allows for the simulation of an indeterminate number of timesteps, while preserving the necessary dimensions for training an ML-based model. By organizing the allocation process in a virtual matrix representation, the Hindsight Window provides a consistent framework for capturing the temporal aspect of task scheduling. Adjusting the sizes of both the Decision Window and the Hindsight Window allows for the representation of various HPC systems with different characteristics, ensuring that the simulation can adapt to a wide range of scenarios. This flexible approach facilitates the investigation of scheduling strategies and resource allocation techniques in diverse HPC environments, enhancing the applicability and relevance of the simulation system.

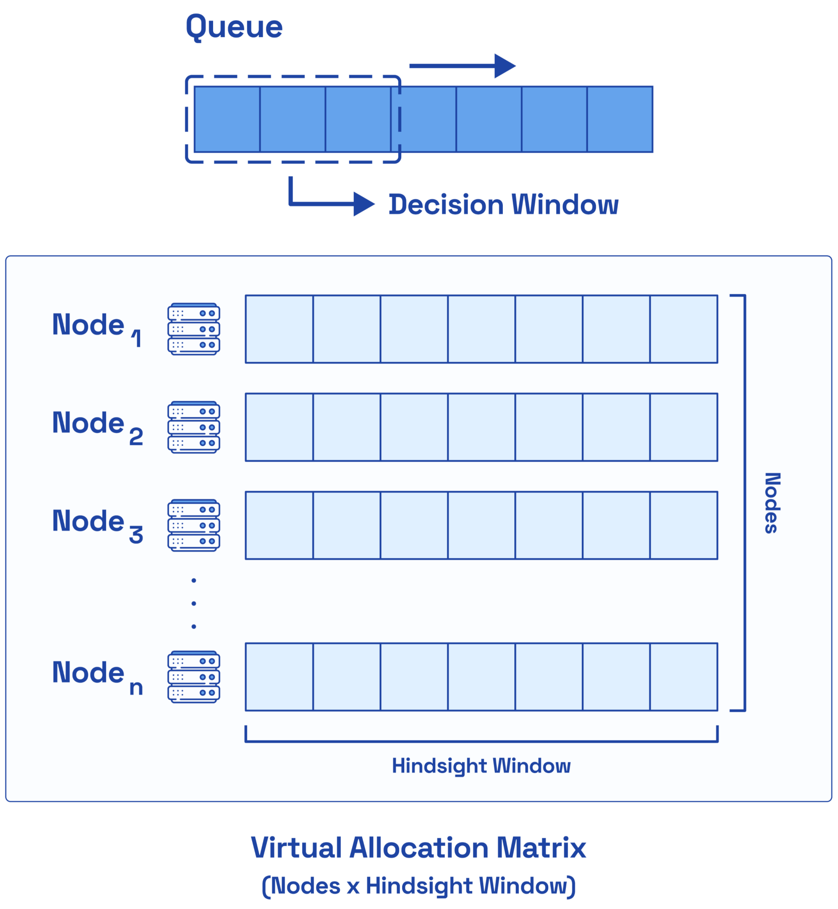


Figure 10 – KAIROS framework simulation environment snapshot. Source: the author.

Given these characteristics, some constraints must be followed to ensure successful execution of the simulation process. Firstly, no task in the job collection can have a resource demand greater than the available resources to any one node in the system since it would make it impossible to fit the task in any one node and make it impossible to allocate in the proposed simulation. Also, the duration of any task must not succeed the size of the Hindsight Window, since the event-based nature of the simulation requires the total task to be allocated in the virtual Allocation Matrix, as to provide a full context to the ML-based model that will be learning from snapshots of the final allocations. The Decision Window has no upper limit, having only the necessity of being larger than zero as a size constraint, since at least one task need to be allocated per cycle for the simulation to eventually end.

One could argue that increasing the size of the Decision Window in the simulation system leads to a closer approximation of an offline system. This is because a larger Decision Window allows for more tasks to be allocated before any changes occur in the task queue. In the extreme case where the Decision Window size equals the queue size, it represents a complete offline likeness, as all tasks are known prior to allocation. On the other hand, the size of the Hindsight Window determines how far ahead the scheduling algorithm can consider when making allocation decisions. A larger Hindsight Window provides more room for exploration of available resources, but may also result in larger and more frequent gaps in the allocation process. The selection of appropriate sizes for both windows involves striking a balance between replicating offline characteristics and maintaining a realistic level of resource exploration and allocation efficiency within the simulation system.

4.4 IMPLEMENTATION

The prototype was developed as an event-based simulator designed to mimic a HPC environment. Jobs are submitted to a queue and await to execute atop a HPC Data Center (DC), and tasks are characterized by their submission time, required walltime, demand for CPU resources, and dependencies. The simulation progresses by either sliding the window forward when all servers are unable to accept further tasks or when the job queue becomes empty, signifying the completion of the current set of tasks. The prototype was implemented in Python 3.10, and leveraged key packages, mainly *networkx* for managing DAGs, *pytorch* and *PyG* as the GNN backbone. We empirically set the constants of the score function $\alpha = 1.5$ and $k = 2$. The simulation and training of the models were executed on a computer with an Intel i5-9400F CPU, 64 GB of RAM and equipped with an NVIDIA RTX 3080 TI GPU with 12 GB memory.

Algorithm 1: KAIROS framework algorithm

```
    Input: tutors, DC, $w_{hindsight}$, $w_{decision}$, job_queue, score_function, gnn_variant
 1  model ← GNN(gnn_variant);
 2  aggregator ← initialize_kairos(DC, $w_{hindsight}$, $w_{decision}$);
 3  task_queue ← topological_sort(job_queue);
 4  for tutor in tutors do
 5      while not task_queue.empty() do
 6          tasks ← task_queue.get($w_{decision}$);
 7          experience, metrics ← tutor.schedule(tasks);
 8          aggregator.evaluate(experience, metrics);
 9          aggregator.save(metrics);
10          update_sliding_windows($w_{hindsight}$, $w_{decision}$);
11      end
12  end
13  X_train, Y_train ← aggregator.get_training_data(score_function);
14  model.fit(X_train, Y_train);
15  for job in job_queue do
16      tasks ← topological_sort(job);
17      task_queue.add(tasks);
18  end
19  while not task_queue.empty() do
20      tasks ← task_queue.get($w_{decision}$);
21      experience, metrics ← model.schedule(tasks);
22      aggregator.save(metrics);
23      update_sliding_windows($w_{hindsight}$, $w_{decision}$);
24  end
25  model.save();
```

The Algorithm 1 summarizes the KAIROS framework execution. Initially, a set of parameters configure the simulation scenario, detailing the tutor algorithms, sliding windows, job queue, and score function. Tasks are topologically sorted (PEARCE; KELLY, 2007) (line 3), and then scheduled by all tutors (lines 4 – 12). To select which tutor experience would be used as training data for the model (line 13) we used the scoring function derived from (YEN; ABBASLOO; CHAO, 2023). After scoring each tutor for their respective performance for each job, the experiences are selected with the probabilities being directly proportional to their score. This is done to prevent a dominant scheduler from overfitting the model, causing it to merely mimic one of the tutors. Finally, after selecting the tutors' experiences, the model is trained (lines 14) and is ready to be evaluated and reiterated if necessary (lines 15 – 24).

In this study, we investigated three variants of Graph Neural Network (GNN): GNN, GCN, and GAT. For each variant, we repeated the training process, but the only change we made was in the model architecture. This allowed us to compare the performance of the different architectures when exposed to the same training data to ensure a fair comparison between each other.

4.5 CONSIDERATIONS

In this chapter, we have presented a proposal for addressing the scheduling problem in the context of HPC systems using the KAIROS framework. We have discussed how this proposal tackles the challenges associated with HPC scheduling by employing ML-based models introduced in the previous chapter.

To ensure the success and applicability of these models, several considerations needed to be taken into account. Firstly, the quality and quantity of the training data is crucial. It is important that the data accurately represents real-world scheduling scenarios and covers a wide range of situations. The data should be diverse, encompassing different job characteristics and scheduling scenarios. Generalization is also a key aspect to consider. The models should not only perform well on the training data, but also demonstrate good performance on new, unseen data. Since the model mimics the decision-making process of other algorithms, it is essential to have representative data that includes diverse scenarios, allowing the training data to capture a variety of policies being applied. Additionally, the choice of model architecture can have a significant impact on performance.

In this study, we examined and contrasted variants of the GNN to evaluate their efficacy in addressing the problem and their comparative performance. To achieve the goals of this research, a systematic investigation of the performances of different structures was essential, which is detailed in the subsequent chapter.

5 EXPERIMENTAL RESULTS

This chapter is dedicated to presenting the empirical results derived from a comparative analysis of scheduling algorithms. The algorithms developed using the KAIROS framework were compared among themselves and against established tutor algorithms in various scenarios. All ML-based models were trained on each dataset and novel validation data was generated or spliced to ensure rigorous testing of the schedulers on data not yet seen by them.

The benchmark tests encompassed a comparison of the performance metrics of the developed and tutor algorithms. After the test, a statistical analysis was performed to assess the significance of the observed results.

5.1 METRICS

In this section, we introduce and discuss the four chosen metrics: bounded slowdown, utilization percentage, makespan and window steps. Each metric serves a different purpose in evaluating HPC schedulers, capturing different aspects of the system's performance and efficiency (FEITELSON; RUDOLPH, 1998a).

The bounded slowdown metric is specifically selected to represent the perspective of the HPC system user. This metric is closely related to other metrics commonly used in online systems, both open and closed (FEITELSON; RUDOLPH, 1998a). The slowdown metric measures a normalized value that indicates the amount of delay each task experienced, with 1 being no delay and higher values representing an increasing amount of delay. To avoid distortion of values for tasks with a short execution time, a lower bound is introduced. This gives us the bounded slowdown metric. It provides insights into how well the scheduler meets the time requirements of user-submitted jobs.

The utilization percentage metric was chosen to represent the point of view of the HPC system provider. It focuses on evaluating the efficient use of system resources. Utilization percentage is the average percentage of the entire DC resources that are used over a period of time, which in the context of this research is the size of an Hindsight Window.

The makespan metric is a fundamental measure of the efficiency of the scheduler. Makespan represents the time taken to complete the process of a task or a set of tasks. It is particularly suited for evaluating offline systems, where the scheduler has access to all the information on the workload and can make optimal scheduling decisions. This metric provides valuable insights into the scheduler's ability to optimize task completion time.

Finally, the window steps metric is a custom metric designed for the context of this research. It measures the amount of window steps needed to complete all tasks that compose a job, and in this way we have a metric representing at job level what a user of the system would experience as the response time to complete a submitted job.

By selecting and analyzing these four metrics, bounded slowdown, utilization percentage, makespan and window steps, we aim to comprehensively evaluate the performance and efficiency of HPC schedulers, at either the data extraction for the training process or the evaluation of the trained model. Each metric offers unique perspectives from both the user and system owner viewpoints, enabling a holistic assessment of the scheduler's capabilities. Calculation of these metrics involves leveraging available data on job completion times, resource utilization, and specified bounds, providing quantitative measurements for analysis.

Table 6 provides the mathematical expressions for the selected metrics. The formula for bounded slowdown incorporates three variables: w_t, representing the waiting time for a task; p_t, representing the processing time of the task; and τ, representing the lower bound for the processing time of each task. The utilization percentage metric considers two values: $R_{current}$, representing the currently available system resources, and R_{total}, representing the total amount of system resources. The makespan metric solely relies on c_t, which denotes the completion time of each task (i.e., the exact time at which the task finishes its execution process). Lastly, the window steps metric utilizes c_t, representing the completion time of a task; W_α, representing the size of the Hindsight Window; t, representing a task; and J, representing a job.

Metric	**Formula**
Bounded Slowdown	$\max(\frac{w_t+p_t}{\max(p_t,\tau)}, 1)$
Utilization Percentage	$\frac{R_{current}}{R_{total}}$
Makespan	$\max(c_t)$
Window Steps	$\lceil\frac{\max(c_t)-\min(c_t)}{W_\alpha}\rceil, \forall t \in J$

Table 6 – Chosen metrics and their formulas. Source: the author.

5.2 WORKLOADS

The simulation system will have as input data from two distinct workloads. First, a collection of semi-synthetic data derived from the distributions of common scientific workloads will be utilized. These synthetic workloads are designed to capture the characteristics and patterns observed in real scientific computations, ensuring a representative simulation of HPC environments. Secondly, real data from a trace of the Alibaba datacenter will be incorporated into the simulation. This real-world dataset provides insights into the operational dynamics of a large-scale datacenter, offering a comprehensive view of workload variations and resource demands in practice.

5.2.1 Scientific Workloads

The scientific workload data comes from a statistical analysis of common scientific workflows executed in HPC environments (JUVE et al., 2013). The analysis was carried out on six different workflows; all are executed in HPC centers for scientific purposes. The workflows were characterized and then profiled, extracting the probabilistic distributions of their resource demands and execution times, logging the amount of jobs, CPU time, I/O read and write volumes, memory utilization and CPU utilization percentage. This data was compiled, characterized and profiled, resulting in the representative generational parameters to create new jobs that are similar in their distribution of characteristics to the original workloads.

With this workload, it is possible to create large quantities of semi-synthetic data that are still mathematically representative of real-world jobs, which is convenient for this research, since it can provide any number of samples for the training data extraction process. However, it is important to note that although an unlimited amount of data can be generated from these probabilistic parameters, they will always have a bias of the data used to generate them, which will be representative of the workflows, but that may be overfitted by the ML-based model, which would make it very effective in the scenarios of that specific distribution, but will have failed in the generalization of broader workflows. This problem is, however, unlikely to happen since the tutors will be used as a teaching medium to the model and since the tutors do not appeal to any statistical fitting, the model is very unlikely to be able to overfit its parameters to accommodate these distributions.

5.2.2 Alibaba Trace Workloads

The Alibaba trace workloads consist of real data obtained from an 8-day trace of the Alibaba datacenter collected in 2018, collecting information on 4.000 machines, 9.000 services and a batch of 4.000.000 distinct jobs along with their allocation information over these 8 days (GUO et al., 2019). This trace provides information on the actual workload patterns and resource utilization observed in a large-scale production environment.

By incorporating this real-world dataset into the simulation, we can capture the intricacies and complexities of HPC workloads in practice. Alibaba trace workloads offer an opportunity to analyze the characteristics of tasks, their arrival rates, durations, and interdependencies, allowing for a more accurate representation of workload dynamics.

5.3 SIMULATION SCENARIOS

Subsets of the available workloads will be selected and used as scenarios for training and comparisons of the schedulers involved in the simulation. This approach enables a systematic and fair evaluation of the scheduling algorithms involved, since there is a need to compare the performance of the trained models, and the best comparison is with the tutor algorithms, since it will indicate if the model was able to generalize the data and surpass its tutors. Each of the scenarios are described below.

5.3.1 Scientific Workloads Scenario

There will be seven different scenarios for the Scientific Workloads data, each with increasing levels of congestion in the DC. Specifically, the initial scenario will consist of 100 jobs to be scheduled, and the subsequent scenarios will have 200 jobs, 300 jobs, 400 jobs, 500 jobs, 600 jobs, and 1000 jobs. These scenarios were created to examine how each scheduler performs as the number of jobs competing for resources increases. This measures the adaptability levels achieved by the tutor algorithms and, more significantly, by the trained proposed models.

5.3.2 Alibaba Trace Workloads Scenario

For the Alibaba trace workloads, a random 5-day cut will be selected from the 8 available days, which represent a volume of data that is more than sufficient for model training and validation. Although different 5-day cuts will be extracted from the data, the data fed to the tutor algorithms will be the same at each instance of scheduling, since the fair comparison between them is vital for the selection explained in the training data extraction process section.

5.4 METHODOLOGY OF RESULTS ANALYSIS

The core of our methodology involved initializing the KAIROS simulated environment, which was configured with a predefined set of jobs and tasks for each scenario. The environment was uniformly structured to operate on an infrastructure comprising 64 nodes, each provisioned with normalized resources of value 1.0. This standardization was essential to ensure consistency and comparability across different scenarios.

In the next phase, tasks were loaded into the environment and allocated by each scheduler for execution. The primary objective here was to record how each scheduler behaved when faced with data from distinct scenarios but consistent between each scheduler, thereby enabling an accurate assessment of each scheduler's comparative performance. This approach ensured that the efficiency and effectiveness of each scheduling algorithm were evaluated under identical conditions.

The queue of tasks was constructed by the decomposition of each job in order into their tasks. This process was done individually for each job by constructing a graph with the dependencies and then applying a topological sort to its tasks; each task is then decoupled from the job maintaining information about its dependencies so the graph, or a subgraph of it, can be reconstructed at any time. The final queue is then constructed by concatenating all tasks. For each scheduling step, a subset of the queue is provided to the scheduler, the size of this subset is based on the Decision Window size.

After the interaction of the schedulers, both ML-based and tutor algorithms, a dataset of their behavior and performance was generated. This dataset encompassed task-level data, where the makespan and bounded slowdown metrics were recorded for each task. At the job level, the dataset included the number of window steps required to complete all tasks associated with each job, shedding light on the overall job handling efficiency of each scheduler. Additionally, the server-level data comprised the average utilization percentage for each server for every window step, providing a detailed view of resource management and utilization.

To maintain a consistent benchmarking framework, two parameters were set uniformly for all scenarios. The size of the Hindsight Window was set to 400 timesteps, defining the range of timesteps to allocate a batch of tasks until the window slid forward. The Decision Window, limited to 100 tasks, provided information about up to that amount of future tasks when deciding on a specific scheduling.

After completion of the data collection phase, a Cumulative Distribution Function (CDF) was plotted for each metric in each scenario. The CDFs were instrumental in understanding and comparing the distribution of data between different schedulers. Furthermore, to evaluate the statistical significance of the differences observed between pair schedulers, pairwise ANOVA tests were performed. The results of these tests were visually represented through a heatmap matrix, plotting the P-value for each unique pair of different schedulers. This methodological approach allowed for a clear representation of the performance variances among the different scheduling algorithms and if they are statistically significant. This process was carried out individually for each scenario.

5.5 RESULTS

The results of our comparative analysis are systematically organized to facilitate understanding and interpretation. For each scenario, we present the findings through a combination of Cumulative Distribution Function (CDF) plots and pairwise P-value matrices. The CDF plots offer a visual comparison of the performance of the scheduling algorithms in the selected metrics, allowing the identification of trends and outliers. Meanwhile, pairwise P-value matrices provide a statistical basis to assert the statistical significance of differences in performance between algorithms.

Table 7 displays the preferred direction and reasoning for each metric's values. A lower preferred value suggests that a metric performs better when its CDF is skewed towards the left, meaning that the majority of value occurrences are on the lower end of the spectrum. Conversely, a higher preferred value indicates that a metric performs better when its CDF is skewed towards the right, with the majority of value occurrences on the higher end of the spectrum.

The P-value is calculated to assess the statistical significance of differences in performance between scheduling algorithms. This test compares the variances across groups to determine whether the observed variations are larger than what might be expected due to random chance alone. A significant P-value suggests that the algorithms exhibit distinct performance characteristics under the conditions tested.

When interpreting the P-value matrices, each cell represents the outcome of the test between a pair of different schedulers. A value below the significance threshold (here set at 0.05) indicates a statistically significant difference in performance between the two algorithms. This method allows us to confirm which differences observed in the CDF plots are indeed statistically significant, thus reinforcing our conclusions with statistical rigor.

Metric	**Preferred Direction**	**Reasoning**
Bounded Slowdown	Lower	Lower values indicate tasks experience minimal delay, leading to a faster processing time and an improved user experience.
Utilization Percentage	Higher	Higher values signify better resource utilization, indicating that the computational resources in the data center are being used effectively.
Makespan	Lower	Lower makespan values mean that the task is completed faster, which is indicative of a more efficient scheduling algorithm and a faster job processing time.
Window Steps	Lower	Fewer window steps required to complete all tasks signify a faster job turnaround time, enhancing the scheduling algorithm's responsiveness and overall system throughput.

Table 7 – Metric preferences for values.

5.5.1 100 Jobs Scenario

For this scenario, 100 jobs were generated based on the Scientific Workloads data, each job contained a number of tasks determined by a series of distributions based on the analysis of multiple scientific research jobs; the characteristics of each task were also determined by the same values originating from the analysis.

Figure 11a shows for the makespan that floor performance is relatively the same for many of the schedulers, only FCFS has a clearly worse performance compared to the other schedulers, which was expected due to its simplicity. The performances differ towards the ceiling of their performances, around 85% or more, where the schedulers based on the GNN and its variants, GAT and GCN, registered the best performances for this metric for most of the distribution, followed closely by the IPEFT tutor algorithm.

Regarding utilization percentage, Figure 11a shows that the GNN-based schedulers utilized the datacenter's resources more efficiently staying between 64% and 100% by window step. The same trend repeats, where the IPEFT follows close by and the other tutors are less efficient with FCFS demonstrating the worst performance.

Figure 11a also shows that for window steps needed to complete the jobs, the GCN and GAT schedulers had similar performances, being followed close by the GNN-based scheduler and most of the tutors. The results begin to differ more towards the end of the distributions, above 90%, where the FCFS tutor shows to perform worse compared to the usual best performers.

Finally, Figure 11a also illustrates the comparative performance in the bounded slowdown metric for each task. In particular, the best performing scheduler, the GCN, scheduled around 73% of tasks without any slowdown. The second best scheduler, the GAT, shows a better upper-end performance by scheduling all tasks with a slowdown of 1.46 or less; however, it has worse performance at the lower and middle values. These two schedulers are followed by the GNN and then by the tutor algorithms, with FCFS again being comparatively the worst of the algorithms.

For Figure 11b, it shows that for most metrics, there is a statistically significant difference between the tutors and the GNN-based schedulers, with GAT and GCN having no statistically relevant difference between them. A notable exception is the window steps metric, where the only difference that can be considered statistically significant is for FCFS and the schedulers: GAT, GCN, GNN and IPEFT. These results support the conclusions for the metrics, where we can see a greater performance from the schedulers GAT, GCN and GNN, with the two former being slightly more dominant.

5.5.2 1000 Jobs Scenario

Figure 12a presents the results for the 1000 jobs scenario, which represented a significant increase in scale, introduced as a test for scheduling algorithms under high infrastructure congestion. Consistent with the patterns identified in all other scenarios, the comparative performance of the schedulers maintained a consistent trajectory, only this time further cementing the GAT and GCN as dominant algorithms with similar performance. The CDF plots for this scenario revealed a punished slowdown curve, with only 20% of the tasks having a bounded slowdown of 1.72 or less for the best performing scheduler (*i.e.*, GCN), indicating the severe congestion introduced in the infrastructure. This was again reflected in the increase in the overall makespan for tasks, a continuation of the trend seen in the previous scenarios.

In this scenario, the performance gap between less efficient and more efficient schedulers reached its peak. As demonstrated in Figure 12b, there is a significant difference between the metrics for every pair of schedulers for the makespan and bounded slowdown metrics. The server utilization showed a difference between all pairs, except for the GNN-based schedulers trained under the KAIROS framework, which achieved performances similar enough so that their differences cannot be considered statistically relevant. Finally, the window steps metric showed more disparities than ever, with most tutors having a statistically relevant difference when compared to the trained models.

All other scenarios based on the Scientific Workloads data showed a similar trend as the congestion in the system increased, their results discussion can be found at Appendix A.

5.5.3 Alibaba Trace Scenario

For this scenario, a slice of the Alibaba trace was randomly selected to serve as input for the jobs to be scheduled in the same architecture, only normalizing the requested resources to prevent an invalid scheduling or situations of deadlock where a task will never be able to be allocated.

Figures 13a and 13b showcase the CDFs and heatmap matrices for all metrics that originated from the execution of the schedulers on a slice of the Alibaba trace. As there is a considerable jump in the number of jobs and even task density and length, new behavior is apparent when compared to the Scientific Workload scenarios. For the makespan and window step metrics, a similar behavior is observed, just on a larger scale, as the worst scheduler (*i.e.*, FCFS) attains a task makespan in the worst-case scenario of 2.42×10^7. Regarding the utilization percentage metric, the comparative performance follows the same patterns, with the GAT and GCN schedulers better utilizing the available resources; however, there is a shift when analyzing the performances individually, as we see a larger gap between lower end performances and a smaller gap between higher end performances, indicating that overall fragmentation in the datacenter has increased. However, in the bounded slowdown metric, there is still a clear performance gap between the three GNN-based schedulers (*i.e.*, GNN, GCN and GAT) and its counterparts.

The same statistical pattern seen in previous scenarios is repeated here, as demonstrated in Figure 13b, the pair testing reveals that there are statistically significant differences between the GNN-based schedulers and their tutors in almost all metrics, again pointing to a dominant performance by the schedulers proposed in this work.

5.5.4 Statistical analysis

Comprehensive analysis across all scenarios, supported by pairwise P-value results, confirms significant performance disparities between the scheduling algorithms. The GNN-based algorithms, particularly GCN and GAT, demonstrated superior performance with statistically significant differences compared to traditional and other ML-based schedulers. These findings underscore the effectiveness of GNN-based models when compared to its tutors, which confirms the GNN architecture as capable of generalizing the scheduling of HPC jobs process under the KAIROS algorithm.

As the scale of the scenarios grew and the infrastructure became more congested, the differences between schedulers became increasingly statistically significant. This was evident from the P-value matrices, which showed that the scale of congestion accentuated the limitations of simpler algorithms. The worse algorithms were observed to be more affected by congestion, as their comparative performance continued to deteriorate.

Furthermore, for scenarios at the higher end of the congestion, even some GNN schedulers started to be affected differently. Although their comparative performance remained similar, the statistical significance of differences between their performances increased. This suggests that as congestion increased, the impact on the performance of GNN schedulers became more pronounced, although not as drastic as in the case of simpler algorithms.

5.5.5 Discussion

This section has provided a comprehensive evaluation of various scheduling algorithms under different job load scenarios, revealing trends and performance differentiations. A critical observation across all scenarios is the escalation in makespan and bounded slowdown as job density increases, which is to be expected. However, the relative performance of the scheduling algorithms, particularly the GNN-based models (GCN, GAT, and GNN), remains remarkably consistent as the job density increases. This points to a successful generalization by the ML-based algorithms which based solely on the interaction of its tutor algorithms with an environment were able to surpass these same algorithms, amalgamating their knowledge to reach further heights.

The GCN and GAT algorithms consistently demonstrate superior performance, dominating across most metrics. Their ability to efficiently handle increased job loads while maintaining lower makespan and slowdown rates is of particularly interest. The GNN model also performs commendably, trailing closely behind its counterparts. This consistent outperformance of the GNN-based models highlights the efficacy of these variants of ML techniques to leaning complex scheduling behaviors.

As the number of jobs escalates, thus increasing the congestion in the datacenter, the gap between the lower-performing schedulers, such as FCFS, and their more advanced peers widens significantly. This divergence underscores the limitations of simpler scheduling algorithms in handling high-density job environments and highlights the potential of the GNN-based models to be applied and thrive in complex scenarios.

Furthermore, statistical analysis reveals no substantial reason to prefer GAT over GCN in these scenarios. Although their performance metrics are closely matched, GAT incurs a higher computational cost in both training and task scheduling. Table 8 shows the average and standard deviation values for the time taken and memory consumed by each model to schedule a batch of tasks with the length of a Decision Window. The data for each model was taken from the execution of the 1000 jobs scenario. This finding suggests that GCN may offer a more efficient balance between performance and computational overhead, making it a preferable choice in high-load scheduling contexts.

Model	**Time (s)**	**Memory Usage (MiB)**
GNN	67.9 ± 1.2	$4,375.8 \pm 230.8$
GCN	103.1 ± 2.8	$4,687.5 \pm 509.6$
GAT	283.6 ± 6.2	$8,125.0 \pm 767.1$

Table 8 – Model average execution time and memory usage per batch of Decision Window.

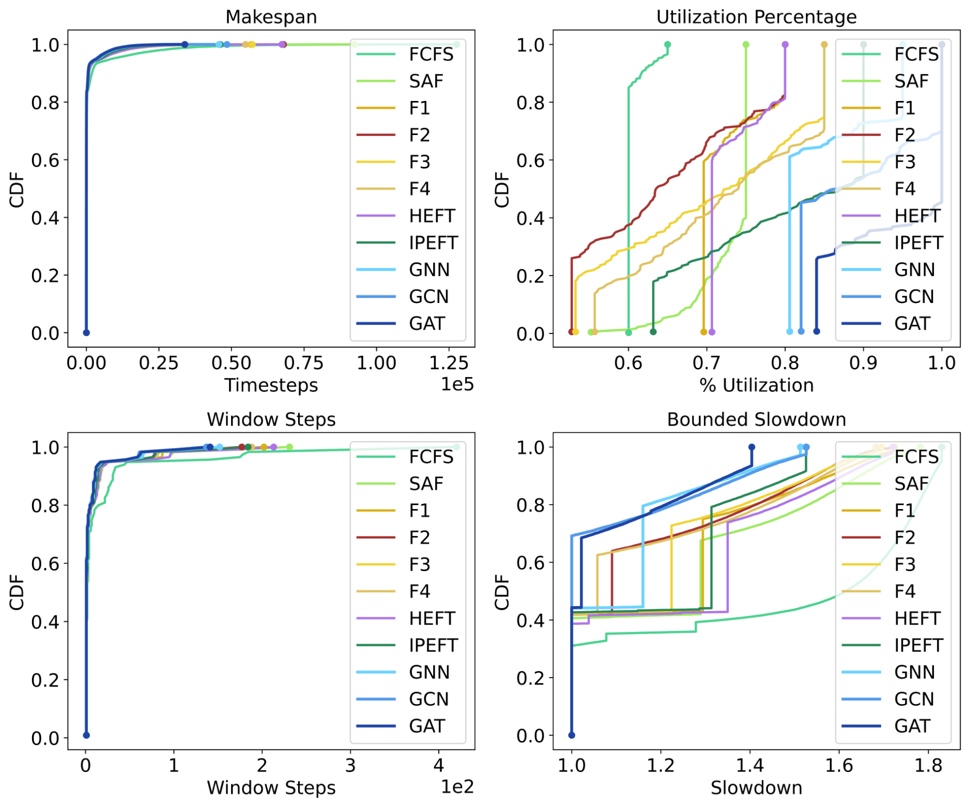
Table 9 presents a summary of the leading performers for each metric in different scenarios. The prevalence of the GNN variants is evident, as GAT and GCN consistently outperform other schedulers in all metrics across all scenarios. It is worth mentioning that although the table indicates the highest performer based on numerical comparison, statistical analysis suggests that in most scenarios, the performance of all GNN-based schedulers is similar, with a notable difference from their counterparts, particularly in high congestion scenarios.

In summary, the results of this comparative analysis underscore the significant advantages

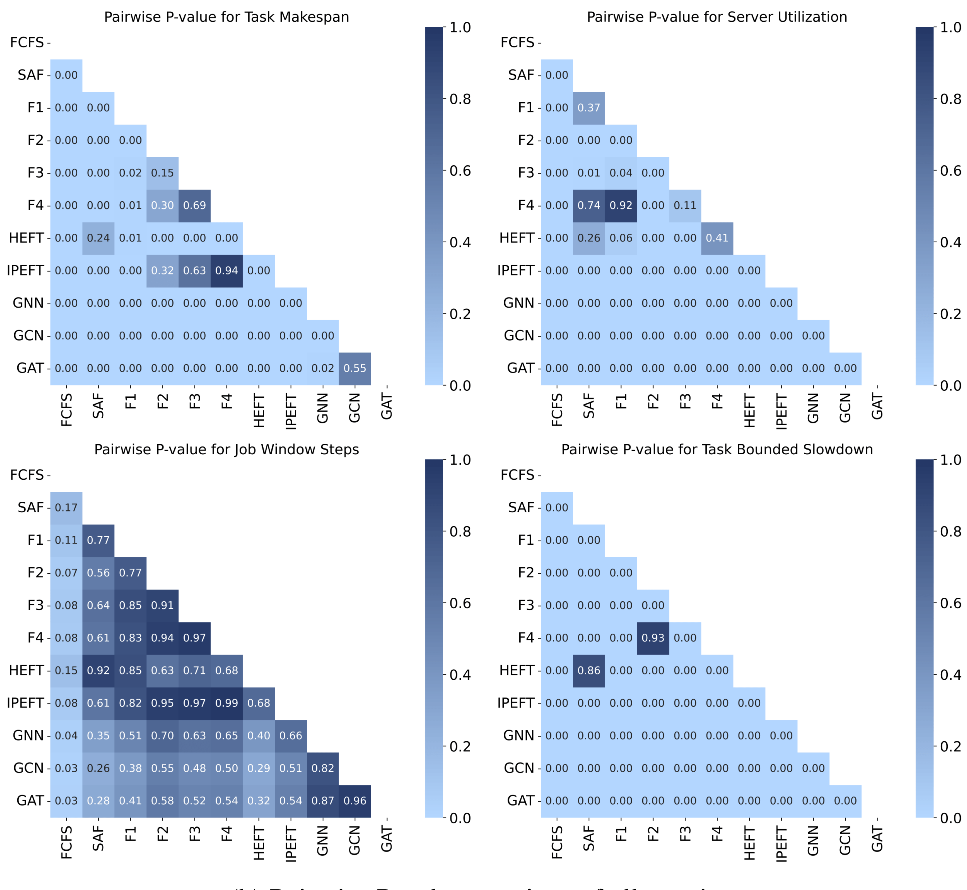
Scenario	**Makespan**	**Utilization Percentage**	**Window Steps**	**Bounded Slowdown**
100 Jobs	GAT	GAT	GCN	GAT
200 Jobs	GCN	GAT	GCN	GCN
300 Jobs	GAT	GAT	GAT	GAT
400 Jobs	GAT	GAT	GCN	GAT
500 Jobs	GAT	GAT	GAT	GCN
600 Jobs	GAT	GAT	GCN	GCN
1000 Jobs	GAT	GAT	GCN	GCN
Alibaba	GAT	GAT	GCN	GCN

Table 9 – Top performing schedulers for each metric and scenario.

of employing advanced GNN-based scheduling algorithms, particularly the variants GCN and GAT, to manage complex high-volume job scenarios within datacenters.

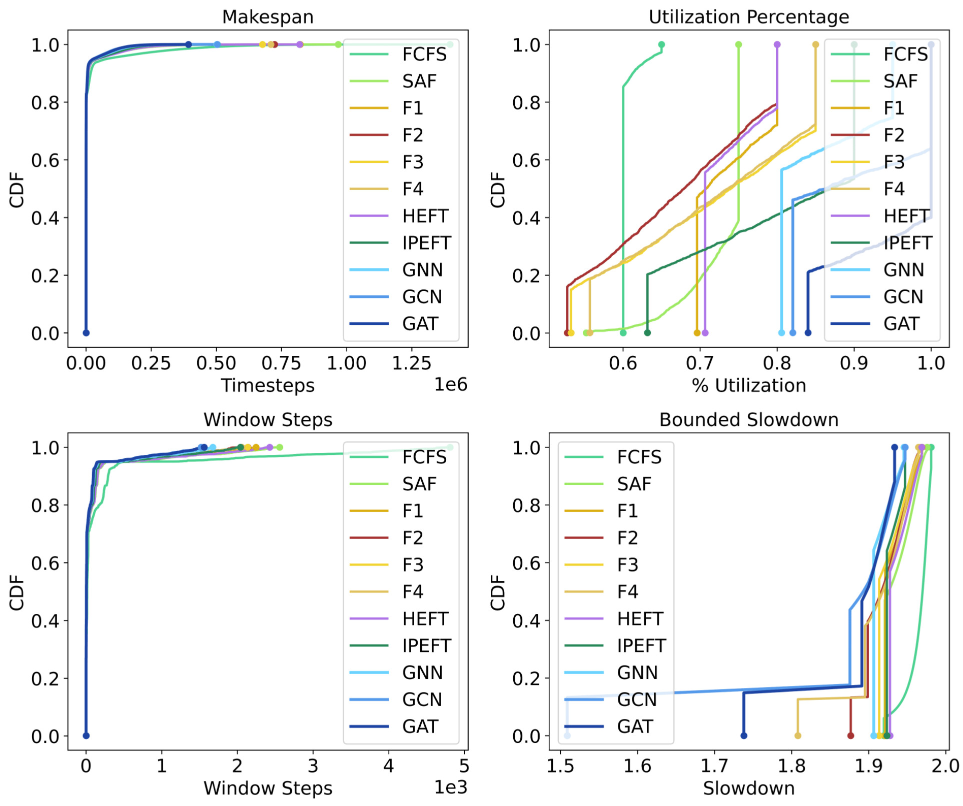


(a) Cumulative distribution functions of all metrics.

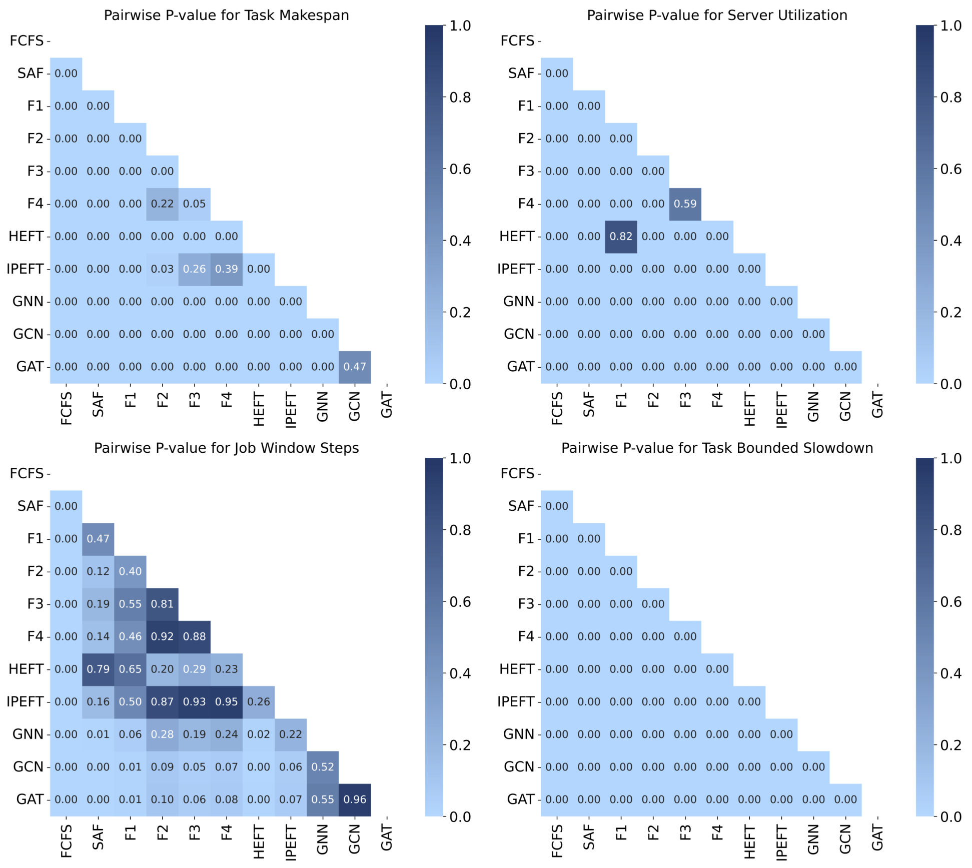


(b) Pairwise P-value matrices of all metrics.

Figure 11 – Results for the 100 jobs scenario.

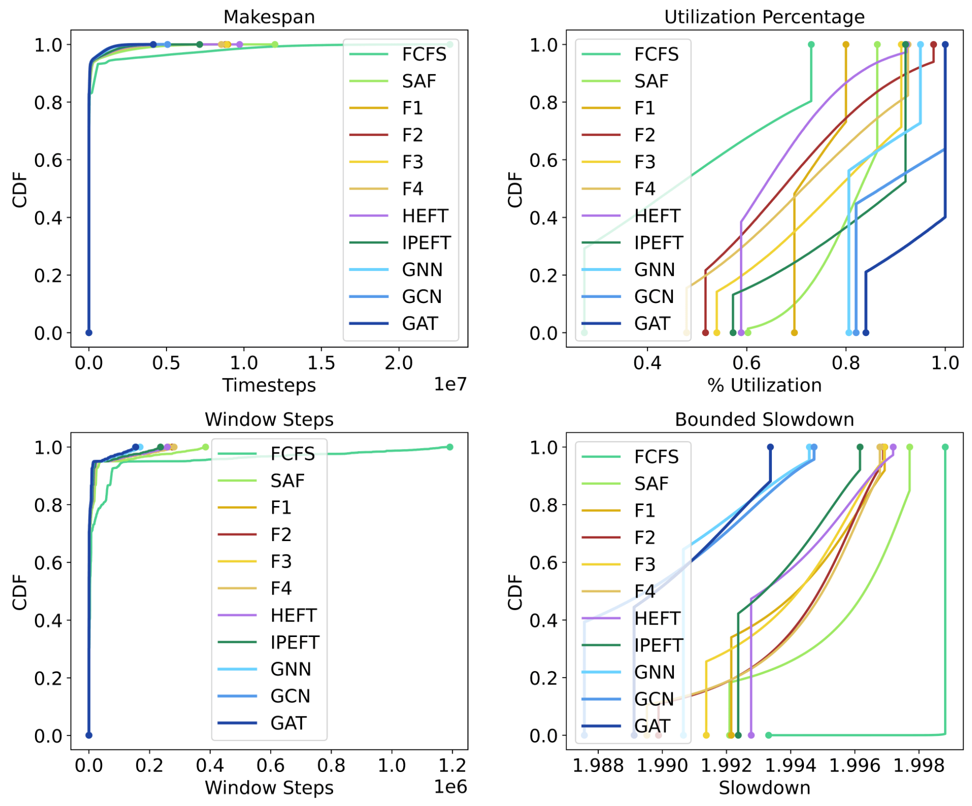


(a) Cumulative distribution functions of all metrics.

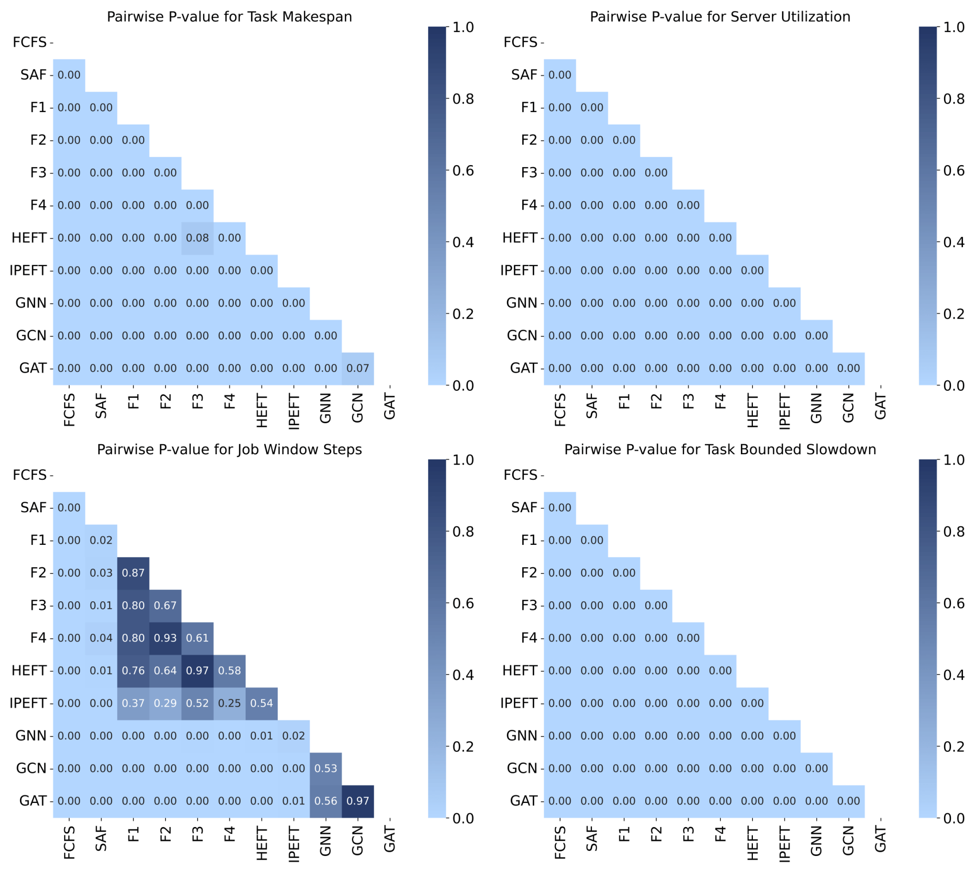


(b) Pairwise P-value matrices of all metrics.

Figure 12 – Results for the 1000 jobs scenario.



(a) Cumulative distribution functions of all metrics.



(b) Pairwise P-value matrices of all metrics.

Figure 13 – Results for the Alibaba trace scenario.

5.6 CONCLUSION

The results presented in this chapter have culminated in key findings regarding the performance of various scheduling algorithms under different job load scenarios. The study distinctly illustrates the effectiveness of GNN-based scheduling algorithms, particularly GCN and GAT, trained under the KAIROS framework in adapting to differently congested environments. These algorithms have consistently outperformed traditional scheduling methods by demonstrating lower makespan and bounded slowdown metrics while maintaining and efficient usage of resources, which can be beneficial to both users and administrators of datacenters.

One conclusion of note was the comparison between the performances of the GNN-based model variants, which are the GNN, GAT and GCN, concluding that both GAT and GCN have statiscally better performance than GNN, however with no significant statistical difference between the performances of GAT and GCN. Given the higher computational demands of GAT, the GCN algorithm emerges as a more resource-efficient option, balancing high performance with lower computational overhead.

This work advances our understanding of GNN-based scheduling algorithms trained under Supervised Learning on the experience of weaker algorithms. Also, it presents a novel comparison between three GNN variants and their relative performance when applied to the job scheduling problem under the HPC context.

6 CONCLUSIONS

This final chapter aims to encapsulate the essence of this dissertation by revisiting the primary objectives, discussing key findings, and reflecting on their implications. The structure of this chapter is laid out to first summarize the research questions and the answers derived, delve into the contributions made to the field of GNN-based scheduling in the context of HPC, identify the limitations of this study, propose directions for future research, and finally, conclude with a comprehensive summary of the entire work. This systematic approach ensures a coherent and thorough reflection on the study's outcomes and their broader impact.

6.1 KEY FINDINGS AND RESEARCH QUESTIONS

The investigation embarked upon in this dissertation was driven by a set of research questions, focusing on the efficiency and potential of GNN variants in the realm of HPC scheduling. The study compared various GNN models, revealing insights into their performance and applicability. Our motivation stemmed from the need to explore novel approaches that could potentially improve upon existing models and techniques.

6.1.1 GNN Variants in HPC Scheduling

Our comparative analysis of different GNN variants, including GCN and GAT, demonstrated their varying efficiencies in job scheduling within the HPC context. The study found that specific GNN models, notably GCN, exhibited superior performance in managing complex scheduling tasks, thereby addressing our first specific objective.

The performance analysis revealed that for this specific context there's no statistically significant gain that justifies the usage of a GAT model, since it requires more computational power to train and execute due to the usage of its attention mechanisms. Therefore, as far as this work has explored the models, the GCN has proved itself to be more efficient when applied to the scheduling problem in the HPC context.

6.1.2 Behavior of GNN Models

The second specific objective involved investigating the adaptability of these models to changes in target metrics. The research revealed that GNN models respond dynamically to varying metrics, highlighting their potential for flexible application in diverse scheduling scenarios.

6.1.3 Prototype Development and Evaluation

In response to the third and fourth objectives, the development and subsequent evaluation of a prototype GNN-based scheduler not only demonstrated the viability of these models but also provided a concrete basis for their assessment against conventional scheduling algorithms.

With this objective in mind, we enhanced the KAIROS framework specifically for this research by introducing sliding windows that can adapt to different scenarios and effectively leverage the power of GNNs. In addition, the introduction of sliding windows in the KAIROS framework allowed us to capture dynamic temporal dependencies and effectively model the complex relationships inherent in job scheduling through the use of graph structures while being compatible with Neural Networks constraints.

6.1.4 Performance Improvement Based on Metrics

Finally, the study's evaluation of the scheduler's performance against a set of metrics, as per the fifth objective, confirmed the hypothesis that GNN-based schedulers could indeed enhance performance, particularly in high-congestion scenarios. More specifically, by learning only from the interaction of simpler algorithms all GNN variants showed the potential to surpass their tutors by generalizing and amalgamating their scheduling behavior.

6.2 CONTRIBUTION TO THE FIELD

The contributions of this dissertation to the field of HPC scheduling are multi-faceted. By comparing advanced geometric machine learning techniques, specifically GNN-based algorithms, into the scheduling process, the study has introduced new key conclusions to the field of GNN-based job scheduling in HPC environments, which is an intersection of fields that still lacks research when compare to other intersections of scheduling and ML techniques.

Furthermore, this work contributes with the preexisting KAIROS framework, by extending its concepts and functionalities, specially with the introduction of sliding windows, graph and subgraph decomposition support and exploration over exploitation approach to experience selection.

The application of Supervised Learning using the experience of weaker algorithms to train more advanced GNN models represents an underexplored approach in this domain. This strategy has proved itself viable in generating effective ML-based schedulers.

Furthermore, the comprehensive comparison of different GNN variants offers a valuable reference for future developments in the field. By elucidating the relative strengths and weaknesses of each model, this research aids in making informed decisions about the selection of appropriate algorithms for specific HPC scheduling needs.

6.3 STUDY LIMITATIONS

While this study has achieved its objectives, it is essential to acknowledge its limitations. These limitations primarily stem from the experimental setup and the scope of the study.

The reliance on the KAIROS framework, though beneficial for a controlled analysis, might not entirely replicate the complexity and unpredictability of real-world HPC environments.

This dependency could affect the generalizability and applicability of the findings to other HPC contexts.

Additionally, the exploration of how different parameters, such as window sizes and tutor algorithms, influence the learning and ultimate performance of the GNN-based schedulers was limited. A more exhaustive investigation into these aspects could provide a more nuanced understanding of the schedulers' adaptability and efficiency.

6.4 RECOMMENDATIONS FOR FUTURE RESEARCH

Some recommendations for future research are proposed. These suggestions aim to extend the understanding and applicability of GNN-based scheduling in HPC environments.

Future studies should consider examining a broader spectrum of GNN models. This exploration could uncover new insights into the capabilities and limitations of various GNN approaches in different scheduling scenarios.

Additionally, incorporating a variety of tutor algorithms and learning methodologies could offer a more comprehensive perspective on the training and performance of GNN-based schedulers. This would also help in understanding the impact of different training paradigms on the efficacy of these models.

Finally, applying the findings of this research to actual HPC environments would enhance their practical value. Conducting field tests and experiments in real-world settings could validate and refine the proposed models, making them more robust and applicable.

6.5 CLOSING SUMMARY

This study aimed to investigate the applicability of GNNs as job scheduler algorithms in the HPC environment. For this we expanded the preexisting KAIROS framework to explore a Supervised Learning approach for this problem. Throughout our study, we proposed a methodology for experiments to evaluate the performance of GNN-based job scheduling models against classical scheduling algorithms from the literature. We defined specific metrics and scenarios to assess the effectiveness of the models accurately.

After acquiring relevant data and training the GNN-based models, we conducted a series of statistical analysis to draw conclusions from the comparison of all trained models among themselves and against their tutors. This analysis indicated a superior performance from the GNN-based solutions, specially the models using GAT andd GCN techniques.

In conclusion, this work is a step forward in the application of GNN-based algorithms for HPC scheduling. These findings show the potential of these advanced models in enhancing scheduler performance. As HPC systems continue to evolve and grow in complexity, the insights and methodologies developed in this may contribute to the design of more efficient, adaptable, and intelligent scheduling solutions.

BIBLIOGRAPHY

ALBERT, Réka; BARABÁSI, Albert-László. Statistical mechanics of complex networks. **Reviews of modern physics**, APS, v. 74, n. 1, p. 47, 2002. Citado na página 33.

ALBUQUERQUE, Paulo Roberto. **Expanding Kairos Scheduler with communication requirements processing**. 55 f. Monografia (Trabalhos de Conclusão de Curso (Graduação)) — Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Joinville, 2022. Citado na página 40.

AMARAL, Marcelo et al. Topology-aware gpu scheduling for learning workloads in cloud environments. In: **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**. New York, NY, USA: Association for Computing Machinery, 2017. (SC '17). ISBN 9781450351140. Disponível em: <https://doi.org/10.1145/3126908.3126933>. Citado na página 15.

BRUCKER, Peter. Scheduling algorithms. **Journal-Operational Research Society**, Springer, v. 50, p. 774–774, 1999. Citado 4 vezes nas páginas 14, 22, 45, and 46.

CARASTAN-SANTOS, Danilo; CAMARGO, Raphael Y De. Obtaining dynamic scheduling policies with simulation and machine learning. In: **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**. [S.l.: s.n.], 2017. p. 1–13. Citado na página 45.

CASANOVA, Henri et al. Developing accurate and scalable simulators of production workflow management systems with wrench. **Future Generation Computer Systems**, v. 112, p. 162–175, 2020. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X19317431>. Citado na página 15.

CASANOVA, Henri et al. On the feasibility of simulation-driven portfolio scheduling for cyberinfrastructure runtime systems. In: **Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)**. [S.l.: s.n.], 2023. p. 3–24. Citado na página 22.

DING, Xu et al. An energy harvesting roadside unit communication load prediction and energy scheduling based on graph convolutional neural networks for spatial-temporal vehicle data. **IET Signal Processing**, v. 16, n. 8, p. 909–924, 2022. Disponível em: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/sil2.12149>. Citado 2 vezes nas páginas 31 and 35.

ERDŐS, Paul; RÉNYI, Alfréd et al. On the evolution of random graphs. **Publ. Math. Inst. Hung. Acad. Sci**, v. 5, n. 1, p. 17–60, 1960. Citado na página 33.

FAN, Yuping et al. Deep reinforcement agent for scheduling in hpc. In: IEEE. **2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)**. [S.l.], 2021. p. 807–816. Citado na página 27.

FEITELSON, Dror G; RUDOLPH, Larry. Metrics and benchmarking for parallel job scheduling. In: SPRINGER. **Job Scheduling Strategies for Parallel Processing: IPPS/SPDP'98 Workshop Orlando, Florida, USA, March 30, 1998 Proceedings 4**. [S.l.], 1998. p. 1–24. Citado 5 vezes nas páginas 9, 15, 23, 24, and 51.

FEITELSON, Dror G.; RUDOLPH, Larry. Metrics and benchmarking for parallel job scheduling. In: FEITELSON, Dror G.; RUDOLPH, Larry (Ed.). **Job Scheduling Strategies for Parallel Processing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. p. 1–24. ISBN 978-3-540-68536-4. Citado na página 23.

GALLET, Matthieu; MARCHAL, Loris; VIVIEN, Frédéric. Efficient scheduling of task graph collections on heterogeneous resources. In: IEEE. **2009 IEEE International Symposium on Parallel & Distributed Processing**. [S.l.], 2009. p. 1–11. Citado 4 vezes nas páginas 31, 32, 35, and 46.

GOMBOLAY, Matthew C; WILCOX, Ronald J; SHAH, Julie A. Fast scheduling of robot teams performing tasks with temporospatial constraints. **IEEE Transactions on Robotics**, IEEE, v. 34, n. 1, p. 220–239, 2018. Citado na página 33.

GRINSZTAJN, Nathan et al. Geometric deep reinforcement learning for dynamic dag scheduling. **2020 IEEE Symposium Series on Computational Intelligence (SSCI)**, p. 258–265, 2020. Citado na página 31.

GUO, Jing et al. Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces. In: **2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)**. [S.l.: s.n.], 2019. p. 1–10. Citado na página 53.

HAMILTON, William L.; YING, Rex; LESKOVEC, Jure. Inductive representation learning on large graphs. In: **NIPS**. [S.l.: s.n.], 2017. Citado na página 29.

HASSAN, Mohammad Mehedi et al. Efficient resource scheduling for big data processing in cloud platform. In: FORTINO, Giancarlo et al. (Ed.). **Internet and Distributed Computing Systems**. Cham: Springer International Publishing, 2014. p. 51–63. ISBN 978-3-319-11692-1. Citado na página 15.

HELLERMAN, Herbert. Some principles of time-sharing scheduler strategies. **IBM Systems Journal**, IBM, v. 8, n. 2, p. 94–117, 1969. Citado na página 33.

HOOGEVEEN, Johannes Adzer; LENSTRA, Jan Karel; VELTMAN, Bart. Preemptive scheduling in a two-stage multiprocessor flow shop is np-hard. **European Journal of Operational Research**, Elsevier, v. 89, n. 1, p. 172–175, 1996. Citado 2 vezes nas páginas 14 and 22.

HU, Liang et al. Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. **Journal of Manufacturing Systems**, Elsevier, v. 55, p. 1–14, 2020. Citado 2 vezes nas páginas 31 and 33.

JING, Xuan et al. Multi-agent reinforcement learning based on graph convolutional network for flexible job shop scheduling. **Journal of Intelligent Manufacturing**, Springer, p. 1–19, 2022. Citado 2 vezes nas páginas 31 and 35.

JOO, Changhee; SHROFF, Ness B. Local greedy approximation for scheduling in multihop wireless networks. **IEEE Transactions on Mobile Computing**, IEEE, v. 11, n. 3, p. 414–426, 2011. Citado na página 34.

JUVE, Gideon et al. Characterizing and profiling scientific workflows. **Future Generation Computer Systems**, v. 29, n. 3, p. 682–692, 2013. ISSN 0167-739X. Special Section:

Recent Developments in High Performance Computing and Security. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X12001732>. Citado na página 52.

KIAMARI, Mehrdad; KRISHNAMACHARI, Bhaskar. Gcnscheduler: Scheduling distributed computing applications using graph convolutional networks. **arXiv preprint arXiv:2110.11552**, 2021. Citado na página 31.

KIM, S J. General approach to multiprocessor scheduling. **Department of Computer Sciences of the University of Texas at Austin**, 1 1988. Disponível em: <https://www.osti.gov/biblio/5501704>. Citado na página 35.

KIPF, Thomas N; WELLING, Max. Semi-supervised classification with graph convolutional networks. **arXiv preprint arXiv:1609.02907**, 2016. Citado na página 28.

KOCOT, B; CZARNUL, P; PROFICZ, J. Energy-aware scheduling for high-performance computing systems: A survey. **Energies**, v. 16, p. 890, 2023. Citado na página 14.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015. Citado 4 vezes nas páginas 15, 27, 36, and 38.

LEI, Kun et al. A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem. **Expert Systems with Applications**, Elsevier, p. 117–135, 2022. Citado 2 vezes nas páginas 31 and 34.

LIN, Ziniu et al. A scheduling algorithm based on reinforcement learning for heterogeneous environments. **Applied Soft Computing**, Elsevier, v. 130, p. 109–122, 2022. Citado 2 vezes nas páginas 31 and 32.

LIU, Zeyi et al. A graph neural networks-based deep q-learning approach for job shop scheduling problems in traffic management. **Information Sciences**, v. 607, p. 1211–1223, 2022. ISSN 0020-0255. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0020025522005990>. Citado 2 vezes nas páginas 31 and 34.

LUO, JianChao et al. Deadlock-free scheduling of automated manufacturing systems using petri nets and hybrid heuristic search. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, IEEE, v. 45, n. 3, p. 530–541, 2014. Citado na página 33.

MAHESH, Batta. Machine learning algorithms-a review. **International Journal of Science and Research (IJSR).[Internet]**, v. 9, p. 381–386, 2020. Citado 2 vezes nas páginas 25 and 26.

MICHELI, Alessio. Neural network for graphs: A contextual constructive approach. **IEEE Transactions on Neural Networks**, IEEE, v. 20, n. 3, p. 498–511, 2009. Citado 2 vezes nas páginas 15 and 27.

NGUYEN, Hoa T.; USMAN, Muhammad; BUYYA, Rajkumar. **iQuantum: A Case for Modeling and Simulation of Quantum Computing Environments**. 2023. Citado na página 15.

NI, Fei et al. A multi-graph attributed reinforcement learning based optimization algorithm for large-scale hybrid flow shop scheduling problem. In: **Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining**. [S.l.: s.n.], 2021. p. 3441–3451. Citado 2 vezes nas páginas 31 and 34.

PARK, Junyoung et al. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. **International Journal of Production Research**, v. 59, p. 1–18, 01 2021. Citado 2 vezes nas páginas 31 and 34.

PASCHALIDIS, Ioannis Ch; HUANG, Fuzhuo; LAI, Wei. A message-passing algorithm for wireless network scheduling. **IEEE/ACM Transactions on Networking**, IEEE, v. 23, n. 5, p. 1528–1541, 2014. Citado na página 34.

PEARCE, David J; KELLY, Paul HJ. A dynamic topological sort algorithm for directed acyclic graphs. **Journal of Experimental Algorithmics (JEA)**, ACM New York, NY, USA, v. 11, p. 1–7, 2007. Citado 2 vezes nas páginas 41 and 50.

PENG, Haosong et al. Lore: a learning-based approach for workflow scheduling in clouds. In: **Proceedings of the Conference on Research in Adaptive and Convergent Systems**. [S.l.: s.n.], 2022. p. 47–52. Citado 3 vezes nas páginas 31, 32, and 34.

PEREIRA, Kleiton; KOSLOVSKI, Guilherme Piêgas. **Escalonamento de tarefas comunicantes guiado por Deep Learning**. 54 f. Monografia (Trabalhos de Conclusão de Curso (Graduação)) — Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Joinville, 2020. Citado 2 vezes nas páginas 16 and 40.

RODRIGUEZ, Maria A.; BUYYA, Rajkumar. Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. **Future Generation Computer Systems**, v. 79, p. 739–750, 2018. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X17301681>. Citado na página 14.

RUIZ, Rubén; STÜTZLE, Thomas. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. **European journal of operational research**, Elsevier, v. 177, n. 3, p. 2033–2049, 2007. Citado na página 34.

SCARSELLI, Franco et al. The graph neural network model. **IEEE transactions on neural networks**, IEEE, v. 20, n. 1, p. 61–80, 2008. Citado 2 vezes nas páginas 15 and 27.

SHAH, Syed Nasir Mehmood; MAHMOOD, Ahmad Kamil Bin; OXLEY, Alan. Modified least cost method for grid resource allocation. In: **2010 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery**. [S.l.: s.n.], 2010. p. 218–225. Citado na página 22.

SHIRVANI, Mirsaeid Hosseini. A hybrid meta-heuristic algorithm for scientific workflow scheduling in heterogeneous distributed computing systems. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 90, p. 103501, 2020. Citado na página 32.

SONG, Yutao et al. A reinforcement learning based job scheduling algorithm for heterogeneous computing environment. **Computers and Electrical Engineering**, v. 107, p. 108653, 2023. ISSN 0045-7906. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045790623000782>. Citado 2 vezes nas páginas 31 and 33.

SULAIMAN, Muhammad et al. An evolutionary computing-based efficient hybrid task scheduling approach for heterogeneous computing environment. **Journal of Grid Computing**, Springer, v. 19, n. 1, p. 1–31, 2021. Citado na página 32.

TOPCUOGLU, Haluk; HARIRI, Salim; WU, Min-You. Performance-effective and low-complexity task scheduling for heterogeneous computing. **IEEE transactions on parallel and distributed systems**, IEEE, v. 13, n. 3, p. 260–274, 2002. Citado na página 35.

VASWANI, Ashish et al. Attention is all you need. In: **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2017. v. 30. Disponível em: <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>. Citado 2 vezes nas páginas 32 and 38.

VELIČKOVIĆ, Petar et al. Graph attention networks. **arXiv preprint arXiv:1710.10903**, 2017. Citado 3 vezes nas páginas 29, 32, and 37.

WANG, Xiaohan et al. Dynamic scheduling of tasks in cloud manufacturing with multi-agent reinforcement learning. **Journal of Manufacturing Systems**, Elsevier, v. 65, p. 130–145, 2022. Citado 2 vezes nas páginas 31 and 32.

WANG, Zheyuan; GOMBOLAY, Matthew. Learning scheduling policies for multi-robot coordination with graph attention networks. **IEEE Robotics and Automation Letters**, IEEE, v. 5, n. 3, p. 4509–4516, 2020. Citado 2 vezes nas páginas 31 and 33.

WANG, Zheyuan; GOMBOLAY, Matthew. Stochastic resource optimization over heterogeneous graph neural networks for failure-predictive maintenance scheduling. **Proceedings of the International Conference on Automated Planning and Scheduling**, v. 32, n. 1, p. 527–536, Jun. 2022. Disponível em: <https://ojs.aaai.org/index.php/ICAPS/article/view/19839>. Citado 2 vezes nas páginas 31 and 35.

WEST, Douglas Brent et al. **Introduction to graph theory**. [S.l.]: Prentice hall Upper Saddle River, 2001. v. 2. Citado na página 22.

WOLPERT, David H. Stacked generalization. **Neural networks**, Elsevier, v. 5, n. 2, p. 241–259, 1992. Citado na página 40.

WU, Z et al. A comprehensive survey on graph neural networks. **IEEE Transactions on Neural Networks and Learning Systems**, Institute of Electrical and Electronics Engineers, 2020. Citado na página 29.

XING, Qiang et al. Real-time optimal scheduling for active distribution networks: A graph reinforcement learning method. **International Journal of Electrical Power and Energy Systems**, v. 145, p. 108637, 2023. ISSN 0142-0615. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0142061522006330>. Citado 2 vezes nas páginas 31 and 35.

YEN, Chen-Yu; ABBASLOO, Soheil; CHAO, H. Jonathan. Computers can learn from the heuristic designs and master internet congestion control. In: **Proceedings of the ACM SIGCOMM 2023 Conference**. NY, USA: Association for Computing Machinery, 2023. (ACM SIGCOMM ’23), p. 255–274. ISBN 9798400702365. Citado 2 vezes nas páginas 41 and 50.

YOU, Dan; WANG, Shouguang; ZHOU, Mengchu. Synthesis of monitor-based liveness-enforcing supervisors for s^3 pr with xi-resources. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, IEEE, v. 45, n. 6, p. 967–975, 2015. Citado na página 33.

ZARANDI, Mohammad Hossein Fazel et al. A state of the art review of intelligent scheduling. **Artificial Intelligence Review**, Springer, v. 53, p. 501–593, 2020. Citado na página 25.

ZHANG, Si et al. Graph convolutional networks: a comprehensive review. **Computational Social Networks**, SpringerOpen, v. 6, n. 1, p. 1–23, 2019. Citado na página 29.

ZHANG, Zhijun et al. Dag scheduling with communication delays based on graph convolutional neural network. **Wirel. Commun. Mob. Comput.**, John Wiley and Sons Ltd., GBR, v. 22, jan 2022. ISSN 1530-8669. Disponível em: <https://doi.org/10.1155/2022/9013361>. Citado 2 vezes nas páginas 31 and 34.

ZHAO, Zhongyuan et al. Distributed scheduling using graph neural networks. In: IEEE. **IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. [S.l.], 2021. p. 4720–4724. Citado 2 vezes nas páginas 31 and 33.

ZHOU, Jie et al. Graph neural networks: A review of methods and applications. **AI open**, Elsevier, v. 1, p. 57–81, 2020. Citado na página 16.

ZHOU, Naqin et al. A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table. **Concurrency and Computation: Practice and Experience**, v. 29, n. 5, p. e3944, 2017. E3944 CPE-15-0408.R2. Citado na página 46.

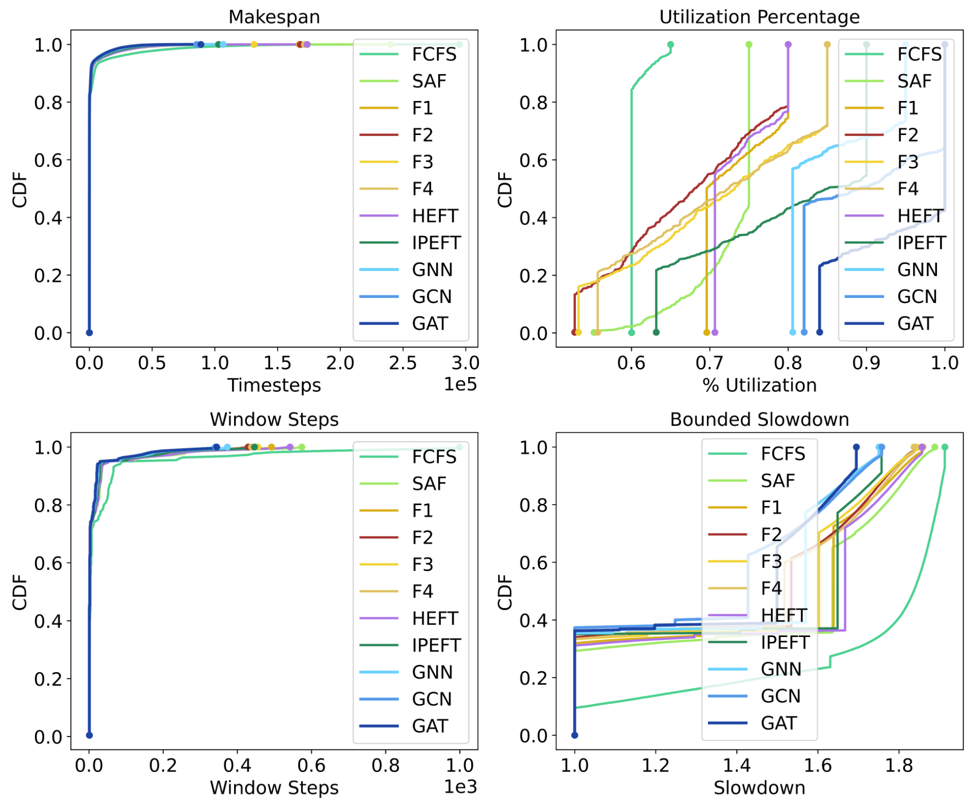
ZHOU, Yunfan et al. Learning to optimize dag scheduling in heterogeneous environment. In: IEEE. **23rd IEEE International Conference on Mobile Data Management (MDM)**. [S.l.], 2022. p. 137–146. Citado 2 vezes nas páginas 31 and 32.

APPENDIX A – ADDITIONAL SCIENTIFIC WORKLOAD RESULTS

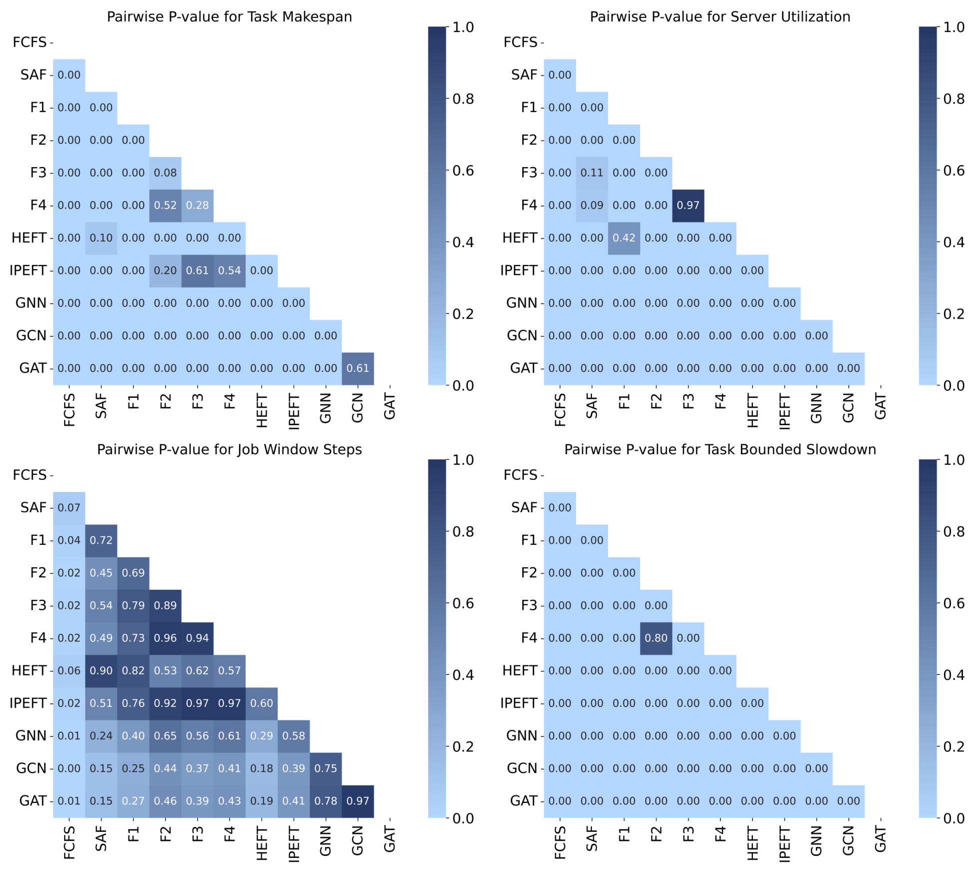
A.1 200 JOBS SCENARIO

In this scenario, a consistent pattern in the relative performance of the scheduling algorithms started to emerge, mirroring the trends identified in the 100 job scenario. However, a shift occurred in some performance metrics due to the increased number of jobs; this increase was global since the number of jobs increased while the infrastructure remained the same, so the relative performance between schedulers barely changed. The CDF plots presented in Figure 14a demonstrated a steeper slowdown curve, indicative of an increase in infrastructural stress under higher work load. This trend translated into longer waiting times for tasks, consequently leading to an elevated overall makespan.

An increase in average server utilization was also evident, a reflection of the system's response to the heightened job density. Alongside this, there was a rise in the number of window steps required for job completion, also attributed to the fact that it now takes longer to complete tasks due to the increased density. Interestingly, although the relative performance ranking did not change, this complexity did not affect all schedulers equally; less efficient algorithms were disproportionately impacted. As a result, the performance gap between the less efficient and more efficient schedulers became more pronounced. The implications of these trends are visually represented in Figures 14a and 14b.



(a) Cumulative distribution functions of all metrics.



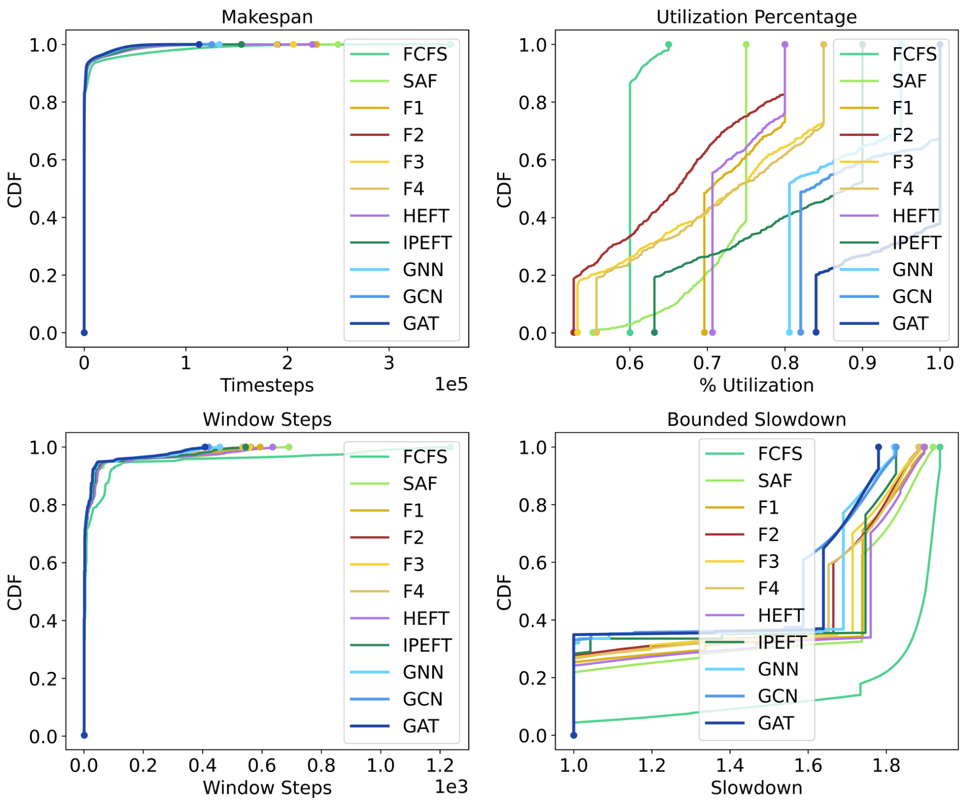
(b) Pairwise P-value matrices of all metrics.

Figure 14 – Results for the 200 jobs scenario.

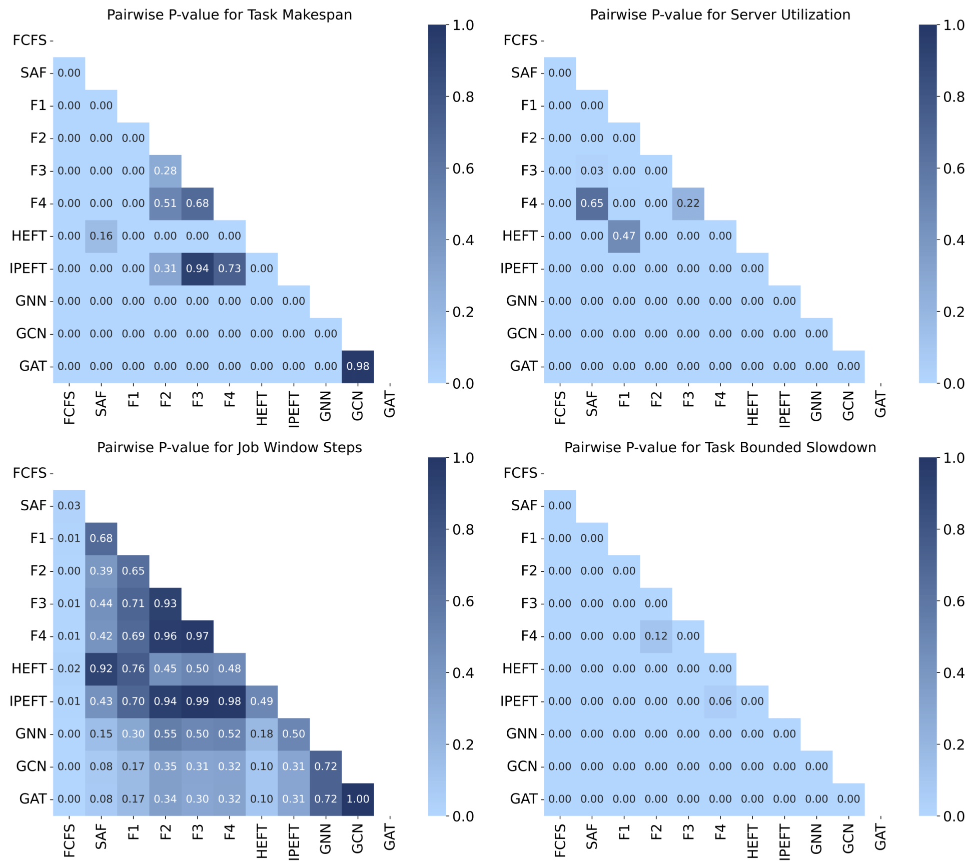
A.2 300 JOBS SCENARIO

The evaluation of this scenario further reinforced the trends observed in the preceding scenarios, particularly highlighting the system’s response to an increasing job density. The relative performance of the scheduling algorithms remained consistent with previous observations, but the impact of the higher job volume was more pronounced. The CDF plots illustrated in Figure 15a exhibited a slowdown curve tending even more to higher values, signaling an intensification of congestion within the infrastructure. This increased congestion was directly mirrored in the extended makespan values for tasks across the board, although the distributions for the makespan are very similar, the values are increased overall.

Analyzing the average server utilization indicated a continued rise, a trend in line with the additional demands imposed by the increased number of jobs. The scenario also saw a further increase in the number of window steps necessary for effective task management, which was to be expected. Notably, the performance gap between the less efficient and more efficient schedulers became increasingly evident, as pointed by the increased values of difference on the pairwise P-values indicated by Figure 15b, highlighting the challenges in high-load scenarios that simpler algorithms usually struggle with.



(a) Cumulative distribution functions of all metrics.



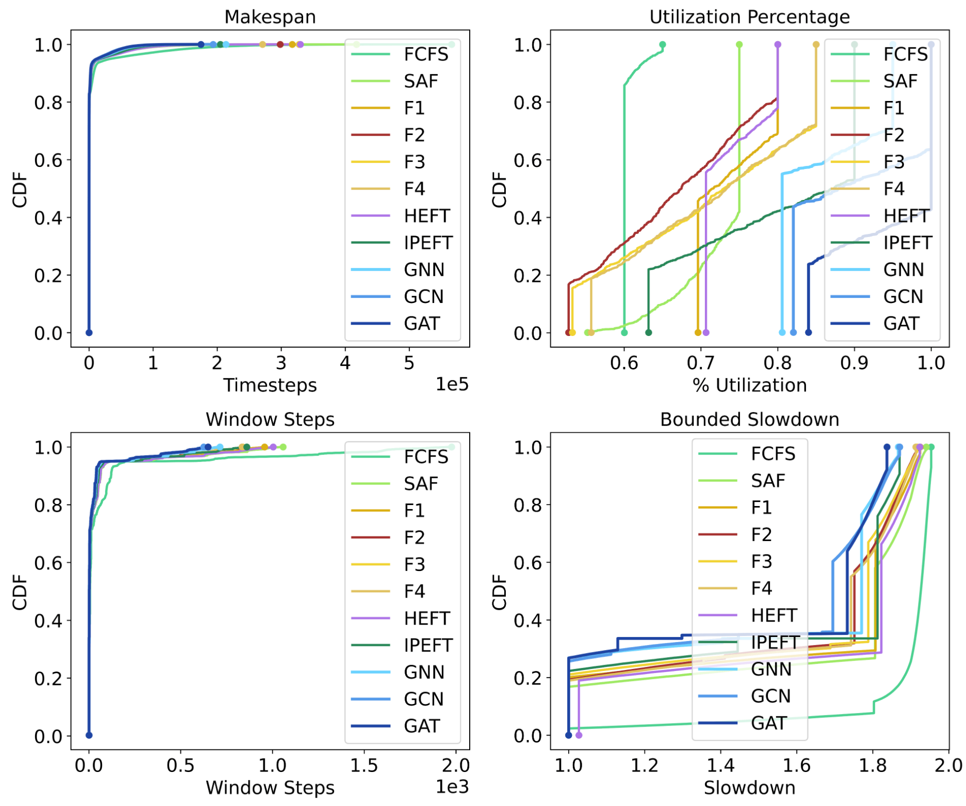
(b) Pairwise P-value matrices of all metrics.

Figure 15 – Results for the 300 jobs scenario.

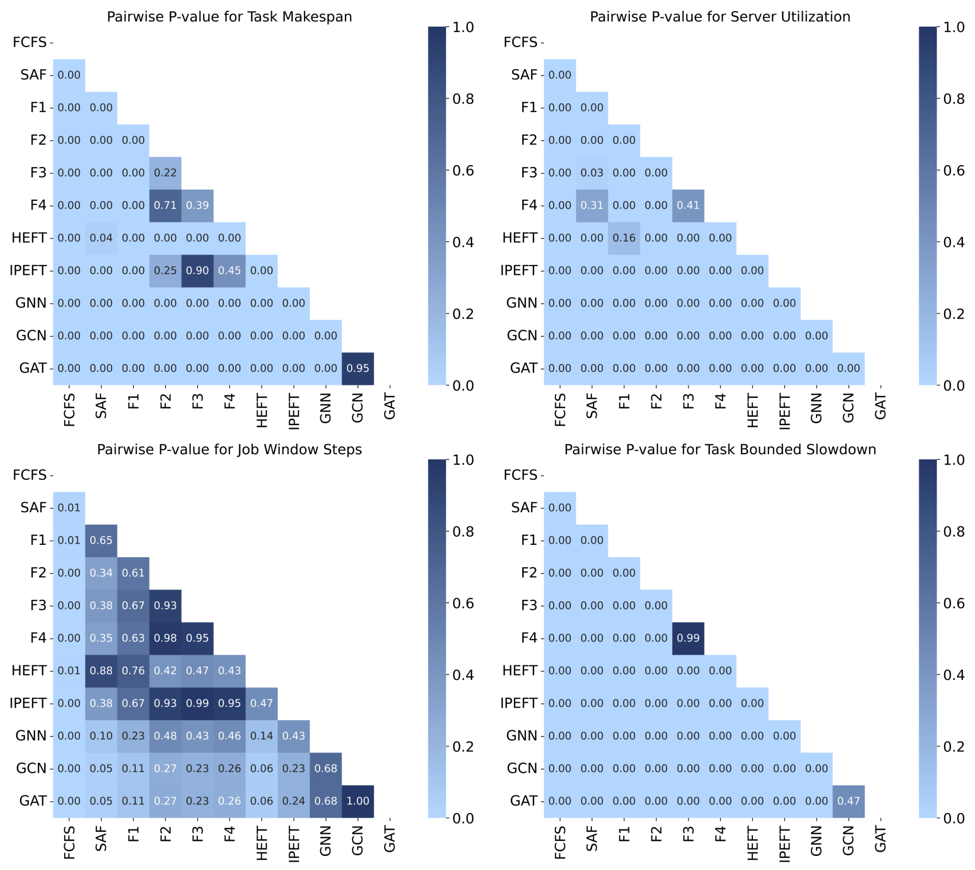
A.3 400 JOBS SCENARIO

The analysis of this further confirmed the behaviors of the schedulers performances under increasing job loads. As observed in the 100, 200 and 300 jobs scenarios, the comparative performance of the schedulers and the change in the metrics retained a similar trend. In this scenario, the CDF plots displayed in Figure 16a revealed values for slowdown curve even more concentrate in larger values, meaning that a greater percentage of tasks had a larger waiting time, an indication of heightened infrastructural congestion. This pronounced congestion contributed to a further increase in the overall makespan for tasks, marking a consistent trend across all scenarios.

In terms of resource utilization, the average server utilization was similar to other scenarios, with a slight increase, indicating that this scenario reached the ceiling of resource optimization for the schedulers, as the average for each window is indicative of efficient resource utilization. This was complemented by an increase in the number of window steps required for task completion, since the infrastructure remains the same, there's a need to further and further window steps to complete the increasing number of tasks. The distinction between the performance of less efficient and more efficient schedulers was even more pronounced in this scenario, as shown in Figure 16b with the rising values of statistically relevant differences between the worst performing schedulers when paired with better ones.



(a) Cumulative distribution functions of all metrics.



(b) Pairwise P-value matrices of all metrics.

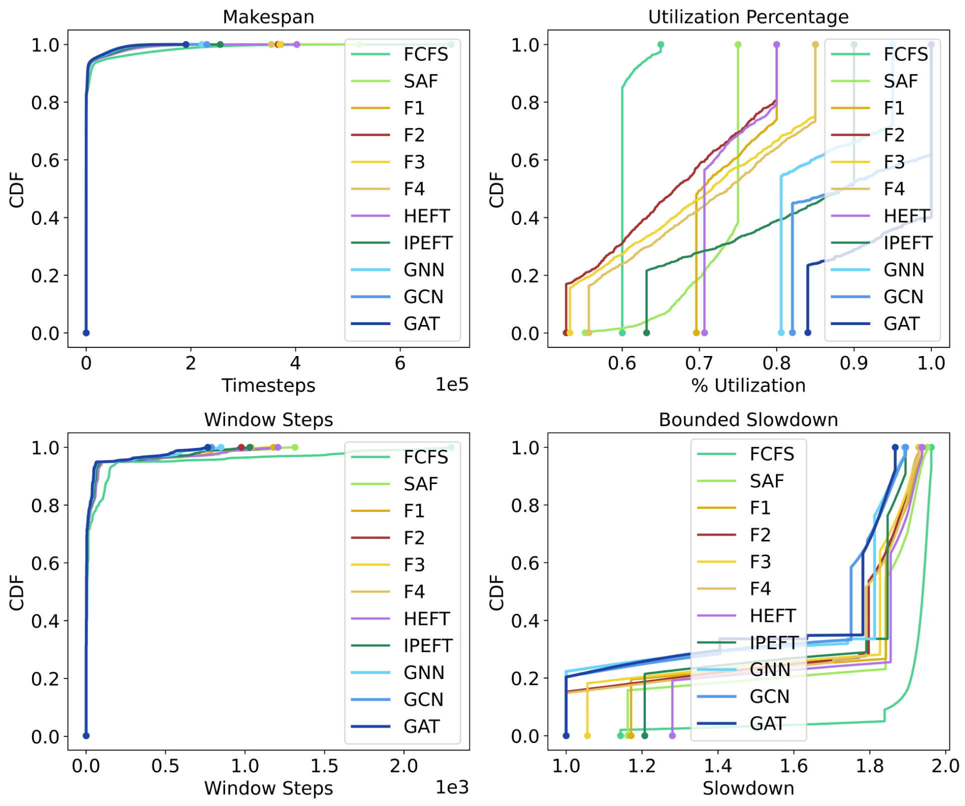
Figure 16 – Results for the 400 jobs scenario.

A.4 500 JOBS SCENARIO

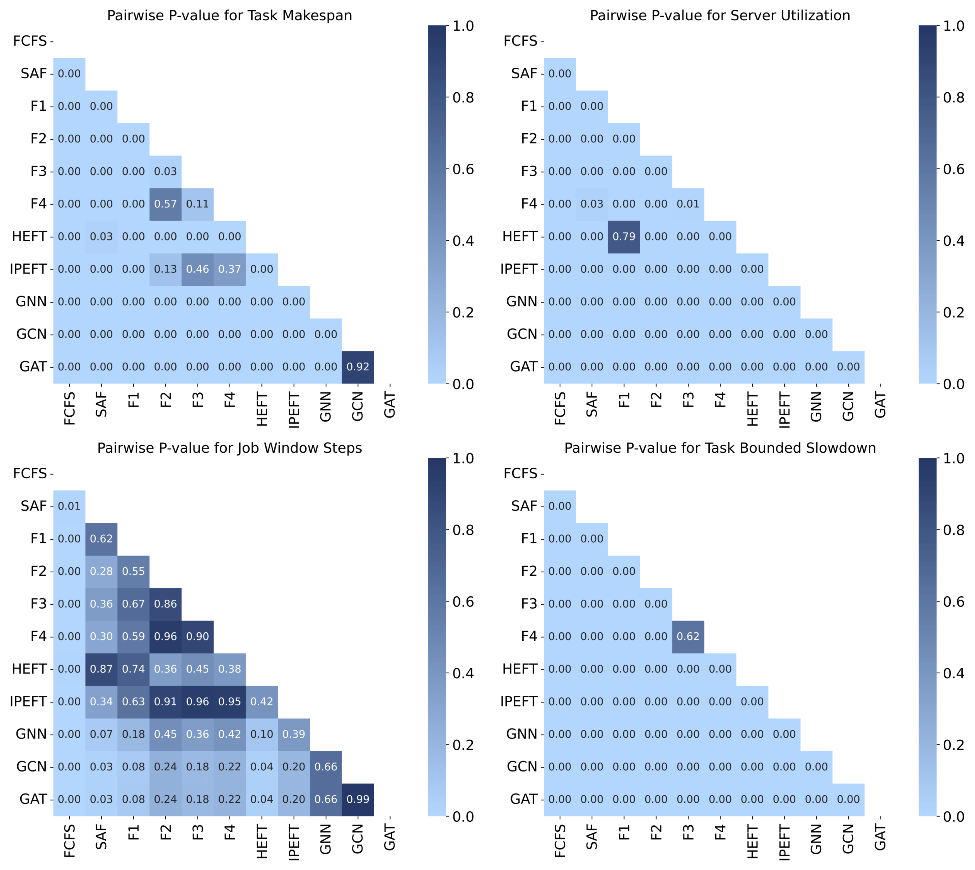
Consistent with the patterns observed in the previous scenarios, the relative performance of the schedulers and the rising values for waiting times for a larger portion of tasks showed a familiar trend. Figure 17a shows the plots for this scenario which demonstrated an even larger concentration of increased values for the slowdown curve, indicating a further escalation in infrastructural congestion. This intensification directly translated into a substantial increase in the overall makespan for tasks, a trend that has been consistent across all evaluated scenarios.

Resource management within this scenario also mirrored the trends observed in previous scenarios, with the average server utilization constrained by the limits of each scheduler once again. Furthermore, the number of window steps required for managing the tasks also experienced a notable increase, highlighting the escalating factors as the volume of jobs increases.

The gap in performance between the less efficient and the more efficient schedulers became more distinct in this scenario, clearly visible in Figures 17a and 17b showing larger gaps in the distributions and even more significant P-values between pairs of schedulers in all metrics.



(a) Cumulative distribution functions of all metrics.



Pairwise P-value for Task Makespan

	FCFS	SAF	F1	F2	F3	F4	HEFT	IPEFT	GNN	GCN	GAT
FCFS											
SAF	0.00										
F1	0.00	0.00									
F2	0.00	0.00	0.00								
F3	0.00	0.00	0.00	0.03							
F4	0.00	0.00	0.00	0.57	0.11						
HEFT	0.00	0.03	0.00	0.00	0.00	0.00					
IPEFT	0.00	0.00	0.00	0.13	0.46	0.37	0.00				
GNN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00			
GCN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00		
GAT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.92	

Pairwise P-value for Server Utilization

	FCFS	SAF	F1	F2	F3	F4	HEFT	IPEFT	GNN	GCN	GAT
FCFS											
SAF	0.00										
F1	0.00	0.00									
F2	0.00	0.00	0.00								
F3	0.00	0.00	0.00	0.00							
F4	0.00	0.03	0.00	0.00	0.01						
HEFT	0.00	0.00	0.79	0.00	0.00	0.00					
IPEFT	0.00	0.00	0.00	0.00	0.00	0.00	0.00				
GNN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00			
GCN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00		
GAT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

Pairwise P-value for Job Window Steps

	FCFS	SAF	F1	F2	F3	F4	HEFT	IPEFT	GNN	GCN	GAT
FCFS											
SAF	0.01										
F1	0.00	0.62									
F2	0.00	0.28	0.55								
F3	0.00	0.36	0.67	0.86							
F4	0.00	0.30	0.59	0.96	0.90						
HEFT	0.00	0.87	0.74	0.36	0.45	0.38					
IPEFT	0.00	0.34	0.63	0.91	0.96	0.95	0.42				
GNN	0.00	0.07	0.18	0.45	0.36	0.42	0.10	0.39			
GCN	0.00	0.03	0.08	0.24	0.18	0.22	0.04	0.20	0.66		
GAT	0.00	0.03	0.08	0.24	0.18	0.22	0.04	0.20	0.66	0.99	

Pairwise P-value for Task Bounded Slowdown

	FCFS	SAF	F1	F2	F3	F4	HEFT	IPEFT	GNN	GCN	GAT
FCFS											
SAF	0.00										
F1	0.00	0.00									
F2	0.00	0.00	0.00								
F3	0.00	0.00	0.00	0.00							
F4	0.00	0.00	0.00	0.00	0.62						
HEFT	0.00	0.00	0.00	0.00	0.00	0.00					
IPEFT	0.00	0.00	0.00	0.00	0.00	0.00	0.00				
GNN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00			
GCN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00		
GAT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

(b) Pairwise P-value matrices of all metrics.

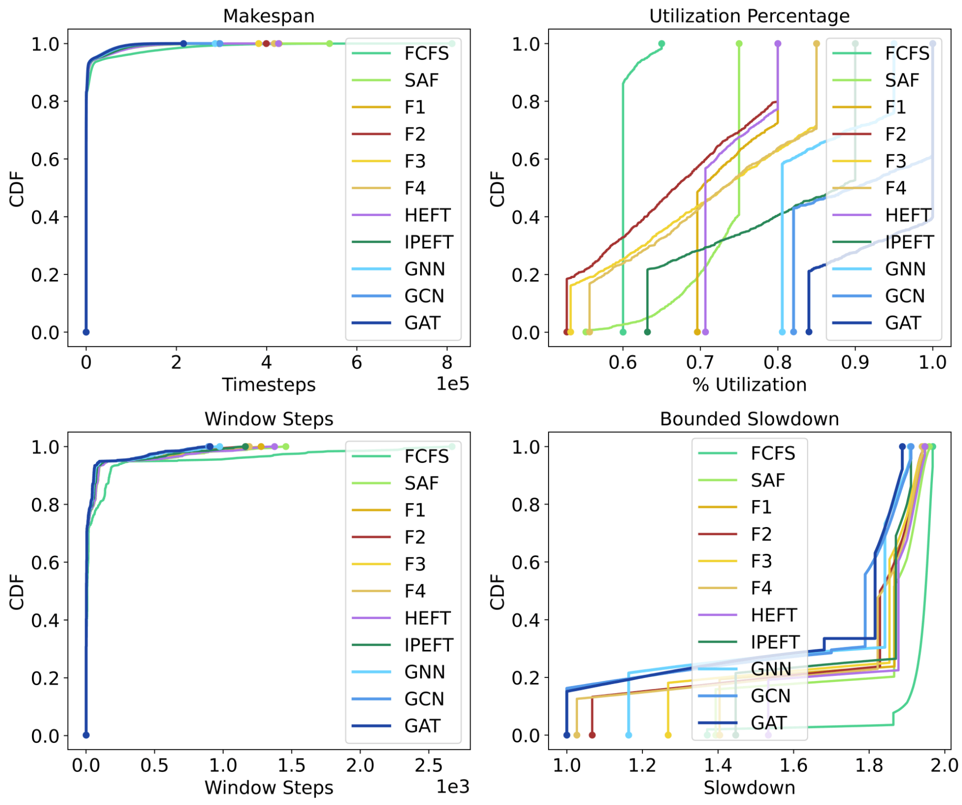
Figure 17 – Results for the 500 jobs scenario.

A.5 600 JOBS SCENARIO

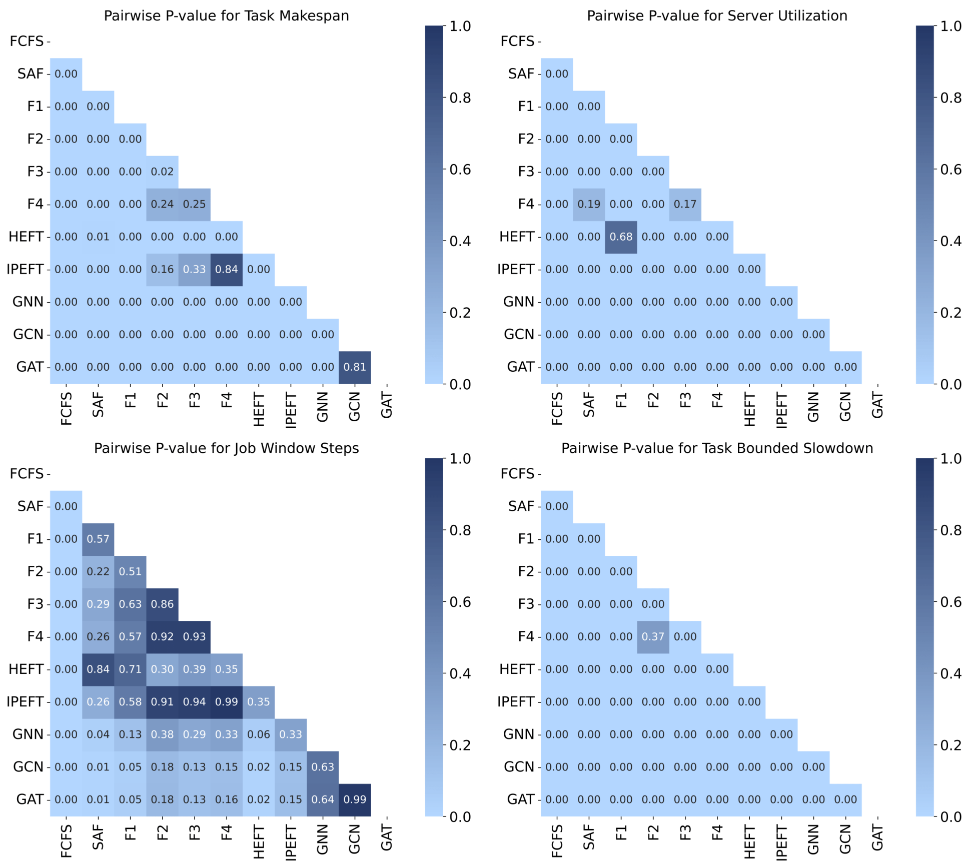
In the context of the 600 jobs scenario, the trends and implications of scheduler performance in high-volume job environments were further accentuated. Aligning with observations from previous scenarios, Figure 18a demonstrates that the relative performance of the schedulers continued to exhibit a consistent pattern. A rise in values for makespan and the higher concentration of larger bounded slowdown values also followed the trend set by previous scenarios. This points to a conclusion that higher densities will only exacerbate the values but keep the trends consistent.

The scenario also kept the trends for window steps and server utilization, with more and more window steps needed to complete the scheduling. The impact of increasing or reducing the size of the window for these scenarios is left for future research.

Figure 18b highlights the performance disparity between the less efficient and more efficient schedulers in this scenario, with higher statistical differences being indicated between more and more pairs of schedulers, specially in the bounded slowdown metric, which points to a more accurate conclusion about the performance of the trained schedulers.



(a) Cumulative distribution functions of all metrics.



Pairwise P-value for Task Makespan

	FCFS	SAF	F1	F2	F3	F4	HEFT	IPEFT	GNN	GCN	GAT
FCFS											
SAF	0.00										
F1	0.00	0.00									
F2	0.00	0.00	0.00								
F3	0.00	0.00	0.00	0.02							
F4	0.00	0.00	0.00	0.24	0.25						
HEFT	0.00	0.01	0.00	0.00	0.00	0.00					
IPEFT	0.00	0.00	0.00	0.16	0.33	0.84	0.00				
GNN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00			
GCN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00		
GAT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.81	

Pairwise P-value for Server Utilization

	FCFS	SAF	F1	F2	F3	F4	HEFT	IPEFT	GNN	GCN	GAT
FCFS											
SAF	0.00										
F1	0.00	0.00									
F2	0.00	0.00	0.00								
F3	0.00	0.00	0.00	0.00							
F4	0.00	0.19	0.00	0.00	0.17						
HEFT	0.00	0.00	0.68	0.00	0.00	0.00					
IPEFT	0.00	0.00	0.00	0.00	0.00	0.00	0.00				
GNN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00			
GCN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00		
GAT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

Pairwise P-value for Job Window Steps

	FCFS	SAF	F1	F2	F3	F4	HEFT	IPEFT	GNN	GCN	GAT
FCFS											
SAF	0.00										
F1	0.00	0.57									
F2	0.00	0.22	0.51								
F3	0.00	0.29	0.63	0.86							
F4	0.00	0.26	0.57	0.92	0.93						
HEFT	0.00	0.84	0.71	0.30	0.39	0.35					
IPEFT	0.00	0.26	0.58	0.91	0.94	0.99	0.35				
GNN	0.00	0.04	0.13	0.38	0.29	0.33	0.06	0.33			
GCN	0.00	0.01	0.05	0.18	0.13	0.15	0.02	0.15	0.63		
GAT	0.00	0.01	0.05	0.18	0.13	0.16	0.02	0.15	0.64	0.99	

Pairwise P-value for Task Bounded Slowdown

	FCFS	SAF	F1	F2	F3	F4	HEFT	IPEFT	GNN	GCN	GAT
FCFS											
SAF	0.00										
F1	0.00	0.00									
F2	0.00	0.00	0.00								
F3	0.00	0.00	0.00	0.00							
F4	0.00	0.00	0.00	0.37	0.00						
HEFT	0.00	0.00	0.00	0.00	0.00	0.00					
IPEFT	0.00	0.00	0.00	0.00	0.00	0.00	0.00				
GNN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00			
GCN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00		
GAT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

(b) Pairwise P-value matrices of all metrics.

Figure 18 – Results for the 600 jobs scenario.