International Journal on Optimization and Applications
IJOA. Vol. 06, Issue No. 01, Year 2026, www.ijoa-journal.com

AI Workload Management: A Comparative Analysis of Process Scheduling Algorithms Evaluating Scheduling Algorithms for AI Optimization

Lucci Ania L. J. Fagela
University of the Cordilleras
ljf7925@students.uc-bcf-edu.ph

Jenny Ann T. Guyong
University of the Cordilleras
jtg6745@students.uc-bcf-edu.ph

***Abstract*—Efficient workload management is critical for optimizing the performance of AI-based systems, especially in environments with diverse and dynamic processing requirements. This study presents an extensive examination of five CPU scheduling algorithms with regard to their performance in managing AI workloads, focusing on First-Come, First-Served (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Priority Scheduling, and Round Robin (RR). Through a scenario-based approach, each algorithm was analyzed based on its waiting time and turnaround time. To gather the data, a simulation tool was used with random arrival times and burst times, which provided insights into their adaptability for various AI processes. The results revealed SRTF to be the superior one among the five despite its frequent context switching. FCFS and RR excel in simplicity and fairness but face inefficiencies such as the convoy effect and high context-switching overhead. On the other hand, SJF and Priority Scheduling perform well in predictable or hierarchical workloads, minimizing turnaround and waiting times, but encounter challenges with starvation and dynamic task management. The study concludes that no single algorithm universally outperforms the others, as their effectiveness depends on workload characteristics and system requirements. This highlights the importance of selecting scheduling algorithms aligned with specific AI workload demands to maximize system efficiency and performance.**

***Index Terms*—Keywords—CPU scheduling algorithms; Artificial intelligence optimization; Process management; Scheduling performance metrics; Modeling and simulation.**

I. Introduction

A. Background of the Study

Technology paved the way for the development of computers and mobile devices that enabled people to manage all aspects of daily life, such as shopping, online learning, booking appointments, and chatting. At the core of this technological shift are computers and mobile devices, which have become indispensable tools in modern life. These devices are able to seamlessly function with the advent of operating systems, a type of system software that manages the resources of the computer and facilitates the system's management and control.

When a user uses a computer and performs a task, such as opening a document, there are often background jobs for sending or receiving email. In these instances, the numerous processes need to be handled by the Central Processing Unit (CPU), which is responsible for processing computer operations. However, only a single process is allocated to the CPU at a time. This is called scheduling, which is the allocation of computer resources to tasks in order to make full use of the CPU [1]. Scheduling algorithms are developed to help the system efficiently manage process execution even while other processes are on hold.

In recent years, artificial intelligence (AI) has emerged as a transformative technology that enables machines to perform tasks that typically require human intelligence . As evident, modern processors, such as the Intel Core Ultra, now integrate AI into their operating systems to support more advanced software and meet the growing demands of AI applications. These systems rely on powerful computational resources to process complex data; hence, the choice of scheduling algorithm for managing AI workloads in these systems becomes critical to ensuring the smooth execution of multiple and complex processes.

1) Research gaps: While scheduling algorithms have been extensively studied, limited investigations have examined their performance specifically within the context of managing AI workloads. Although existing studies have evaluated various scheduling algorithms [2]–[4],

they have not fully explored how well these algorithms meet AI-specific demands.

In 2018, OpenAI reported that the computational power required to train the largest AI models had been doubling every 3.4 months since 2012, debunking the predictions of Moore's Law, where computing power doubles every two years [5]. This situation calls for an urgent need to deeply understand how advanced systems must be implemented to effectively manage AI workloads.

As AI continues to advance and integrate into various fields, determining which scheduling algorithms are best suited for managing AI workloads becomes essential. This ensures that systems are built with algorithms capable of managing resources effectively even under advanced processing power. Furthermore, only a few studies have conducted a comprehensive comparison of the five selected scheduling algorithms considered in this study, highlighting a notable gap in the current research landscape.

2) Current body of knowledge: Artificial intelligence (AI), along with machine learning (ML), is one of the defining technological trends of the modern era. Technologies ranging from computers to mobile devices increasingly incorporate AI capabilities. This trend continues to evolve, impacting various fields including business, industry, and everyday life.

The foundation of AI lies in analyzing, categorizing, and predicting outcomes based on user input data, thereby facilitating improved decision-making processes. AI aims to replicate human thought and behavior, focusing on aspects such as perception, reasoning, learning, and prediction. Xu [6] suggests that by understanding these human concepts, society may progressively replace certain forms of human labor and resources.

As AI continues to evolve, its applications expand across diverse sectors including healthcare, finance, transportation, and many others. These applications rely heavily on AI's ability to process large volumes of data, adapt to new inputs, and generate predictions through complex algorithms, which in turn leads to highly resource-intensive workloads.

AI workloads refer to tasks performed by AI systems that often involve processing vast amounts of data and executing complex computations [7]. These tasks include predictive analysis forecasting, Natural Language Processing (NLP), anomaly detection, image or video recognition, and recommendation algorithms [8].

Such workloads are highly resource-intensive, requiring substantial computational power, memory, and storage to process complex computations and analyze extensive datasets [9]. As Sekar [10] explains, AI workloads are characterized by their significant computational requirements during both the training and inference phases of machine learning models. Training may involve millions of iterations of algorithm optimization, while inference requires rapid computations to deliver real-time results.

Consequently, efficient handling of data movement, computational resources, and algorithm optimization is required to achieve effective outcomes [7]. Despite their advantages, AI workloads present several challenges that complicate their implementation. According to Vohra [9], resource demand remains a primary concern, with workloads requiring high-performance hardware such as GPUs, TPUs, or ASICs to support computational intensity.

Data movement also poses a significant challenge, as AI tasks require efficient handling of large datasets across distributed computing systems. Furthermore, the complexity of deploying multiple interdependent components—including data preprocessing, model training, and monitoring—necessitates advanced management techniques.

Scalability represents another major challenge. Increasing data volumes and algorithmic complexity demand infrastructures that can scale seamlessly [7]. Traditional computing systems often struggle to meet these requirements, leading to performance limitations.

Efficient management of AI workloads requires strategies that optimize both resource utilization and operational efficiency. Pacheco [8] highlights the importance of high computational power that enables parallel processing and supports AI workloads effectively. Specialized hardware configurations are essential for AI systems to operate efficiently and achieve high performance.

Key hardware components include Central Processing Units (CPUs), Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), Field-Programmable Gate Arrays (FPGAs), and Application-Specific Integrated Circuits (ASICs), each serving distinct roles in optimizing AI functionality.

CPUs provide essential processing power and system control, managing complex AI tasks at a fundamental level. GPUs, with their strong parallel processing capabilities, are particularly suitable for the intensive computations associated with AI operations, especially neural network training.

TPUs are specifically designed to accelerate AI computations, significantly improving performance for deep learning tasks. FPGAs offer a combination of performance and flexibility through their reconfigurability, enabling implementation of customized algorithms for evolving AI applications.

Finally, ASICs are custom-designed hardware solutions optimized for specific tasks, maximizing efficiency by tailoring hardware to dedicated AI functions and achieving peak performance for specialized applications [11]. Leveraging these hardware accelerators can significantly enhance the performance of AI workloads [8]. However, when such hardware resources are limited, another key factor that can support the management of AI workloads is parallelization and distributed computing. Parallelization and distributed computing play a crucial role in handling AI workloads by breaking tasks into smaller and manageable units that can be processed simultaneously across multiple systems [7]. Sekar [10] further supports this by demonstrating that dynamic resource allocation and containerization can improve resource utilization by up to 85%. This approach not only accelerates data processing and model training but also ensures scalability, enabling AI applications to manage larger datasets and increasingly complex algorithms without being constrained by hardware limitations.

Efficient AI workload management has become a critical component of modern computing, allowing organizations to maximize the potential of artificial intelligence while optimizing resource utilization and operational costs. According to Vohra [9], efficient workload management reduces latency by ensuring that AI applications receive the required resources at the appropriate time, thereby preventing performance bottlenecks. Additionally, scalability and flexibility are significant advantages, as they allow systems to dynamically adapt to fluctuations in workloads. This adaptability ensures consistent performance even during varying computational demands.

The growing complexity and diversity of computing environments have emphasized the importance of efficient scheduling algorithms in AI and broader computing systems. As explained by Mohammadjafari and Khajouie [12], early computing systems relied on simple batch processing techniques to execute tasks sequentially. However, the evolution of computing environments now requires more sophisticated scheduling approaches. Process scheduling is fundamental to modern computing systems, as it governs how CPU time is allocated among multiple processes. Scheduling mechanisms ensure fair resource distribution while minimizing idle time, thereby improving system performance and maintaining operational stability.

Recent developments in computing architectures have introduced heterogeneous processing units such as CPUs, GPUs, and ASICs, which provide unprecedented computational capabilities. Nevertheless, Mohammadjafari and Khajouie highlight that effectively utilizing these resources requires scheduling strategies specifically designed for heterogeneous architectures. Effective schedulers must consider the diverse computational resources available within a system and analyze thread behavior patterns to optimize their performance across different cores, as emphasized by Nemirovsky and others [13].

Algorithms serve as the foundation of artificial intelligence by enabling machines to mimic human intelligence and execute complex computational tasks. These algorithms process data to derive meaningful insights and continuously adapt to new information. The effectiveness of AI systems largely depends on the selection and implementation of appropriate algorithms. A wide range of algorithms contributes to enhancing AI capabilities in data analysis, pattern recognition, search optimization, and performance improvement [11]. Common AI algorithms include search and optimization algorithms, supervised and unsupervised learning algorithms, neural networks, reinforcement learning algorithms, computer vision algorithms, and natural language processing (NLP) algorithms. Exploring diverse algorithmic strategies is highly recommended for improving current and future AI systems and expanding their functional capabilities [14].

Among these computational approaches, scheduling algorithms play a crucial role in determining the performance of AI workloads, as they directly influence resource allocation, system latency, and computational throughput. The efficiency of AI systems strongly depends on how computational resources are distributed and how tasks are prioritized. However, a major challenge lies in adapting these algorithms to the specific requirements of AI workloads.

This study investigates and compares five scheduling algorithms in order to determine which algorithm is the most efficient for managing AI workloads. The algorithms examined in this study include: (a) First-Come, First-Served (FCFS), (b) Shortest Job First (SJF), (c) Shortest Remaining Time First (SRTF), (d) Priority Scheduling, and (e) Round Robin.

The First-Come, First-Served (FCFS) algorithm is considered the simplest scheduling approach [15], [16], as it processes tasks in the order in which they arrive without preemption. In CPU scheduling, preemptive scheduling occurs when the currently executing process is forced to release the CPU because a higher-priority process arrives in the queue, such as system interrupts or system calls [17]. The simplicity of FCFS ensures fairness because all processes are eventually executed, thereby eliminating starvation [18]. Starvation occurs when a process is deprived of CPU resources because other processes continuously utilize them.

Despite its fairness, FCFS suffers from several performance limitations. One major drawback is the *convoy effect*, where processes with long CPU burst times significantly increase the waiting time and turnaround time for other processes [19]. Waiting time refers to the total time a process spends in the ready queue before execution, while turnaround time represents the total time required to complete a process from submission to completion [15]. Goel and Garg [18] highlight that the absence of prioritization in FCFS can lead to reduced system throughput, as long processes may monopolize the CPU. Consequently, although FCFS ensures simplicity and fairness, it often leads to inefficient resource utilization and poor response times in systems with diverse process burst durations.

The Shortest Job First (SJF) algorithm improves upon FCFS by addressing the convoy effect. SJF prioritizes tasks with the shortest execution time, allowing smaller tasks to complete earlier. Specifically, SJF selects the process with the smallest burst time, which significantly reduces average waiting time and turnaround time [15]. However, SJF also presents several practical challenges. One major limitation is the difficulty of accurately estimating the burst time of processes in advance [16], [18]. Furthermore, although SJF improves overall efficiency, it introduces the possibility of starvation for processes with longer burst times, particularly in heavily loaded systems. This trade-off between efficiency and fairness makes SJF suitable for optimizing throughput but less appropriate for systems requiring equal attention to all processes. Nevertheless, the SJF algorithm remains an effective strategy for reducing queue waiting time and improving overall The Shortest Remaining Time First (SRTF) is a preemptive variant of SJF that processes the task that is closest to the completion of its execution. This dynamically reallocates the CPU to processes with shorter remaining burst times upon their arrival, as described by Gonzalez-Rodriguez [16]. This approach offers further improvements in turnaround and waiting times over SJF but at the cost of increased context switching [19]. Context switching refers to the process of storing and restoring the state of a CPU so that it can switch from executing one process to another. Whenever a new process arrives with a shorter remaining time than the currently running process, the algorithm preempts the current process, leading to frequent switches. Each context switch introduces overhead as the system must save and restore the state of processes, consuming valuable CPU cycles. CPU overhead refers to the additional time and resources consumed by the CPU that are not directly related to executing the actual tasks or processes. It is the extra "cost" brought by managing tasks rather than performing them. In environments with numerous processes arriving in quick succession, this overhead accumulates, which reduces overall efficiency and resource utilization. While SRTF is more efficient for systems with frequently varying process bursts, it suffers from similar starvation issues as SJF, particularly for processes with long burst times in environments with continuous short processes [19]. This adaptability makes SRTF suitable for systems prioritizing quick response but raises concerns about fairness and overhead. In comparison with Shortest Job First, SJF has a much slower execution than the SRTF. This algorithm allows for easier management of updates and replacements and enables efficient memory usage.

Priority Scheduling is a method that assigns each task a priority level, and tasks with higher priority values are processed first. Shakor [15] and Goel and Garg [18] emphasize that this method ensures responsiveness for high-priority tasks, and meeting deadlines effectively. However, the algorithm's drawback lies in the potential starvation of lower-priority processes, a recurring issue that can be mitigated through the aging technique, where the priority of a process increases as it waits in the ready queue [19]. Additionally, equal-priority processes may experience increasing waiting times due to sequential scheduling [18]. Priority scheduling's emphasis on responsiveness and deadline adherence makes it valuable for critical systems, though careful management is required to prevent starvation and imbalance. Priority Scheduling is often used in scenarios where certain tasks are more critical than others, ensuring that high-priority tasks receive the necessary resources before lower-priority ones. Round Robin (RR) is a widely implemented scheduling algorithm that aims to distribute resources evenly among tasks by assigning a fixed time quantum to each. This is the period of time a process is permitted to run in a preemptive multitasking system. Round Robin is designed for fairness and responsiveness, utilizing time quantum to cyclically allocate CPU resources to processes. It has the ability to mitigate FCFS's drawbacks by balancing process execution [15], [16]. However, Pemasinghe and Rajapaksha [19] caution that the effectiveness of RR depends on the quantum size; too small a quantum increases context switches, while a large quantum becomes similar to FCFS. As discussed by Goel and Garg [18], a shorter time quantum can cause many context switches that lower the CPU efficiency while a longer time quantum leads to poor response time. They further note that due to high waiting times, deadlines are rarely met in this method. Despite these challenges, RR is well-suited for time-sharing systems requiring

equitable CPU distribution. This approach ensures that all tasks receive a fair share of processing time, making it suitable for time-sharing systems where responsiveness is essential.

3) Applications of CPU Scheduling Algorithms: CPU scheduling is fundamental to AI for optimizing resource use, improving system performance, and ensuring efficient execution of computationally intensive tasks. Effective CPU scheduling enables maximum CPU utilization, which is crucial for managing AI workloads that demand heavy computations and translation processes. By strategically allocating CPU time, scheduling maximizes throughput, allowing AI systems to handle tasks with high efficiency and avoid resource wastage. According to Mehta and Mehta [4], throughput refers to the number of processes that are successfully executed per time unit.

Given the variability and intensity of AI workloads, which often require dynamic resource allocation, CPU scheduling helps maintain efficiency under changing demands. It optimizes task execution in order to reduce latency, thus decreasing waiting and response times, the key factors for real-time AI applications. Additionally, by ensuring multiple tasks are processed smoothly, scheduling enhances throughput, which facilitates the completion of more tasks within a given period.

Priority management is another significant benefit CPU scheduling brings to AI systems. It ensures equitable resource distribution among various users and processes, preventing system overload and supporting collaborative environments. Priority handling allows AI tasks to be ranked based on urgency and importance, enabling systems to handle high-priority tasks promptly without disrupting others.

Lastly, CPU scheduling supports scalability, allowing AI systems to process large datasets and manage complex models without compromising performance. This scalability is essential as AI applications continue to grow in complexity and scale. Overall, CPU scheduling in AI contributes to optimized resource utilization, improved efficiency, and sustained performance, even with demanding and variable workloads [20]. FCFS processes tasks in the exact order they arrive. This is particularly useful in AI task scheduling applications, as AI systems often process numerous tasks or operations. By executing tasks based on arrival order, FCFS facilitates efficient resource allocation and shared resource access, which is crucial for generating timely responses. FCFS also offers a systematic method for data handling, where data preprocessing tasks are queued to ensure sequential processing. Furthermore, FCFS effectively models real-world applications, aiding in the analysis and optimization of service workflows. Its primary benefit lies in its predictability, as the straightforward, sequential execution order makes it highly reliable [21].

Similar to FCFS, SJF is beneficial in AI for task scheduling, as it optimizes execution order by prioritizing shorter tasks, thereby enhancing overall efficiency. In batch processing systems, SJF minimizes throughput time by completing shorter tasks first, freeing up resources for subsequent processes. Given AI's demand for timely task processing, SJF aids in improving task completion efficiency with reduced wait times. Additionally, SJF models queuing behavior and resource allocation effectively, providing performance enhancements across varying workloads [22].

Meanwhile, SRTF selects the process with the least remaining execution. This is often implemented on modern operating systems for managing CPU scheduling, often in applications where quick responses are utilized. Moreover, SRJF is often applied to real-time systems and cloud computing systems, whereas this algorithm can be used to ensure that critical tasks are prioritized effectively and able to run multiple virtual applications concurrently. It allows for better optimization of resource allocation by ensuring that shorter tasks are completed quickly. Overall, SRJF helps in managing workload effectively and minimizes latency [23].

The Priority Scheduling algorithm assigns priority levels to tasks, ensuring that higher-priority tasks are executed before lower-priority ones. In AI, this algorithm is essential for real-time processing, as it allows urgent tasks to be completed without delay, enhancing responsiveness. It optimizes batch processing by prioritizing resource allocation to higher-priority tasks, enabling the system to handle processes effectively based on urgency and importance. This leads to improved efficiency and responsiveness, making Priority Scheduling theoretically beneficial for AI applications requiring timely task execution [24].

The Round Robin algorithm allocates CPU time to each process in a rotating, cyclic order, ensuring each process receives an equal share of CPU resources. This approach is advantageous in AI for managing CPU resources, as it enables AI workflows to handle multiple tasks concurrently. Round Robin supports real-time processing in AI applications, allowing critical and complex tasks to run without significant delays. Additionally, it provides valuable insights into system performance under varying workloads and is often employed in simulations to model effective task management and resource allocation strategies [25].

B. Objectives of the Study

From the outlined research gaps, the main objective of this study is to perform a comparative analysis of the five algorithms: First-Come, First-Served (FCFS), Shortest Job First (SJF), Shortest Remaining Job First (SRJF), Priority Scheduling, and Round Robin, and identify the most suitable algorithm for managing AI workloads within AI-based systems. The specific objectives and the research questions that this study aims to address are as follows:

- To determine the advantages and disadvantages of the scheduling algorithms based on the performance metrics.
- To examine and assess the capability of the scheduling algorithms in managing AI workloads.
- To identify which scheduling algorithm is the most optimal for use as a primary method in AI data handling tasks.

C. Methodology

In order to evaluate and compare the five scheduling algorithms, this study employed a quantitative comparative research design. Quantitative research utilizes statistical software, mathematical models, and computational algorithms to analyze numerical data [26]. This method significantly helped in evidence-based decision-making, which aided in determining which scheduling algorithm was the most efficient. On the other hand, a comparative research design is used to compare the performance of the scheduling algorithms across various performance metrics. In the context of the quantitative approach, this design involves comparing the values of two or more cases based on relevant variables and assessing them after [27].

D. Dataset

CPU scheduling has two key concepts that will be utilized in this research study as part of the dataset: **arrival time** and **burst time**. Arrival time indicates when a process enters the ready queue and begins waiting for execution, representing the moment a process is available to start its assigned tasks [28]. It depicts the time frame at which the process becomes available to complete the assigned job. This concept is fundamental in scheduling algorithms that prioritize processes based on their entry order, as it dictates which process is selected next. For instance, the First-Come, First-Serve (FCFS) algorithm uses arrival time to allocate CPU resources to processes in the order they arrive.

Arrival time can be determined by the formula:

$$Arrival\ Time = Completion\ Time - Turnaround\ Time$$

Burst time, also known as execution time, represents the total CPU time a process needs to complete its execution. Measured in milliseconds, burst time specifically indicates the time required by the CPU to execute a given task, excluding any I/O time. During this period, the process transitions from a running state to a completion state. Several factors influence burst time, including the complexity of the task, code efficiency, and the system's available resources.

Burst time can be determined by the formula:

$$Burst\ Time = Complete\ Time - Waiting\ Time$$

The key difference between arrival time and burst time lies in their roles within the process lifecycle. Arrival time marks the process's entry point into the ready queue, while burst time indicates the duration required by the CPU to execute the process to completion. Arrival time is determined before the process begins execution, whereas burst time is known once the process has been completed. Simply put, arrival time denotes when a process enters the queue, while burst time defines how long it will take for the process to execute [28], [29]. Incorporating both arrival time and burst time into CPU scheduling strategies enables the development of efficient algorithms that optimize overall system performance and responsiveness.

By considering these key parameters, scheduling can better manage processes within the operating system, ensuring resources are allocated in a timely and organized manner.

For this study, the data used to test the algorithms was adapted from the randomly generated arrival times and burst times simulated by Algabri et al. [30] who used a Java-based simulation framework. The current research study also used the five scenarios they detailed, which already have the arrival times and burst times needed for the study's analysis. Making use of eight processes similar to theirs, this approach allowed for a simulation of a wide array of operational conditions that reflect the dynamic and often complex nature of real-world computing environments.

TABLE I
DATASET ADAPTED FOR THE STUDY (TAKEN FROM ALGABRI ET AL. [30]).

Process	Scenario 1 (AT, BT)	Scenario 2 (AT, BT)	Scenario 3 (AT, BT)	Scenario 4 (AT, BT)	Scenario 5 (AT, BT)
1	AT: 0, BT: 10	AT: 8, BT: 4	AT: 1, BT: 10	AT: 0, BT: 4	AT: 6, BT: 1
2	AT: 3, BT: 10	AT: 10, BT: 4	AT: 1, BT: 10	AT: 2, BT: 8	AT: 8, BT: 6
3	AT: 4, BT: 1	AT: 11, BT: 5	AT: 2, BT: 9	AT: 3, BT: 5	AT: 10, BT: 10
4	AT: 9, BT: 4	AT: 13, BT: 8	AT: 2, BT: 7	AT: 5, BT: 4	AT: 11, BT: 4
5	AT: 12, BT: 4	AT: 14, BT: 1	AT: 10, BT: 2	AT: 6, BT: 5	AT: 14, BT: 2
6	AT: 16, BT: 10	AT: 16, BT: 4	AT: 14, BT: 10	AT: 6, BT: 7	AT: 15, BT: 7
7	AT: 18, BT: 7	AT: 18, BT: 5	AT: 15, BT: 8	AT: 10, BT: 6	AT: 15, BT: 10
8	AT: 18, BT: 10	AT: 20, BT: 2	AT: 16, BT: 1	AT: 15, BT: 4	AT: 17, BT: 2

TABLE II
FIVE SCENARIOS ADAPTED FOR THE STUDY (TAKEN FROM ALGABRI ET AL. [30]).

Scenario	Arrival time characteristics	Burst time characteristics	General observation
1	Steady, starting from time 0 with a noticeable initial gap	Ranges from 1–10, with several longer bursts	Potentially longer waits for processing
2	Later starts, spread out arrivals	Shorter on average, mostly under 5 units	Quicker processing, less initial congestion
3	Early and congested, most arriving within the first 2 units	Varied, from 1–10, unpredictable delays	Early congestion, varied processing times
4	Early like scenario 1, more evenly spread over time	Moderate, none exceeding 8 units	Balanced processing load
5	Later starts like scenario 2, arrivals spread out	Mostly short, between 2 and 7 units	Quick processing, less congestion

The randomness of the data reflected various scenarios, each representing varying levels of system load and operational demands. These are designed to capture different levels of complexity and unpredictability, providing a robust test environment for evaluating different scheduling algorithms. This setup facilitated a comprehensive understanding of a wide range of operational algorithms, contributing to the development of more resilient and flexible scheduling solutions capable of supporting a dynamic and robust system [30]. 2.2 Data gathering instruments The study employed simulation tools as the main instrument to analyze the performance of the five process scheduling algorithms. A simulation tool is software or a program that imitates the dynamics of a real-world process or system. The system's predefined history allows for gaining an insight into its characteristics and workings under a real-world scenario [31]. With a simulation model, the system's behavior is studied. Yin and McKay [33] defined the modeling and simulation domains. Among the modeling and simulation techniques, the current research study specifically requires a process and system simulation, which relates to the simulation of operational systems with the purpose of understanding their processes. With this in mind, the researchers looked for existing process scheduling simulation tools that are utilized in the study and deemed Deepak Aggarwal's program to be the most appropriate. The chosen tool allowed for the testing of algorithms using predefined inputs for arrival time and burst time and provided the key performance metrics needed in the study, which are waiting time and turnaround time. The tool supports the simulation of five process scheduling algorithms chosen for the study, which are First-Come, First-Served, Shortest Job First, Shortest Remaining Job First, Round Robin, and Priority Scheduling. By employing this simulation tool, the collected data on waiting time and turnaround time are used to compare how each algorithm performs across the eight distinct scenarios. This provided insights into the efficiency of the five algorithms and helped in determining the algorithms appropriate for managing AI workloads.

E. Data Gathering Procedures

The data gathering procedure for this study involved several key steps. First, the study adapted the research method from Algabri et al. [30], which utilized a Java-based simulation tool to generate random arrival times and burst times across five distinct scenarios. To maintain consistency, the arrival times and burst times from the original study were used, as they already effectively capture the values appropriate for each scenario, simulating eight processes that were also made use of in the current study (see Tables I and II).

A simulation tool that supports the five chosen scheduling algorithms was to be selected to assess their performance under the varying scenarios. However, since

the original simulation tool used in the original paper was not publicly available, an alternative tool was selected instead, which is Deepak Aggarwal's program that simulates FCFS, SJF, SRTF, Priority Scheduling, and Round Robin algorithms. The simulation tool is available in GitHub, a platform that allows developers to share their work and code within the field. It is also published as a web application, which the researchers utilized due to its user-friendly interface, simplifying the testing process.

The priority ranking used for the Priority Scheduling for processes 1 to 8 is 4, 3, 5, 1, 6, 4, 1, and 2, respectively. In real-world systems, priorities are usually based on criteria such as urgency, importance, deadlines, or resource requirements. However, for theoretical or academic purposes, randomly assigned priorities can help test a scheduling algorithm, particularly across diverse and unstructured scenarios.

In evaluating the Round Robin algorithm, varying sizes of time quantum were employed. This included a time quantum of 1, a time quantum of 5, and a time quantum of 10. This ensured that the choice of time quantum, which is a key concept of Round Robin, is considered as a factor that can influence its results.

Next, the input data consisting of the arrival times and burst times was fed into the simulation tool. Each simulation was run multiple times to ensure accuracy and to minimize variability in the results. The simulation tool output the two metrics needed in the study, waiting time and turnaround time, which helped to assess the effectiveness and performance of each scheduling algorithm in the context of managing AI workloads.

The output generated by each simulation was meticulously logged in a structured format, capturing all relevant timing data for each process, including the aforementioned two metrics for every scheduling algorithm. Each scenario was then analyzed using descriptive statistical methods, focusing on calculating averages for these key performance metrics. This approach allowed for a thorough assessment of the differences in performance, showing how each algorithm handles varying loads and process characteristics.

For instance, averages for waiting time and turnaround time provided insights into how promptly and efficiently each algorithm processes tasks in each scenario. The results were then represented visually through bar charts to illustrate and compare each algorithm's performance across different scenarios. Special emphasis was placed on arrival and burst times to observe how fluctuations in these variables impact the overall efficacy of the scheduling algorithms. This graphical representation enabled clear comparisons, revealing strengths, weaknesses, and patterns in each algorithm's performance.

Finally, the study contextualized these findings by examining their relevance to AI systems and assessing the capability of each algorithm in managing AI workloads.

F. Data Analysis Metrics

The selection of the appropriate scheduler for systems is an important task as it needs to ensure an efficient utilization of resources [33]. There are several CPU scheduling criteria that must be taken into consideration when designing a scheduler. Some criteria that are commonly used are turnaround time, waiting time, throughput, response time, and CPU utilization [4], [30], [33]. These metrics help to evaluate how well an algorithm manages system resources while maximizing system performance and minimizing delays. Each algorithm has its own characteristics and preferences. For this particular reason, which parameters to be used for the analysis has a huge influence on what algorithm is considered to be the best [4].

For this research, the focus was specifically on waiting time and turnaround time. The reason why these two metrics were chosen is driven by their relevance to the nature of the phenomena being studied—workloads of AI-based systems. Both waiting time and turnaround time are key metrics that directly influence how quickly tasks are completed and how responsive the system is in processing tasks. Therefore, these two measures should be well considered when analyzing CPU scheduling algorithms in relation to system performance [34].

For Round Robin, however, since it is directly affected by the time quantum, it was evaluated with varying quantum values (1, 5, and 10). This variation allows for an assessment of how different time slice lengths impact the overall performance of the algorithm in terms of the two key metrics mentioned. With all these considerations, the capabilities of the algorithms in managing AI workloads are established. The two metrics mentioned are further discussed in the sections below.

1) Turnaround Time: The turnaround time is the total amount of time taken from when a process arrives in the systems to when it is terminated [16]. This is the duration that is required for a process to complete its procedure, which is important for evaluating a system's performance and responsiveness [30]. The turnaround time should be lower to achieve an efficient scheduling algorithm [4]. A short turnaround time would mean that processes finish faster, which is particularly essential in computational tasks implemented in AI for faster system performance. The formula for determining the turnaround time is as follows:

$$Turnaround\,Time = Completion\,Time - Arrival\,Time \tag{I.1}$$

The completion time refers to the time the process completes its execution while the arrival time is the time when the process is now in the ready state. Alternatively, if the inputs for the waiting time and burst time are available, the turnaround time can also be calculated using the formula:

$$Turnaround\ Time = Burst\ Time + Waiting\ Time \tag{I.2}$$

While this provides the total execution time, turnaround time alone does not output any details of delays that occur during the waiting period. This is why waiting time is an important additional metric to consider for a more comprehensive analysis.

2) Waiting Time: According to Gonzalez-Rodriguez et al. [16], waiting time is the total amount of time the processes spend in the ready queue before their execution. This significantly helps in assessing how efficient a scheduling algorithm is. A low waiting time is preferred for a good scheduling algorithm [4]. Evaluating the waiting time is vital for system performance because this determines how long processes are delayed before they can actually start running. For systems that need high hardware requirements such as AI software or gaming applications, longer waiting times can cause noticeable delays, which will negatively affect user experiences. The formula for determining the waiting time is as follows:

$$Waiting\ Time = Turnaround\ Time - Burst\ Time \tag{I.3}$$

II. Findings

A. Simulation Results for the Average Waiting Time

The bar chart (Fig.1) showcases the comparison of the average waiting time for CPU scheduling algorithms: First-Come, First-Serve (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Priority Scheduling, Round Robin with quantum 1 (RR1), Round Robin with quantum 5 (RR5), and Round Robin with quantum 10 (RR10) across five distinct scenarios. The analysis of average waiting times, where lower values indicate better performance, revealed clear differences among the scheduling algorithms across the five scenarios.

In Scenario 1, SRTF achieved the lowest waiting time with 10.375 ms, followed by SJF with 10.375 ms. Meanwhile, RR1 had the longest delay at 20.375 ms, followed closely by RR5. In Scenario 2, SRTF once again led with a better performance of 5.875 ms of waiting time, followed by SJF at exactly 6 ms. RR1 further obtained the highest waiting time, performing at 14.75 ms.

Scenario 3 shows SRTF and Priority Scheduling outperforming the others with the lowest waiting time within the range of 14 ms. Meanwhile, FCFS, SJF, and RR10 all tied at 23 ms. Similar to some of the previous scenarios, RR1 (29 ms) and RR5 (29.125 ms) obtained the highest waiting times compared to other algorithms.

In Scenario 4, SRTF had the lowest waiting time at 10.25 ms, followed by FCFS, SJF, and RR10, which all recorded waiting times of 13.125 ms. For this scenario, the Round Robin algorithms RR1 and RR5 had the highest waiting times.

Finally, in Scenario 5, SRTF and SJF achieved the lowest waiting times, performing at 7.125 ms and 7.25 ms, respectively. Priority scheduling recorded a waiting time of 10.125 ms, while FCFS and RR10 both had slightly higher times at 12.25 ms. RR1 and RR5 had the highest waiting times at 14.75 ms and 15.375 ms, respectively.

These results highlight the differences in performance among the scheduling algorithms across various scenarios, with SJF and SRTF generally delivering the lowest average waiting times.

In the analysis of the average turnaround time (Fig.2), where a lower turnaround time indicates better performance, distinct patterns emerged across the five scenarios for the various scheduling algorithms.

In Scenario 1, RR1 achieved the lowest turnaround time at 14.75 ms, followed closely by SJF and SRTF, both at 16.75 ms. The highest turnaround time in this scenario was recorded by Round Robin with a quantum of 5 (RR5), reaching 25.5 ms.

For Scenario 2, SRTF performed the best with the lowest turnaround time of 8 ms, followed by SJF at 10.125 ms. FCFS, RR5, and RR10 obtained similar turnaround times of approximately 14 ms, while RR1 reached a higher value of 18.75 ms.

In Scenario 3, SJF and Priority Scheduling performed equally well with turnaround times of 21.375 ms and 21.15 ms, respectively. In contrast, RR1 and RR5 recorded the highest turnaround times of 36.125 ms and 36.25 ms.

For Scenario 4, SRTF again demonstrated strong performance with a low turnaround time of 15.625 ms, followed by SJF at 18.5 ms and Priority Scheduling at 18.75 ms. Meanwhile, RR1 recorded a higher value of 27.625 ms.

Finally, in Scenario 5, SJF and SRTF delivered the best results with turnaround times of 12.375 ms and 12.5 ms, respectively. In contrast, RR1 and RR5 exhibited significantly higher values within the range of 20 ms.

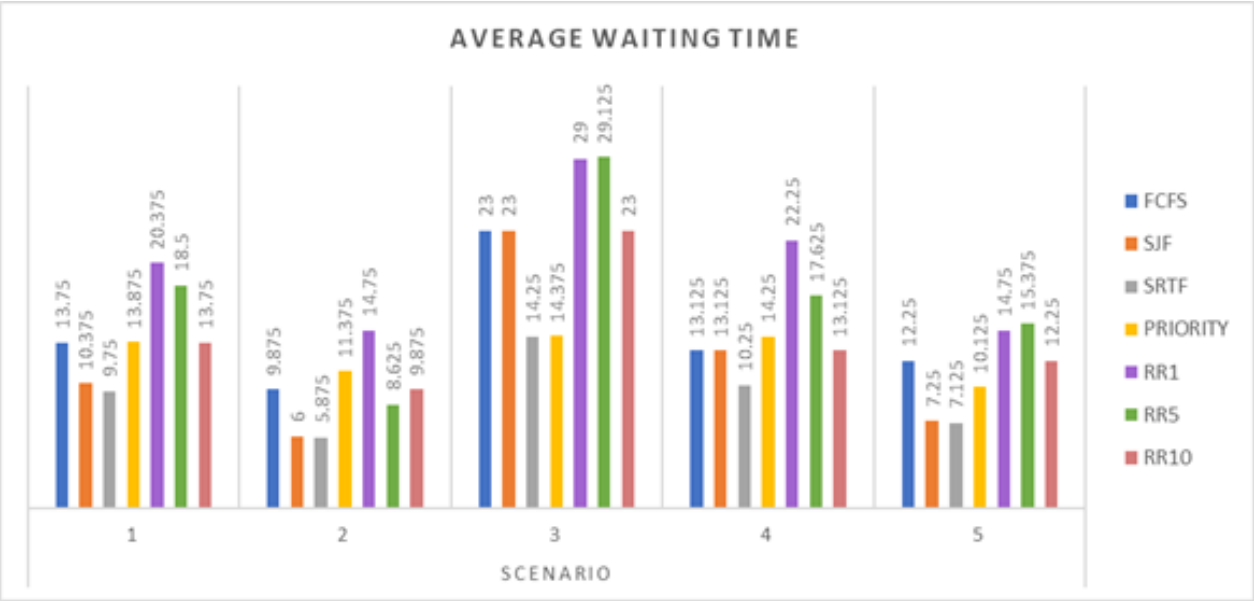

Fig. 1. Simulation results for the average waiting time.

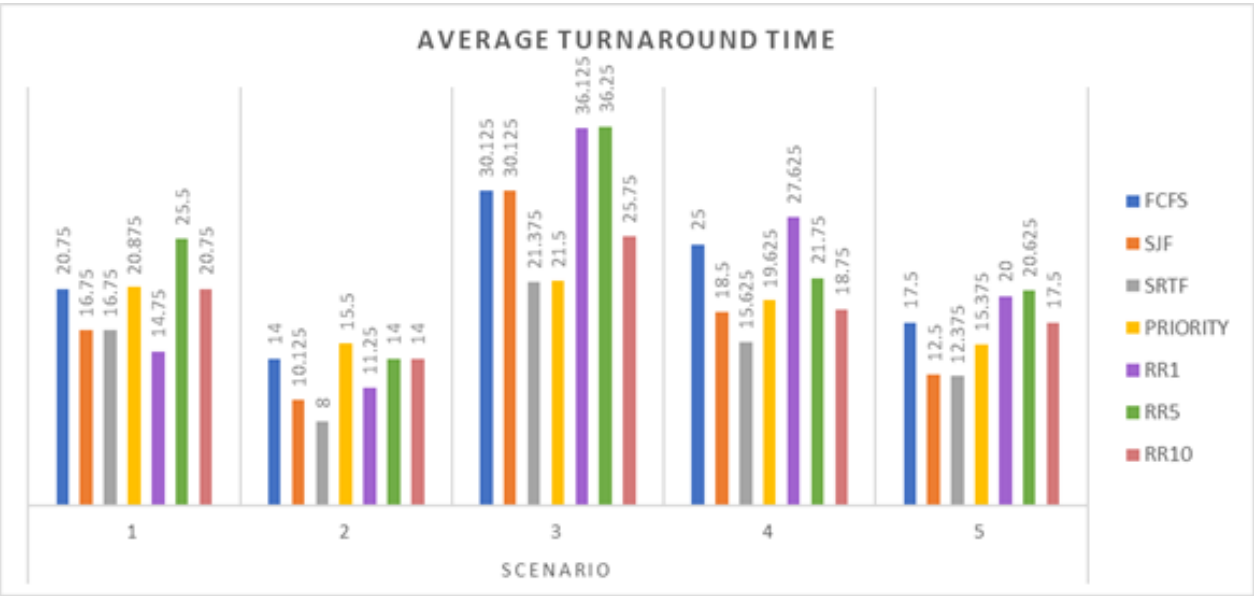

Fig. 2. Simulation results for the average turnaround time.

Overall, these results highlight that SJF and SRTF consistently provide lower turnaround times across most scenarios, indicating their efficiency in handling workloads that require faster task completion.

III. Discussion

A. Strengths and Limitations

1) First-Come, First-Served: The FCFS algorithm demonstrated simplicity in processing tasks in the order of their arrival without preemption, as highlighted by Shakor [15]. However, the results revealed that FCFS suffered from high waiting and turnaround times, particularly in scenarios involving processes with diverse burst

times. For instance, in Scenario 3, where the arrival times are early and congested while the burst times are high with unpredictable delays, FCFS recorded its highest average waiting time of 23 ms and its highest turnaround time of 30.125 ms.

This reflects the "convoy effect" described by Goel and Garg [18]. This phenomenon, where long burst-time processes delay shorter tasks, leads to increased inefficiencies and poor throughput, as the algorithm lacks prioritization mechanisms. In contrast, Scenario 2 showed the opposite results since the burst times are shorter on average (mostly under 5 units), allowing FCFS to attain its lowest waiting time of 9.875 ms and lowest turnaround time of 14 ms.

While FCFS eliminates starvation due to its fairness in eventually processing all tasks [18], the lack of optimization in waiting and turnaround times makes it less suitable for dynamic and resource-intensive environments. These results reaffirm the theoretical limitation of FCFS: its inability to prioritize or adapt, resulting in poor responsiveness in workloads with diverse burst times.

2) Shortest Job First: The SJF algorithm took advantage of its ability to prioritize shorter tasks, consistently achieving low waiting and turnaround times across most scenarios. In Scenario 2, characterized by short burst times mostly under 5 units, SJF delivered its lowest waiting time of 6 ms and its lowest turnaround time of 10.125 ms, reflecting its ability to address the convoy effect by prioritizing tasks with the shortest burst times.

This finding is consistent with Shakor [15], who stated that SJF prioritizes processes with the least burst times, significantly improving the average waiting time and average turnaround time. However, in Scenario 3, where burst times are high, SJF recorded its highest waiting time and highest turnaround time, tied with FCFS in both metrics. As noted by Goel and Garg [18], SJF can cause starvation for longer processes in saturated systems with frequent short tasks.

In addition, the practical implementation of SJF is hindered by the difficulty of accurately estimating process burst times, a limitation noted by Gonzalez-Rodriguez [16] and Goel and Garg [18]. This implies that while SJF is effective in reducing waiting times and turnaround times, its real-world application may require enhancements, such as burst-time prediction mechanisms, to mitigate its fairness issues.

3) Shortest Remaining Time First: The SRTF algorithm, being a preemptive variant of SJF, consistently delivered the lowest waiting and turnaround times across most scenarios. In Scenario 5, where processes had highly varied arrival times, SRTF outperformed every other algorithm, showcasing its adaptability in dynamic environments. This performance underscores its capability to dynamically reallocate CPU resources, as highlighted by Gonzalez-Rodriguez [16].

However, like SJF, it suffers from starvation issues [19]. This is reflected in Scenario 3, where SRTF recorded its highest waiting time and turnaround time. In this scenario, arrival times are early and congested, mostly starting within the first two time units. When processes arrive in quick succession, the scheduler frequently checks for the shortest or highest-priority task. In algorithms such as SRTF, this behavior leads to rapid preemptions and frequent context switching, which likely contributed to its relatively higher turnaround and waiting times compared to other scenarios.

In Scenario 2, however, SRTF achieved the lowest waiting time and turnaround time because the arrival times were more spread out and began later. When processes arrive with larger intervals, the scheduler has sufficient time to complete tasks before new ones enter the queue, thereby reducing context switching. This observation supports the findings in the literature [19], which indicate that although SRTF is highly efficient, it introduces overhead due to frequent process preemptions.

These results imply that SRTF is particularly suitable for environments requiring quick responses but may encounter efficiency challenges in systems with excessive context-switching occurrences. Despite these challenges, SRTF remains an effective choice for systems prioritizing quick response and low latency, as it outperformed most algorithms in scenarios characterized by rapidly changing workloads.

4) Priority Scheduling: The performance of Priority Scheduling in the study highlights its distinct advantages and limitations. Priority Scheduling assigns processes a priority value and schedules them based on these rankings, with lower numerical values representing higher priority. In the context of the given priority rankings for processes 1 to 8—4, 3, 5, 1, 6, 4, 1, and 2, respectively—the algorithm prioritized processes with higher ranks (lower numbers) over those with lower ranks (higher numbers).

In Scenario 3, where arrival times are congested with most processes arriving within the first two units, the algorithm struggled, resulting in high waiting times and high turnaround times. This poor performance stems from the simultaneous arrival of multiple processes, coupled with their assigned priority rankings. In this case, higher-priority processes, such as those with priorities 1, 2, and 3, monopolized the CPU, which forced lower-priority processes (e.g., with priorities 5 and 6) to wait for extended periods.

Shakor [15] and Goel and Garg [18] highlight that

Priority Scheduling heavily depends on the priority values, meaning that processes with lower rankings are deprioritized, leading to starvation or excessive waiting times [18], especially in congested scenarios like Scenario 3. The diversity in burst times corresponding to the priority rankings also added to the delay, as some higher-priority processes required longer CPU bursts, further increasing the waiting times for lower-priority tasks.

In contrast, Scenario 5 provided a more favorable environment for Priority Scheduling, with later and more spread-out arrivals as well as shorter and evenly distributed burst times. The algorithm achieved its lowest waiting time and lowest turnaround time in this scenario, demonstrating its efficiency when handling balanced workloads. Additionally, the shorter burst times reduced the likelihood of monopolization by any single process, enabling smoother transitions between tasks. Priority Scheduling's emphasis on responsiveness makes it suitable for systems with critical deadlines but still necessitates careful management to avoid starvation and imbalance.

5) Round Robin: The performance of Round Robin scheduling across scenarios reflects the direct influence of the quantum size and the characteristics of the workload. The findings reveal that RR with a quantum of 1 (RR1) consistently performed poorly, with the highest waiting times across scenarios, particularly in Scenario 3 (29 ms) and Scenario 4 (22.25 ms).

Similarly, RR with a quantum of 5 (RR5) exhibited high waiting times, suggesting inefficiencies when processes require significant CPU bursts. Pemasinghe and Rajapaksha [19] explained that shorter quantum sizes, such as 1 or 5, may lead to incomplete processing of tasks before another context switch occurs, adding delays in the waiting queue.

RR5 performed slightly better than RR1 but still lagged behind other algorithms due to similar inefficiencies in handling diverse workloads. In contrast, the larger quantum size of Round Robin with a quantum of 10 (RR10) provided a balance between minimizing context switches and ensuring fairness in CPU allocation, resulting in moderate performance across scenarios.

While RR10 did not achieve the best results with regard to the evaluated metrics, its performance was closer to FCFS, avoiding the drawbacks of excessive context switching associated with smaller quantum sizes. This aligns with Pemasinghe and Rajapaksha's [19] assertion that larger quantum sizes approximate FCFS behavior. Despite its fairness in distributing CPU time, the findings reveal that careful selection of the quantum size is crucial to optimize performance.

B. Process Scheduling on AI Workloads

The findings from the literature and the comparisons of various scheduling algorithms provide a comprehensive understanding of how they can manage AI workloads. FCFS is the simplest scheduling algorithm, where processes are executed in the order they arrive. While this algorithm is straightforward, its simplicity does not necessarily translate into efficiency when managing AI workloads.

Its vulnerability to the convoy effect can lead to inefficiencies in waiting and turnaround times, particularly when dealing with tasks of varying burst lengths [18]. The findings highlight that long tasks that arrive first block shorter tasks, leading to excessive waiting and poor system performance. This is particularly problematic in AI workloads that involve mixed tasks, such as training machine learning models while simultaneously performing real-time inference [8]. Moreover, as highlighted by Mohammadjafari and Khajouie [12], these delays in process execution can lead to poor performance and reduced user satisfaction.

FCFS is also unable to dynamically prioritize tasks based on urgency or computational requirements, which further limits its usefulness in AI systems that demand rapid processing and responsiveness.

In contrast, SJF addresses these inefficiencies by prioritizing tasks with the shortest burst times, thereby improving average waiting and turnaround times. In environments where AI workloads are predictable or involve computationally lighter tasks, such as natural language processing (NLP) or simpler predictive analytics, SJF can be an effective algorithm.

However, SJF faces challenges when tasks vary significantly in burst times, as it can lead to starvation of longer tasks (as observed in the findings), especially in AI systems that involve mixed workloads. Furthermore, it relies on accurate estimation of burst times, which is often difficult to obtain in real-world environments. Its tendency to cause starvation in heavily loaded systems limits its applicability in dynamic AI environments [15].

SRTF, as the findings suggest, adapts dynamically across different scenarios, which is critical in AI systems where computational loads are dynamic and change rapidly. This dynamic adjustment to task arrival and execution times makes SRTF highly effective for AI workloads with unpredictable or fluctuating resource demands.

As discussed by Vohra [9], AI workload management often involves unpredictable task patterns, which SRTF can handle effectively, as demonstrated in the results. SRTF also minimizes waiting time by processing shorter tasks as soon as they arrive, which is crucial for AI

systems relying on real-time data processing and immediate decision-making, such as anomaly detection or recommendation systems.

However, the downside of SRTF lies in the potential overhead caused by frequent context switching, especially when tasks have similar execution lengths. In such cases, the overhead can reduce the overall performance gains obtained through reduced waiting times.

Priority Scheduling provides an effective mechanism for managing tasks with varying levels of importance, which is common in AI applications involving critical and time-sensitive tasks, such as autonomous driving or healthcare diagnostics. However, its strict prioritization can lead to starvation of lower-priority tasks, particularly in environments where high-priority tasks continuously arrive.

This aligns with observations by Vohra [9], who notes that AI systems often require balancing multiple processes with different priorities and computational requirements. Consequently, the inflexibility of Priority Scheduling may reduce its effectiveness compared to adaptive algorithms such as SRTF.

Round Robin is particularly suitable for AI workloads operating in multi-user or multitasking environments, such as cloud-based AI platforms or shared computing infrastructures. Its time-sharing mechanism ensures fairness by allocating equal CPU time to all processes, preventing resource monopolization.

This property supports parallel task management [35], which is beneficial for scalable AI systems [10]. However, Round Robin can introduce high overhead due to frequent context switching. The results show that this leads to increased waiting times, which may negatively affect AI workloads requiring low latency and rapid response times.

C. Optimal Scheduling for AI Systems

Each of the algorithms analyzed presents unique strengths and limitations, but not all are equally suited to the demands of modern AI systems. Algorithms such as FCFS and Round Robin excel in simplicity and fairness, with FCFS ensuring tasks are processed in the order they arrive and RR providing equal CPU time allocation. However, these benefits are overshadowed by significant inefficiencies.

FCFS suffers from the "convoy effect," leading to high waiting times and poor responsiveness. In contrast, Round Robin's frequent context switching, especially with smaller quantum sizes, introduces a high overhead that hampers performance when implemented in intensive AI workloads.

SJF and Priority Scheduling perform relatively better in terms of minimizing waiting and turnaround times, particularly in scenarios with predictable or hierarchical workloads. SJF excels when tasks are short and burst times are predictable, while Priority Scheduling offers flexibility in prioritizing critical tasks. However, both algorithms face challenges in dynamic settings, with SJF prone to starving longer tasks and Priority Scheduling struggling with shifting task priorities, which may lead to delays for lower-priority tasks.

Among the five algorithms, SRTF consistently outperformed the others in most scenarios, making it stand out as the most effective algorithm for AI workloads due to its efficiency. SRTF demonstrated low waiting and turnaround times, which are critical metrics for AI applications such as real-time data processing, anomaly detection, and recommendation systems.

A key strength of SRTF lies in its responsiveness. Unlike non-preemptive algorithms such as SJF, SRTF dynamically preempts ongoing tasks when a shorter task arrives, reallocating CPU resources in real time. This adaptability allows it to outperform other algorithms in scenarios with rapidly changing workloads, ensuring optimal resource utilization and minimizing delays. For example, in scenarios with highly varied arrival times, SRTF demonstrated the best performance by efficiently handling all processes without being overly constrained by the arrival sequence. However, SRTF also has its drawbacks. The frequent context switching innate in its preemptive design can result in CPU overhead. This drawback can reduce overall system efficiency, making SRTF less ideal for workloads that involve sustained, long-running computations, such as training large machine learning models or processing massive datasets in parallel. Additionally, the potential for starvation of longer tasks—while mitigated compared to SJF—remains a concern in busy systems. These limitations indicate that while SRTF showed low average waiting times and low average turnaround times, it is only best suited to specific AI use cases that prioritize low latency and rapid response times. Nonetheless, SRTF consistently demonstrated superior performance across all of the scenarios, gaining its position as the most efficient scheduling algorithm among the five. Its dynamic nature aligns well with the demands of managing AI workloads requiring high adaptability and responsiveness, making it an indispensable choice for many modern applications.

IV. Conclusion

With the rise of artificial intelligence (AI), there is an increasing demand for complex applications across

various fields. To ensure that this technology delivers high performance while providing users with efficient applications, different computational approaches must be explored. This research focused on analyzing five CPU scheduling algorithms—First-Come, First-Served (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Priority Scheduling, and Round Robin—in the context of AI workload management.

By examining the advantages and disadvantages of these algorithms and evaluating their performance metrics, particularly turnaround time and waiting time, the study highlighted their suitability for varying workloads. Using randomly generated arrival time and burst time data, the algorithms were tested across five distinct scenarios. The simulation results illustrated differences in turnaround times and waiting times, providing insights into how each scheduling algorithm manages varying computational loads. The results revealed the strengths and limitations inherent in each scheduling method.

FCFS, while simple and fair in processing tasks sequentially, is hindered by inefficiencies such as the "convoy effect," particularly in environments with diverse task burst times. This limitation reduces its effectiveness for AI systems requiring high responsiveness and efficiency. In contrast, SJF excels in minimizing waiting and turnaround times by prioritizing shorter tasks. However, its effectiveness depends on accurate burst-time predictions, and its tendency to starve longer tasks makes it less suitable for heterogeneous workloads.

SRTF, a dynamic and preemptive alternative, demonstrated superior adaptability across the tested scenarios, making it particularly effective for real-time and latency-sensitive AI applications. Nevertheless, frequent context switching may introduce overhead, which can reduce efficiency in workloads involving prolonged computations. Priority Scheduling provides flexibility in prioritizing critical tasks, making it suitable for time-sensitive AI applications. However, it may lead to starvation of lower-priority processes, especially in heavily loaded systems.

Round Robin, designed to ensure fairness in multi-user environments, distributes CPU time evenly among tasks. However, its performance is strongly influenced by the size of the time quantum. Smaller quantum sizes may result in excessive context switching, increasing waiting times and reducing overall efficiency in resource-intensive AI workloads.

Overall, the analysis indicates that no single scheduling algorithm universally outperforms the others, as each possesses specific strengths and limitations depending on the nature of the workload. However, among the five algorithms analyzed, SRTF consistently demonstrated superior performance in dynamic environments that require adaptability and low latency. In contrast, algorithms such as SJF or Priority Scheduling may be more appropriate for predictable or structured AI tasks.

These findings highlight the importance of selecting scheduling algorithms according to the specific requirements of AI workloads. Future work may explore hybrid scheduling strategies or adaptive scheduling mechanisms capable of dynamically balancing overhead, fairness, and responsiveness to further optimize AI system performance.

REFERENCES

[1] T. O. Omotehinwa, "Examining the developments in scheduling algorithms research: A bibliometric approach," *Heliyon*, vol. 8, no. 5, 2022. https://doi.org/10.1016/j.heliyon.2022.e09510

[2] T. A. Alghamdi, S. M. Ali, F. H. Almuhsin, R. F. Alsahrani, and E. E. El-Sharawy, "A review on the CPU scheduling algorithms: Comparative study," *International Journal of Computer Science and Network Security*, vol. 21, no. 1, 2021. https://doi.org/10.22937/IJCSNS.2021.21.1.4

[3] N. Kulkarni, S. A. Mahajan, and A. Patil, "A study of available process scheduling algorithms and their improved substitutes," *International Research Journal of Engineering and Technology*, vol. 7, no. 8, 2020.

[4] S. Mehta and H. Mehta, "Detailed analysis and simulation of various process scheduling algorithms," *International Journal of Algorithms Designs and Analysis*, vol. 6, no. 2, 2021.

[5] K. Hao, "The computing power needed to train AI is now rising seven times faster than ever before," *MIT Technology Review*, Aug. 23, 2024. https://www.technologyreview.com/2019/11/11/132004/the-computing-power-needed-to-train-ai-is-now-rising-seven-times-faster-than-ever-before/

[6] Y. Xu *et al.*, "Artificial intelligence: A powerful paradigm for scientific research," *The Innovation*, vol. 2, no. 4, 100179, 2021. https://doi.org/10.1016/j.xinn.2021.100179

[7] Cloudian, "6 types of AI workloads, challenges & critical best practices," Jun. 26, 2024. https://cloudian.com/guides/data-lake/6-types-of-ai-workloads-challenges-and-critical-best-practices/

[8] M. Pacheco, "AI workloads: Data, compute, and storage needs explained," TierPoint, Jul. 11, 2024. https://www.tierpoint.com/blog/ai-workloads/

[9] K. Vohra, "AI workload management in data centres: What you need to know," Hyperstack, Jun. 25, 2024. https://www.hyperstack.cloud/blog/case-study/ai-workload-management-in-data-centres

[10] J. Sekar, "Optimizing cloud infrastructure for AI workloads: Challenges and solutions," *International Journal of All Research Education & Scientific Methods*, vol. 12, no. 8, pp. 296–307, 2024.

[11] GeeksforGeeks, "Hardware requirements for artificial intelligence," Aug. 13, 2024. https://www.geeksforgeeks.org/hardware-requirements-for-artificial-intelligence/

[12] A. Mohammadjafari and P. Khajouie, "Optimizing task scheduling in heterogeneous computing environments," *arXiv*, May 13, 2024. https://arxiv.org/abs/2405.08187

[13] D. Nemirovsky, N. Markovic, M. Nemirovsky, and T. Arkose, "A machine learning approach for performance prediction and scheduling on heterogeneous CPUs," in *Proc. 29th Int. Symp. Computer Architecture and High Performance Computing*, 2017. https://doi.org/10.1109/SBAC-PAD.2017.23

[14] Tableau, "Artificial intelligence (AI) algorithms: A complete overview." https://www.tableau.com/data-insights/ai/algorithms

[15] M. Y. Shakor, "Scheduling and synchronization algorithms in operating system: A survey," *Journal of Studies in Science and Engineering*, vol. 1, no. 2, pp. 1–16, 2021. https://doi.org/10.53898/josse2021121

[16] M. González-Rodríguez, E. González-Rufino, L. Otero-Cerdeira, and F. J. Rodríguez-Martínez, "Study and evaluation of CPU scheduling algorithms," *Heliyon*, vol. 10, no. 9, 2024. https://doi.org/10.1016/j.heliyon.2024.e29959

[17] H. K. Omar, K. H. Jihad, and S. F. Hussein, "Comparative analysis of the essential CPU scheduling algorithms," *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 5, pp. 2742–2750, 2021. https://doi.org/10.11591/eei.v10i5.2812

[18] N. Goel and R. B. Garg, "A comparative study of CPU scheduling algorithms," *International Journal of Graphics & Image Processing*, vol. 2, no. 4, 2013.

[19] S. Pemasinghe and S. Rajapaksha, "Comparison of CPU Scheduling Algorithms: FCFS, SJF, SRTF, Round Robin, Priority Based, and Multilevel Queuing," in *Proc. IEEE Region 10 Humanitarian Technology Conference*, 2022.

[20] G. Salton, "Introducing Run:ai's CPU Scheduling," *Run:ai*, Jul. 19, 2024. https://www.run.ai/blog/introducing-run-ais-cpu-scheduling-improved-productivity-and-utilization-for-cpu-only-clusters

[21] "Different types of non-preemptive CPU scheduling algorithms." https://www.turing.com/kb/different-types-of-non-preemptive-cpu-scheduling-algorithms

[22] M. J. Rene and D. Kagaris, "Equitable Shortest Job First: A preemptive scheduling algorithm for soft real-time systems," Department of Electrical and Computer Engineering, 2014.

[23] N. Brooks, "Shortest job first (SJF): Preemptive, Non-Preemptive example," *Guru99*, Aug. 12, 2024. https://www.guru99.com/shortest-job-first-sjf-scheduling.html

[24] S. N. Chandra and V. Karthik, "Analysis of priority scheduling algorithm on the basis of FCFS & SJF," *International Journal of Engineering Research in Computer Science and Engineering*, 2017.

[25] T. Baware, S. Chauhan, S. Naik, S. Patil, R. Thakur, and K. Vayadande, "A survey paper on CPU process scheduling," *River Publishers*.

[26] W. M. Lim, "What is quantitative research? An overview and guidelines," *Australasian Marketing Journal*, 2024. https://doi.org/10.1177/14413582241264622

[27] S. M. Miri and Z. D. Shahrokh, "A short introduction to comparative research," 2019.

[28] A. Peter, "CPU scheduling: arrival, burst, completion, turnaround, waiting, and response time," *Baeldung*, Sep. 4, 2024. https://www.baeldung.com/cs/cpu-scheduling

[29] GeeksforGeeks, "Difference between arrival time and burst time in CPU scheduling," Sep. 19, 2024. https://www.geeksforgeeks.org/difference-between-arrival-time-and-burst-time-in-cpu-scheduling/

[30] M. Algabri *et al.*, "Performance assessment of CPU scheduling algorithms," *Journal of Computer Science*, vol. 20, no. 9, 2024. https://doi.org/10.3844/jcssp.2024.972.985

[31] M. Leonelli, "What is simulation," Apr. 19, 2021. https://bookdown.org/manuele$_leonelli/SimBook/what-is-simulation.html$

[32] C. Yin and A. McKay, "Introduction to modeling and simulation techniques," 2018.

[33] N. Kulkarni, S. A. Mahajan, and A. Patil, "A study of available process scheduling algorithms and their improved substitutes," *International Research Journal of Engineering and Technology*, 2020.

[34] GeeksforGeeks, "Difference between turn around time (TAT) and waiting time (WT) in CPU scheduling," Sep. 30, 2024. https://www.geeksforgeeks.org/difference-between-turn-around-time-tat-and-waiting-time-wt-in-cpu-scheduling/

[35] GeeksforGeeks, "Data partitioning techniques in system design," Nov. 1, 2024. https://www.geeksforgeeks.org/data-partitioning-techniques/