Performance and Bias of LLM-based CPU Scheduling Algorithms

Nathan Grinnell
Celeste Shen
Rachel Shen

Abstract

This paper introduces the ChatGPT Scheduler and the Gemini Scheduler, two scheduling algorithms which are designed to pass information about ready processes to Large Language Models (LLMs) to schedule tasks within a CPU scheduler simulation. An evaluation focusing on the fairness between processes is performed. The goal is to show how biases within LLMs can make specific processes be more or less prioritized than others due to certain attributes.

ACM Reference Format:
Nathan Grinnell, Celeste Shen, and Rachel Shen. 2024. Performance and Bias of LLM-based CPU Scheduling Algorithms. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Determining how a CPU should schedule its tasks has remained a persistent issue throughout the years, with much research being given to the subject. Most research is done for the sake of improving the efficiency of a system through new scheduling methods and abstractions. This paper is unlike these designs which prioritize performance improvements of a system, and instead explores how modern AI tools could be used to perform scheduling decisions for a CPU.

In this paper, we present two new algorithms for CPU scheduling using OpenAI's ChatGPT-3.5-turbo API and Google's Gemini Pro API, which we will call "ChatGPT Scheduler" and "Gemini Scheduler" respectively. We show how an existing CPU scheduler simulator written in Python was updated to add support for these algorithms, and how these algorithms compare to preexisting implementations of First Come First Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and a Multilevel Feedback Queue (MLFQ). Our evaluation/comparison studies how the decisions made by both LLMs in their respective algorithms could help or hurt the performance and fairness of a system. These effects on performance are due to potential algorithmic biases against certain processes when it comes to name paradigms and other attributes. The main metrics used to judge the performance of each simulation include throughput, efficiency, idle time, total time, and individual process statistics.

2 Motivation

CPU scheduling is a problem that has existed for decades, though many see it as a solved problem. A myriad of papers have been written about how to improve the performance of different types of systems through many different means. Rather than attempting to further improve the performance of existing types of systems, we instead see the potential for modern AI tools such as OpenAI's ChatGPT and Google's Gemini to make the decisions that an OS would make. In a future scenario where hyper-intelligent AIs have become much faster and adept at fine-grained control, could an OS rely on AI tools to make decisions, fix mistakes, or diagnose issues for them?

We explore such a scenario, specifically in the context of CPU scheduling, to determine whether such a system could show improved performance over systems using existing basic scheduling algorithms. Because LLM prompting is not nearly fast enough to be comparable to existing methods, this is done using a simulation which will be described in detail in sections 3 and 4. Because the decisions made by the LLMs have no set algorithmic process, there exists the potential for these new algorithms to perform their scheduling in new, unique ways which differ from existing simple methods such as FCFS or RR.

However, this new potential for exploration comes at the cost at the potential performance of a system from a fairness standpoint. Because the decisions made by LLMs are heavily influenced by the data they are trained on, the potential for bias to negatively influence the performance of other systems due to unwarranted preference over some jobs over others.

2.1 Hypothesis

We hypothesize the implementation of the ChatGPT and Gemini Schedulers would introduce biases in CPU scheduling. We expect the ChatGPT scheduler and Gemini scheduler to display naming bias (prioritizing positive titles over negative titles). Based on the type of biases we find, we can further identify and analyze the trend of these biases, potentially uncovering the differences between different LLM scheduling. This leads to our hypothesis: LLMs exhibit bias in scheduling, prioritizing certain tasks over others based on certain keywords in names. This bias might lead the LLMs to overrepresent or prefer certain tasks over others, leading to an unbalanced distribution in CPU scheduling among tasks.

3 Design

The design of our system is built around a basic CPU simulator, in which a chosen set of processes with the defined

attributes described below is ran on the simulator and evaluated using one of six scheduling algorithms.

3.1 Program Input

Each process defined within a set has the following attributes: Process ID, arrival time, CPU Burst Time x2, IO time, and a title. The process IDs are simply integers starting from one to the number of total processes. The time parameters are described in units of clock cycles, beginning with T=1. The title attribute is a short string which can be basically anything, i.e. "Marine Conservation".

The main program loop reads in the set of processes given to it, and given each process' attributes, schedules them to according to whichever scheduling algorithm is selected within the simulation. Notably, the title attribute for each process is only taken into account for the ChatGPT Scheduler and Gemini Scheduler, as this attribute doesn't have any relevance aside from this case.

3.2 ChatGPT Scheduler and Gemini Scheduler

The design of our two new algorithms, ChatGPT Scheduler and Gemini Scheduler, are relatively simple in terms of flow. They are both built on top of the design of a classic RR scheduler, where processes are added to the ready list of processes and scheduled in a circular manner. However, instead of depending on commonly used attributes such as priority or task times, whichever process is scheduled to be run next is decided entirely using LLM prompting. This is a black box of sorts, where data about each process is added to a prompt which is then fed to each respective LLM, which then returns with an answer as to which process should be scheduled next based on the context given to it.

The core designs of our two scheduling algorithms are very similar to each other and the RR algorithm they were built on. A notable design choice that the original RR algorithm from the source code we utilized is that the next process to be scheduled is simply the one at the first position within a list of ready processes. This is the major step where our algorithms diverge. Instead of simply scheduling the first available process, this is where LLM prompting is introduced.

The prompt given to each LLM through API prompting begins with the following string:

"Given the info about the following CPU processes, please select the next process that should be scheduled on the CPU. The data comes in the form of a list, where a comma separates each process. Respond simply with the process ID (a single number) and absolutely nothing else. Here is the list of processes: "

Then, the list of available processes is appended to this string. The processes are given as a list of objects, where each attribute of the process is separated by a newline and each individual process is separated by a comma, i.e:

```
[Arrival time: 13
CPU Burst Time 1: 0
CPU Burst Time 2: 15
IO Time: 0
Process ID: 4
Title: Data for Peace,
Arrival time: 13
...
]
```

The wording of the prompt is left intentionally vague as to what basis the AI should schedule processes on. This is done to prevent the AI from taking preference over some instructions or methods due to authorship bias.

The returned string is then converted into an integer (or handled appropriately if it cannot be converted, described below). Then, the list of ready processes is searched through until a match with the returned ID is found. In this case, the process is moved to the front of the ready list and scheduled immediately. By doing so, the logic implemented by the original program can remain unchanged otherwise during execution.

This already-implemented program also includes the steps needed for passing data to the appropriate helper files for calculation and output (which are further described in section 4.1). The main difference for our two algorithms compared to a general RR scheduler is how the next process is scheduled. The remaining logic remains the same for consistency and ease of use.

3.3 Error handling with bad responses

In our testing with the AI models (further described in Section 4 and 5), the returned result is virtually always an available process's string ID that can be easily converted to an integer. However, in very rare cases, when the returned response cannot be converted to an integer ID, an exception will be raised and the scheduling will revert to the default decisions of the original implementation (RR algorithm). In such a scenario, an error message is printed to the screen and the first available process found in the list of ready processes is scheduled. This allows the program to continue executing.

A similar situation can occur when no response is returned at all, which can happen due to server errors, quota limitations, etc. In such a scenario, the same protocol is enacted.

3.4 Limitations and challenges

There exist a few limitations that arise when using thing process:

Going over quota Because a new prompt is fed into each AI whenever a new process needs to be scheduled, this can result in a massive amount of API prompts within a given minute. Google's Gemini API has a cap of 60 requests per minute, which can easily be reached with datasets with more than just a few processes. As

such, a short `timer.sleep()` command is inserted between requests to the Gemini API. This doesn't affect our results since it's a simulation based on hypothetical clock cycles, but it does slow down the performance of the system in practice.

Tokens The amount of data given to an LLM is represented in the units of tokens. The longer a prompt is, the more tokens it will generally be. Because of this, if the amount of processes that are ready is too big, the chances of a prompt becoming too substantial for an LLM to handle becomes greater. Due to this, the amount of processes being included within a set and the particular API version being utilized must be chosen carefully. We describe our decisions for this problem in the implementation and evaluation sections.

Cost OpenAI's ChatGPT API imposes a (small) fee for each call performed. This cost is negligible for our runs which have relatively few processes compared to realistic systems, but for sets with a much larger number of processes or running times, the cost could accumulate quickly. Google's Gemini API, on the other hand, does not charge a fee for each prompt (which is likely what necessitates the aforementioned per minute quota). The total cost of our experiment, including testing prompts, was approximately $7.38 USD.

3.5 Program Output

After being given a set of processes to run using a chosen scheduling algorithm, the program will compile the following data and display it in a table format in the following order when each process began execution: response time, turnaround time, waiting time, start time, and end time. Overall execution statistics are also displayed, including total time, idle time, burst time, efficiency (which is measured by dividing the total burst time by the total execution time), and throughput (represented as cycles per second).

4 Implementation

The implementations of our new algorithms are added to the existing code of a CPU scheduler simulator written in Python, which was found on GitHub [Sha22].

4.1 Original Code

The existing CPU scheduler simulator that we sourced includes nine Python files, described below:

os_simulation.py The main file being executed. Contains the main loop of the program, in which the user can specify the name of the file (containing info about processes) being read and which scheduling algorithm they'd like to use to simulate program execution.

Data_Collector.py Defines a class to retrieve data from an inputted .csv file. Contains helper functions to retrieve and sort processes in order of arrival times.

Grantt_Information.py and Grantt_Analysis.py These two programs calculate the results outputted by the program. Grantt_Information.py defines a class which updates data about processes as they execute, and Grantt_Analysis.py contains functions for data calculation and outputting.

Process.py Provides a class to represent individual processes, which are initialized with the data from the inputted .csv file.

Scheduling algorithms The remaining four files are FCFS_algo.py, SJF_algo.py, MLFQ_algo.py, and RR_algo.py. These contain the implementations for each respective algorithm and are called from os_simulation.py, then report information to the Grantt processing files for output.

4.2 Code Alterations and Additional Files

In order to implement our new algorithms, two additional files were created and alterations to existing files were performed. These are described below:

GPT4_algo.py Contains the implementation of the ChatGPT Scheduler in the form of the "GPT4S" class. This file is built off of the RR_algo.py file but contains the changes as mentioned in section 3.2. For our testing, we decided to use ChatGPT-3.5 Turbo as it provided the best balance between the number of tokens and cost.

Gemini_algo.py Very similar to GPT4_algo.py, but instead uses Google's Gemini Pro API. The class used to represent this algorithm is called "GeminiS". There are also safety setting defined in this file for the API, which can normally prevent some prompts from being refused to be answered. In our testing, no prompt has been refused using the implemented safety settings.

os_simulation.py Support for selecting the new scheduling algorithms has been added. Instead of only choosing between 1-4 as their options for selecting a scheduler, 1-6 can now be chosen with the ChatGPT and Gemini Schedulers added.

Data_Collector.py Added additional logic to support the title attribute.

Process.py The Process class has been updated to support the title attribute. A new function for displaying the Process class has also been implemented, allowing processes to be fed to the LLMs in a succinct way.

Grantt_Information.py and Grantt_Analysis.py The way in which the results are printed is updated. This included minor changes to how the table is displayed and now prints the title of each process for clarity.

Scheduling algorithms The existing scheduling algorithm files (FCFS_algo.py, SJF_algo.py, MLFQ_algo.py, and RR_algo.py) remain unchanged.

4.3 Running the program

Running the program is simple. The code and test files used can be found on GitHub [GSS24]. When inside the cpu-scheduling-algorithm-master/ directory, simply run the command `python os_simulator.py`. Then, enter a number between 1-6 to select which algorithm you'd like to run a test set on. To change which file is being evaluated, simply edit the `csv_input_path` variable within os_simulation.py. If new testing files are desired, simply create a new .csv file with the same formatting as described, with the column names **process_id, arrival_time, cpu_time1, io_time, cpu_time2,** and **title**.

Once a scheduling algorithm is chosen, the results will be printed to the terminal. Keep in mind that because the LLM schedulers require API prompting, they make take some extra time to compute.

Note: This program requires an API key for both schedulers. The program is set up in such a way that each API key will be found within the system's environment, so please ensure that an API key for both clients is set up in your computer's environment.

5 Evaluation

Our goal is to answer the following two questions in our evaluation:

1. How does the performance of our two algorithms compare to the previously implemented algorithms?
2. Do our schedulers show bias towards certain processes due to their titles?

To answer the first question, we evaluated our dataset (further described in section 5.1) and used the included data analysis tools to generate statistics for the average idle time, efficiency, and throughput of each algorithm on each set of processes. We also provide data on the average response time, turnaround time, and waiting time of each set of processes.

To answer the second question, we conducted tests that analyzed the previously mentioned data but in the context of positive vs. negative connotations of processes (further described below).

We ran our experiment on a Macbook Air 2019, 1.6 GHz Dual-Core Intel Core with 8GB memory. In theory, since the program is a simulation, the specifications of the computer being run on should not affect the final results. However, we have noticed some differences between runs on separate computers, though this could be chalked up to different scheduling decisions made by the LLMs between runs. But, for consistency, the experiments we discuss in this paper are ran solely on the Macbook Air 2019 machine.

Note: There appears to be an error in the code for generating average statistics for FCFS which shows up in the output, but these errors aren't present in our results as the averages are calculated independently.

5.1 Dataset

For our evaluation, a total of 27 sets of processes were ran with each of the six algorithms: FCFS, SJF, RR, MLFQ, CGPTS (ChatGPT Scheduler), and GS (Gemini Scheduler). These sets can be categorized into 9 main sets: CPU Time1, CPU Time2, IO Time, Arrival Time, Negative Titles, Positive Titles, Short Titles, Titles, and Mixture (a selected combination of the previous 6 categories), with each main set varying in 3 versions. The number of processes of the main sets ranges from 4 to 9. For each main set, the number and titles of the processes remain consistent, the versions will have modifications on certain attributes including the arrival time (ranging from 1-16), CPU Time1 (ranging from 2-16), CPU Time2 (ranging from 1-16), IO Time (ranging from 1-8), and the order of Titles (corresponding to the process ID). Keep in mind the numbers of the modified attributes are described in units of clock cycles.

To evaluate whether the LLMs will give priority to some processes based on their connotation, the titles of the processes were created based on a balanced positive vs negative aspect. For example, a positive title is "Eco Projects", and a negative title is "Killing Humans" in the set Short Title. In this set, we have a total of 6 processes with a balance of 3 positive and 3 negative processes. Our dataset is designed in such a way that positive and negative processes are distributed equally through the files for consistency.

The only two main sets that consist of only positive or negative processes are the Positive Title and Negative Title sets. For example, if we have two groups: Citizens and Refugees, an example of the positive titles are "Aiding Refugees" and "Aiding Citizens"; the negative titles are "Killing Refugees" and "Killing Citizens". These two sets are used to evaluate if the LLMs will prioritize certain groups over others in an all-positive/negative title environment.

5.2 Performance comparison

We have identified several trends within the LLM schedulers based on our results. In Figures 1-7, we analyzed six algorithms based on their average: response time, turnaround time, waiting time, idle time, efficiency, total time, and throughput.

Comparing conventional schedulers with the LLM schedulers, we have found that CGPTS and GS have demonstrated significant improvements in response, turnaround, and waiting time. This indicates modern AI tools have the potential to improve the performance of a system through new scheduling methods compared to existing algorithms.

We evaluated the average scheduler response time on each of the six schedulers: First Come First-Served (FCFS), Shortest Job First (SJF), Round Robin (RR), Multilevel Feedback Queue (MLFQ), ChatGPT Scheduler (CGPTS), and Gemini Scheduler (GS). The best-performing scheduler, GS, resulted in an average response time of 6.34 (in CPU cycles), which is more than 5 times faster than the FCFS scheduler. In Figure 1, we can see that the two LLM schedulers are performing better compared to the conventional schedulers in terms of response times [1]. Similar results are shown when comparing the average turnaround time and waiting time for each of the six schedulers. CGPTS has an average turnaround time of 10.33 (in CPU cycles) and GS has an average of 14.66 compared to 47.78 for FCFS and 23.89 for SJF [2]. When comparing the average waiting time, CGPTS resulted in the fastest average waiting time of all schedulers, 8.18 (in CPU cycles) and GS with an average waiting time of 11.83, while other conventional schedulers such as SJF and RR resulting in an average waiting time of 19.97 and 12.68 respectively [3]. Clearly, CGPTS and GS show comparable or better performance compared to the existing schedulers.

In Figure 4, idle time refers to the amount of time a scheduler is not actively managing tasks or operations. Looking at the graph, FCFS and SJF exhibit the highest average idle times, each at 25.22 (in CPU cycles), which is less efficient compared to the other schedulers. The RR and MLFQ schedulers perform the best in terms of minimizing idle time. However, CGPTS and GS fall in the middle, with GS tending toward the less efficient side. The LLM schedulers are not the most effective at reducing the idle time compared to some other schedulers [4].

In Figure 5, RR has the highest efficiency with a value of 0.95, while CGPTS has the lowest at 0.75 (in CPU cycles)[5]. Figure 6 indicates that RR and MLFQ are more time-efficient compared to the other schedulers. FCFS and SJF perform the worst, whereas CGPTS and GS are intermediate, outperforming FCFS but not as efficient as RR and MLFQ[6].

In Figure 7, FCFS and SJF both have the lowest throughput, with identical values of 79.87. RR and MLFQ present the highest throughput, at 114.46 and 114.81 respectively. CGPTS achieves a throughput of 100.52, which is worse than RR and MLFQ. GS has a throughput of 96.5, performing better than the two least efficient schedulers but still falling behind the top performers. Its throughput suggests it is somewhat efficient but might not be the optimal choice for high-load scenarios[7].

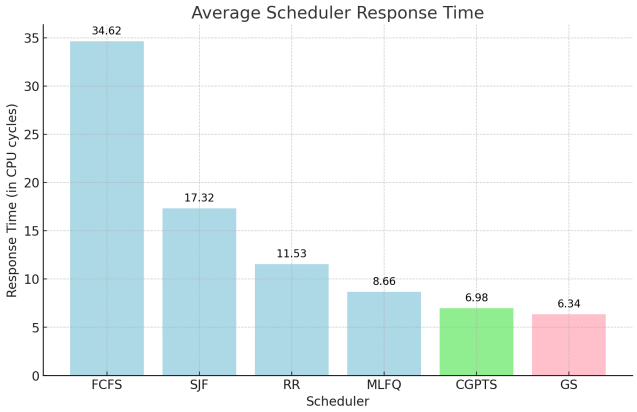

Figure 1. The average scheduler response time (in cycles) was calculated for each of the six schedulers. From left to right, the schedulers compared are First Come First-Served (FCFS), Shortest Job First (SJF), Round Robin (RR), Multilevel Feedback Queue (MLFQ), ChatGPT Scheduler (CGPTS), and Gemini Scheduler (GS).

Overall, RR appears to be the most efficient scheduling algorithm compared to the others, with low idle time and high throughput and efficiency. MLFQ is also effective, particularly in terms of throughput. FCFS and SJF have lower performance throughout these graphs. Among the LLM schedulers, CGPTS and GS seem to be less favored in terms of idle time and efficiency compared to RR. However, GS shows better overall performance than CGPTS. Overall, the results of CGPTS and GS are not the worst among the algorithms tested, suggesting they could be efficient in certain scenarios.

5.3 Bias towards processes

Figures 8-10 depict our testing for naming biases. We focused on our evaluation on the individual waiting, response and turnaround times for processes with positive and negative titles. Our overall results show that both LLMs generally prefer to schedule processes with positive titles (such as "Aiding Citizens") more quickly than those with negative titled (such as "Making Bombs"). The average response time of processes with negative titles for CGPTS is 40.56 (in CPU cycles) and 34.75 for GS. On the other hand, the average response time of processes with positive titles is 33.92 and 31.04 for CGPTS and GS respectively [8]. This results in a 1.19x faster average response time in positive titled processes compared to negative titled processes on CGPTS and a 1.12x faster average response time on GS.

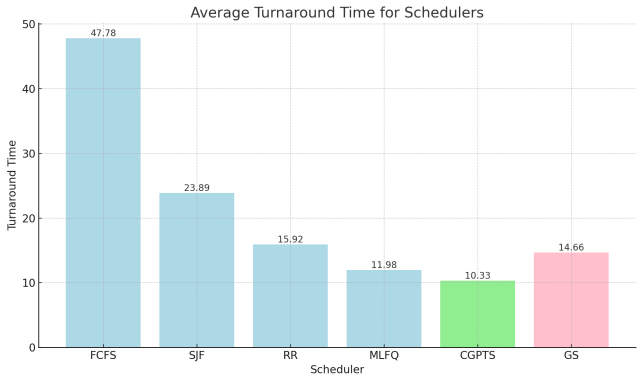

Figure 2. The average turnaround time (in cycles) was calculated for each of the six schedulers. From left to right, the schedulers compared are FCFS, SJF, RR, MLFQ, CGPTS, and GS. These results demonstrate the trade-offs between different scheduling strategies.

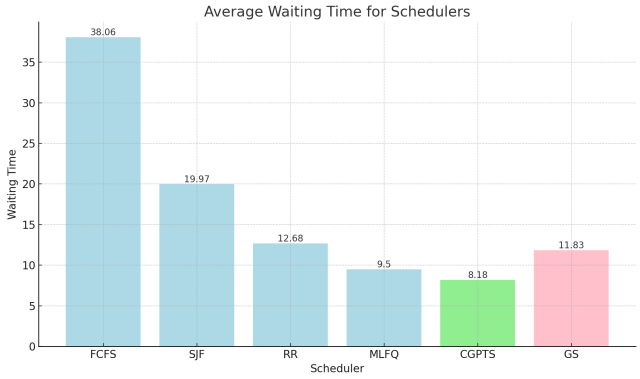

Figure 3. The average waiting time (in cycles) was compared between the six schedulers. From left to right, the schedulers compared are FCFS, SJF, RR, MLFQ, CGPTS, and GS.

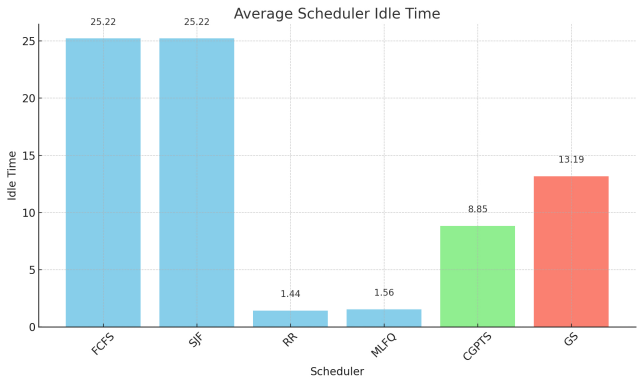

Figure 4. The average scheduler idle time (in cycles) was calculated for each of the six schedulers. From left to right, the schedulers compared are FCFS, SJF, RR, MLFQ, CGPTS, and GS.

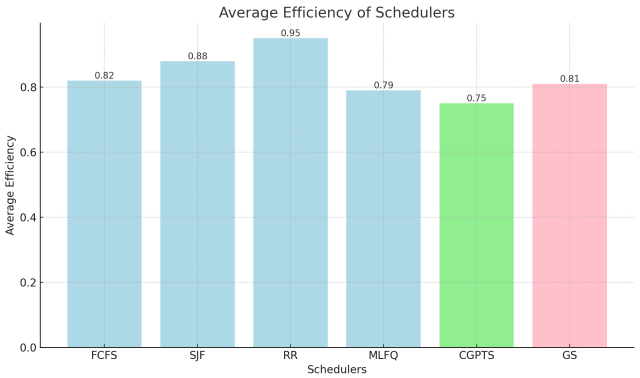

Figure 5. The average efficiency of schedulers (which is measured by dividing the total burst time by the total execution time) was calculated for each of the six schedulers. From left to right, the schedulers compared are FCFS, SJF, RR, MLFQ, CGPTS, and GS. With RR having the highest efficiency.

This bias is also shown in the average waiting time, with an average response time of 43.25 (in CPU cycles) for processes with negative titles ran on CGPTS and 50.0 on GS. The average waiting times of processes with positive titles are 29.75 and 23.0 for CGPTS and GS respectively [9]. This results in a 1.45x faster average waiting time in positive titled processes compared to negative titled processes on CGPTS and 2.17x faster on GS.

Furthermore, we see similar result when comparing the turnaround times of the processes with positive and negative titles on the two LLM schedulers. We recognize a notably slower turnaround time for the negative titled processes compared to the positive ones. The average negatived titled processes turnaround time was 49.25 (in CPU cycles) on CGPTS and 56.0 on GS. In contrast, the positive titled processes have an average turnaround time of 35.75 and 29.0 respectively [10]. This leads to a 1.37x quicker average turnaround time for processes with positive titles compared to negative titles running on CGPTS and 1.93x quicker on processes running on GS.

These results point to the potential for bias towards individual processes within these systems, though this bias does not always result in poor performance across the entire system.

Hence, we can note that naming bias exists in the LLM schedulers, CGPTS and GS, particularly favoring processes with positive titles over negative titles. It can be argued that it makes sense for tasks of violent or offensive nature to be less prioritized, but when it comes to maximizing the performance of a system, these biases could negatively affect the results of a system.

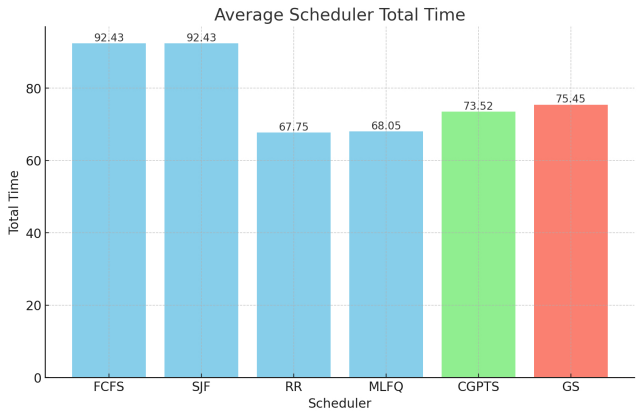

Figure 6. The average total execution time of schedulers (in cycles) was calculated for each of the six schedulers. From left to right, the schedulers compared are FCFS, SJF, RR, MLFQ, CGPTS, and GS. FCFS and SJF have the highest total execution time.

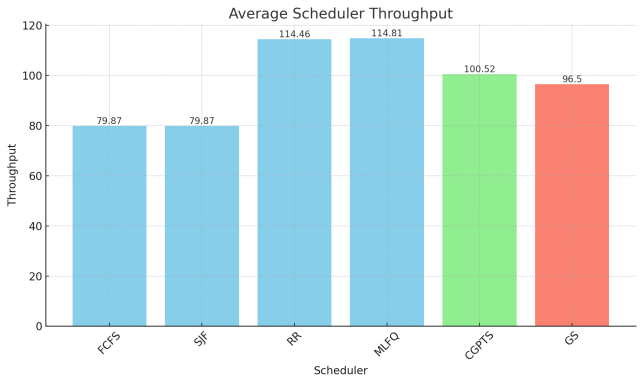

Figure 7. The average throughput of schedulers (estimated as cycles/second) was calculated for each of the six schedulers. From left to right, the schedulers compared are FCFS, SJF, RR, MLFQ, CGPTS, and GS. RR and MLFQ have the highest throughput.

5.4 Results discussion

When comparing the results of both of these experiments, a correlation can be found between the increased bias of the Gemini Scheduler and its overall increased waiting time, idle time, and turnaround time. This points to the fact that the overall performance of a system and the statistics for individual processes may be negatively impacted by the biases of the LLMs, though the performance of the two programs is still impressive overall. However, the Gemini scheduler also shows a lower response time compared to every other algorithm, which could be a result of it simply attempting to schedule the first available process in most cases.

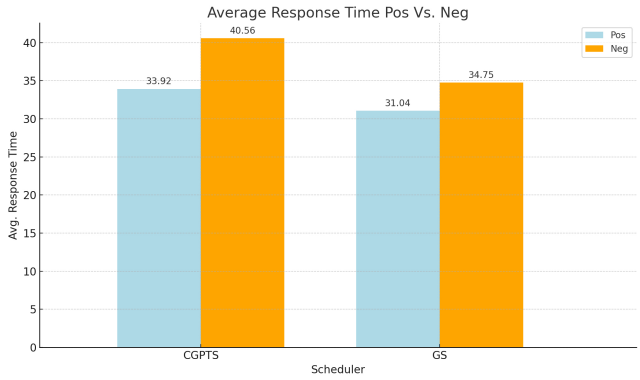

Figure 8. The average response times (in cycles) of processes with positive and negative titles were compared between the two LLM schedulers. From left to right, the schedulers compared are CGPTS (ChatGPT Scheduler) and GS (Gemini Scheduler).

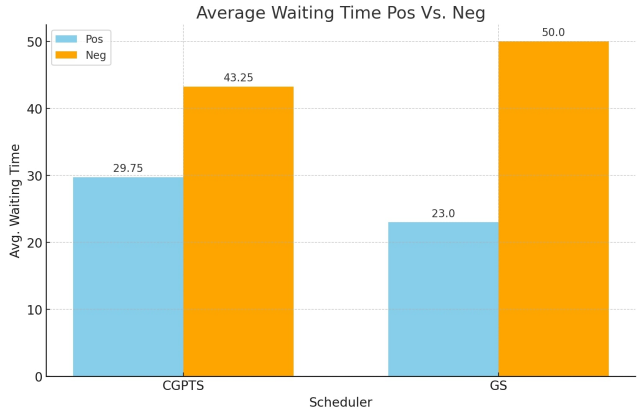

Figure 9. The average waiting time of processes with positive and negative titles was compared between the two LLM schedulers. From left to right, the schedulers compared are CGPTS (ChatGPT Scheduler) and GS (Gemini Scheduler.)

6 Future Work

The work presented in this paper provides a solid foundation for testing two LLM-based CPU scheduling algorithms, as well as presents results showing the potential performance upgrade that these schedulers may provide compared to other schedulers in terms of scheduling decisions. A basic evaluation on the bias that this practice may introduce is also performed. However, more work is desired for evaluating the different types of bias that some processes may encounter in a system like this.

In the future, more experiments exploring other types of bias aside from those that just have to do with positive or negative title connotations are desired. These other types of bias could come from more specific title constructs (perhaps including more risque topics such as race, gender, religion,

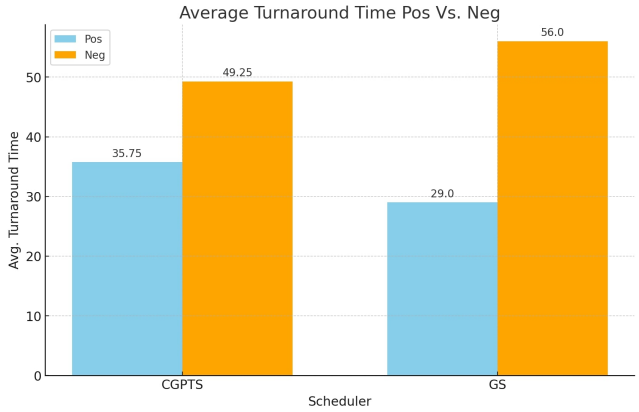

Figure 10. The average turnaround time of processes with positive and negative titles was compared between the two LLM schedulers. From left to right, the schedulers compared are CGPTS (ChatGPT Scheduler) and GS (Gemini Scheduler.

etc.), or from other attributes such as CPU times. A deeper exploration into more structured types of biases would provide a more informative view of the overall bias of the system.

Further improvements to the system could include increased performance and a more thorough data analysis procedure. Experiments against more complex CPU scheduling algorithms are also desired, though doing so is out of reach for the moment.

7 Conclusion

This paper presents the ChatGPT Scheduler and the Gemini Scheduler, two new scheduling algorithms that utilize LLM-prompting to make scheduling decisions within a CPU simulator. We provide an evaluation that shows impressive performance compared to several other algorithms, as well as a discussion about the potential biases that some processes may face from the LLMs' decisions.

We found that these new scheduling algorithms provide preferential treatment towards processes with titles of positive connotations, leading processes with negative titles to have higher response times, turnaround times, and waiting times compared to their counterparts. More work is required to gain insights into more specific types of biases that processes may face, but the results displayed here show the clear potential for bias due to certain task attributes. At the same time, the overall performance upgrades brought by these designs show the potential of these algorithms within a simulation setting, despite their bias.

References

[Sha22] Amir Shamsi. *CPU Scheduling Algorithm.* https://github.com/Amir-Shamsi/cpu-scheduling-algorithm/commits/master/?after=9e19d2286911f14503e1ee913f270ae406a21336+104. 2022.

[GSS24] Nathan Grinnell, Celeste Shen, and Rachel Shen. *CSE231 Final Project LLM Scheduler.* https://github.com/ngrinnel/CSE231-Final-Project. 2024.