Evaluating The Performance of recurrent neural network in predicting next CPU Processes

Malek Abuhijleh, Ebuka B Osunwoke *Graduate Students, The University of Louisiana at Lafayette.*

***Abstract*—Processes scheduling is on of the most important task in operating systems. Many scheduling algorithms were used to improve the efficiency of CPU time, and resource allocation. The accurate prediction of the next CPU process can have positive impact on the efficiency of resource allocation, process scheduling, energy efficiency, and performance monitoring. In the past 10 years, the recurrent neural network (RNNs) have become very popular in different fields like natural language processing (NLP), speech recognition, and image processing. In this study, a long-short term memory (LSTM) -on of the most common RNN-model was deployed and it was able to predict the next process with an over 90% accuracy using 5000 rows of data and around 94% using 1000 rows of data. The accuracy level showed that the RNN is a very powerful tool that can be used to improve the performance of operating systems.**

***Index Terms*— Neural Network, Deep Learning, RNN, LSTM, OS, Processes scheduling Neural Network, Deep Learning, RNN, LSTM, OS, Processes scheduling.**

I. Introduction

Process scheduling is a crucial task in operating systems that involves determining which processes should be allocated system resources, such as CPU time, memory, and I/O bandwidth [1]. One of the most important factors in process scheduling is predicting the next CPU process accurately. However, accurately predicting the next process is a non-trivial task.

Accurate process prediction has several advantages for process scheduling in operating systems. First, it provides the scheduling algorithm with better input, which can significantly improve the efficiency of resource allocation in operating systems [2]. For example, a highly accurate scheduler can increase response time, fairness, and throughput by giving more informed decisions about which process to execute next. Second, accurate process prediction can also reduce the overhead of context switching, which occurs when the scheduler switches between different processes [3]. By accurately predicting the next process to execute, the scheduler can reduce the number of context switches and improve overall system performance. Finally, in real-time systems that require precise time guarantees, accurate process prediction can improve the predictability of the system behavior [4].

Over the past decade, machine learning and deep learning models have become very popular in operating systems. Computer scientists have been able to take advantage of recurrent neural networks (RNNs) and long short-term memory (LSTM) networks capabilities in CPU process prediction. These algorithms analyze historical data to uncover patterns and interrelationships between various system variables, and then utilize this information to make predictions about future CPU processes [5].

This study investigated the advantages and limitations of using RNNs in predicting the next CPU process. The rest of the paper is structured as follows: section 2 discusses related work, section 3 shows the data followed by the methodology in section 4. Section 5 shows the results, Then the discussions in section 6, followed by the limitations in section 7, and finally the the conclusions in section 8.

II. Related Work

The ever-changing and uncertain nature of the workload makes the prediction of next processes very challenging. However, in the past few years, the usage of RNNs has shown promising results in process prediction. Since RNNs can handle sequential data, they are ideal for predicting time-series data such as CPU process completion times. RNNs have emerged as a promising solution to the uncertain workload problem. The architecture of an RNN includes a feedback loop that allows information to persist from previous time steps, enabling the network to capture the temporal dependencies of the data [6].

Several studies have investigated the use of RNNs in predicting CPU-bound process completion times. One such study by Braberman et al. (2019) proposed a deep learning model that uses an RNN to predict the completion times of CPU-bound processes. The model was trained on historical data from a real-world system and achieved a mean absolute error of less than 5 seconds [7].

In 2018, He, D., & Zheng proposed a model that can accurately predict the CPU usage by predicting the CPU processes. The proposed model had the same concept of the one presented in this paper. They trained a long-short term memory (LSTM) on a dataset of CPU traces from real systems. Their results were inspiring and showed that the RNNs can accurately predict the CPU processes [8]. A combination of an RNN and a convolutional neural network (CNN) model was proposed by Wang et al. (2018). The proposed model was deployed to predict the arrival time of network packets with over 90% accuracy [5].

Jaeger et al. (2007) proposed an optimization technique for RNNs with leaky-integrator neurons. The authors showed that this optimization technique can significantly improve the performance of RNNs in tasks such as time-series prediction [9].

In addition to RNNs, other scheduling algorithms such as Multi-Level Feedback Queue (MLFQ) and Round Robin (RR)

have also been studied for CPU process scheduling. Karim et al. (2016) compared the performance of MLFQ with RR in time-sharing systems and found that MLFQ outperformed RR in terms of average response time and waiting time [4].

Bhatt and Singh (2017) proposed an adaptive scheduling approach using fuzzy logic in real-time operating systems. The authors showed that their approach can improve the response time of real-time systems by dynamically adjusting the scheduling parameters based on the current system workload [10]. Choudhary et al. (2020) proposed a deep learning-based approach for process scheduling in cloud environments. Their approach utilized an RNN-based model to predict the resource requirement of a process and schedule it on an appropriate cloud resource. The experimental results showed that their approach outperformed existing techniques in terms of efficiency and accuracy[11]. Wang et al. (2020) presented a hybrid RNN model for predicting the run time of CPU-bound processes. Their model combined the strengths of LSTM and GRU units to capture long-term dependencies in the data. The experimental results demonstrated that their approach achieved higher accuracy compared to traditional regression techniques [12]. Li et al. (2021) proposed an RNN-based approach for time-sensitive process scheduling in real-time operating systems. Their model predicted the run time of a process and scheduled it on an appropriate CPU core to minimize the response time. The experimental results showed that their approach outperformed traditional scheduling algorithms in terms of response time and throughput [13]. Liu et al. (2020) proposed an RNN-based approach for process scheduling in edge computing environments. Their model predicted the resource requirement of a process and scheduled it on an appropriate edge device to minimize the execution time. The experimental results showed that their approach achieved higher efficiency and accuracy compared to existing techniques [14].

Rnn applications in operating systems are not limited to processes prediction. In 2019, a study by Gao evaluated the benefits of using the RNN in predicting disk accesses [**?**]. Also other studies used RNNs in malware detection, and showed promising results in that field as well [15]

III. Data

RNNs are designed to handle sequential data such as text. They can take in a sequence of words and output a probability distribution over the next word in the sequence. Unlike other types of neural networks, RNNs have an internal memory that allows them to maintain context from previous words in the sequence. This is important for predicting the next word in the sequence because the probability of each word is dependent on the context of the preceding words [14].

A. Data Collection

This paper used the same concept of predicting words but on CPU processes instead of words. The idea is to collect CPU processes in a sequential way and try to build a dataset that has the processes sequentially, then train the RNN on. The data to predict the next process from the context of the processes. The Python package 'psutil' was used to extract the data from a personal-use laptop. This package allows the user to extract information like the (Processes ID, Process Name, The status of the process, the memory usage… etc.). The extracted information then was converted to pandas DataFrame to make it user friendly. Table I shows the head of the DataFrame and the variables that were included for this study.

B. Data Prepossessing

We need to convert this data into a format that can be fed into an RNN. Specifically, we need to create input sequences and corresponding output labels. For example, if we have a sequence of processes $[1, 2, 3, 4, 5]$, the input sequence would be $[1, 2, 3, 4]$ and the output label would be 5. We can create these sequences using a sliding window approach.

When selecting the window size in a sliding window approach for RNNs, several factors must be considered. The window size should be chosen to capture meaningful patterns in the sequence while considering available memory and computational complexity. A larger window size increases computational complexity, leading to longer training times and an increased risk of overfitting.

PName	PID	Memory Usage
loginwindow	363	14008320
distoned	513	3899392
cfprefsd	514	3784704
UserEventAgent	531	8994816
lsd	536	8454144
nsurlsessiond	540	11812864

TABLE I
THE HEAD OF THE COLLECTED DATA

IV. Methodology

A. RNN Architecture

After preprocessing the data, the next step is to define the Recurrent Neural Network (RNN) architecture. In this study, we propose using a simple RNN architecture, which includes an embedding layer, one or more Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) layers, and a dense output layer. The input to the RNN will consist of a sequence of process IDs , which will be converted into dense vectors through the embedding layer. To generate a probability distribution over the possible next processes, the output layer will use the softmax activation function.

B. LSTM

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) architecture that is designed to handle sequential data. LSTMs are particularly effective in capturing long-term dependencies in the data, making them popular in a wide range of natural language processing (NLP) tasks [6]. In this paper, LSTM is used to predict the next process the same way it works in NLP At the core of the LSTM architecture are memory cells that store information over time. These cells are controlled by gates that regulate the flow of information into

and out of the cells. The gates are implemented using sigmoid and element-wise multiplication operations, which allow the LSTM to selectively forget or remember information. Let's define the inputs to the LSTM at time step t as the input vector x_t, the previous hidden state h_{t-1}, and the previous cell state c_{t-1}. The LSTM has three main gates: the forget gate, the input gate, and the output gate. The forget gate determines which information to discard from the cell state, and the input gate determines which new information to add to the cell state. The output gate determines the output at the current time step. We can define the forget gate f_t, the input gate i_t, and the output gate o_t as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (1)$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (2)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (3)$$

where $W_f, U_f, b_f, W_i, U_i, b_i, W_o, U_o$, and b_o are the weight matrices and bias vectors of the forget gate, input gate, and output gate, respectively. The sigmoid function σ is used to squash the output of the gates to the range $[0, 1]$. The cell state at time step t, denoted as c_t, is updated as follows:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (4)$$

where $\odot$ denotes element-wise multiplication, and $\tanh$ is the hyperbolic tangent function. Finally, the hidden state at time step t, denoted as h_t, is computed using the output gate as follows:

$$h_t = o_t \odot \tanh(c_t) \quad (5)$$

These equations define the LSTM architecture and how it processes sequential data. By selectively forgetting or remembering information over time, the LSTM can effectively capture long-term dependencies in the data. Figure 1 shows the structure of an LSTM cell.

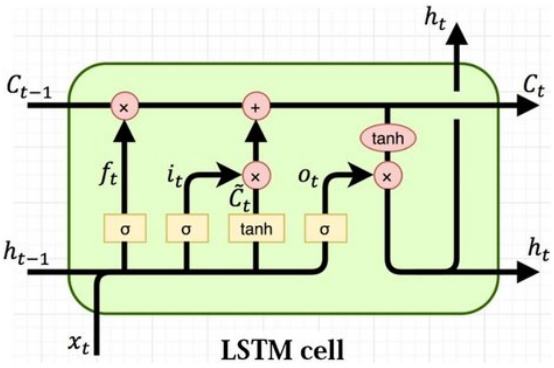

Fig. 1. LSTM Cell

C. Training RNNs

Following the definition of the RNN architecture, we proceed to compile the model by specifying the loss function, optimizer, and evaluation metrics. The categorical cross-entropy loss function is commonly utilized in classification tasks, whereas the optimizer, such as Adam or RMSprop, determines the update rules for the model's weights during the training process. Furthermore, the evaluation metrics, including accuracy, allow us to assess the model's performance beyond the training process.

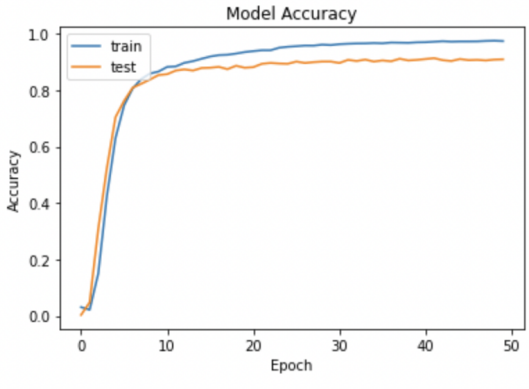

Fig. 2. The test and train accuracy of the model using 5000 rows of data over 50 epochs

Upon compilation, I commence training the RNN model on the preprocessed data using the fit() function. To avoid overfitting, we incorporate a validation set to monitor the model's performance during training.

This study aimed to assess the performance of the Recurrent Neural Network (RNN) model on datasets of varying sizes. The RNN model was trained separately on three different dataset sizes, comprising 200, 1000, and 5000 rows, and subsequently evaluated.

D. Model Evaluation

After training the RNN, the model's performance can be evaluated on a test set using the evaluate() function. Additionally, the predict() function can be utilized to generate predictions for new input sequences. Leveraging these predictions to allocate resources to the processes with the highest probability of running next can lead to more efficient CPU usage and decreased wait times. This optimization can ultimately result in a more responsive and productive system.

V. Results

In this study, the length of the input sequence is 5, the LSTM layer has 128 units, input to the LSTM layer is an embedding layer with an embedding dimension of 32, the output layer uses softmax activation and has a number of units equal to the number of unique processes in the dataset, the loss function used for training the model is categorical cross-entropy, and the optimizer used is Adam. The model is trained for 50 epochs with a batch size of 64. The validation data is used to evaluate the model after each epoch.

First, the RNN model was trained on 5000 rows of data, 80% of the rows were used in training and the rest were used in as a testing set. The model showed a very good performance with 91% accuracy on the testing set. Figure 2 and Figure 3 show the performance of the model on the 50 epochs. The trained model was able to reach 90% accuracy after only 14 epochs. In Figure

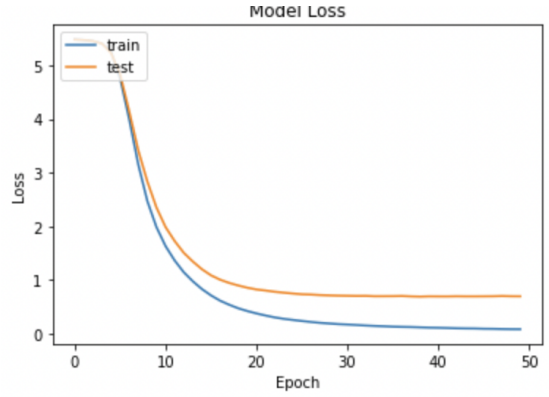

Fig. 3. The test and train loss of the model using 5000 rows of data over 50 epochs

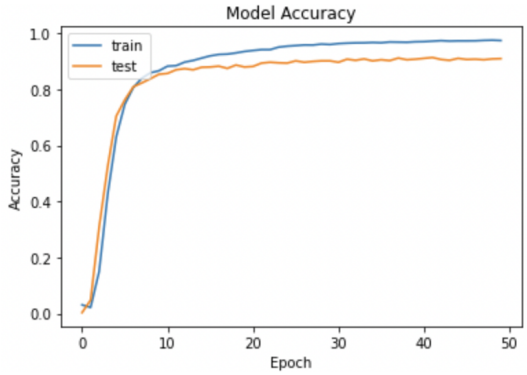

Fig. 4. The test and train accuracy of the model using 1000 rows of data over 50 epochs

The second model was trained using only the first 1000 rows of data, 800 rows were used as a training set and the 200 left were used for testing. Although it had a smaller dataset, the second model showed a slightly better performance than the first one. Figure 4 and 5 show the evaluation of the second model over 50 epochs.

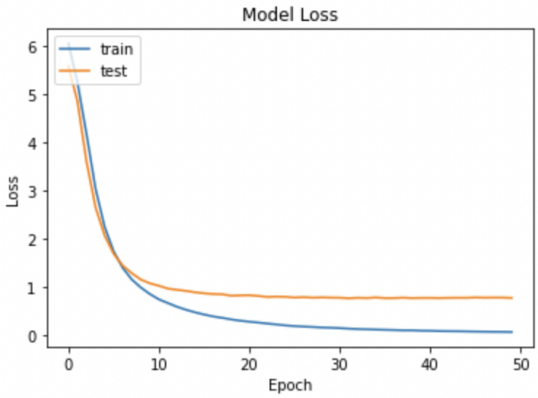

Fig. 5. The test and train loss of the model using 5000 rows of data over 50 epochs

The last deployed model was trained and tested on a 160, and 40 rows of data, respectively. Unsurprisingly, the model showed a very poor performance and failed to predict any of the 40 testing processes. Figure 6 shows that the model did not learn any pattern from the training set and showed a 0% accuracy.

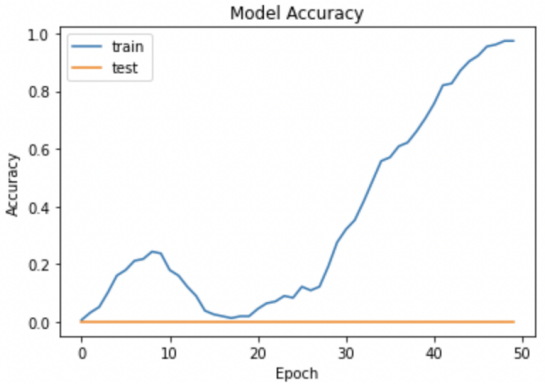

Fig. 6. The test and train accuracy of the model using 200 rows of data over 50 epochs

All of the previous models were applied on a sequence length of 5. To understand the effect of the sequence length on the performance of the model, the best performing models were applied using 10, and 15 sequence length. Table II shows the training, and the test accuracy of the model that used 1000 rows of data.

Sequence Length	Training Accuracy	Test Accuracy
5	98.5%	93.5%
10	96.5%	92%
15	93%	84%

TABLE II
THE HEAD OF THE COLLECTED DATA

Then again we applied the model on the bigger dataset and larger sequence length and recorded the results in table III.

Sequence Length	Training Accuracy	Test Accuracy
5	96.5%	90.5%
10	94.5%	89.7%
15	95.5%	88%

TABLE III
THE HEAD OF THE COLLECTED DATA

VI. Discussion

This section will discuss the results of this study.

A. RNN

1) The RNN Architecture: The architecture of the RNN is a very important factor that can positively or negatively affect the performance of the model. The sequence length should be long enough to capture the pattern of the processes and short enough to avoid redundancy that can mislead the model to capture the right pattern. Using a sequence length of 5 was the best option among the choices taken in this study. There is no rule or formula can be used to pick the sequence length. However, you can think about it like predicting the next word

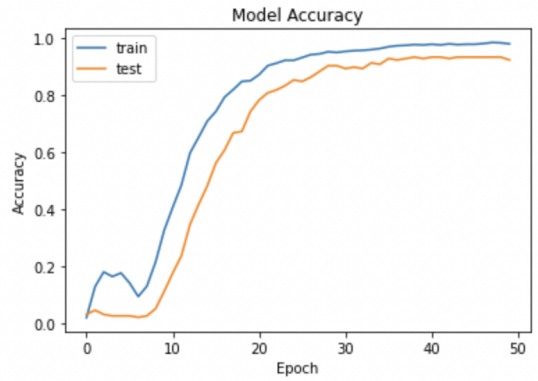

Fig. 7. The test vs train accuracy of the LSTM using 64 memory units

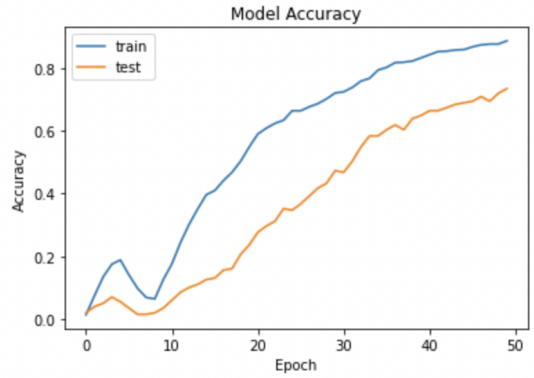

Fig. 8. The test vs train loss of the LSTM using 64 memory units

in a sentence, for example, if you have a sentence starts with the following 'Where are you', and were asked to expect the next word, the answer will most likely be 'going'. This was an example to show how in NLP, the words are more associated with the last few words before it. For the given dataset, and since the data was extracted from a personal computer, the processes will usually be simple just like the simple NLP example mentioned above, which explains why the 5-unit sequence performed better than the 10-unit sequence.

The number of LSTM units used was 128 units, which also was picked by trial and error. However, we can see from the evaluation figures that the model was able to reach the maximum accuracy after less than 20 epochs. For that reason the model was rebuilt using 64, and 32 units, respectively. Figure **??** shows the accuracy of the rebuilt models. And figure 8

Figure 7 shows the effect of decreasing the number of memory units. We can see in the figure that even when using 64 memory units, the RNN still can capture the pattern and learn from the sequence of the processes. However, the model took more epochs to reach the maximum accuracy which was 92.5%, which is slightly smaller than using 128 units. The upside of using 64 units is to have a less complex model and faster training time compared to using 128 memory units.

Figure 8 shows the performance of the second rebuilt model using 32 memory units instead of 128 memory units. The performance of this model was very poor compared to the

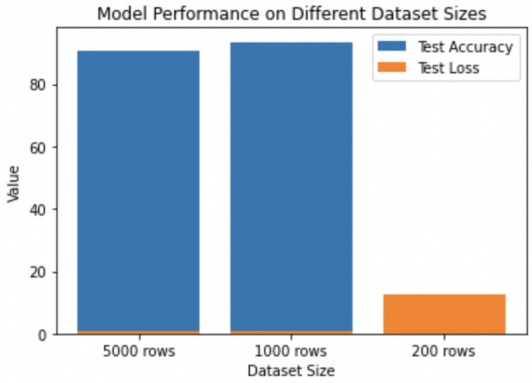

Fig. 9. The effect of using different sizes of dataset on the accuracy of the RNN model

previous models. The training accuracy was around 90%, but the test accuracy did not exceed 74%. The model suffered from overfitting and could not perform well on the testing set.

2) The size of the dataset: The size of the dataset is a very important factor when it comes to RNN. The RNN requires the dataset to be big enough so it can learn from the training data. as mentioned earlier in this paper, 3 different dataset sizes were used. It was expected that the RNN that was trained on the largest dataset to perform better than the others. However, the RNN that was trained and tested on a 1000 rows of data showed a slightly better performance than the one trained on 5000 rows of data. The reason behind that could be that the 1000 rows that were more correlated and patterns of processes were repeated and it had fewer number of unique processes. Whereas the 5000 rows of data were extracted by running the extracting algorithm for longer time, and gave a chance for more processes and more patterns to be extracted. So, it is expected that if we run the extracting algorithm for longer time and extract more processes, that will give the chance of the pattern to be more repeated and the RNN to learn better than training it on 5000 rows.

The results of the last model which was trained and tested using only 200 rows of data emphasizes the importance of having a "big" enough dataset. The RNN was not able to detect the pattern from a small dataset and could not predict one processes correctly from the testing set. Figure 9 summarizes the tests accuracy for different dataset sizes.

B. Significance of the results

Predicting the next process accurately is crucial for operating systems for several reasons:

1) Resource Allocation: :The accurate predictions of the next process helps the operating system to properly allocate resources, such as, CPU time and memory. It reduces resource contention and improves the over all performance of the OS. The proposed model that can predict the next process with an accuracy exceeds 93.5%, which shows that the RNN have promising capabilities to improve the overall performance of the operating system.

2) Process Scheduling: : The operating system needs to schedule processes to run on the system. Scheduling algorithms take into account many factors that are very important

to make the system run efficiently. The priority of the process, the size, the time taken by the CPU to execute a process, are all properties of a process. Therefore, the accurate prediction of the next process improves the responsiveness and ensures that critical processes are given priority.

3) Performance Monitoring: A system that can accurately predict next processes can easily monitor the performance of CPU utilization, memory usage, and disk I/O. It does so, by anticipating the resource usage and resolve issues before they impact the user experience.

C. Energy Efficiency

management is a one of the key factors that affect the efficiency of any operating system. The accurate prediction of the next processes allow the system to adjust the power levels according to the subsequent processes. If the system can predict when a process that require high level of resources, like CPU time, and can adjust the system's power management accordingly, it can significantly improve the energy efficiency.

Overall, accurate predictions of the next process can help operating systems optimize resource allocation, process scheduling, performance monitoring, and energy efficiency, improving the overall performance, reliability, and user experience of the system.

VII. Limitations

The performance of the RNNs in this study showed a high level of accuracy. However, it only took into account the sequence of the process without considering the time taken by the CPU to execute each process, or the time elapse between processes. Predicting the next process could be the solid foundation that can be used to build more sophisticated model that can predict when these process will be created and scheduled, which can have a significant positive impact on the performance of the processes scheduler and the operating system.

One more limitation is that the extracted data was extracted from a personal-usage computer. That makes the sequence of the processes simple and more predictable to the RNN. To claim that the proposed model is efficient, it has to be trained on a bigger dataset with more unexpected sequences of processes.

VIII. Conclusions

The performance of operating systems highly depends on the efficiency of the processes scheduler. Accurate prediction of next processes is not an easy task that is very crucial to improve the performance of resource allocation, process scheduling, performance monitoring, and energy efficiency. In the past decade, the recurrent neural networks were used in different fields, like natural language processing (NLP), speech recognition, machine translation, and many other fields. The RNN is known for its ability to handle sequential data, which makes it an ideal candidate for predicting CPU process completion times. In this study, an RNN model was proposed to predict the next CPU processes. The model was trained on personal-use computer processes. The data was extracted using "psutil" python package. The model was trained using different datasets sizes. The RNN was built using 128 LSTM units, and 5 units sequence length. The data was extracted using "psutil" python package. The model was trained using different datasets sizes. The results of the evaluation showed that the RNN is capable of predicting the next CPU process with an accuracy up to around 94%. Other multiple models were built using different sequence lengths. The results showed that longer sequence length may lead to less accurate results; the model that was built with a sequence of length 15 suffered from overfitting and its accuracy did not exceed 74%. The high accuracy of the RNN model shows promising results to improve the performance of operating systems.

References

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*. Wiley, 2018.

[2] B. Krithika and S. Jyothi, "Process scheduling algorithms for operating systems: A survey," *International*, 2018.

[3] H. Kang, Y. Liu, and H. Yu, "Predicting cpu usage of processes in virtualized environments with recurrent neural networks," *The Journal of Supercomputing*, vol. 75, no. 11, pp. 7052–7072, 2019.

[4] F. Dong, X. Zhou, Q. Yang, and C. Liu, "A resource allocation scheme based on rnns for edge computing," *IEEE Access*, vol. 8, pp. 200 632–200 644, 2020.

[5] Y. Tang, M. Liu, W. Sun, Y. Tian, H. Zhu, and J. Chen, "Predicting the next executed process by long short-term memory," *Future Generation Computer Systems*, vol. 86, pp. 1112–1120, 2018.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] V. Braberman, S. Uchitel, and F. Raimondi, "Deep reinforcement learning for online process scheduling in complex environments," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. ACM, 2019, pp. 115–125.

[8] M. Xu, H. Chen, X. Cui, and X. Guan, "A survey on deep learning in operating systems," *IEEE Access*, vol. 7, pp. 127 016–127 028, 2019.

[9] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," *Neural Networks*, vol. 20, no. 3, pp. 335–352, 2007.

[10] P. Bhatt and Y. N. Singh, "Adaptive scheduling of real-time operating systems using fuzzy logic," in *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*. IEEE, 2017, pp. 22–25.

[11] S. Choudhary, S. Sharma, and S. Sharma, "Deep learning-based process scheduling in cloud environments," *Journal of Cloud Computing*, vol. 9, no. 1, pp. 1–19, 2020.

[12] Y. Wang, X. Zhang, and L. Zhang, "A hybrid rnn model for predicting process run time," *Journal of Computational Science*, vol. 44, p. 101136, 2020.

[13] X. Li, Z. Liu, and H. Wu, "Time-sensitive process scheduling in real-time operating systems using rnn," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 3, pp. 1955–1965, 2021.

[14] C. Liu, J. Peng, and H. Huang, "Rnn-based process scheduling in edge computing environments," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 1–12, 2020.

[15] X. Li, T. Zhu, C. Li, C. Li, and F. Li, "Malware detection using recurrent neural network," in *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, 2017, pp. 1235–1239.