SmartOS: Automating Allocation of Operating System Resources to User Preferences via Reinforcement Learning

by

Sepideh Goodarzy

B.S., University of Tehran, 2017

M.S., University of Colorado Boulder, 2020

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2023

Committee Members:

Richard Han, Chair

Eric Keller

Qin Lv

Shivakant Mishra

Eric Rozner

Goodarzy, Sepideh (Ph.D., Computer Science)

SmartOS: Automating Allocation of Operating System Resources to User Preferences via Reinforcement Learning

Thesis directed by Prof. Richard Han

Operating systems currently leverage a one-size-fits-all method for resource allocation that does not automatically identify and adapt to user preferences. This thesis explores a machine learning-based operating system called SmartOS that learns individual user preferences, gathering data and feedback from the user, and then adjusts its resource allocation policy accordingly. The feasibility of using machine learning to learn and adapt user-based allocation of computer resources such as processing, memory, storage, and network bandwidth is studied through a user study and implementation in the Linux operating system, and different reinforcement learning strategies are evaluated to determine the fastest convergence of the operating system to the optimized resource allocation policy solution.

Contents

Chapter

1 Introduction **1**

2 Related work **4**

2.1 Machine Leaning in Operating systems 4

2.2 Understanding User Emotions 5

3 SmartOS: A Study of Reinforcement Learning Using Synthetic Data **8**

3.1 Overview 8

3.1.1 Actions 11

3.1.2 Rewards 11

3.2 Prototype 11

3.2.1 Resource allocation 12

3.2.2 Reinforcement Learning algorithm 13

3.3 Evaluation 13

3.3.1 Foreground contended with resource-intensive background 16

3.3.2 Foreground under-performed due to a dependent application 16

3.3.3 Multiple important applications with needs in separate dimensions 18

3.3.4 Variation of dimensions 20

3.3.5 SmartOS dynamicity and convergence 22

3.4 Conclusions 24

4 Human Frustration Prediction **25**

4.1 Application Overview . . . 26

4.1.1 Application Usage Procedure . . . 26

4.1.2 Data Collection . . . 29

4.2 Architecture . . . 32

4.2.1 Application Architecture . . . 32

4.2.2 Data Cleaning and Preprocessing . . . 32

4.2.3 Model Architecture . . . 36

4.3 Evaluation . . . 41

4.3.1 Parameter Search . . . 45

4.3.2 Models' Comparison . . . 45

4.4 Conclusion . . . 45

5 SmartOS: Improving Reinforcement Learning Algorithm Convergence Using Pre-training **46**

5.1 Pre-training SmartOS Reinforcement Learning algorithm . . . 47

5.2 Reinforcement Learning Model Architecture . . . 48

5.3 Evaluation . . . 54

5.4 Conclusion . . . 54

6 Conclusion and Future Directions **56**

6.1 Conclusion . . . 56

6.2 Future Direction . . . 58

Bibliography **60**

Tables

Table

3.1 Different Reinforce Learning methods convergence in a dynamic setting. 20

Figures

Figure

3.1 Learning-based SmartOS architecture. 9

3.2 Foreground app contended with a resource-intensive background app. 15

3.3 Foreground under-performed due to a dependent application. 17

3.4 Multiple important applications with resource needs in separate dimensions. 19

3.5 Variation of dimensions. 21

3.6 Convergence of Monte Carlo in a dynamic setting (each episode consisted of 4 feedbacks). 23

4.1 SmartOS Application Central Page 26

4.2 SmartOS Application First Survey Form 27

4.3 SmartOS Application Report Form 27

4.4 SmartOS App: Data Collection for Human Frustration Prediction Architecture . . . 33

4.5 Human Frustration Prediction Supervised Model Architecture 1 38

4.6 Human Frustration Prediction Semi-supervised Model Using Using Generative Adversarial Network 40

4.7 Human Frustration Prediction Supervised Model Architecture 2 42

4.8 Human Frustration Prediction Dropout rate Search 43

4.9 Human Frustration Prediction L2 regularizer Parameter Search 43

4.10 Human Frustration Prediction Models Comparisons 44

5.1 Pre-training The Reinforcement Algorithm in the SmartOS Using Third Methodology 47

5.2 Actor Model Architecture in The SmartOS A2C Reinforcement Learning Model . . . 49

5.3 Critic Model Architecture in The SmartOS A2C Reinforcement Learning Model . . . 50

5.4 Reinforcement Learning Algorithm Models' Average Convergence 53

6.1 Architecture of Active SmartOS App, including App and Pre-trained A2C Algorithm Located in Cortex Server 57

Chapter 1

Introduction

As modern humans, we deal with computer operating systems (OSs) in our daily lives when we are at work, home, school, or even watching a TV show with our family at night. Operating systems are everywhere. They exist in our desktop computers, laptops, cellphones, cars, TV, smartwatches, etc. According to studies, there are 4.88 billion Internet users worldwide who use the devices mentioned above with operating systems installed on them. [5] Thus, they play an undeniably important role in our life quality.

The main job of the operating system is to manage and distribute the computer resources among the applications. Unfortunately, today's user-facing OSs, such as laptop, mobile, and desktop OSs, are typically designed with a one-size-fits-all approach to resource management. For example, the Linux Completely Fair Scheduler (CFS) effectively divides up the CPU equally among all processes, assigning them the same static priority [3, 44]. In contrast, users interacting with an OS often move from task to task and application to application, wanting sufficient resources devoted to their current task, such as editing a document, chatting via zoom, or listening to music. In addition, users often have many applications open simultaneously, some from previous tasks that may be resumed in the near future, including multiple tabs in a browser persistently refreshing their content, resulting in a landscape of many applications continuing to consume system resources. In this context, the current approach of static prioritization often fails to devote sufficient resources to the applications that the user cares most about at that moment, resulting in degraded performance. For example, many users experience a slow down on their computer when numerous open applica-

tions collectively act as CPU hogs or memory hogs, in some cases due to runaway processes, and interfere with the tasks that the user deems most important. Furthermore, other processes occupy enough network bandwidth to interfere with real-time audio/video, including software updates and cloud synchronization. Anyone who has tried to conduct a Zoom call while there is another local network-hogging application understands this difficulty. Ideally, a next-generation OS would benefit the user and alleviate these bottlenecks by being able to learn what applications the user currently considers to be most important and adaptively prioritizing the allocation of resources to those applications.

This thesis explores the role of machine learning in the design and implementation of next-generation operating systems, harnessing the exploding interest in artificial intelligence and machine learning to improve the user experience through automated learning and resource allocation in the OS. An investigation is conducted on how the OS may be structured to accommodate learning-based management of joint resource allocation for memory, processing, input/output (I/O), and network bandwidth in response to user behavior. This thesis considers which machine learning algorithms may most effectively integrate and learn from user behavior. This thesis also seeks to understand whether there is any net benefit in performance to applying machine learning in OS design and if so, to quantify the benefits.

The challenges faced are significant. The context that governs what the user values as most important at any given time is complex and challenging to learn. For example, a user may currently be engaged in editing a document, and would prefer to have CPU (central processing unit) and memory resources prioritized towards editing. Adding complexity, the user may also be streaming a music video from a cloud provider simultaneously and wish to listen to music while editing. This different modality should also receive high priority in terms of CPU, memory, and notably, network bandwidth. Static prioritization policies become difficult to craft as we consider such increasing complexity. Heightening the complexity, each application may consist of multiple dependent communication processes, which would need to receive elevated allocation as a group.

This thesis document is organized as follows. First, Chapter 2 studies the related work.

Then, Chapter 3 shows the feasibility of a learning-based operating system that automates resource management based on user preferences through reinforcement learning using synthetic scenarios and envisaged user interactions. Chapter 4 describes the prediction of user frustration with the computer using machine learning based on real-world user interaction with the computer. Chapter 5 leverages this prediction to develop a new pre-trained reinforcement learning algorithm and evaluates its convergence in automating resource management based on user preferences in real scenarios from a human study compared to an untrained reinforcement learning algorithm.

Thesis statement: This thesis shows that reinforcement learning can learn real-world user preferences and adjust resource allocation in the operating system to improve the user experience. The contributions of this thesis are as follows:

- Evaluation of the effectiveness of learning-based operating system using reinforcement learning algorithm based on synthetic user preferences compared to other common heuristics
- Developing a machine learning model predicting user frustration with their computer based on real-world data
- Evaluating the convergence of a learning-based operating system based on real- world user preferences using a pre-trained reinforcement learning algorithm vs. untrained reinforcement learning algorithm

Chapter 2

Related work

The related works in the literature are divided into two main categories. The first category will include improving the operating system using machine learning, heuristics, etc. The second category will be the works only focused on ascertaining the user's feelings.

2.1 Machine Leaning in Operating systems

Limited performance in today's operating systems has become enough of a concern that machine learning (ML) has begun to be applied to improve application execution. Perhaps the most closely related to this work is the recent Acclaim system that seeks to improve user experience in the Android OS by predicting what memory pages will be used next by employing machine learning [42]. Acclaim assumes that the most important applications are the foreground application and audio/video apps, and statically prioritizes page reclamation for these applications. However, as we show later, static prioritization is not ideal in various circumstances. In addition, machine learning has been applied to improve Linux process scheduling [52, 14], I/O scheduling [46, 34], and network cloud systems [21]. Still, each only considers one resource dimension, rather than jointly allocating CPU, memory, networking, and I/O, and also do not learn user preferences for resource allocation. The following research reviews the future possible directions of learning based OSes and their challenges, however, they have not implemented none of their ideas. [69]

Some previous work has combined machine learning with control theory to preserve the quality of service while minimizing energy consumption [48]. The issue with this method is that a

conservative configuration was used until enough data was gathered for offline learning. Then, once training was finished, the resource configuration was changed in their control system. As a result, this method can not be used in an interactive environment with the user since, despite the user giving the system feedback, the system will not change the configuration of resources in a computer until it has enough data to do offline training. Other works in the literature used control theory to find the sweet spot of accuracy and performance trade-off space in dynamic environments [29, 30]. Still, their method is unsuitable for our problem space as the goal in our problem space is subjective (user preference) and changes more frequently than objective goals such as application performance and energy consumption which are the focus of those works. This is because control theory is based on modeling the environment. However, reinforcement learning methods in machine learning are also applicable to environments where there is a lack of knowledge to be modeled perfectly (model-free reinforcement learning).

2.2 Understanding User Emotions

What is emotion? There are a lot of definitions of human emotions in the literature [32]. One explanation focuses on two features of emotions. First, emotions are the reactions to the events that involve one's goals, needs, or concerns. Second, emotion contains physiological, affective, behavioral, and cognitive components [8]. Emotions differ from other feelings, such as mood, a long-lasting user state that will bias the user's responses to the events. Emotion is an intentional reactive response to objects, while the user's mood is an unintentional non-reactive state. Moods can change the thresholds of the user's emotions, and repetitive emotions can cause moods. There are also some confusions between sentiments and emotions. Sentiments are mostly judgments of objects based on one's own experience or other's experience, while emotions are the actual response to an event after an event's occurrence. To clarify these two, one example of sentiment is "I like receiving emails" vs. the emotion is "I got happy after receiving this particular email." [18]

There are two main types of emotions: primary and secondary emotions. Primary emotions are the more impulsive ones, such as fear, and secondary ones go through cognitive procedures

such as frustration, pride, and satisfaction. The secondary emotions are the main focus of Human-Computer Interaction (HCI) [8].

Many works in the literature focused on human emotion prediction using different modalities [15] :

- Firstly, audio and visual-based input modalities including eye gaze tracking, facial expressions, body movement detection, and speech and auditory analysis.
- Secondly, physiological input modalities using sensor-based signals, such as electroencephalogram (EEG), galvanic skin response, and electrocardiogram.
- Alternatively, extra inputs may be gained by interpreting user behavior with mouse movements, keyboard keystrokes, content viewing, or even combining all of these different modalities.

Some works specifically focused on emotion prediction in human-robot interaction, which do not apply to our problem of predicting human frustrations with their computers because the nature of the human-computer interaction is different from that of the human-robot interaction. Humans see computers as task-oriented devices while they perceive robots as intelligent beings capable of interacting with humans and understanding human emotions [56, 59]. This work [59] tries to predict human-feeling prediction based on their facial expression solely. They showed that because of the differences between human-computer interactions and human-human interactions, methods that proved to be working in emotion prediction in human-human interactions are not applicable in human-computer interactions. On the other hand, some other works focused explicitly on the co-relation of mouse movement speed and direction and could predict emotions in human-computer interaction [27]. Still, they didn't focus on the specific emotions related to human frustration with the computer as focused on in this thesis. Some other works studied the causes of human frustration with their computers, which showed that the main reasons are error messages, dropped network connections, lengthy download times, and hard-to-find features. An optimal resource allocation policy can prevent dropped network connections and long download times. [37, 10, 35, 36] .Thus,

these will use the computer's internal state, such as running applications' resource usage profiles and remaining resources in the computer, which directly impact the resource allocation policy and thus will affect the user's frustration with the computer. This thesis will also use other modalities in combination, such as keyboard strokes, mouse movements' speeds and directions, and users' heads' movements and audio of the users, which were shown co-related with the users' emotions [55].

Chapter 3

SmartOS: A Study of Reinforcement Learning Using Synthetic Data

This chapter of the thesis introduces SmartOS, a learning-based operating system that takes the user interactions with the OS into account to perform automated resource management. SmartOS leveraged reinforcement learning to continuously gather feedback from the environment and change the resource parameters exposed by the Linux operating system's kernel to improve the user experience. The contributions of this chapter are as follows:

- An architecture is described that integrated machine learning into the OS as a user space module controlling allocation of memory, CPU, network, and I/O.
- Reinforcement learning (RL) was utilized to solve the difficult challenge of learning the user's context and applying the appropriate resource allocations.
- The RL algorithm was able to rapidly converge to the desired allocation of system resources, and its overhead was modest.

3.1 Overview

Figure 3.1 depicts an overview of the SmartOS. The main learning component of the system is the Cortex, which is the "brain" of the SmartOS and is responsible for monitoring the application status, user context, and the user's interaction with the computer. In response to this state information, the Cortex determines which resource allocation policy is the best to satisfy the user's

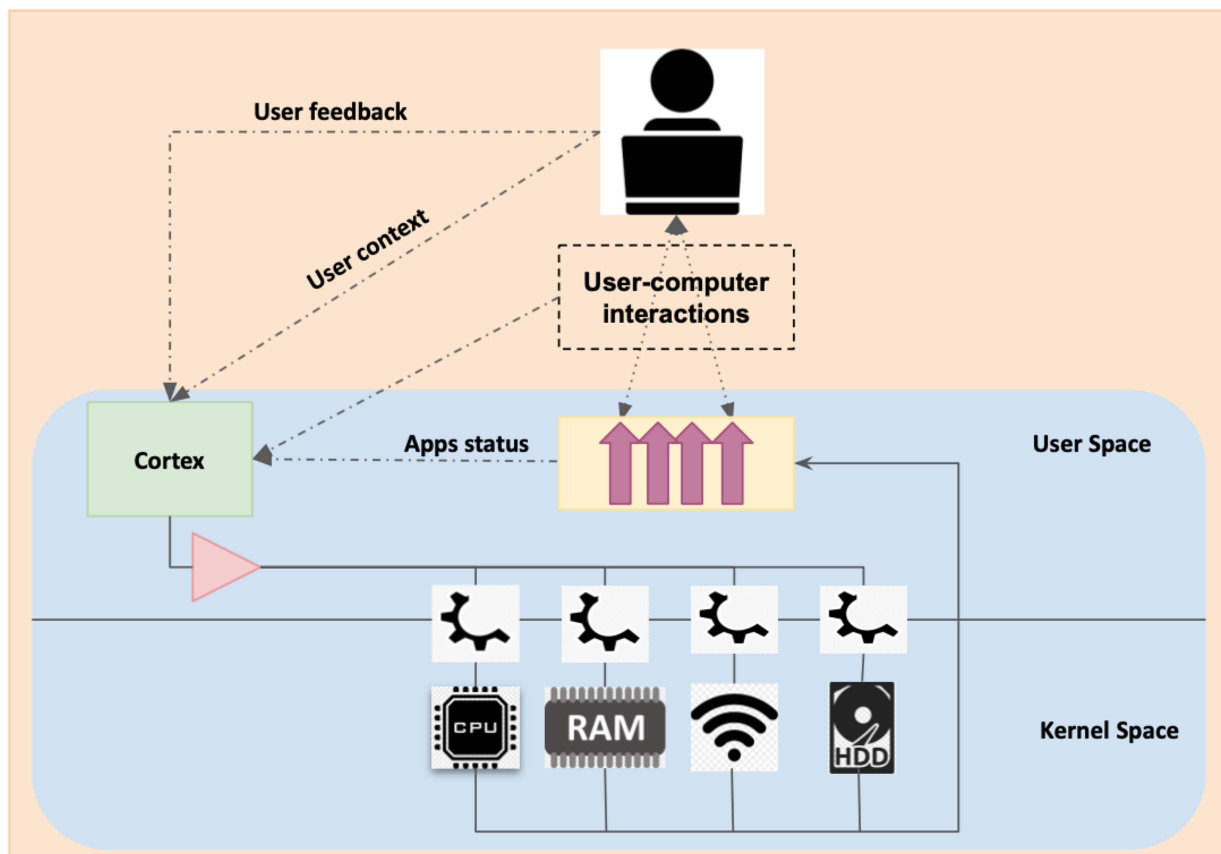

Figure 3.1: Learning-based SmartOS architecture.

expectations. The Cortex applies the determined resource allocation policy through parameters exposed by the kernel. The kernel then allocates CPU, memory, network bandwidth, and disk I/O among applications according to the specified parameters. The new resource distribution changes the quality of the user experience. The user provides feedback based on their experience to the Cortex to indicate the resource policy has successfully achieved its objectives.

The overall architecture and design of the SmartOS are similar to Reinforcement Learning (RL). RL consists of an agent that measures the current state of the environment, acts on the environment, and watches the environment's next state. Also, it receives a reward based on how that action has affected the environment to reach its objective. In general, the RL agent's objective is to maximize the total gained reward. As a result, the instant gain of action is not as valuable as the entire gain of a set of actions over time. The nature of RL makes it suitable for decision-making problems.

Due to the similarity between the SmartOS and RL design, an RL algorithm was utilized in the Cortex. Another reason for this design decision was that the OS constantly decides on sharing resources among processes and scheduling processes. Therefore, this was highly suitable for RL, which continually adapts its decisions based on the available state information. Other machine learning techniques such as supervised and unsupervised learning were not ideal for this problem as they could not handle the dynamic nature of this problem. However, correctly identifying the environment's current state, set of possible actions, and reward made this approach challenging [48].

The Cortex component was located in the user space in pursuing this Cortex model for integrating user-adaptive learning into an OS. This resulted in easily managing, testing, debugging, and updating the SmartOS and learning algorithms. Another approach was to place the learning component within kernel space. This could have improved the performance. For example, KMLib provides a fast library framework for ML applications [6] in kernel space. However, using such a library was difficult because of the challenges of managing and debugging code in the kernel space. It is also a trend in the industry to move applications to user space to prevent tight coupling and slow release cycles [47]. According to section 3.3, the performance of the RL-based SmartOS in

user space is more than adequate given the human time scales for adaptation.

3.1.1 Actions

To distribute the computer resources among different applications, one can use the parameters in the kernel or change the kernel code. In this thesis, the predefined parameters were utilized to control resource allocation to be compatible with different Linux versions (see Section 3.2). Implementing the SmartOS in user space was easier using this method, which made it easily pluggable.

Among different resources in a computer, the CPU, memory, network, and disk I/O were the main focus of the SmartOS as their allocations are critical to application performance.

3.1.2 Rewards

The best reward for SmartOS was user feedback because SmartOS's final goal is to find the best resource allocation policy based on user preferences. Therefore the user was the only person who could show how effective the Cortex was in reaching its objective. The feedback could be implicit and non-intrusive based on passively monitored interactions such as keyboard strokes or mouse clicks and movements. For example, frustrated users move the mouse quickly or type more rapidly. The feedback also could be explicit via a specialized application for providing feedback.

3.2 Prototype

The Cortex of SmartOS was implemented on Linux, Ubuntu 20.04 in user space. As a first step to test the feasibility of employing RL to tune parameters automatically, simplified discrete-valued models for the environmental states, actions, and rewards were constructed, reasoning that if the benefits in such a system can be demonstrated for discrete values, then it can be extended later to address the complexity of continuous-valued states. The environment state was limited to the applications' resource usage profiles, whether the applications were foreground or background, and whether the applications were video/audio applications. If an application was in the foreground, the foreground value in the environment's current state vector was one and zero otherwise. As

noted in the Acclaim work, whether a process is in the foreground or background state is a key indicator of what the user views the process as being important at that moment, though as will be shown statically, prioritizing the foreground is not the whole story. Similarly, the value was one if an application was video/audio or zero otherwise. This allowed us to examine how the modality of the application should be considered in learning the best resource allocation. As for the CPU, memory, network, and disk I/O values in the state vector, if the application was intensive in the consumption of any of these resources, the corresponding vector index for that resource type was one otherwise zero. Choosing four independent resource dimensions allowed us to examine cases where high-priority applications may demand intensive resources in certain dimensions while being interfered with by other applications in various complex ways.

In addition, the range of the possible actions was limited. A value of one in the action vector for any resource meant high priority in that corresponding resource and otherwise normal priority.

As for the reward, a script was created that generated user feedback synthetically to test the space of a wide variety of user behavior. This gave us more freedom to explore many different types of user behavior, both from a user resource perspective as well as a temporal perspective. The script only gave +1 as a reward for the best action and 0 otherwise. The best action was the resource allocation policy that resulted in the highest performance of the applications that are important to the user in a given scenario.

3.2.1 Resource allocation

A variety of system tools in Linux were leveraged to implement changes in the relative allocation of system resources to each application, effectively furnishing the ability to change the prioritization of different processes independently in four separate dimensions: CPU, memory, I/O, and network bandwidth. The RL algorithm manipulated these system "knobs" and then inspected whether it had converged towards the best allocation of resources. Different resource parameters were controlled as follows:

- **CPU:** the **nice** value was set to -20 for high priority and 0 for normal priority to control CPU allocation.

- **Memory:** To control the memory, the parameters **oom adjacent score** and **cgroup memory swappiness** were used. To set a high priority for memory for an application, these values were set to -1000 and 0 for **oom adjacent score** and **cgroup memory swappiness**, respectively, and for normal priority, set to 0 and 60.

- **Network:** **cgroup netprio ifpriomap** was utilized to distribute the network bandwidth among applications. For high priority and regular priority applications, these were set to 10 and 0 respectively as priority values.

- **Disk I/O:** To manage the Disk I/O, **ionice** was used to set a real-time class with priority 0 for high priority applications and left the priority of the standard application as default (idle class with the priority of 4).

3.2.2 Reinforcement Learning algorithm

For implementation of the automated learning, a variety of reinforcement learning algorithms were explored, including DQN [50], QLearning [65], Sarsa [57], Double Qlearning [26], A2C [49], and Monte Carlo [62]. For testing some of the algorithms such as DQN and A2C, [28] was used, and for the other algorithms, Python 3.6 [4] and the Numpy [25] package were used. The Monte Carlo Reinforcement Learning algorithm is shown in Section 3.3.5, it had the best convergence rate compared to other methods in Table 3.1.

3.3 Evaluation

Comparisons were made between the automated learning approach of SmartOS, which applied RL to learn the proper allocation of CPU, memory, I/O, and network bandwidth, with a variety of static prioritization schemes. The followings are the different static prioritization heuristics that were compared against:

- **Fg only:** This heuristic set the CPU, memory, network, and disk I/O parameter of the Foreground application to high priority.

- **Fg + video/audio:** This heuristic, inspired by [42], added to the previous heuristic by giving high priority in all four resource dimensions to video/audio applications resulting in competition in some dimensions for resources with the foreground application. The idea was to test the case where a user may be editing a document while listening to music or playing a video.

- **Fg + dependent:** It gave high priority to the foreground application and all other applications that foreground performance depends on. It identified the mentioned applications by a predefined directed acyclic graph (DAG). The DAG was based on our common observation of what each application usually depends on. The aim was to test the case where an application on a computer may consist of a set of dependent processes that communicate via network message passing, as is the case for complex applications like browsers and video/audio.

- **Multi-dimensions:** This heuristic used a predefined map that stored the essential resources per application's performance based on our common observation of the applications. Then, it prioritized the foreground application in all resources necessary to its performance. After that, if any remaining resources were not assigned to the foreground application, it assigned them to the important applications to the user. These important applications were also stored in a hash map defined by asking the user in the beginning. The intent was to examine the case where the foreground application may have variance in its resource needs while competing with applications that may have resource needs in different dimensions.

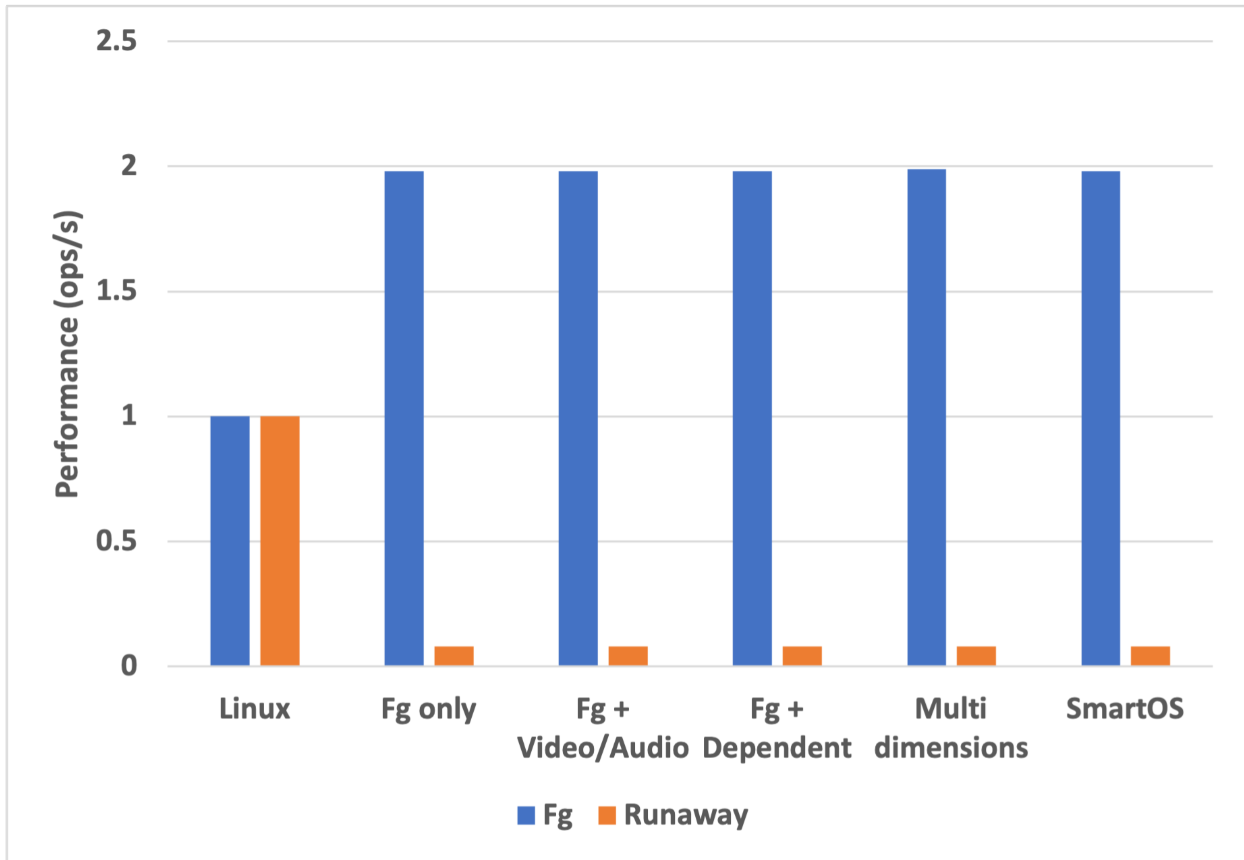

Figure 3.2: Foreground app contended with a resource-intensive background app.

Next, various scenarios were set up to compare the aforementioned static prioritization schemes and Linux's base CFS scheduling algorithm with SmartOS' automated RL.

3.3.1 Foreground contended with resource-intensive background

In this scenario, a foreground application that should have been given the highest priority for all resources to maximize user experience, was forced to compete with a resource-intensive background application that was not as important to the user. Both the foreground and runaway background applications were implemented as stress applications that made equal intensive use of the CPU, memory, and disk I/O, consuming the same resources on the same core in an Ubuntu 20.04 virtual machine with 8 GB of memory, one processor, and a 50 GB VDI disk drive.

In Figure 3.2, the blue and orange bars show the performance of the foreground application and background application correspondingly proportional to the base Linux over 60 seconds of execution. CFS-based Linux gave equal priority to the foreground and background, so the foreground application could not make as much progress as in other policies. In contrast, for each heuristic static prioritization policy, the foreground could run at twice the ops/sec rate of the base Linux case. At the same time, the background application was appropriately given scant resources. Similarly, SmartOS' user-adaptive RL strategy learned the correct policy and converged to the same actions, prioritizing the foreground application. We used the Monte Carlo RL algorithm for these and the following comparisons.

3.3.2 Foreground under-performed due to a dependent application

In this experiment, a stress app ran as a foreground app that consumed CPU, memory, and disk I/O and communicated through blocking pipes with another process that was also another stress app and was intensive in consumption of the same resources as the foreground application. A third runaway application was also running that was intensive in CPU, memory, and disk I/O usages. All of these were running on the same core. Hence, these three applications competed on attaining CPU, memory, and disk I/O. Figure 3.3 shows how the Fg only heuristic could not achieve

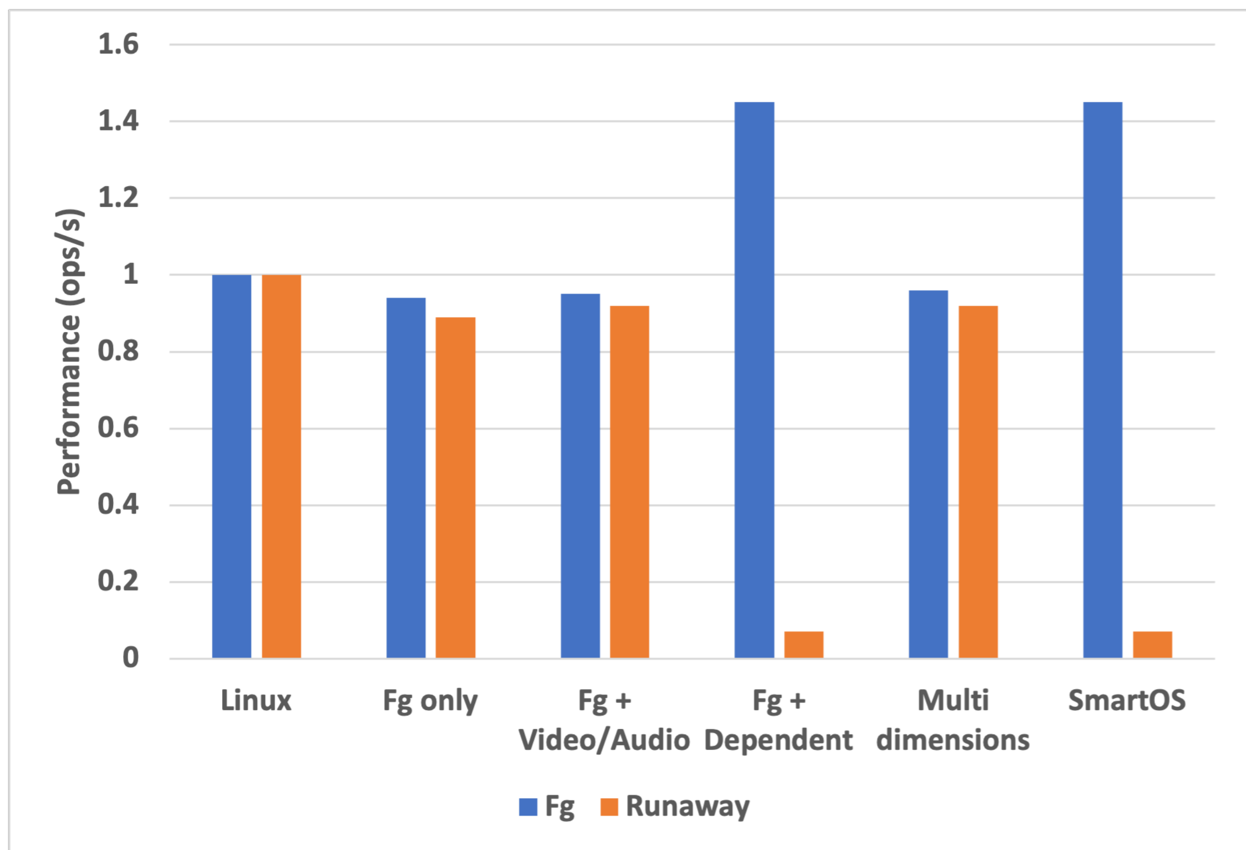

Figure 3.3: Foreground under-performed due to a dependent application.

a good performance as it only changed the priority of the foreground app to high priority and left the background app unchanged. Thus, the foreground app, which was messaging through pipes with the background app, remained in the blocking stage for a significant portion of its execution time and made the CPU available to the runaway application, resulting in an undesired resource allocation, which led to user frustration. The same also happened with the Fg+Video/Audio and Multi-dimension heuristics. The only successful heuristics in this experiment were the Fg+dependent static policy and SmartOS, which converged to the best resource allocation decision.

3.3.3 Multiple important applications with needs in separate dimensions

Sometimes the performance of other applications besides the foreground application is also crucial for enhancing the user experience. An example of this scenario would be when a user is working inside a document editing application, but is also monitoring a stock widget on their laptop screen, or listening to music. The stock widget or music is not a foreground application, but its performance is important to the user. Suppose the other applications important to the user are consuming different kinds of resources from the foreground application. In that case, the priority for all resources should not be given to the foreground application. In order to test such a scenario, a VM with four cores, 3 GB of RAM, and 50 GB of VDI hard disk with installed Ubuntu 20.04 was set up. A stress app that was CPU intensive was run as a foreground application on core one, and two memory-intensive applications were run on cores 2 and 3, respectively. One memory-intensive application played an important role in the user experience, and the other was a runaway application. Since all applications ran on different cores, they did not compete for more computation. As a result, giving more priority to the foreground application did not affect its performance. Thus the Fg only, Fg+Video/Audio, and Fg+Dependent static prioritization policies did not improve the user experience. However, the multi-dimensional heuristic achieved a better user experience as it only gave priority in CPU to the foreground application and gave more priority in memory to the critical background application. The two memory-intensive applications competed over the memory since they did not fit simultaneously in the memory and needed to be swapped out

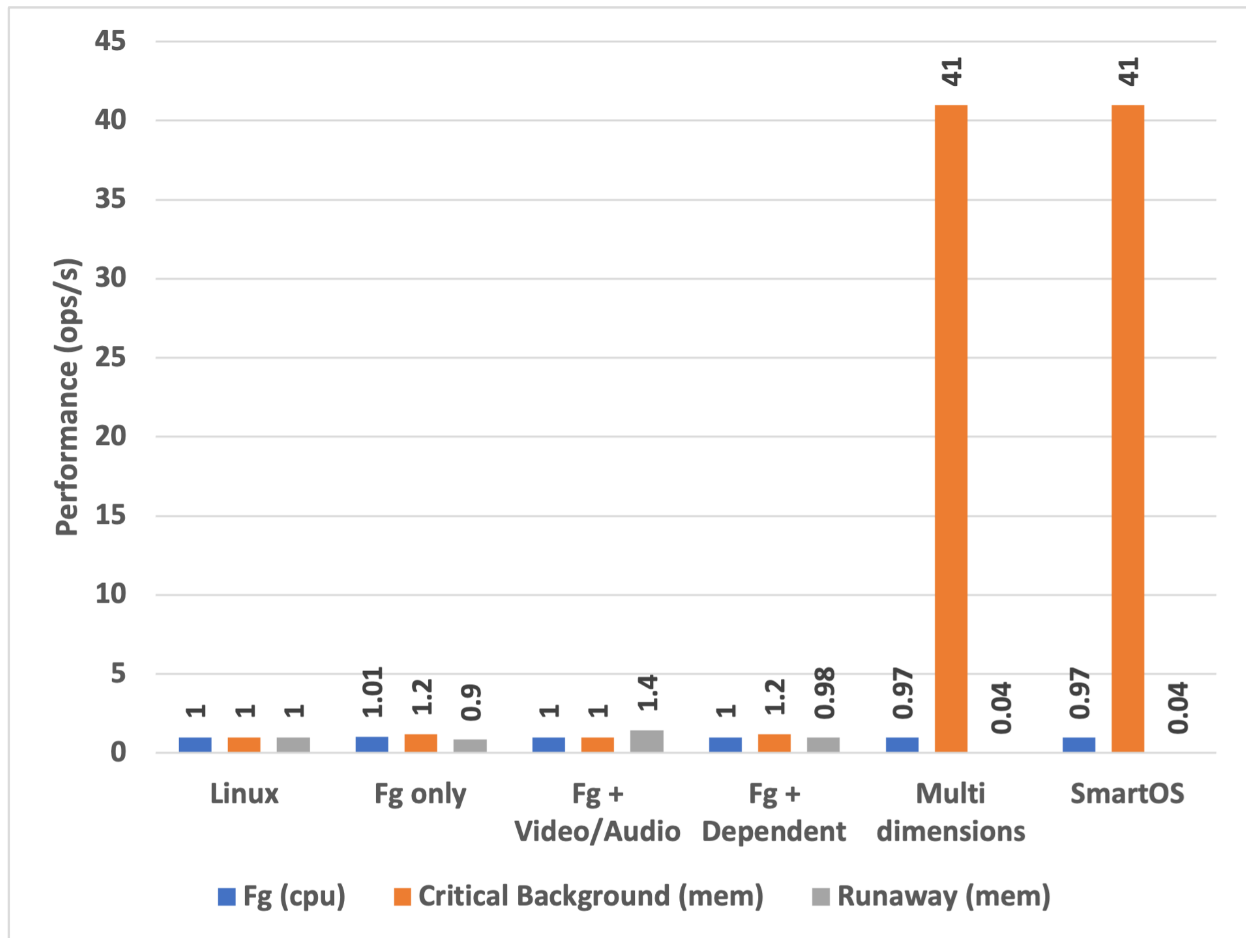

Figure 3.4: Multiple important applications with resource needs in separate dimensions.

to disk space. As a result, giving more memory priority to the critical application enriched the user experience because it prevented swapping out of the critical application. Figure 3.4 demonstrates the same result. SmartOS learned and achieved the same performance as the multi-dimensional static prioritization approach.

3.3.4 Variation of dimensions

The following experiment consisted of random assignments of the importance of various applications and their resource needs. A random generator first selected the CPU as the resource needed for a stress application running as a foreground application on core one. It also randomly chose the CPU for the second stress application competing with the foreground running on the same core. After that, the random generator picked the memory as the resource needed for applications three and four. These two memory-intensive applications ran on core two and three, and competed over memory. One of these two applications' performance is critical to the user experience quality, and the other one is a runaway application. Figure 3.5 shows that the static multi-dimensional heuristic did not allocate resources efficiently because it picked the memory as the required resource for the foreground application and the second application, and the CPU as the necessary resource for the third and the fourth applications, which differed from what these applications needed. The other three heuristics, Fg only, Fg+Video/Audio, and Fg+dependent, successfully improved the foreground application but did not alter the critical application's performance. Only SmartOS successfully enhanced both the foreground and the critical application's performance.

Table 3.1: Different Reinforce Learning methods convergence in a dynamic setting.

RL Algorithm	Feedbacks#	Episodes#
DQN [50]	52000	13000
QLearning [65]	28400	7100
Sarsa [57]	3680	920
Double Qlearning [26]	2400	600
A2C [49]	1600	400
Monte Carlo [62]	400	100

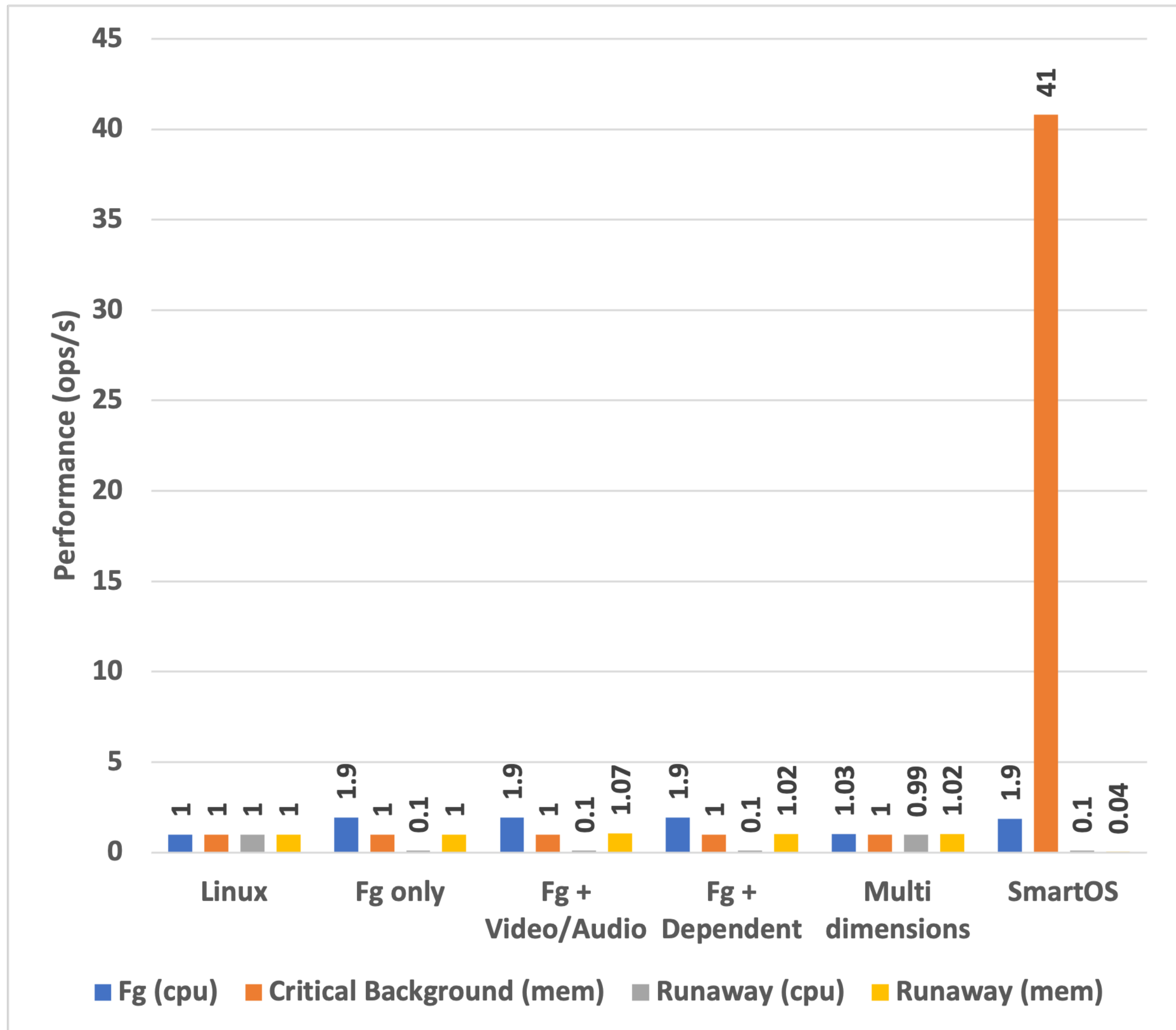

Figure 3.5: Variation of dimensions.

3.3.5 SmartOS dynamicity and convergence

This section evaluates how quickly SmartOS can adapt and converge to appropriate resource allocations based on user feedback. An experiment was designed that combined all the previous experiments. In other words, A script ran all the applications in Section 3.3.1 for 60 seconds. Meanwhile, SmartOS applied a resource allocation policy and requested feedback after applying the policy from the script. After 60 seconds, the script terminated all the running applications in Section 3.3.1 and started all the applications in Sec 3.3.2 for 60 seconds, etc. This same procedure was repeated for the applications in Section 3.3.3 and Section 3.3.4. Each repetition of Section 3.3.1 to Section 3.3.4 is called an **Episode**. After an episode was completed, this process was repeated with a new episode. Table 3.1 shows the number of feedbacks required for each reinforcement learning algorithm to find the best collection of policies. The results show that the Monte Carlo algorithm converged faster than other reinforcement learning algorithms.

Figure 3.6 provides a detailed temporal perspective of how SmartOS achieved the best set of policies and a maximum reward of 4 (one for each distinguished experiment) after receiving 400 feedbacks using the Monte Carlo in 100 episodes. It is demonstrated that there is rapid convergence early in the learning process. Note that what is portrayed in graph 3.6, is a smoothed average of ten episodes. Although it appears that episode 100 did not achieve the maximum award, namely full convergence of 4, the unsmoothed value attained a value of 4. So Monte Carlo reached full convergence by episode 100. While this experiment combined many iterations of different scenarios, note that for just a single scenario of the foreground application explained in section 3.3.1, convergence was achieved in just eight steps.

The Monte Carlo implementation of RL in the SmartOS Cortex was used as a basis for performance measurements. SmartOS adapted to each user feedback in 0.218 ms of total execution time. 0.21 ms of that time consisted of purely CPU execution, and the rest contains context switch time. These results show that SmartOS is reasonably responsive to user feedback without excessively burdening Linux, given that time scales for human adaptation are on the order of

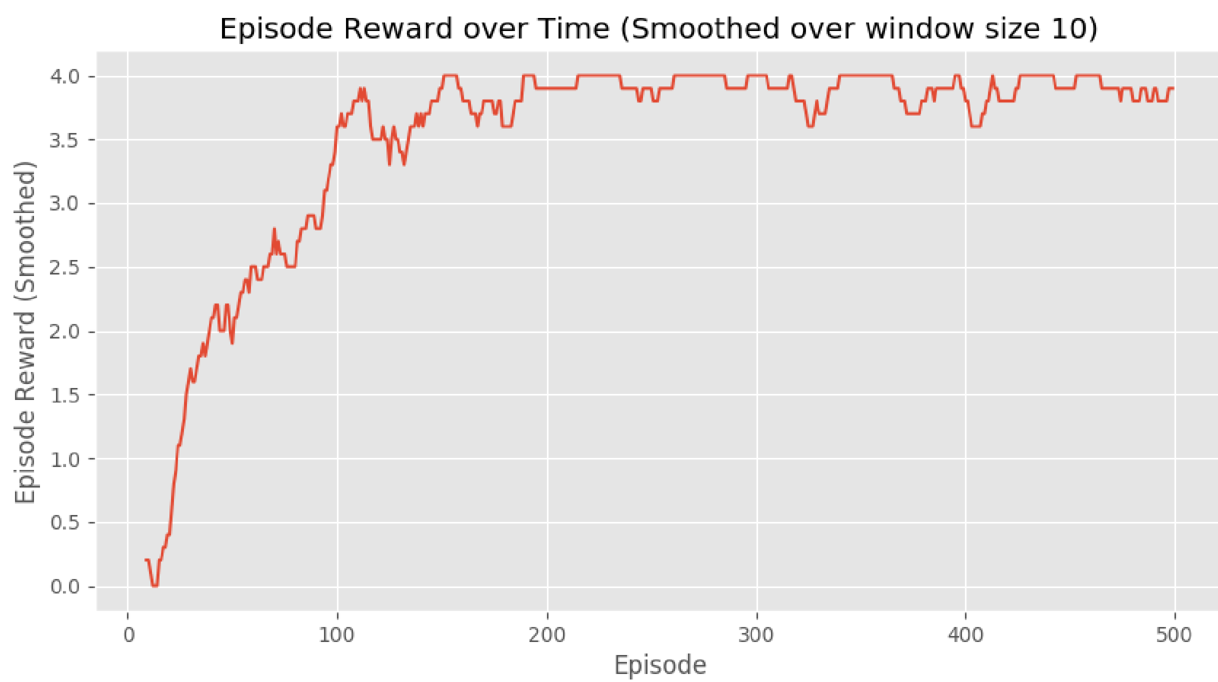

Figure 3.6: Convergence of Monte Carlo in a dynamic setting (each episode consisted of 4 feedbacks).

seconds. Also, the required memory to run the Cortex application was 23.1 MB.

3.4 Conclusions

This chapter presented SmartOS, a learning-based operating system that implements reinforcement learning to automatically adjust the allocation of memory, CPU, I/O, and network bandwidth according to learned user preferences. SmartOS was implemented in Linux user space, and our test results showed SmartOS adapted automatically to increasingly complex allocation synthetic scenarios, unlike static prioritization policies. It was also demonstrated that a Monte Carlo RL algorithm achieved the fastest convergence in terms of its learning rate, and that its overhead was tenths of milliseconds.

Chapter 4

Human Frustration Prediction

This thesis chapter investigated a model for human frustration prediction via an IRB-approved human study. First, an application was developed that ran in the user's computer/laptop/VM with Linux 20.04. Then, the application collected the user's data, including time, the list of running apps and their resource usage profiles and resource utilization of the system and the active app that the user was working with, mouse clicks and movements, keyboard clicks' patterns, audio features, head movements through the computer's microphone and webcam passively without interfering with the user's daily work. User reported the times they were frustrated with their computer's performance and explained their reasons for their frustration through the app. Finally, the application sent the data to the cloud. After two weeks of data collection, a model was developed offline that predicted the user frustration with the computer using the collected data. So the main contribution of this chapter involves the following:

- First, an overview of the application and all the collected data are described.
- Second, the data cleaning and preprocessing for the model development are explained in detail.
- Third, the architecture of the model that predicts users' frustrations with their computers is demonstrated that works with multiple modalities of the data, such as continuous, categorical, audio, video, and text.

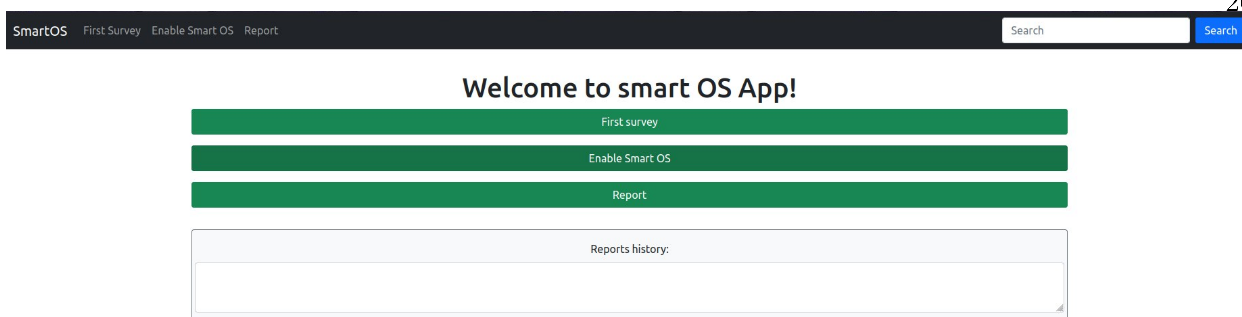

Figure 4.1: SmartOS Application Central Page

- Finally, there is a comparison between the model's effectiveness with other possible machine learning models.

4.1 Application Overview

An IRB-approved study collected data from fifteen human users of desktop computers by installing a SmartOS app on their computers. A variety of data were collected from their computers, including time, the list of running apps and their resource usage profiles and resource utilization of the system and the active app that the user was working with, mouse clicks and movements, keyboard clicks' patterns, audio features, head movements through computer's microphone and webcam and explicit feedback when users became frustrated. The collected data were used to train a model that predicts user frustration.

4.1.1 Application Usage Procedure

After the user DocuSigned the consent form, the link was sent in an email to the user for downloading, installing, and running the SmartOS software app on their desktop/laptop, as shown in Figure 4.1. When the user ran the SmartOS app for the first time, they needed to complete an initial satisfaction "First Survey" shown in Figure 4.2. After this step, monitoring began on the user's desktop/laptop. Once the SmartOS server received the survey data, it generated a unique numerical ID for the user and registered it in the system for the remainder of the study. The

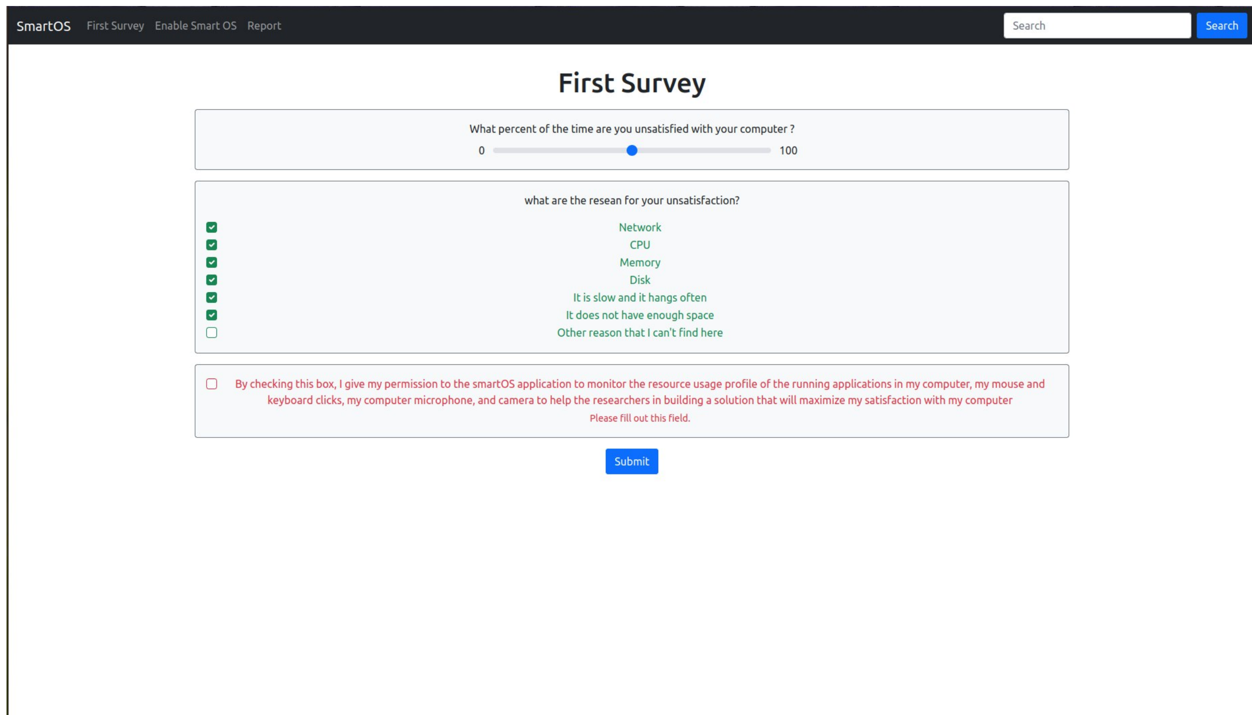

Figure 4.2: SmartOS Application First Survey Form

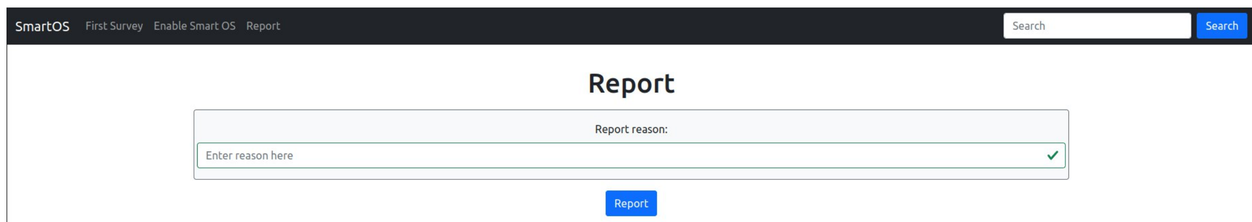

Figure 4.3: SmartOS Application Report Form

user's subsequent interaction with the server was labeled with this numerical ID. All the data were de-identified as the SmartOS application did not collect personally identifiable information such as name, email address, etc. All the audio data were converted on the user's computer to MFCCS (Mel-frequency cepstrum), then sent to the server. The MFCCS data are not reconvertible to their original raw audio form. The webcam detected the user's head position with respect to their screen size using a convolutional neural network face detection model running on the user's computer, and it sent four numbers: top, bottom, left, and right of the user's face position with respect to their screen size to the server. So no raw videos were collected in the human study. As for the keyboard pressing patterns, the smartOS app only sent the timestamp the user pressed a key to the server. It did not collect what the user typed. Thus, no one can identify the user's data, not even SmartOS researchers.

The detailed procedure for using the app after the installation was as follows:

- First, the user had to connect to the Internet. Then they had to download, install, and run the application on their computer.

- Once they ran the application, they saw the screen in Figure 4.1 called the central page of the application, which contained three buttons, First Survey, Enable Smart OS, and Report. First, the user clicked on the First Survey button and completed the first survey form. The other buttons did not work until the First Survey was completed. Once they completed the "first survey" shown in Figure 4.2, the monitoring and data collection began.

- For the study, the user needed to run the application whenever they turned on their computer. The application passively collected all the data types mentioned in Section 4.1.2.

- In addition to this, the user also needed to report the times they were dissatisfied with their computer using the report form in the app, shown in Figure 4.3. The user could type a brief explanation of the reason for their frustration before submitting their report.

- After two weeks, the data monitoring stopped.

The time commitment for the subjects in this study was around two weeks.

4.1.2 Data Collection

The application collected data about user interaction with their computer that were used as the input to the machine-learning model to detect user frustration with their computer. To collect data, first, in the SmartOS monitoring app downloaded and installed by the user, the user filled out a "First Survey" shown in Figure 4.2. This form was only filled out once at the beginning of the study and asked the user to describe the extent of their dissatisfaction with their computer and reasons for being dissatisfied. At the end of the initial survey, the user was reminded that the monitoring begins immediately once they fill out and submit the survey. Once this form was completed, the SmartOS app collected the following user data for two weeks:

- **User-computer interactions:**

  * **Mouse clicks, mouse movements, mouse scrolls:** Previous studies showed when the user is frustrated with the system, they tend to make more mouse clicks and move the mouse faster in divergent directions. For example, in [27, 67] they studied the captured mouse cursor's x/y position and timestamp at a millisecond precision rate, total distance, and average cursor's speed to infer emotions. In [70], they analyzed parameters including the number of mouse clicks per minute, the average duration of mouse clicks (from the button-down to the button-up event), and the maximum, minimum and average mouse speeds to compute the affective state. In [58] they studied the number of clicks (per button and aggregated); overall distance; covered distance (the distance the cursor has traversed) between two button press events, between a button press and its following release event, between a button release and its following press event and between two button release events; the Euclidean distance in the four previously described cases; the difference between the covered and the Euclidean distances calculated; and time duration between the proposed combinations of events

to predict the user's affective state. Thus, inspired by previous studies, we captured the mouse cursor's x/y positions, the mouse clicks' x/y positions, mouse scrolls' x/y positions, and the scrolls' amount in the x/y coordinate along with timestamps at a millisecond precision rates.

* **The pattern of keyboard keys clicks and special keys (such as ESC, delete, enter, space, etc.) clicks:** According to other studies, frustrated users may betray their agitations by typing faster or clustering specific keys together [31, 70, 39]. In [31] they studied features such as typing speed (mode, standard deviation, standard variance, and range) of the number of characters typed in 5-second intervals, total time taken for typing, the total number of backspaces used, and idle time in between (if any) when the user is not typing anything for recognizing emotions from keyboard stroke pattern. In [58] they studied the number of key press events, the average time between press events, the average time between a press and its following release event, and the number of times a certain key or a group of keys has been pressed (backspace key, delete key, alphabetical characters keys, etc.), and the indicators proposed in [16], which were generated from creating combinations of two or three keystrokes events to predict the user affective state. Following the footsteps of the previous work, the SmartOS app collected all the keys clicks and special keys clicks (such as ESC, delete, enter, space, etc.) with timestamps at a millisecond precision rates.

- **User audio:** Frustrated users may voice their frustration in various ways, such as raising their voice or vocalizing more rapidly or frequently, which may help us identify user frustration. In [63] they used audio spectrum for emotion recognition using deep neural networks. In [54] they made an emotion recognition system based on prosodic features (i.e., intensity, pitch, formant frequencies of sounds) combined with short-term perceptual features to classify emotions. In [66], they used spectrograms, the existence of certain words, and a phoneme segmentator to recognize emotions. In [61] they used the perceptual

evaluation of audio quality (PEAQ) model as described by the standard ITU-R BS.1387-1, which provides a mathematical model resembling the human auditory system. In [9] they preprocessed the raw audio signal from voice data in the database and then extracted 34-dimensional low-level handcrafted acoustic features of time-domain, spectral-domain, and cepstral-domain characteristics, including zero crossing rate, energy, the entropy of energy, spectral centroid, spectral spread, spectral entropy, spectral flux, spectral roll-off, MFCCs (Mel frequency cepstral coefficients), 12-dimensional chroma vector, and standard deviation of chroma vector. They kept the maximum input of 100 frames and finally got a (100, 34) vector for each utterance. To show the manual acoustic emotional features more intuitively, they visualized several features, including chroma, zero crossing rate, MFCCs, energy, and spectral flux. In [23] they showed that the MFCC are the best features for recognition of emotional content in the audio. Thus SmartOS app collected the MFCCS of users' audio.

- **User video:** Frustrated users may move their heads in divergent directions quickly. In [63] they used video frames with deep neural networks to recognize emotions from head movements. SmartOS collected head positions which were computed using convolutional neural network face detection model applied on video frames recorded with the user's computer's webcam .

- **System-wide information such as computation, memory usage, network bandwidth, and input/output bandwidth of the running applications in the computer and the computer(Applications' and Computer' states):** SmartOS' goal is to predict the cases when the user is frustrated with the system. Thus the system resource allocation information helps identify scenarios where the user is specifically unhappy due to the slow response of their computer.

- **User Feedback:** Lastly, users reported their frustration with the system through the application using the Report Button, which opens a small window for describing and submitting the event causing the frustration, as shown in "report form" in Figure 4.3. Reports

provided the ground truth value of user frustration with their computers.

4.2 Architecture

The human study in this chapter was designed for data collection, so the SmartOS app did not modify the resource allocations policy on the user's computer. In the following sections, first, the procedure of collection and storing the data mentioned in Section 4.1.2 is explained in Section 4.2.1. Then the steps taken for data cleaning and preprocessing before implementing the model in Section 4.2.2 are described. Finally, in Section 4.2.3, an introduction of the model architectures used for emotion recognition in previous works is presented. Then the model architecture for predicting human frustration with their computers is demonstrated.

4.2.1 Application Architecture

The above data in Section 4.1.2 is collected and communicated to the AWS Data server, where they are stored in the DynamoDB database and subject to further analysis as shown in Figure 4.4 every 10 seconds. After the data collection phase, the data were cleaned and preprocessed. Then a machine learning model was developed offline to predict the user frustration based on the collected data at the cloud server.

4.2.2 Data Cleaning and Preprocessing

Our data is multi-modal, so each modality needs its specific cleaning and preprocessing. The followings are the data cleanings and reprocessing done to each raw data modality before feeding the data as input to the machine learning model.

- **Mouse Movements Data Cleaning and Preprocessing** After every 10 seconds, the server received the x/y positions and recorded timestamps of all the mouse cursor movement events at a millisecond rate. First, these events were divided into one-second intervals. Then

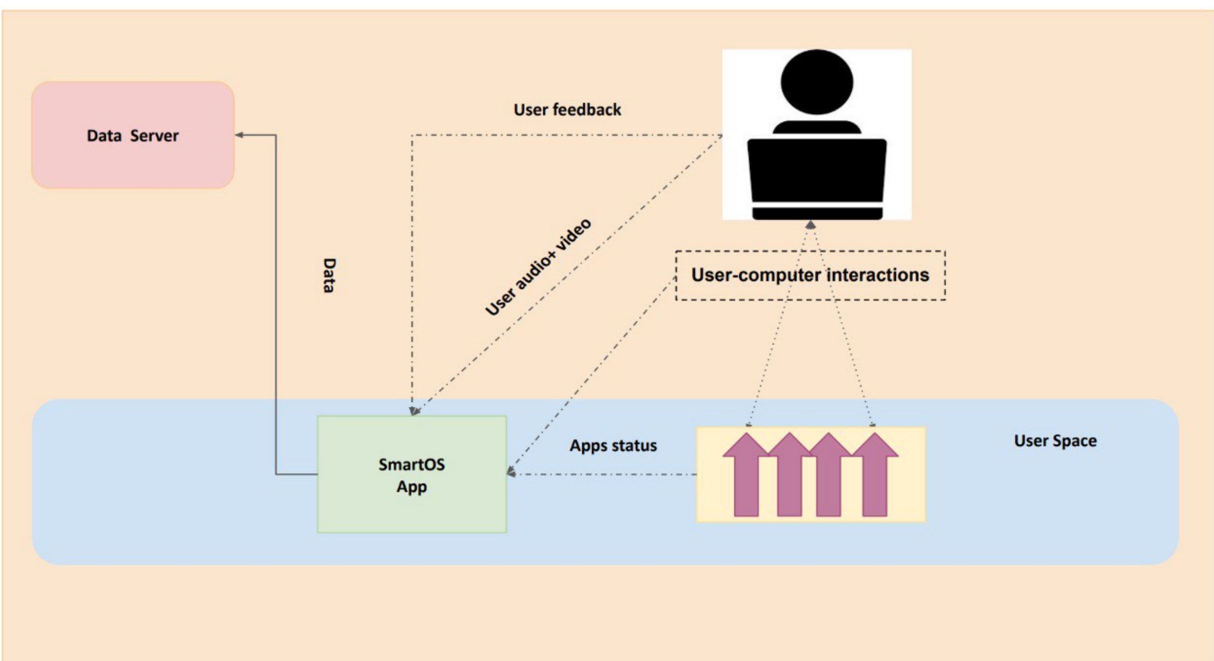

Figure 4.4: SmartOS App: Data Collection for Human Frustration Prediction Architecture

the number of events and the mean and standard deviation of x/y positions of the mouse cursor movement events in each interval were computed and fed to the model.

- **Mouse Clicks Data Cleaning and Preprocessing** After every 10 seconds, the server received the x/y positions and recorded timestamps of all the mouse cursor click events at a millisecond rate. First, these events were divided into one-second intervals. Then the number of left-click and right-click events and the mean and standard deviation of x/y positions of mouse click events in each interval were computed and fed to the model.

- **Mouse Scrolls Data Cleaning and Preprocessing** After every 10 seconds, the server received the x/y positions of all the mouse cursor scroll events, the amount of scrolls in x/y coordinates, and their recorded timestamp at a millisecond rate. First, these events were divided into one-second intervals. Then the mean and standard deviation of x/y positions and x/y amounts of the mouse scroll events in each interval were computed and fed to the model.

- **Keyboard Clicks Data Cleaning and Preprocessing** After every 10 seconds, the server received all the keyboard clicks events timestamps at a millisecond rate. These events were divided into one-tenth-of-second intervals because the highest typing speed is a character per 0.1 seconds and the number of clicks were counted in each interval and fed to the model.

- **Special Keys Clicks Data Cleaning and Preprocessing** After every 10 seconds, the server received all the special keys (such as ESC, delete, enter, space, etc.) clicks events timestamps at a millisecond rate. These events were divided into one-tenth-of-second intervals because the highest typing speed is a character per 0.1 seconds and the number of clicks were counted in each interval and fed to the model.

- **Audio Data Cleaning and Preprocessing** After every 10 seconds, the server received the MFCCS computed based on the raw audio. Then it computed the mean on axis 0 and

padded it with zeros so that the feature's length is 864.

- **Video Data Cleaning and Preprocessing** After every 10 seconds, the server received a vector of the top, bottom, left, and right positions of the user head. Then it padded it with zeros so that the feature's length is 32.

- **Applications Data Cleaning and Preprocessing** After every 10 seconds, the server received a vector of the top 10 applications in CPU usage, Memory usage, Disk usage and Network usage. Each application consisted of a vector of data as follows:

  * Process ID
  * User: If the application user were root, this value was one otherwise was zero.
  * Nice value
  * Virtual, residential and shared memory of each application in bytes
  * State: The index of the Application's state in the following list minus one:
    (1) Uninterruptible sleep
    (2) Idle
    (3) Running
    (4) Sleeping
    (5) Stopped by the job control signal
    (6) Stopped by the debugger during trace
    (7) Zombie
  * CPU usage and memory usage percentages
  * Application command: Application command names were added to a list, and their indexes in the list were used for encoding the command.
  * IO priority
  * Disk read, write, swap in usages in bytes

  * IO usage percentage
  * Fg: If an application were a foreground, the Fg was one, otherwise was zero.
  * Sent and received network data in bytes.

- **Computer Total Resource Usage profile Data Cleaning and Preprocessing** After every 10 seconds, the server received data regarding the total resource usage profile of the computer. It consists of the following:

  * Number of Total Tasks
  * Number of Sleeping Tasks
  * Number of Stopped Tasks
  * Number of Zombie Tasks
  * Percentage of idle CPU
  * Free Memory in Bytes
  * Free Swap Memory in Bytes
  * Total percentages of the IO
  * Total Network Packets Sent and Received in Bytes

- **Ground truth values(Labels)** For setting the labels, the timestamps of all the reports submitted by the users in the human study were used. All the data collected within 90 seconds of each report was labeled as frustration, thus resulting in a total 665 number of data labeled as frustration cases. The selection of 90 seconds intervals was because human emotions usually last 90 seconds. [2]

4.2.3 Model Architecture

Some previous works researched emotion prediction. However, as far as this study is concerned, none have focused on human frustration prediction with their computers using all the data

modalities ,used in this thesis. For example, In [17, 68, 53] they used the same feature as this thesis to predict emotions from the mouse data, however, they used linear regression, support vector machines, and random forest as their machine learning models. In [31, 20], they predicted emotion from the keyboard strokes' patterns using the same data as this thesis, and they used multi-layer perceptron and random forest as their machine learning models. In [38, 51, 13, 24, 41, 40] they focused on emotion prediction using speech. In [38] they used a pre-trained image classification network. In [13, 24] they used deep learning architecture and in [41, 40], they applied domain adversarial neural network and unsupervised learning respectively. In [22, 45] they used head motions to predict emotion using feature engineering and support vector regression, linear regression, hidden Markov model, LSTM, CNN, and the fusion between LSTM and linear regression.

First, due to the multi-modal input data of the problem, a supervised model using deep learning was defined. The supervised model base architecture, which is referenced as architecture one, is shown in Figure 4.5 which includes the following:

(1) All the input provided to the model after cleaning and preprocessing

(2) Each modality in our data input was connected to its specific layers consisting of three or four CNN layers or LSTM layers based on the input. These layers will convert each modality to a tensor with the shape of eight. CNN and LSTM layers were used to capture each individual user changes along time.

(3) These vectors were concatenated through the concatenate layer

(4) The output of the concatenate layer was fed to the dense layer with the output of 128 and an L2 regularizer with the parameter value of 0.2, followed by the Leaky Relu with an alpha of 0.2 as activation function and a batch normalization layer and a dropout layer with a dropout rate of 0.4.

(5) The output of the dropout layer was connected to another combination of layers similar to the 4 with only a difference in the output dimension of the dense layer, which was 32.

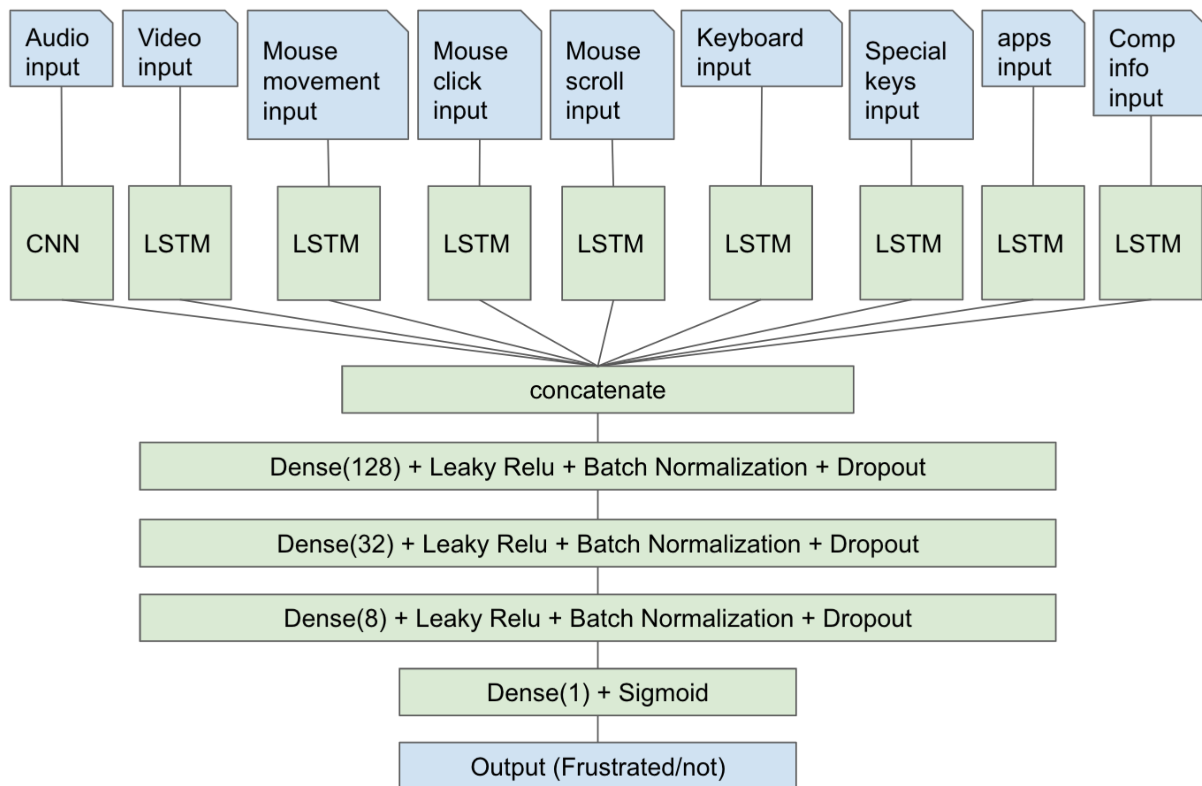

Figure 4.5: Human Frustration Prediction Supervised Model Architecture 1

(6) Afterwards, the output of the dropout layer was connected to another combination of layers similar to the 4 with only a difference in the output dimension of the dense layer, which was 8.

(7) the last dense layer received the output of 6 and produced a single output which was then fed into the Sigmoid activation function to predict the probability of the user being frustrated or not.

Using three or four layers of CNN, LSTM or Dense and decreasing the size of each layer's output dimension is a common practice for deep learning.

The numbers of labeled data were limited because the users did not report all their frustration cases. Sometimes, they forgot to report their frustrations or they became tired of reporting for the same reason multiple times. In addition, Users adjusted themselves and their expectations to avoid dealing with the difficulties. Thus the unlabeled data should not be recognized as satisfaction cases. So modifying the model's architecture to a semi-supervised model [11] using a Generative Adversarial Network [12] seemed natural. As shown in Figure 4.6 the supervised model and unsupervised model shared their backbone layers. These backbone layers were the same layers minus the last dense and Sigmoid in Figure 4.5. In addition, the generator model was connected to the unsupervised model. The unsupervised model input consisted of the fake data generated by the generative network and the real data from the human study. The unsupervised model did not differentiate between the real data labeled as frustrated or not. They are all real data, thus, they have the label of one. The supervised model input consisted of the real data labeled as frustration cases by the user and the same number of data sampled from the unlabeled real data in the human study. Using this method, the numbers of positive and negative samples were the same. Thus accuracy was a reliable metric for assessing the success of the supervised model.

The supervised model is only trained on the labeled data and only a small set of unlabeled data. The generator and unsupervised model strengthen each other to learn the hidden statistic patterns in all the data, both labeled and unlabeled as frustration. Thus, the shared layers between

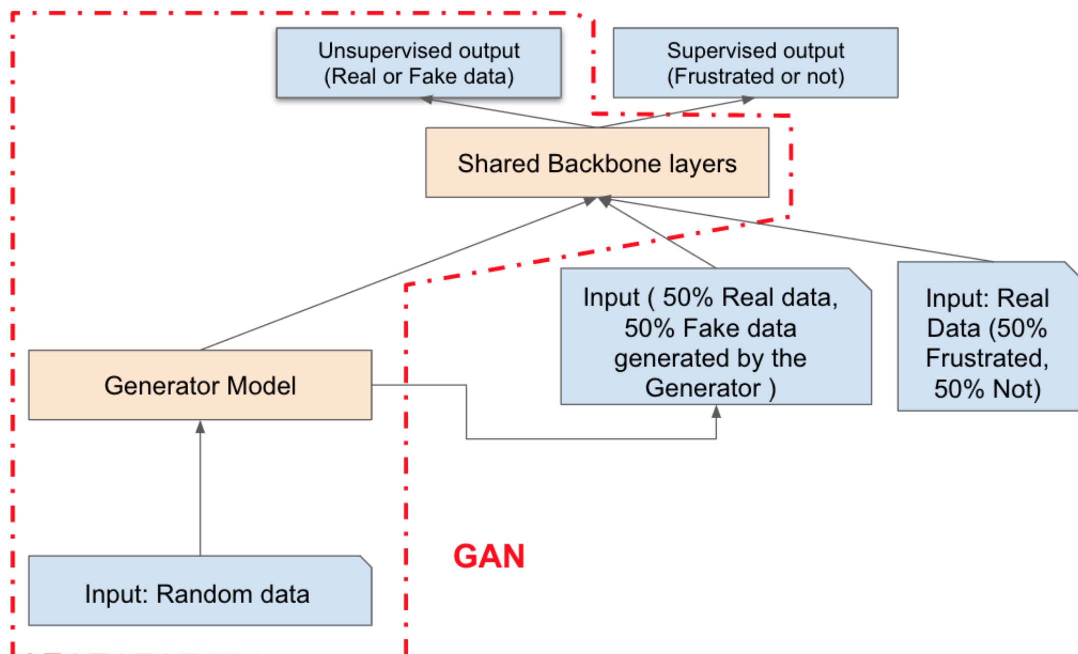

Figure 4.6: Human Frustration Prediction Semi-supervised Model Using Using Generative Adversarial Network

the supervised and unsupervised models brought the supervised model the advantage of learning more about the data and its statistical patterns. As a result, supervised model can predict the the frustration using only a few labeled data.

The cycle of training included three mini-epochs:

(1) supervised model was trained for one mini epoch on the real labeled data and the sampled real unlabeled data.

(2) unsupervised model was trained for one mini epoch on the real data collected from the human study and the generated data from the generator. It should be noted that the generator is not trainable at this stage.

(3) the generative adversarial network (GAN) was trained on the random input and the output of the unsupervised model. The GAN network learned in this phase to generate data in such a way that the unsupervised model can not distinguish between real and fake data. It should be noted that the unsupervised model is not trainable at this stage.

This whole cycle was repeated for the total number of epochs divided by mini epochs. The number of mini epochs in the model was set to 64, and the batch size was set to 128.

4.3 Evaluation

As for the evaluation, The number of epochs was 256. The total number of input data of the supervised model was 1330, and the total number of data collected from our user study was 122577 cases that were fed to the unsupervised model. The input data of the supervised model was divided into 80% training data and 20% test/validation data. The skit-learn train_test_split function did the sampling for the train and test data with the stratify parameter set according to the labels which results in unbiased test results.

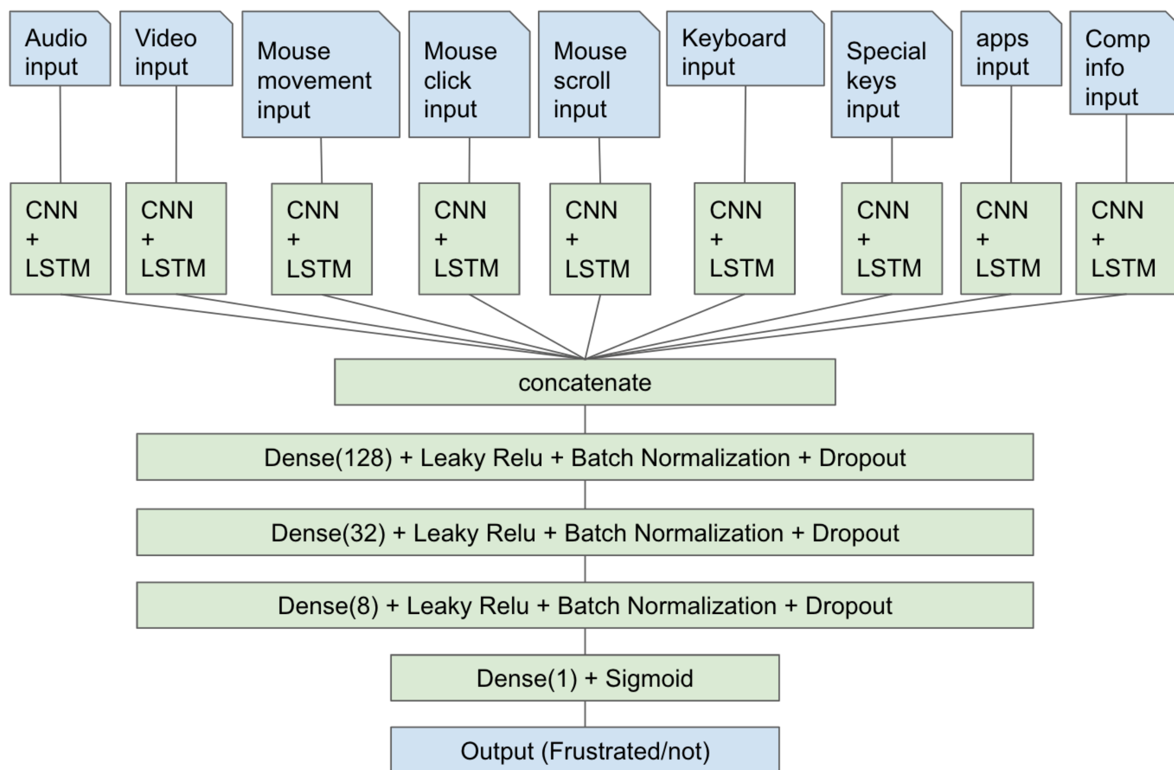

Figure 4.7: Human Frustration Prediction Supervised Model Architecture 2

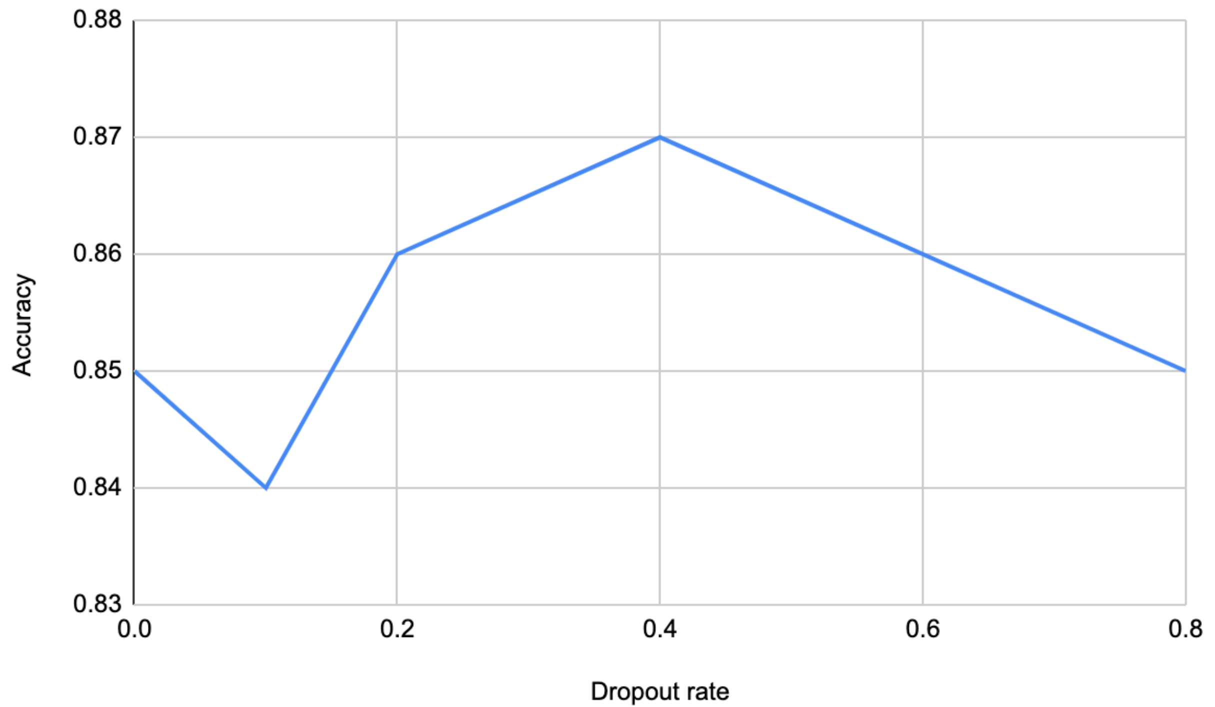

Figure 4.8: Human Frustration Prediction Dropout rate Search

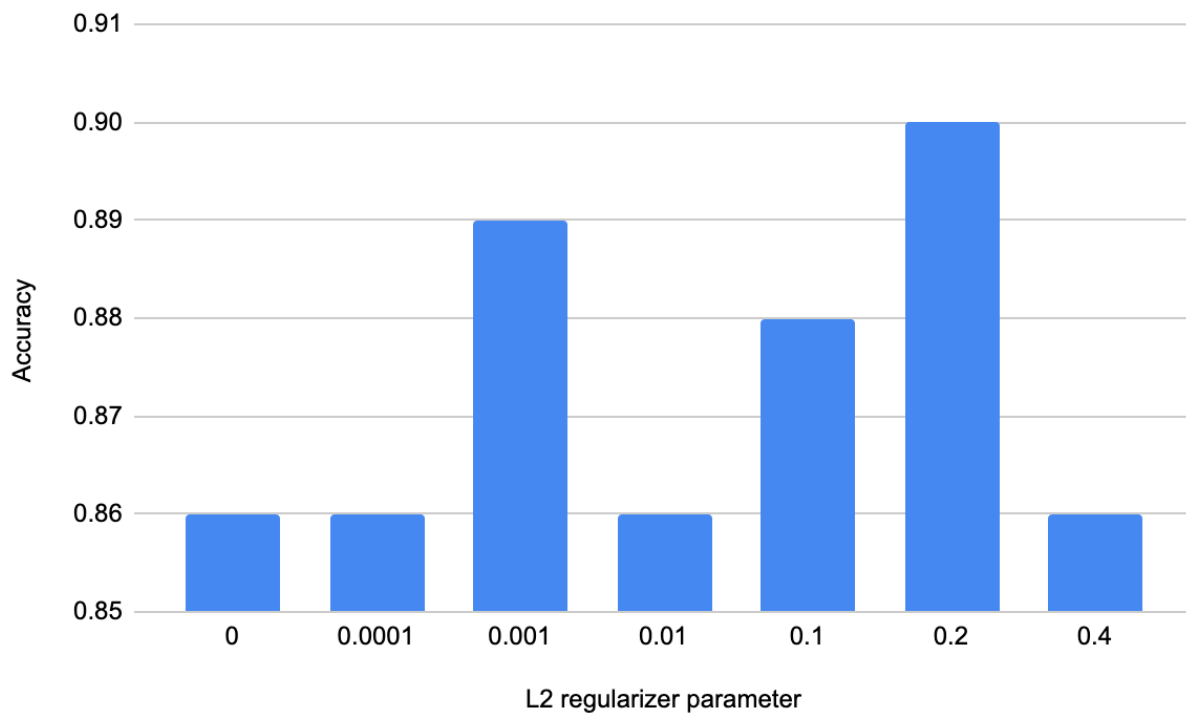

Figure 4.9: Human Frustration Prediction L2 regularizer Parameter Search

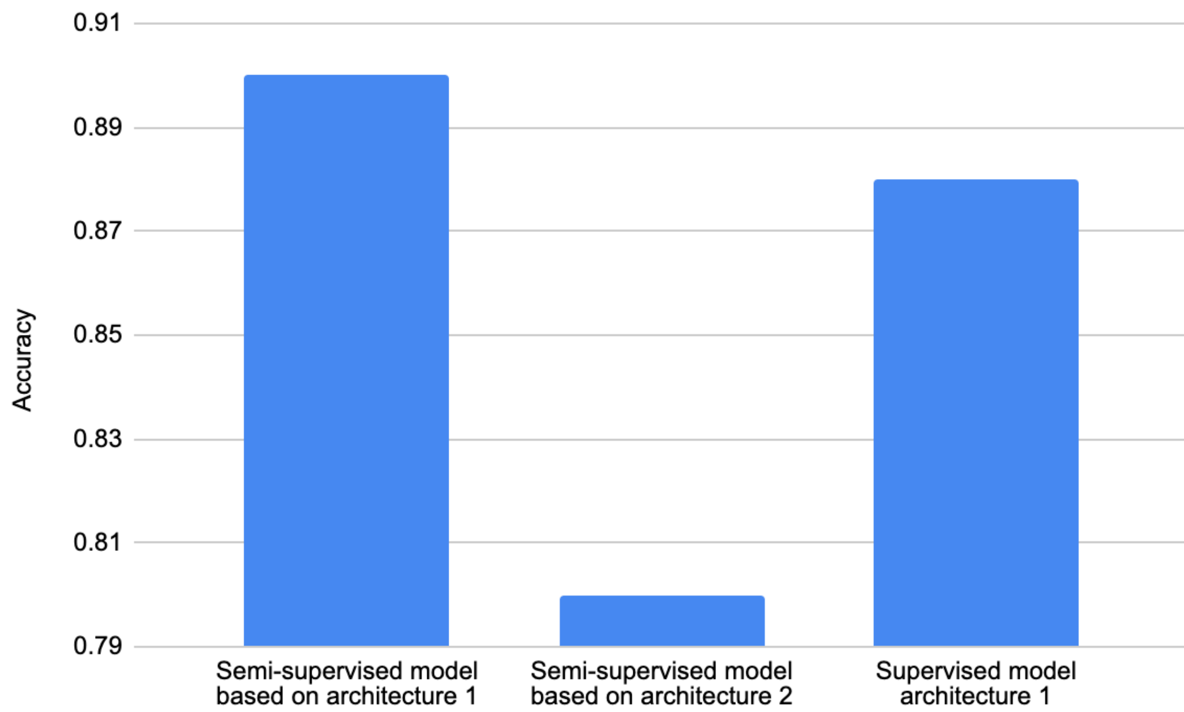

Figure 4.10: Human Frustration Prediction Models Comparisons

4.3.1 Parameter Search

A parameter search was done to find the best combination of the L2 regularizer parameter and dropout rate. First, As shown in Figure 4.8 with the l2 parameter set to 0.01, the dropout rate of 0.4 resulted in the highest accuracy. Then the dropout rate was set to 0.4, and the l2 parameter of 0.2 consequenced in the highest accuracy. In a nutshell, as shown in Figure 4.9, the combination of 0.4 as the dropout rate and 0.2 as the L2 regularizer parameter produced the highest accuracy.

4.3.2 Models' Comparison

In addition, the semi-supervised model based on architecture 1 in Figure 4.6, was compared with the supervised model architecture 1 in Figure 4.5. Another comparison was conducted between different architectures for the semi-supervised model. One architecture was based on using layers in architecture 1 in Figure 4.5 for shared backbone layers in the semi-supervised model. And the second architecture was more complicated, which used layers in architecture 2 of Figure 4.7 for the shared backbone layers in the semi-supervised model inspired by [22, 45]. As shown in the graph of Figure 4.10 the semi-supervised model based on architecture 1 reached the highest accuracy of 90%.

4.4 Conclusion

In this chapter, a human study with fifteen users was completed. In the study, the SmartOS application collected multi-modal data from the users' computers and sent the data to AWS data servers. The collected data were used to predict human frustrations with their computers. Thus different machine learning model architectures were evaluated, and a semi-supervised model using a generative adversarial network (GAN) resulted in the highest accuracy of 90%.

Chapter 5

SmartOS: Improving Reinforcement Learning Algorithm Convergence Using Pre-training

In this thesis chapter, the SmartOS cortex was moved to the remote server so that it can learn across users and then the reinforcement learning algorithm in the SmartOS was improved to converge faster to the desired allocation resource policy to minimize user frustration. As we saw in Chapter 3, SmartOS takes a minimum of eight feedbacks to reach an optimal resource allocation policy for one synthetic scenario consisting of 4 applications. This minimum required feedbacks increases with the number of different scenarios and their complexity. The increment in required feedbacks decreases the smoothness and usability of the SmartOS and diverges us from the primary goal of SmartOS, reducing user frustration. In order to address this problem, different RL algorithms were compared to find which one converges faster for the real frustration scenarios reported in Chapter 4. After that, the best RL algorithm for SmartOS was pre-trained and compared in terms of its average convergence with the untrained version of the same algorithm in the real frustration scenarios reported in the human study in Chapter 4. So the main contributions of this chapter are as follows:

- First, all options in pre-training the RL are listed. Then it is explained that pre-training the RL algorithm with the model developed in Chapter 4 generalizes better in real everyday cases.
- Second, the architecture of one of the RL models tried in this chapter is detailed. All other RL algorithms follow a similar theme.

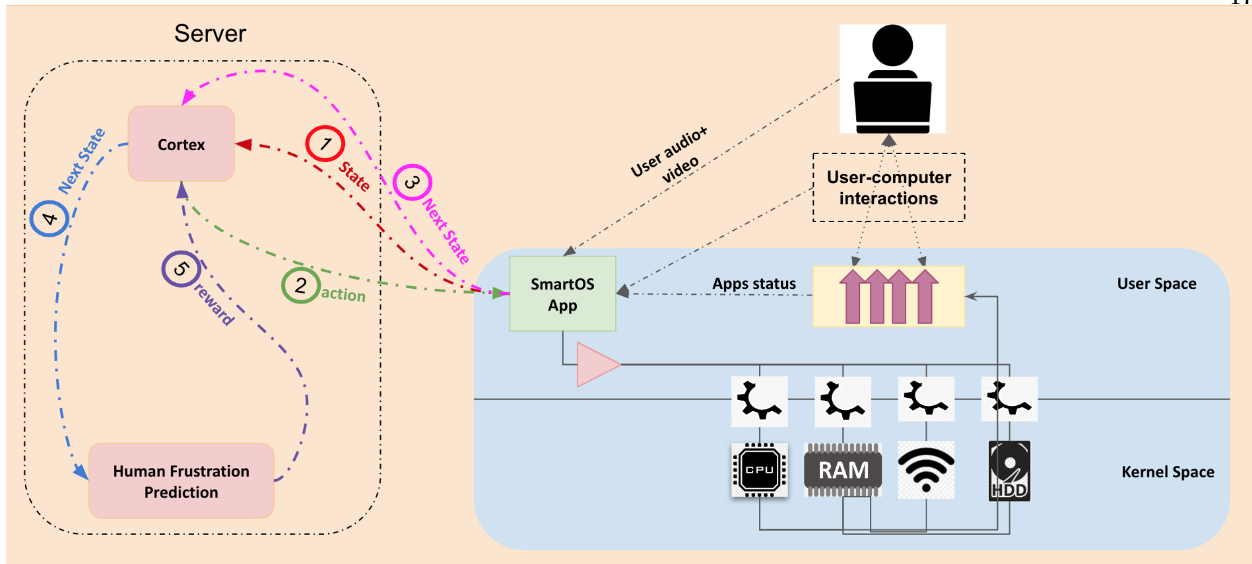

Figure 5.1: Pre-training The Reinforcement Algorithm in the SmartOS Using Third Methodology

- Third, different RL algorithms' average convergence times in the scenarios reported as frustration in Chapter 4 are compared.
- Then, it is studied whether pre-training the fastest RL model from the prior step 5 can converge faster on real-world scenarios reported as frustration in Chapter 4 or not.

5.1 Pre-training SmartOS Reinforcement Learning algorithm

In order to make the reinforcement learning converge faster to the optimal set of actions to maximize the reward, it should be pre-trained in a simulated environment. The methodologies for pre-training the RL in the SmartOS are as follows:

- One methodology is to develop synthetic scenarios and manually find the best actions for those scenarios. Then, place the smartOS app in those scenarios and feed it with synthetic feedbacks generated from a script guiding the SmartOS app towards the best set of actions we found earlier. The issue with this methodology is that developing enough scenarios for the SmartOS to generalize to real everyday scenarios a user may face is tedious and time-consuming.

- The second methodology is to let the SmartOS app run on my computer and give the application my feedback manually through the application report form. This methodology is better than the first methodology in terms of being trained on a higher number of scenarios, leading to a state that the SmartOS can generalize better in real everyday scenarios. Still, it should be noted that giving manual feedback is a tedious and time-consuming job and slows down the whole pre-traning process.

- As shown in graph of Figure 5.1, the third and the best methodology of all is to let the SmartOS app run on my computer, but instead of providing manual feedback to it, let the human frustration prediction machine learning model developed in Chapter 4 give it the prediction of my frustration with my computer based on my next state as a reward. This way, I should not provide anything manually, which speeds the whole pre-training process and makes it possible for the SmartOS to see even more everyday scenarios that help it generalize better in real everyday scenarios.

5.2 Reinforcement Learning Model Architecture

The state of the RL Learning model was a continuous vector, identical to the data collected in Section 4.1.2. The continues state made the representation of the state stronger. The action of the RL Learning model was a discrete vector, the same as the action in Section 3.2. The reward of the RL learning model in the pre-training was set to one minus the predicted value of the human frustration prediction model developed and trained in Chapter 4 based on the next state's value. And the reward of the RL learning model in the testing was set to one if the action was in the set of the best actions that resolved the user's frustration reason, otherwise zero.

One of the RL algorithms developed for the evaluation section is A2C [49]. A2C consists of two separate models, one for the actor and one for the critic. The actor model takes the state as the input and outputs the action resulting in the maximum reward based on its current knowledge. The critic model takes the state as the input and outputs the value of the state based on the insights it

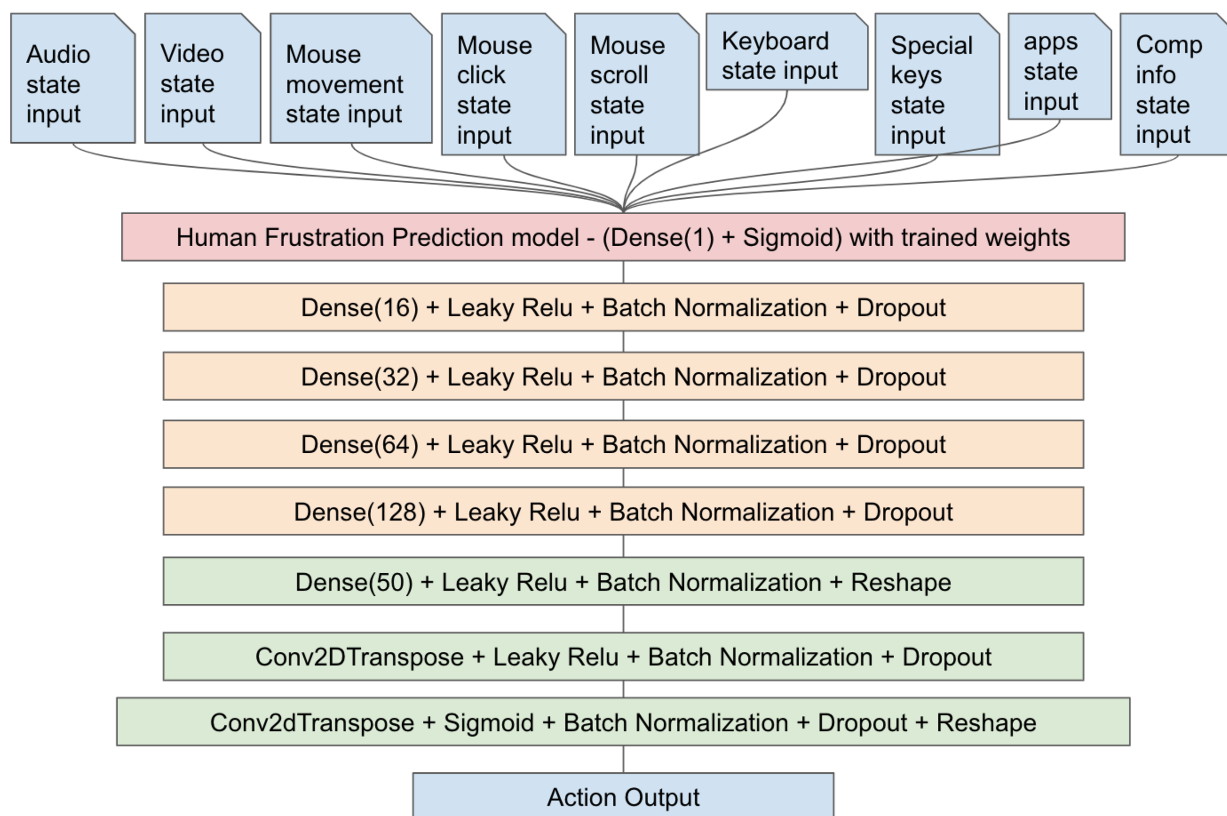

Figure 5.2: Actor Model Architecture in The SmartOS A2C Reinforcement Learning Model

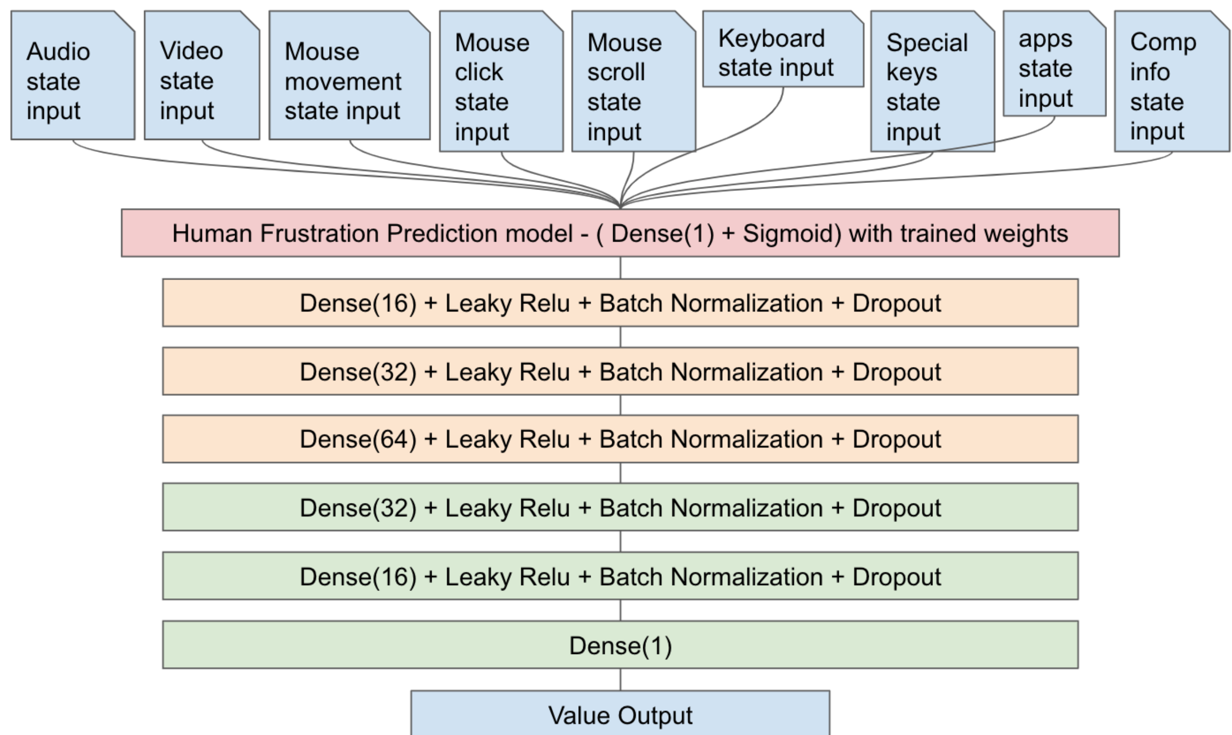

Figure 5.3: Critic Model Architecture in The SmartOS A2C Reinforcement Learning Model

has gained so far. The state inputs in actor and critic models are the same as the input in the human frustration prediction machine learning model, which was developed in Chapter 4. In addition, the output of that model was predicting human frustration, and reducing human frustration is the final goal of the RL algorithm. Hence, using **transfer learning** [7] and transferring the knowledge learned in Chapter 4 to the RL algorithm before pre-training is advantageous. Thus, as shown in Figures 5.2 and 5.3, the human frustration prediction model and its trained weights were used in both the actor and critic architectures. The only modification to the human frustration prediction model was removing its two last layers. The remaining human frustration prediction model layers were not trainable in actor and critic models.

Since the modified human frustration prediction model encoded the state into a tensor with the shape of 8 in such a way that it conveys the information required for predicting frustration. So this tensor first needed to be decoded and Then transformed into the action output.

As a result the following layers were added to the actor model:

(1) The output of the modified human frustration prediction model was fed to a dense layer with the output of 16 and an L2 regularizer with the parameter value 0f 0.2 followed by a Leaky Relu with the alpha of 0.2 as activation function and a batch normalization layer and a dropout layer with the dropout rate of 0.4.

(2) Then, the output of the Dropout layer was connected to another combination of layers similar to the 1 with only a difference in the output dimension of the dense layer, which was 32.

(3) Later, the the output of 2 was connected to a third combination of layers similar to the 1 in which the dense layer's output dimension was 64.

(4) Afterwards, the 3's output was connected to a fourth combination of layers similar to the 1 with 128 as the dense layer's output dimension.

(5) The last dense layer received the output of 4 and produced a tensor with the shape of 50,

which was fed into a Leaky Relu and a batch normalization with the same parameters as 1 and then reshaped into a tensor with the shape of (50,1,1)

(6) After all the dense layers, there was a Conv2dTranpose with one output and 4 and 2 as the filter and stride sizes, respectively, and the same padding. The Conv2dTranpose uses an L2 regularizer with the parameter value 0f 0.2. The Conv2dTranpose output was followed by a Leaky Relu, batch normalization, and a dropout layer with the same parameters as the 1.

(7) The last layer was another Conv2dTranspose with the same parameters as 6 with a Sigmoid activation function followed by a batch normalization, and a dropout with the same parameters as 1. The output of the dropout layer was reshaped to a tensor with the dimension of (200, 4), which is the action output.

The critic model architecture is very similar to the actor model architecture. The critic has the same modified human frustration prediction model used in the actor model followed by layers identical to the immediate layer in the actor model after the modified human frustration prediction model with 16, 32, 64, 32, 16 as the outputs of the dense layer. Then the last layer is another dense layer with only a single output which is the value of the state.

The selection of layers with increasing output sizes are for decoding while the selection of layers with increasing output sizes are for transformingencoding. Using 3 or 4 dense layers for decoding and encoding is a common practice in deep learning.

All the other reinforcement learning algorithms evaluated in Section 5.3, including the policy gradient [64], Actor-critic [33], Proximal Policy Optimization (PPO) [60] , Deep Deterministic Policy Gradient (DDPG) [43], Twin Delayed Deep Deterministic Policy Gradient (T3D) [19], follow the same design patterns, and they all use transfer learning.

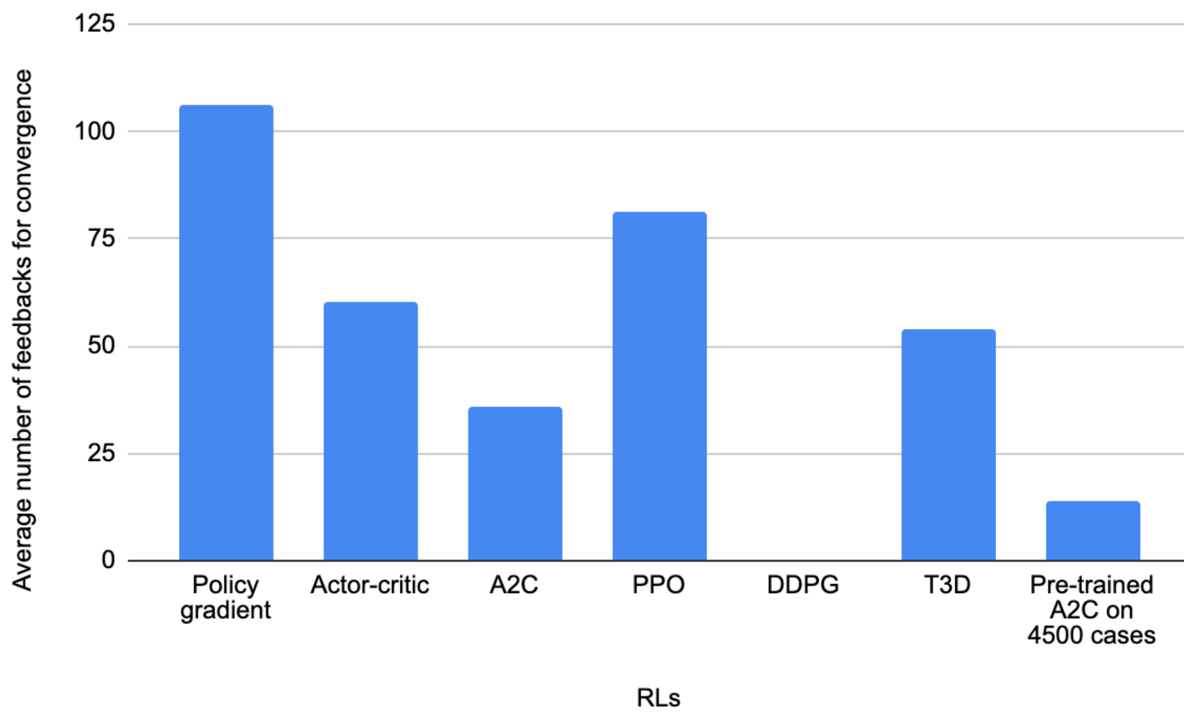

Figure 5.4: Reinforcement Learning Algorithm Models' Average Convergence

5.3 Evaluation

For the evaluation, different RL algorithm models' average convergence times were compared using the frustration reports submitted in the human study in Chapter 4. First, based on the submitted reason, the important application for the user was extracted for each report. Then, based on the computer's total resource usage, the important application's, and other applications' resource usages at the time of report, a set of best actions was produced which would resolve the frustration of the report's owner. After that, a simulated environment was developed using python, which fed the state data at the time of the report to the RL algorithm model as the input and then received an action produced by the RL algorithm model. If the action was a member of the best set of actions for that report, the simulated environment gave the RL the reward of 1, and the RL would converge. However, suppose the RL was unsuccessful in finding an action that resolves the user's frustration. In that case, the simulated environment restarted and fed the state of the report again to the RL. This procedure continued until the RL converged. One hundred and nine reports were used for the evaluation. Then, the average convergence for each report was computed based on ten trials. Finally, the average of all the reports' average convergence times was set as a metric for the RL algorithm models' evaluation. The evaluation results are shown in Figure 5.4.

A2C achieved the fastest convergence time amongst all the reinforcement learning algorithm models, and DDPG did not converge. Therefore, the pre-training of the A2C model was performed using the same method shown in Figure 5.1 on my computer. After 4500 updates to the A2C model, it was compared with the non-trained A2C model. As shown in Figure 5.4, the pre-training of the A2C model improves its convergence by 60.83%.

5.4 Conclusion

In this chapter, different RL algorithm models were developed and evaluated on the real user frustration cases collected from the previous chapter. The A2C was selected as the best RL algorithm for the pre-training phase in terms of convergence time. Then pre-training of A2C was

done on my computer using the human frustration prediction ML model developed in the previous chapter so that the A2C model can generalize and converge faster in everyday cases. The pre-trained A2C converged faster by 60.83%.

Chapter 6

Conclusion and Future Directions

6.1 Conclusion

This thesis identified the one-size-fits-all problem of operating systems in allocating resources to applications based on user preferences. As a result, The smartOS architecture was defined to fill this need.

A prototype of smartOS was implemented, which was used in a performance analysis against other heuristics. In addition, the Convergence time and overhead of the smartOS prototype implementation were measured.

Furthermore, the SmartOS application was developed and used in an IRB-approved human study. In the human study, 122,577 real-world scenarios from 15 users were collected, from which 665 scenarios were labeled as frustration, and the remaining were unlabeled. A semi-supervised human frustration prediction model using GAN was developed, trained, and compared against other ML models.

In the last chapter, different RLs were implemented using transfer learning, and they were applied in the cortex of the SmartOS app to work with complicated everyday scenarios with continuous states. An evaluation was completed to test different RL models' convergence times based on the real-world collected data in the human study. The RL was pre-trained on my computer and was evaluated against untrained RL in terms of convergence time.

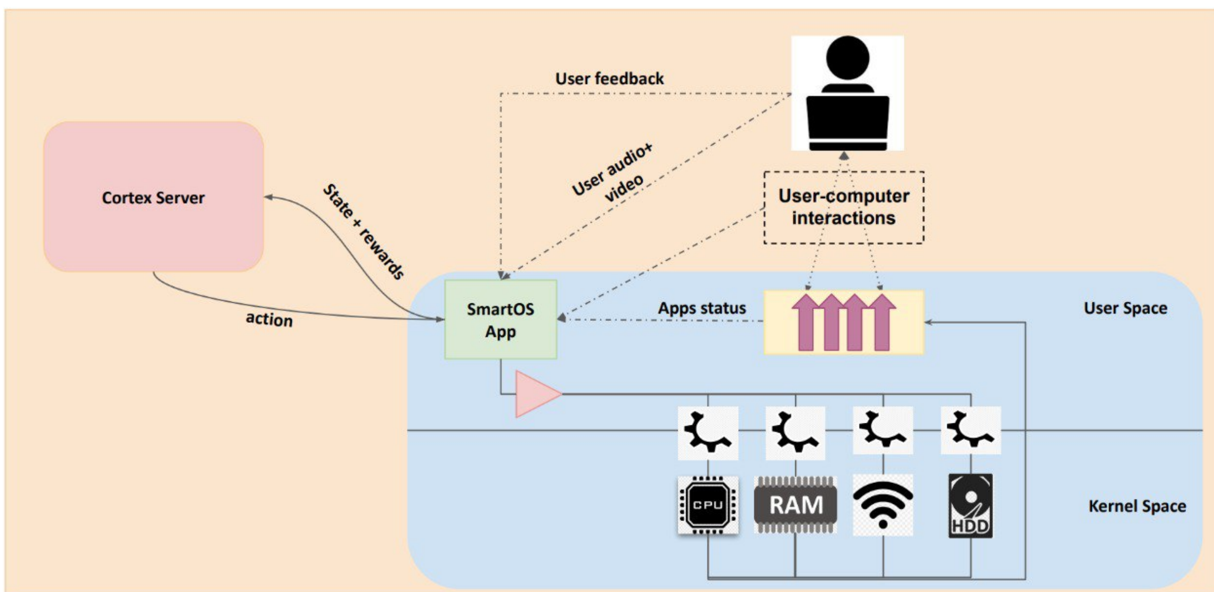

Figure 6.1: Architecture of Active SmartOS App, including App and Pre-trained A2C Algorithm Located in Cortex Server

6.2 Future Direction

An active human study may be completed to close the loop of this research for a possible future direction. Active human study implies the SmartOS app should actively change the resource allocation policy of running applications on the users' computers based on the received action from cortex. The SmartOS app will utilize the pre-trained A2C as the cortex. 4.1.2 and the users will submit their frustration using the report button as the rewards to the A2C algorithm the same way they did in Chapter 4. The A2C algorithm will continue to be trained based on the newly received reports. By comparing the number and type of reports between the passive study in Chapter 4 and this active study, one can measure the effectiveness of an active SmartOS.

One possible architecture for the SmartOS in this human study is illustrated in Figure 6.1 The SmartOS app will be responsible for sending the state and rewards (collected data in ref) to the cortex located on a remote server that runs the pre-trained A2C algorithm. After the cortex has received the state and rewards, it stores them and feeds them to the pre-trained A2C algorithm. The pre-trained A2C algorithm calculates the necessary action to reduce the user's frustration with the computer. Then the cortex sends the calculated action to the SmartOS app residing on the user computer to change the resource allocation policy of the running applications in the user computer using memory, CPU, network, and d/O knobs.

Due to the effectiveness of using the human frustration prediction model in pre-training the A2C algorithm, the A2C algorithm in the active SmartOS app could receive its rewards from the human frustration prediction model as a second source rather than waiting for the user to report their frustration manually. However, the user manual feedback should overwrite the prediction of the human frustration prediction model in case they are different. Using this method, we will not bother users by getting manual feedback. It will take less time for the same amount of rewards. Thus, SmartOS can converge faster using 14.1 rewards on average from the human frustration prediction model on everyday real-world scenarios compared to relying on receiving manual feedback only.

If the SmartOS could not converge, it could return to Linux's default resource allocation policy. We would investigate how to ensure the property that SmartOS performs no worse than the default policy.

Another possible future direction is studying the hidden information in the collected data from the human study in chapter 4. Based on this investigation, the ML models employed in this thesis and their results can be explained.

Bibliography

[1] 10 usability tips based on research studies.

[2] 90 seconds... that's how long an emotion lasts... yeah, right.

[3] Completely fair scheduler.

[4] Python 3.6.0.

[5] Digital around the world, 2021.

[6] Ibrahim Umit Akgun, Ali Selman Aydin, and Erez Zadok. Kmlib: Towards machine learning for operating systems. In Proceedings of the On-Device Intelligence Workshop, co-located with the MLSys Conference, pages 1–6, 2020.

[7] Stevo Bozinovski and Ante Fulgosi. The influence of pattern similarity and transfer learning upon training of a base perceptron b2. In Proceedings of Symposium Informatica, volume 3, pages 121–126, 1976.

[8] Scott Brave and Cliff Nass. Emotion in human-computer interaction. In The human-computer interaction handbook, pages 103–118. CRC Press, 2007.

[9] Linqin Cai, Yaxin Hu, Jiangong Dong, and Sitong Zhou. Audio-textual emotion recognition based on improved neural networks. Mathematical Problems in Engineering, 2019, 2019.

[10] Irina Ceaparu, Jonathan Lazar, Katie Bessiere, John Robinson, and Ben Shneiderman. Determining causes and severity of end-user frustration. International journal of human-computer interaction, 17(3):333–356, 2004.

[11] O Chapelle, B Schölkopf, and A Zien. Semi-supervised learning cambridge, 2006.

[12] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. IEEE signal processing magazine, 35(1):53–65, 2018.

[13] Théo Deschamps-Berger, Lori Lamel, and Laurence Devillers. End-to-end speech emotion recognition: challenges of real-life emergency call centers data recordings. In 2021 9th International Conference on Affective Computing and Intelligent Interaction (ACII), pages 1–8. IEEE, 2021.

[14] Siddharth Dias, Sidharth Naik, Sreepraneeth K, Sumedha Raman, and Namratha M. A machine learning approach for improving process scheduling: A survey. International Journal of Computer Trends and Technology (IJCTT), 43(1):1–4, 2017.

[15] Zoran Duric, Wayne D Gray, Ric Heishman, Fayin Li, Azriel Rosenfeld, Michael J Schoelles, Christian Schunn, and Harry Wechsler. Integrating perceptual and cognitive modeling for adaptive and intelligent human-computer interaction. Proceedings of the IEEE, 90(7):1272–1289, 2002.

[16] Clayton Epp, Michael Lippold, and Regan L Mandryk. Identifying emotional states using keystroke dynamics. In Proceedings of the sigchi conference on human factors in computing systems, pages 715–724, 2011.

[17] Paul Freihaut, Anja S Göritz, Christoph Rockstroh, and Johannes Blum. Tracking stress via the computer mouse? promises and challenges of a potential behavioral stress marker. Behavior Research Methods, 53(6):2281–2301, 2021.

[18] Nico H Frijda et al. Varieties of affect: Emotions and episodes, moods, and sentiments. 1994.

[19] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In International conference on machine learning, pages 1587–1596. PMLR, 2018.

[20] Surjya Ghosh. Emotion-aware computing using smartphone. In 2017 9th International Conference on Communication Systems and Networks (COMSNETS), pages 592–593. IEEE, 2017.

[21] Sepideh Goodarzy, Maziyar Nazari, Richard Han, Eric Keller, and Eric Rozner. Resource management in cloud computing using machine learning: A survey. In 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), pages 811–816, 2020.

[22] Hatice Gunes and Maja Pantic. Dimensional emotion prediction from spontaneous head gestures for interaction with sensitive artificial listeners. In International conference on intelligent virtual agents, pages 371–377. Springer, 2010.

[23] Vedika Gupta, Stuti Juyal, Gurvinder Pal Singh, Chirag Killa, and Nishant Gupta. Emotion recognition of audio/speech data using deep learning approaches. Journal of Information and Optimization Sciences, 41(6):1309–1317, 2020.

[24] Pavol Harár, Radim Burget, and Malay Kishore Dutta. Speech emotion recognition with deep learning. In 2017 4th International conference on signal processing and integrated networks (SPIN), pages 137–140. IEEE, 2017.

[25] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. Nature, 585(7825):357–362, September 2020.

[26] Hado Hasselt. Double q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, Advances in Neural Information Processing Systems, volume 23. Curran Associates, Inc., 2010.

[27] Martin Thomas Hibbeln, Jeffrey L Jenkins, Christoph Schneider, Joseph Valacich, and Markus Weinmann. How is your user feeling? inferring emotion through human-computer interaction devices. Mis Quarterly, 41(1):1–21, 2017.

[28] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[29] Henry Hoffmann. Jouleguard: Energy guarantees for approximate applications. In Proceedings of the 25th Symposium on Operating Systems Principles, pages 198–214, 2015.

[30] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic knobs for responsive power-aware computing. ACM SIGARCH computer architecture news, 39(1):199–212, 2011.

[31] Preeti Khanna and Mukundan Sasikumar. Recognising emotions from keyboard stroke pattern. International journal of computer applications, 11(9):1–5, 2010.

[32] Paul R Kleinginna and Anne M Kleinginna. A categorized list of emotion definitions, with suggestions for a consensual definition. Motivation and emotion, 5(4):345–379, 1981.

[33] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. Advances in neural information processing systems, 12, 1999.

[34] Julian Kunkel, Michaela Zimmer, and Eugen Betke. Predicting performance of non-contiguous i/o with machine learning. In Julian M. Kunkel and Thomas Ludwig, editors, High Performance Computing, pages 257–273, Cham, 2015. Springer International Publishing.

[35] Jonathan Lazar, Adam Jones, Katie Bessiere, Irina Ceaparu, and Ben Shneiderman. User frustration with technology in the workplace. 2003.

[36] Jonathan Lazar, Adam Jones, Mary Hackley, and Ben Shneiderman. Severity and impact of computer user frustration: A comparison of student and workplace users. Interacting with Computers, 18(2):187–207, 2006.

[37] Jonathan Lazar, Adam Jones, and Ben Shneiderman. Workplace user frustration with computers: An exploratory investigation of the causes and severity. Behaviour & Information Technology, 25(03):239–251, 2006.

[38] Margaret Lech, Melissa Stolar, Christopher Best, and Robert Bolia. Real-time speech emotion recognition using a pre-trained image classification network: Effects of bandwidth reduction and companding. Frontiers in Computer Science, 2:14, 2020.

[39] Po-Ming Lee, Wei-Hsuan Tsui, and Tzu-Chien Hsiao. The influence of emotion on keyboard typing: an experimental study using auditory stimuli. PloS one, 10(6):e0129056, 2015.

[40] Mao Li, Bo Yang, Joshua Levy, Andreas Stolcke, Viktor Rozgic, Spyros Matsoukas, Constantinos Papayiannis, Daniel Bone, and Chao Wang. Contrastive unsupervised learning for speech emotion recognition. In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 6329–6333. IEEE, 2021.

[41] Zheng Lian, Jianhua Tao, Bin Liu, Jian Huang, Zhanlei Yang, and Rongjun Li. Context-dependent domain adversarial neural network for multimodal emotion recognition. In INTERSPEECH, pages 394–398, 2020.

[42] Yu Liang, Jinheng Li, Rachata Ausavarungnirun, Riwei Pan, Liang Shi, Tei-Wei Kuo, and Chun Jason Xue. Acclaim: Adaptive memory reclaim to improve user experience in android systems. In 2020 USENIX Annual Technical Conference (USENIX ATC 20), pages 897–910. USENIX Association, July 2020.

[43] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[44] Jean-Pierre Lozi, Baptiste Lepers, Justin Funston, Fabien Gaud, Vivien Quéma, and Alexandra Fedorova. The linux scheduler: A decade of wasted cores. In Proceedings of the Eleventh European Conference on Computer Systems, EuroSys '16, New York, NY, USA, 2016. Association for Computing Machinery.

[45] Surbhi Madan, Monika Gahalawat, Tanaya Guha, and Ramanathan Subramanian. Head matters: Explainable human-centered trait prediction from head motion dynamics. In Proceedings of the 2021 International Conference on Multimodal Interaction, pages 435–443, 2021.

[46] Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert Latham, Robert Ross, Shane Snyder, and Stefan M. Wild. Machine learning based parallel i/o predictive modeling: A case study on lustre file systems. In Rio Yokota, Michèle Weiland, David Keyes, and Carsten Trinitis, editors, High Performance Computing, pages 184–204, Cham, 2018. Springer International Publishing.

[47] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C Evans, Steve Gribble, et al. Snap: A microkernel approach to host networking. In Proceedings of the 27th ACM Symposium on Operating Systems Principles, pages 399–413, 2019.

[48] Nikita Mishra, Connor Imes, John D Lafferty, and Henry Hoffmann. Caloree: Learning control for predictable latency and low energy. ACM SIGPLAN Notices, 53(2):184–198, 2018.

[49] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937. PMLR, 2016.

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[51] Edmilson Morais, Ron Hoory, Weizhong Zhu, Itai Gat, Matheus Damasceno, and Hagai Aronowitz. Speech emotion recognition using self-supervised features. In ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 6922–6926. IEEE, 2022.

[52] Atul Negi and P. Kishore Kumar. Applying machine learning techniques to improve linux process scheduling. In TENCON 2005 - 2005 IEEE Region 10 Conference, pages 1–6, 2005.

[53] Avar Pentel. Patterns of confusion: Using mouse logs to predict user's emotional state. In UMAP Workshops, 2015.

[54] Mahwish Pervaiz and Tamim Ahmed Khan. Emotion recognition from speech using prosodic and linguistic features. International Journal of Advanced Computer Science and Applications, 7(8), 2016.

[55] Rosalind W Picard and Jonathan Klein. Computers that recognise and respond to user emotion: theoretical and practical implications. Interacting with computers, 14(2):141–169, 2002.

[56] Fuji Ren and Yanwei Bao. A review on human-computer interaction and intelligent robots. International Journal of Information Technology & Decision Making, 19(01):5–47, 2020.

[57] Gavin A Rummery and Mahesan Niranjan. On-line Q-learning using connectionist systems, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[58] Sergio Salmeron-Majadas, Olga C Santos, and Jesus G Boticario. Exploring indicators from keyboard and mouse interactions to predict the user affective state. In Educational Data Mining 2014, 2014.

[59] Anas Samara, Leo Galway, Raymond Bond, and Hui Wang. Affective state detection via facial expression analysis within a human–computer interaction context. Journal of Ambient Intelligence and Humanized Computing, 10(6):2175–2184, 2019.

[60] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[61] Mehmet Cenk Sezgin, Bilge Günsel, and Güneş Karabulut Kurt. A novel perceptual feature set for audio emotion recognition. In 2011 IEEE International Conference on Automatic Face & Gesture Recognition (FG), pages 780–785. IEEE, 2011.

[62] David Silver. Lectures on reinforcement learning. URL: `https://www.davidsilver.uk/teaching/`, 2015.

[63] Mandeep Singh and Yuan Fang. Emotion recognition in audio and video using deep neural networks. arXiv preprint arXiv:2006.08129, 2020.

[64] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems, 12, 1999.

[65] CJC Watkins. H. learning from delayed rewards, ph. d. thesis, cambridge university, 1989. 1989.

[66] Oren Wright. Emotion recognition from voice in the wild. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA PITTSBURGH United States, 2019.

[67] Takashi Yamauchi, Anton Leontyev, and Moein Razavi. Assessing emotion by mouse-cursor tracking: Theoretical and empirical rationales. In 2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII), pages 89–95. IEEE, 2019.

[68] Takashi Yamauchi and Kunchen Xiao. Reading emotion from mouse cursor motions: Affective computing approach. Cognitive science, 42(3):771–819, 2018.

[69] Yiying Zhang and Yutong Huang. " learned" operating systems. ACM SIGOPS Operating Systems Review, 53(1):40–45, 2019.

[70] Philippe Zimmermann, Sissel Guttormsen, Brigitta Danuser, and Patrick Gomez. Affective computing—a rationale for measuring mood with mouse and keyboard. International journal of occupational safety and ergonomics, 9(4):539–551, 2003.