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

Sepideh Goodarzy
University of Colorado Boulder

PhD Thesis Defense
Aug 16th, 2022

Operating Systems

They are everywhere!

How Many Users?

- At 2018, 98% of households in US -> at least one Computer [1]
- At 2021, 4.88 billion internet users in the world [2]

[1] Computer and Internet Use in the United States: 2018
[2] Digital Around the World — DataReportal – Global Digital Insights

What Do Operating Systems Do?

- Manage and distribute computer resources among applications.
- One-size-fits-all!
- Linux CFS scheduler divides up the CPU equally among all applications assigning them all equal static priority
- Not good at capturing each individual preferences and the changes of those preferences along time!

Humans Frustration With Their Computer

The time lost due to the frustrating experiences ranged from 30.5% to 45.9% of time spent on the computer. [3]

Top 4 frustration causes [3]

- **Error messages**
- **Dropped network connections**
- **Long download times**
- **Hard-to-find features**

[3] Understanding Computer User Frustration: Measuring and Modeling the Disruption from Poor Designs

What Can Be Done?

- Give more resources (CPU, memory, network bandwidth, Disk I/O, etc.) to the applications that matter **MORE**!
  - Manually By user: Not all users are computer experts! time consuming and a frustrating job!
  - Automatically By OS
- What applications matter more ? Is it always the foreground ?
  - Editing a doc in Microsoft Office Word, while listening to music on youtube in Google Chrome, upgrading some software in the background and monitoring the stock widget on top of your desktop
- ***Based on the context, different users care more about some applications more than the others!***
- How we can find the important applications ?

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.

- 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

Prior Work: ML in Operating systems

- Acclaim: Improving user experience in android OS by predicting the the next memory page using ML. It assumes the foreground and video/audio are the most important applications in memory reclamation (Liang Yu 2020)
  - Static prioritization
- ML in linux process scheduling (Atul Negi 2005, Siddharth Dias 2017) , ML in I/O scheduling (Sandeep Madiredd 2019, Julian Kunkel 2015), ML in network cloud systems (Sepideh Goodarzy 2020)
  - One resource dimension and no user preferences
- Control theory + ML (Nikita Mishra 2018, Henry Hoffmann 2015, Henry Hoffmann 2011)
  - No user preferences and not changing quickly enough to changes in the environment due to offline learning
- Reviews on the possible future directions of learning based OSes and their challenges
  - Not implemented none of their ideas.

Prior work: Human frustration Prediction

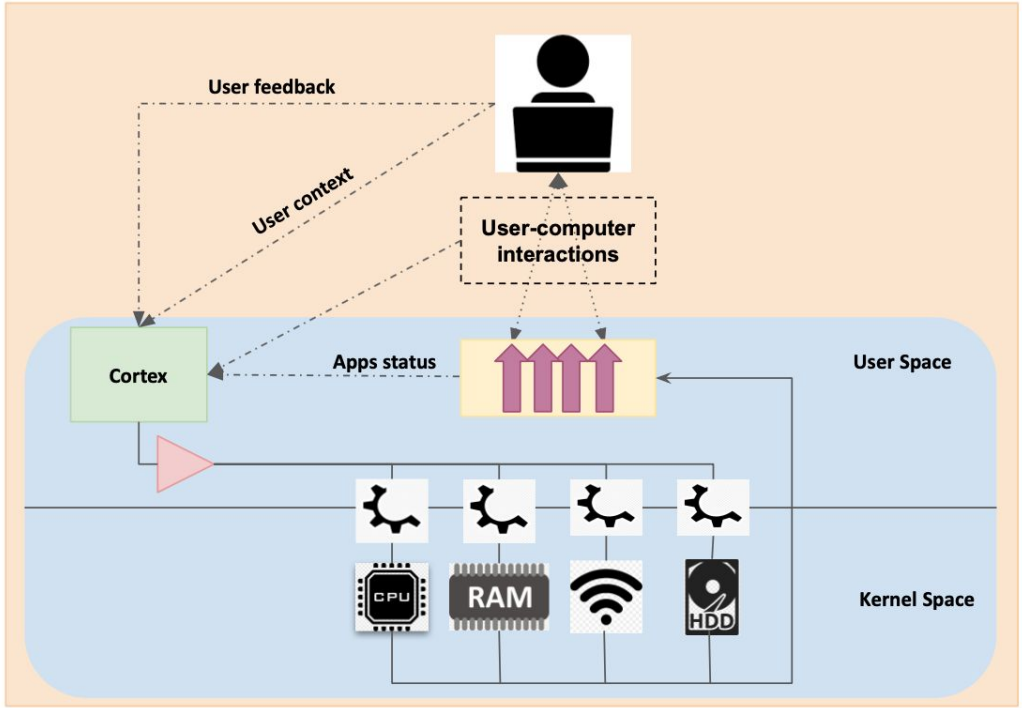
- Emotion prediction in HRI (Fuji Ren 2020) :
  - Robots: Intelligent beings capable of understanding humans
  - Computers: task-oriented devices
- Facial-expressions solely in HCI (Anas Samara 2019):
  - Can’t interpret human emotions
  - Difference between HCI and human-human interaction
- Correlation of keyboard stroke patterns (Preeti Khanna 2010, Sergio Salmeron-Majadas 2014) or mouse movement speed and direction with emotions in HCI (Martin Thomas Hibbeln 2017):
  - One dimension only and not focusing on human frustration with the computer

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.

- **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

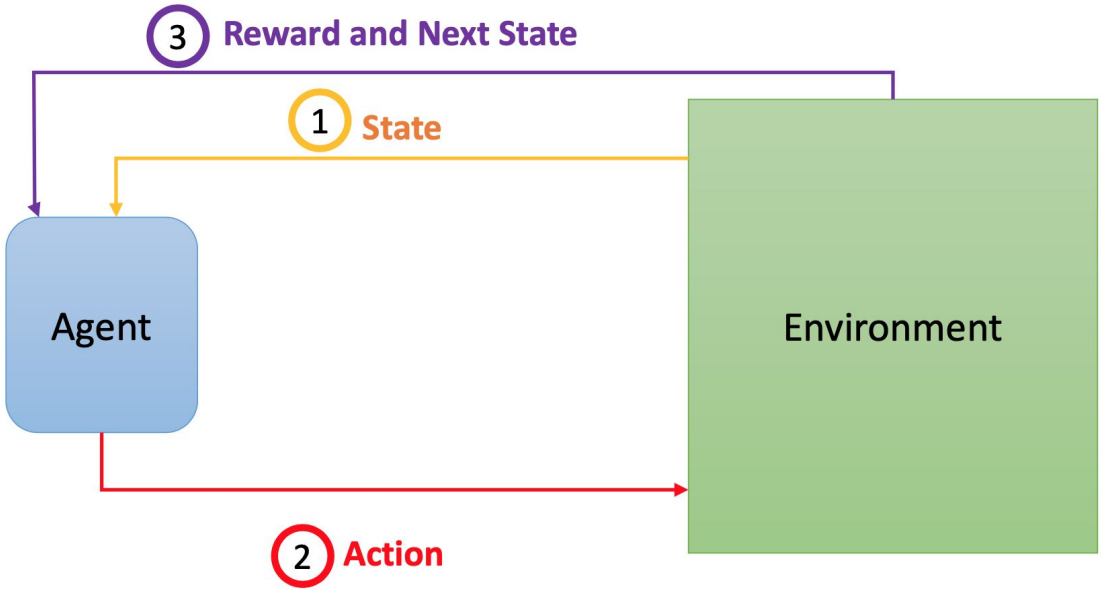
SmartOS Prototype Architecture



Cortex

- Heuristic based: Needs knowledge beforehand! Static! User preferences are changing constantly!
- Supervised/Unsupervised Machine Learning:
  - One time training: Static! User preferences are changing constantly!
  - Cyclic training/eval/deploy: What is the correct frequency to do re-training? What happens if the user preferences differ from trained model faster than the re-traning frequency ? lengthy and compute-expensive re-training while the user is frustrated!

Reinforcement Learning



Reinforcement learning in SmartOS

Discrete State:

CPU	Memory	Network	Disk I/O	Fg	Audio/video
0/1	0/1	0/1	0/1	0/1	0/1

Discrete Action:

CPU	Memory	Network	Disk I/O
0/1	0/1	0/1	0/1

Discrete Reward:

Synthetic, +1 for the action leading to best performance and 0 otherwise

Resource Allocation Interfaces in SmartOS

- **CPU:** *Nice value*
  - *High prio: -20*
  - *Normal Prio: 0*
- **Memory:** *OOM Adjacent score and Cgroup memory swappiness*
  - *High prio: -1000 for score and 0 for swappiness*
  - *Normal prio: 0 for score and 60 for swappiness*
- **I/O:** *I/O nice*
  - *High prio: 0 real-time class*
  - *Normal prio: 4 none class (default)*
- **Network bandwidth:** *Cgroup network I/O prio map*
  - *High prio: 10*
  - *Normal prio: 0*

Evaluation and Preliminary Result

- Test various static prioritization heuristics and automated learning policies in different synthetic scenarios
- Compare how good they are at giving priority to important tasks by evaluating each task’s performance (#operations/seconds)

Different Static Prioritization Heuristics

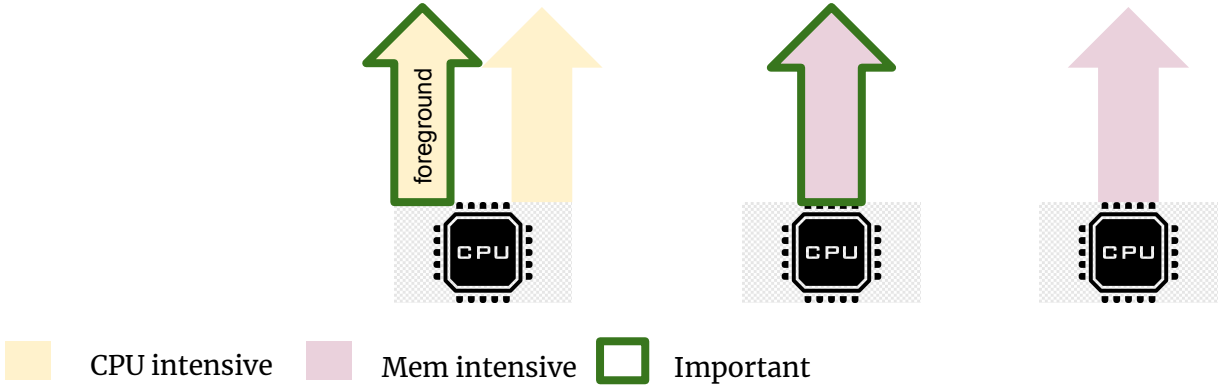
- **Linux:** *default linux CFS scheduler*
- **Fg only:** *high priority for foreground application for all resources*
- **Fg + video/audio:** *high priority for foreground and video/audio applications for all resources*
- **Fg + dependent:** *high priority for the foreground application and all other applications that foreground performance depends on.*
    - *predefined directed acyclic graph to store dependencies between applications*

Different Static Prioritization Heuristics (Cont.)

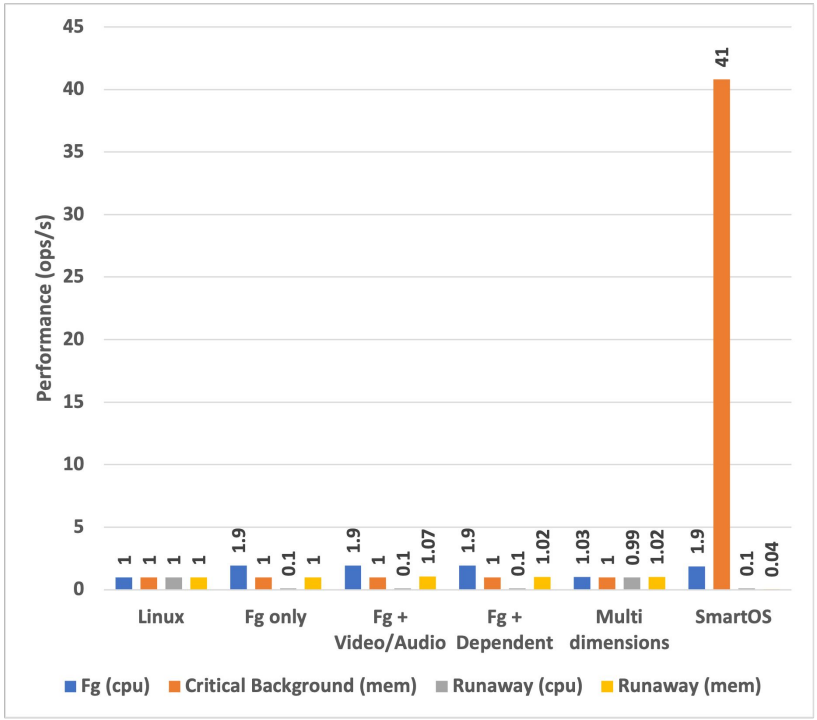
- **Multi-dimension:**
  - *Uses a predefined map that stores the essential resources per application's performance*
  - *High prio for the foreground application in all resources necessary to its performance*
  - *Assign the remaining resources to the important applications to the user*
  - *Important applications are stored in a hash map defined by asking the user in the beginning.*

Different Application Behaviour in Synthetic Scenarios

- CPU intensive or memory intensive or depend on a random resource or etc.
- Ubuntu 20.04 virtual machine with 8 GB of memory, 4 processors, and 50 GB VDI disk drive
- One of the scenarios : Random resource dependency for each application, not enough memory -> swapping out



Applications' performances for one of the scenarios:



For more scenarios please refer to the write-up

SmartOS Dynamicity and Convergence

- 4 different scenarios
- 60 seconds on each scenario (extreme case)
- Ask for the feedback after each scenario and move to the next scenario

With less frequent change of applications, SmartOS is able to achieve convergence faster -> 8 feedbacks for a single scenario

RL Algorithm	Feedbacks
DQN	52000
QLearning	28400
Sarsa	3680
Double Qlearning	2400
A2C	1600
Monte Carlo	400

SmartOS Overhead

- SmartOS adaptation to each user feedback takes:
  - 0.218 ms total execution time
  - 0.21 ms pure CPU time
  - less than human adaptation time which is in order of seconde
  - less than the required time for re-training a supervised/unsupervised model
- Required memory for cortex: 21.3 MB
- Negligible overhead!

Lessons Learned

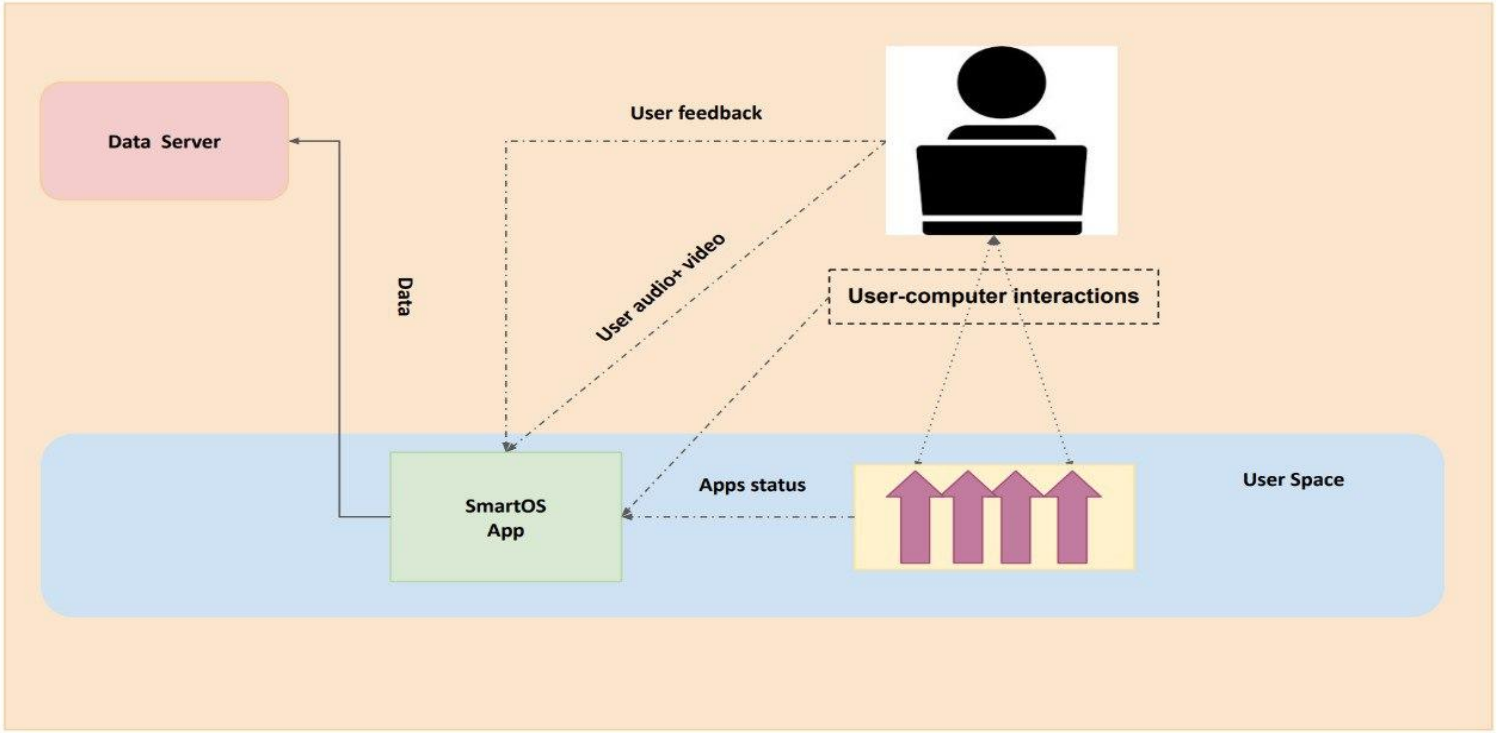
1. Test Real world scenarios using IRB-approved human study
2. Collecting more data for a stronger representation of the state
3. Developing RL that accepts continuous state
4. Converge faster (8 feedbacks are required for converging in a scenario consisting of 4 applications. More feedbacks are required for more complicated everyday scenarios)
5. What about cold start issues?
6. Place the cortex in the cloud and learn across users

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.

- 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

Human frustration Prediction



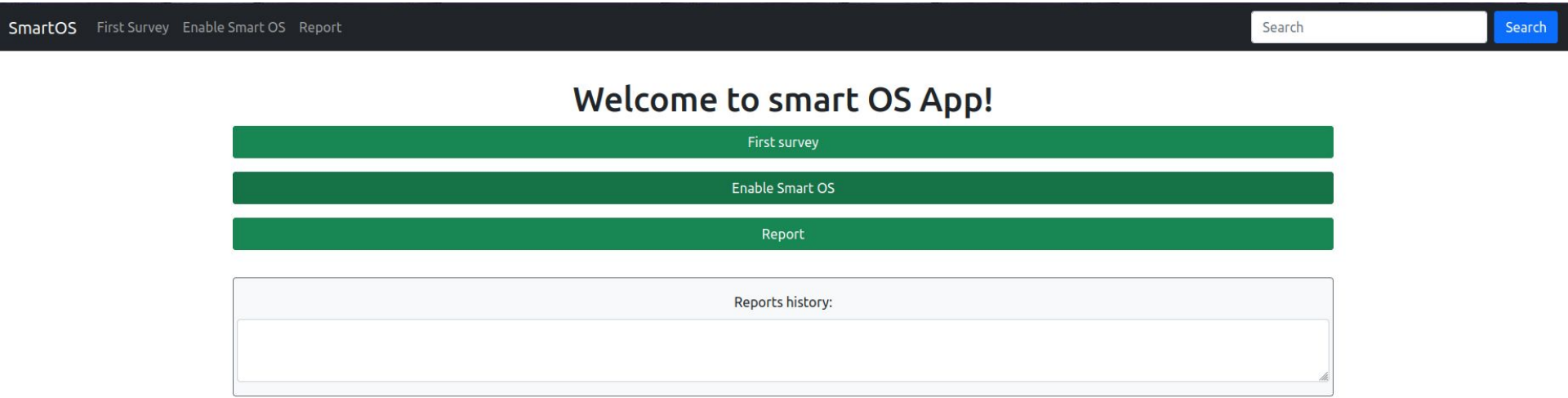
Data

- **User-computer interactions**:
  - *The mouse cursor's x/y positions, the mouse clicks' x/y positions, the mouse scrolls' x/y positions, and the scrolls' amount in the x/y coordinate along with timestamps at millisecond precision rates*
  - *All the keys clicks and special keys clicks (such as ESC, delete, enter, space, etc.) with timestamps at a millisecond precision rates*
- **User audio**:
  - *MFCCS*
- **User video**:
  - *Extracted head positions from recorded video frames using CNN face detection model running on the user computer*
- **Applications' and Computer's states**:
  - *System-wide information such as computation, memory usage, network bandwidth, and input/output bandwidth of the running applications in the computer and the computer*
- **User Feedback**:
  - *User reported frustrations with the system through the application*

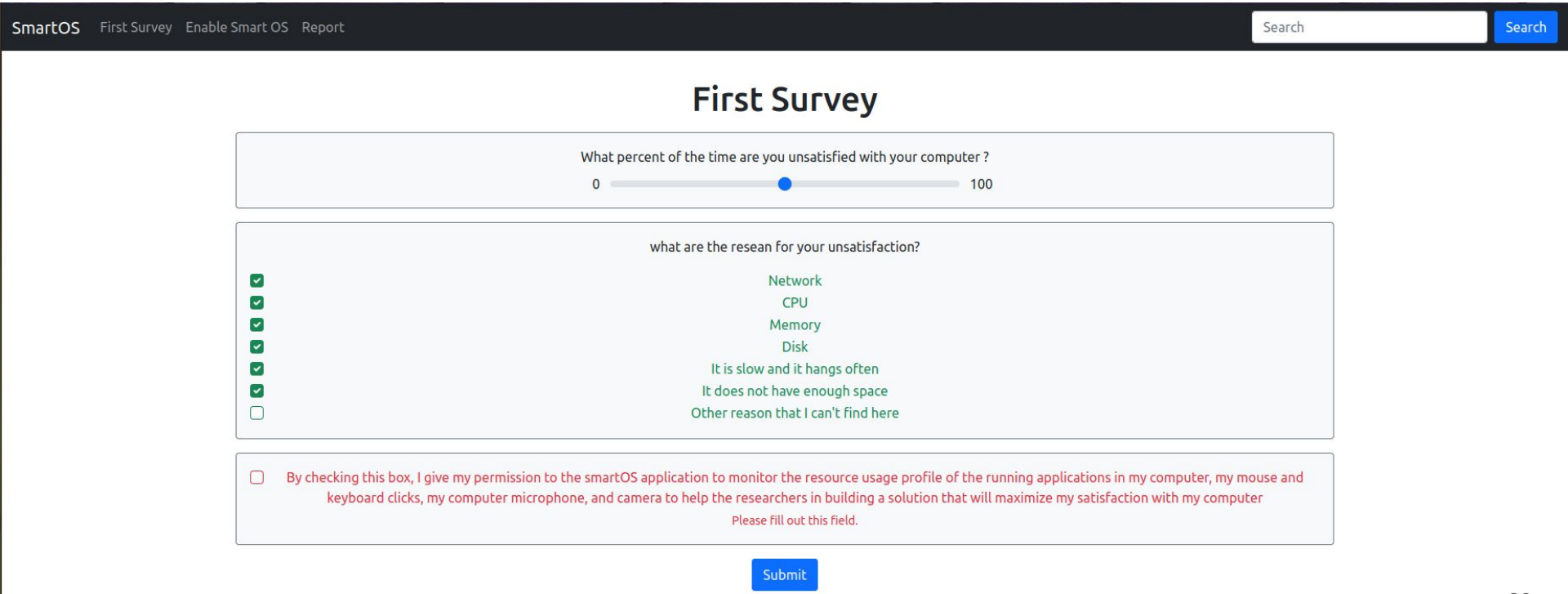
IRB-approved Human Study

- 15 ubuntu 20.04 users
  - 18 years or older
  - Residing in the US
  - The ubuntu machine with at least 8 Gigabytes of memory and four processors with video and audio recording capability.
- 2 weeks
- Included privacy protection
  - Not collecting any identifiable information such as names, email addresses, etc.
  - Not collecting keys' values, raw audios and raw videos
  - Generating random ID for each user and attaching it to the user's data instead of emails or other identifiable information

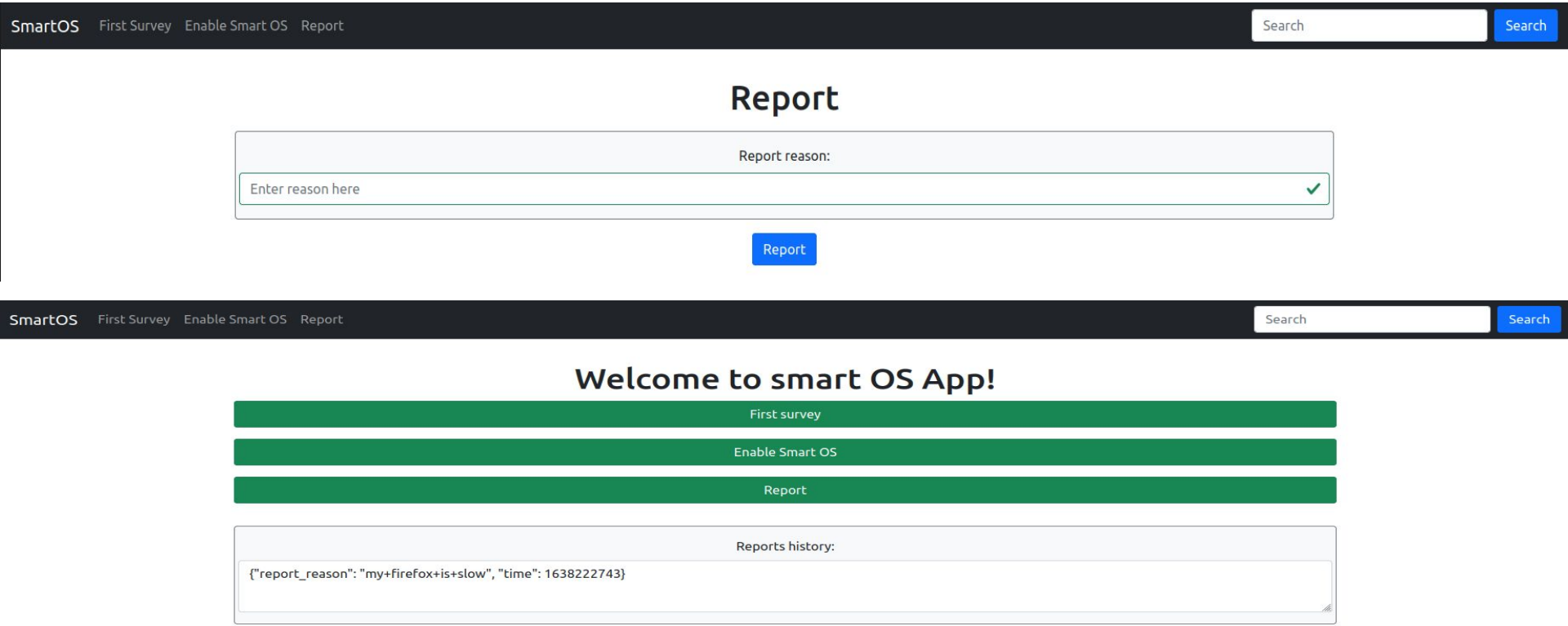
User Study SmartOS App: Central Page



User Study SmartOS App: App First Survey Form



User Study SmartOS App: Report Form



Collected Data Analysis

Data cleaning and Preprocessing

After every 10 seconds, the server received data from the SmartOS App and it did the cleaning and preprocessing for each data modal

- Ground truth values(Labels) [4]
  - Human emotions usually last 90 seconds.
  - Based on the timestamps of all the reports submitted by the users in the human study, All the data collected within 90 seconds of each report was labeled as frustration
  - 665 number of data labeled as frustration cases

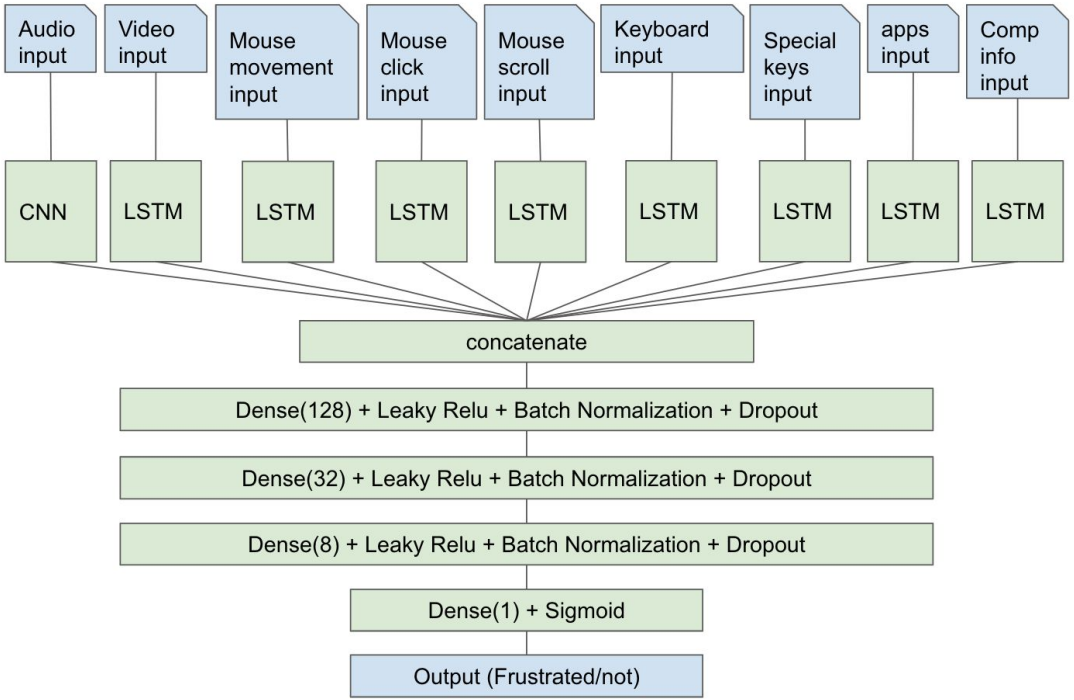
For more detailed data cleaning and preprocessing please refer to the write-up

[4] How long emotions last?

Human Frustration Prediction Model

- Multi-modal Input Features:
  - User-computer interactions, User audio + video, Apps' and Comp's statuses
- Ground truth values (labels):
  - User feedback
- A Supervised model using deep learning architecture

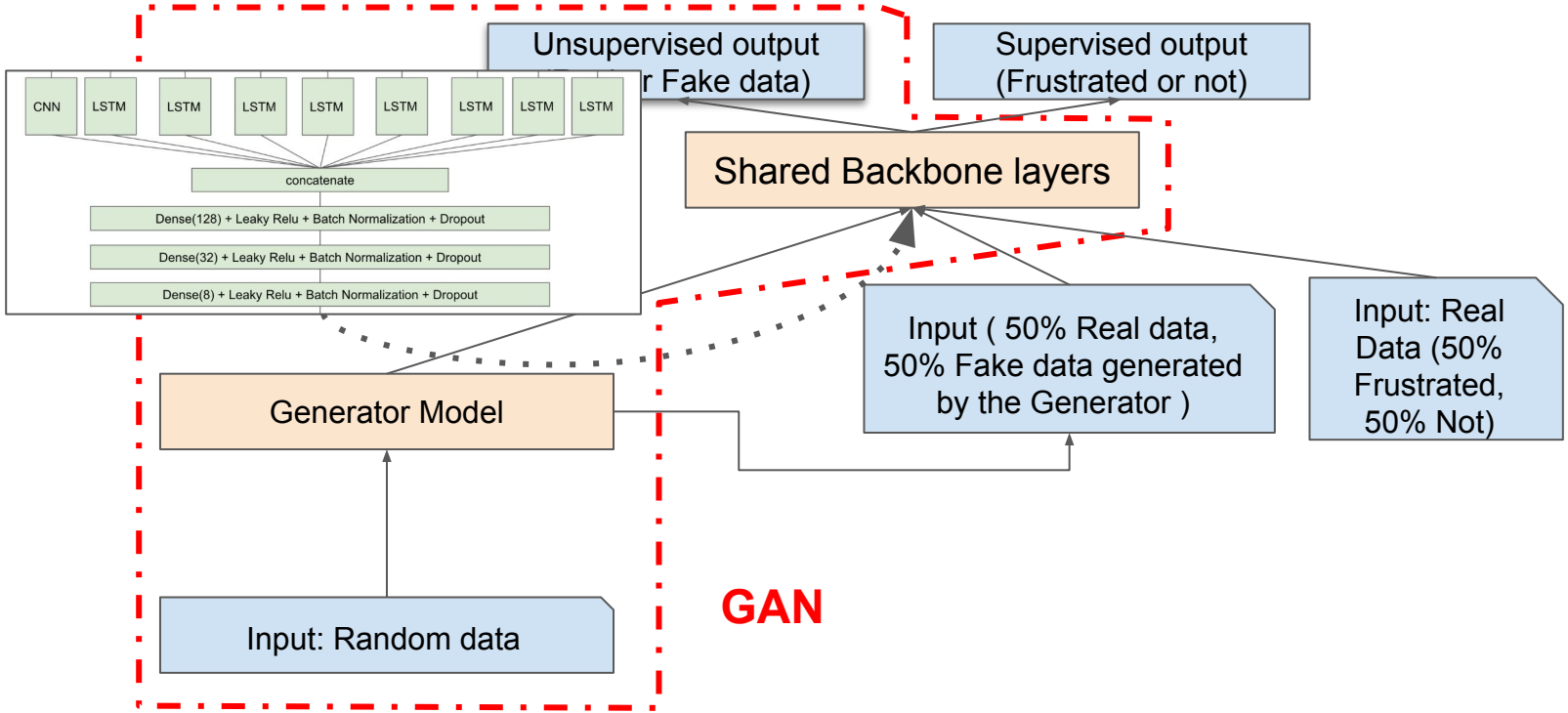
Human Frustration Prediction Supervised Model architecture one



Human Frustration Prediction Semi-Supervised Model using Generative Adversarial Network

- All the unlabeled data can not be labeled as satisfaction cases because
  - Users forget to report
  - Users become tired of reporting for the same reason multiple times.
  - Users adjust their usage and their expectations to avoid dealing with the difficulties
- Limited number of labeled data: 665
- While a lot of data are not labeled: 121912
- How we can use the information existing in a big portion of our data that are unlabeled?
- Semi-supervised using Generative adversarial networks (GAN)

Human Frustration Prediction Semi-Supervised Model Based on Architecture one

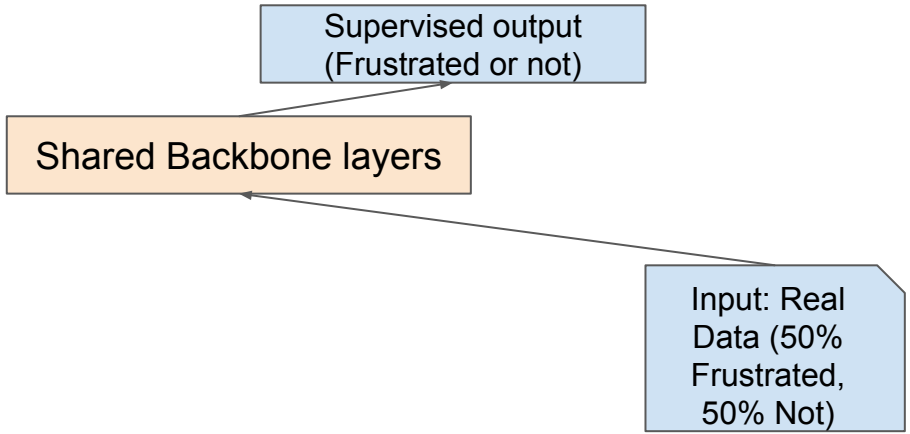


Human Frustration Prediction Semi-Supervised Model Based on Architecture one: Training cycles

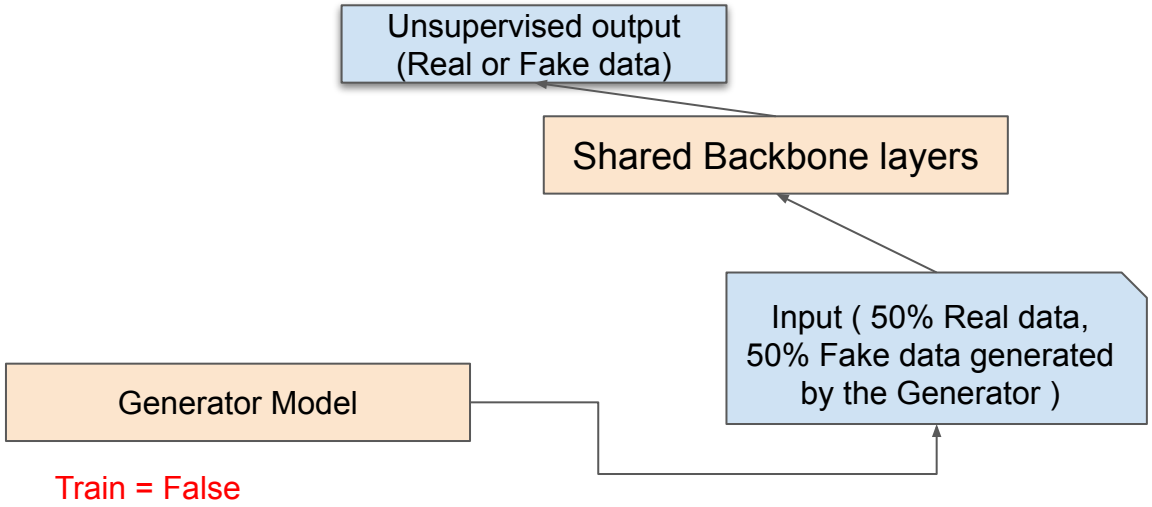
Each cycle consists of three mini-epochs training different models in the semi-supervised model architecture.

Total number of cycles = epoch size / mini-epoch size

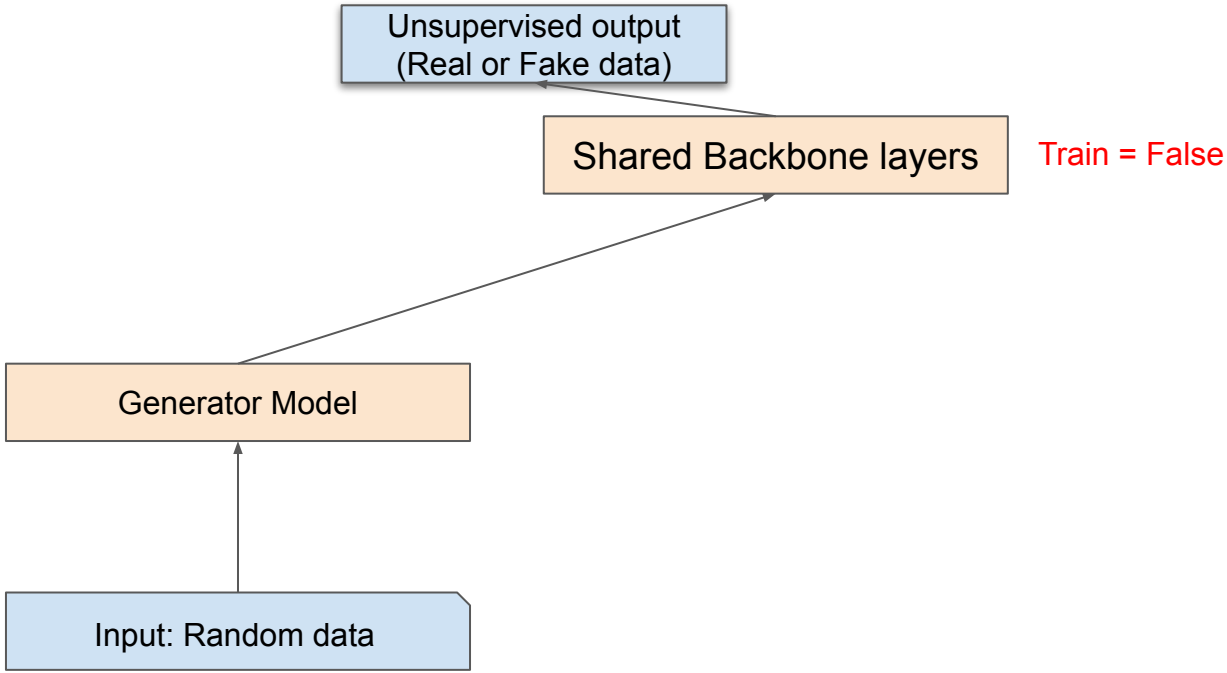
First Mini-Epoch in The Training Cycle



Second Mini-Epoch in The Training Cycle



Third Mini-Epoch in The Training Cycle



Purpose of the Semi-supervised Model Using Generative Adversarial Network Architecture

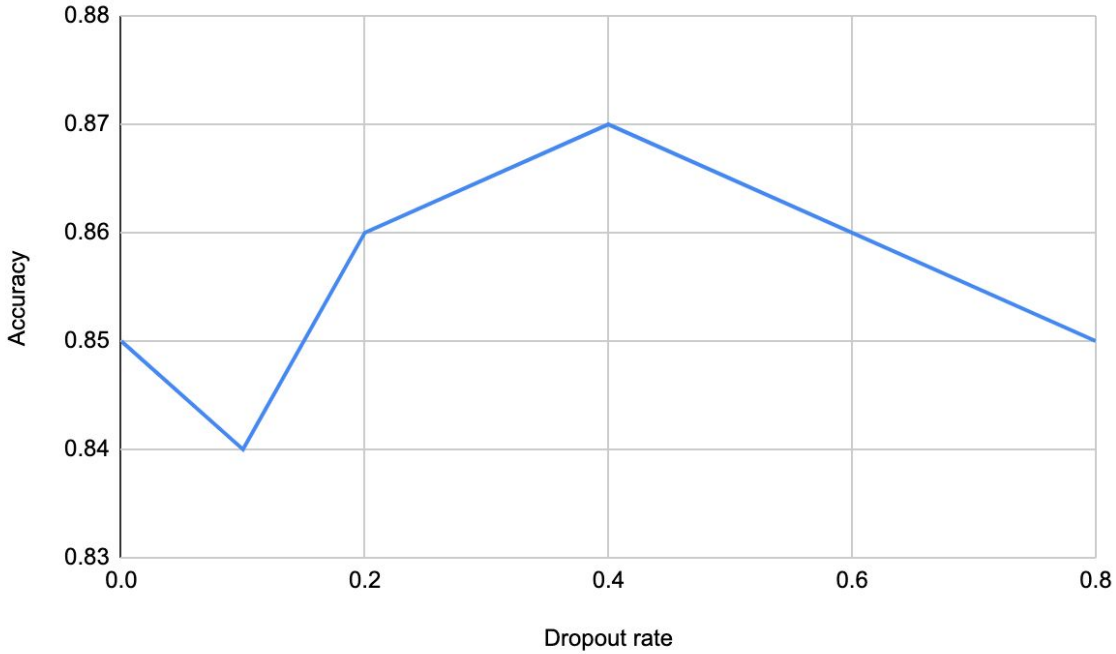
1. The Generator and unsupervised model strengthen each other to learn the hidden statistic patterns in the large portion of the unlabeled data
2. This knowledge will be shared with the supervised model through shared backbone layers
3. Thus supervised model can predict the the frustration using only a few labeled data

Evaluation

- Epoch size: 256
- Mini epoch size : 64
- Batch size: 128
- Supervised model input data count: 1330
- Unsupervised model input data count: 122577
- Supervised model:
  - 80% training data and 20% test/validation data.
  - skit-learn *train_test_split* function with the stratify parameter set according to the labels -> Unbiased Test Results

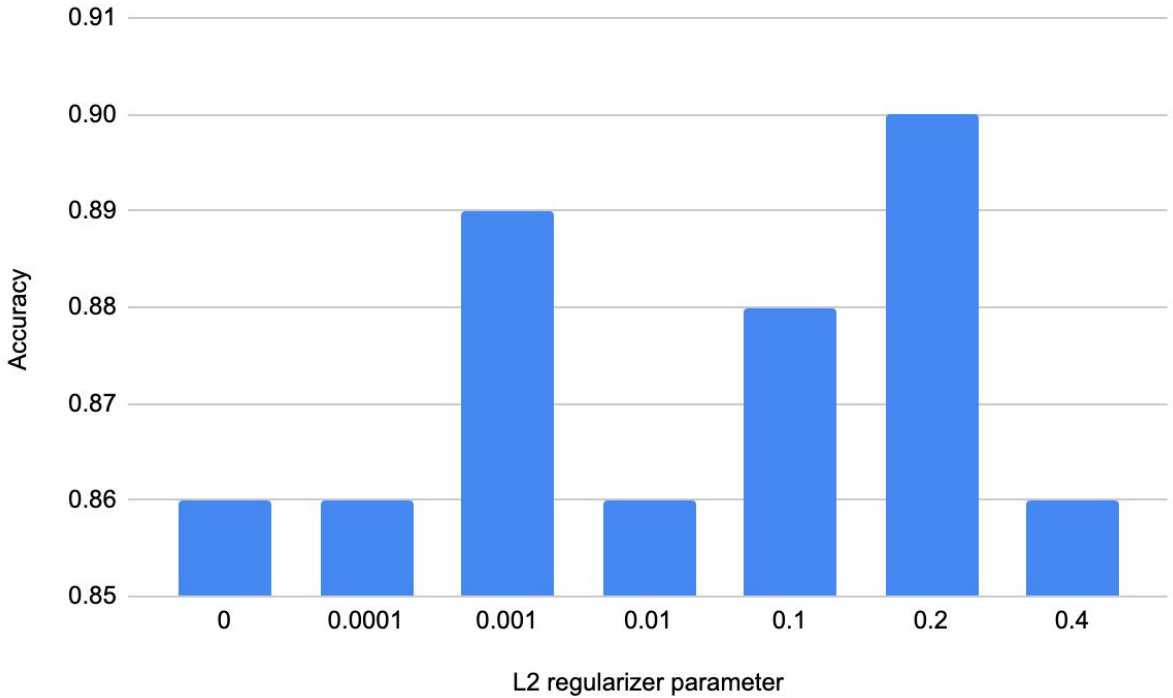
Parameter Search

- Human Frustration Prediction Semi-Supervised Model Based on Architecture one
- L2 Regularizer parameter = 0.01



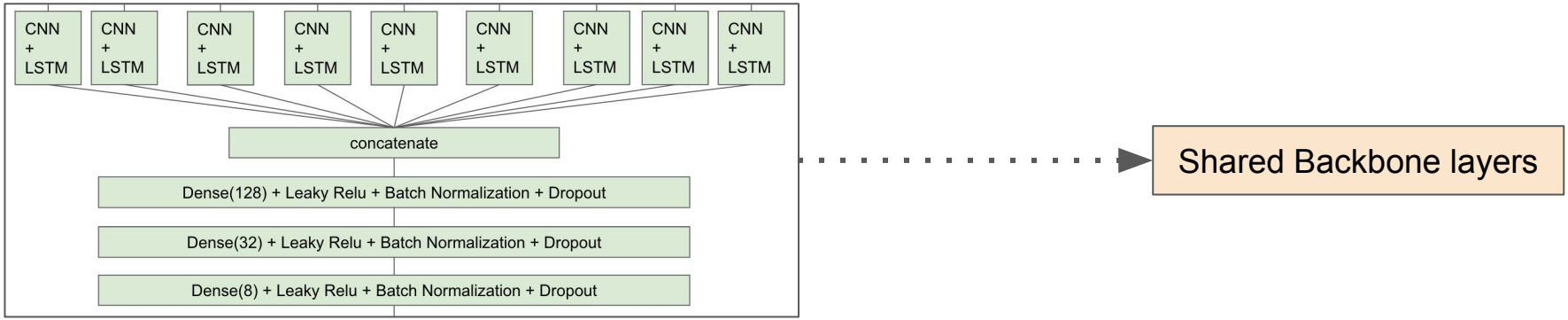
Parameter search

- Human Frustration Prediction Semi-Supervised Model Based on Architecture one
- Dropout rate = 0.4



Human Frustration Prediction Semi-Supervised Model Based on Architecture two

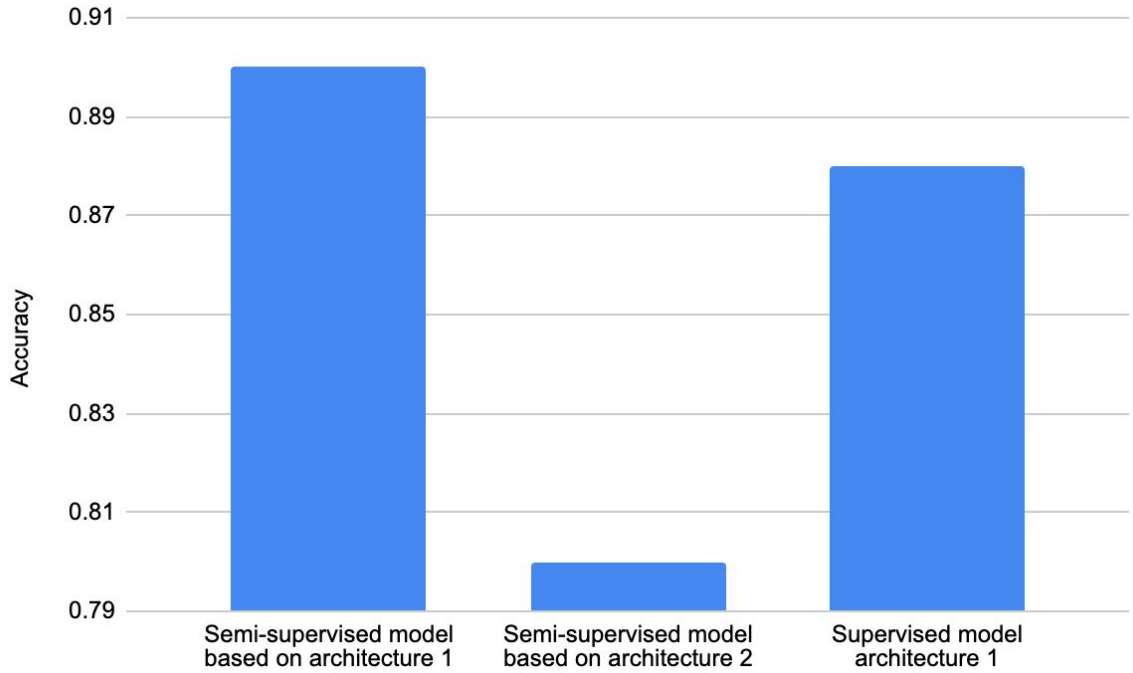
- Used CNN+LSTM inspired by (Hatice Gunes 2010, Surbhi Madan 2021)



Models' Comparison

Semi-supervised model based on architecture 1 with 90% accuracy on test data

- Supervised model architecture 1 vs. Semi-supervised model based on architecture 1
- Semi-supervised model based on architecture 1 vs. Semi-supervised model based on architecture 2



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.

- 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**

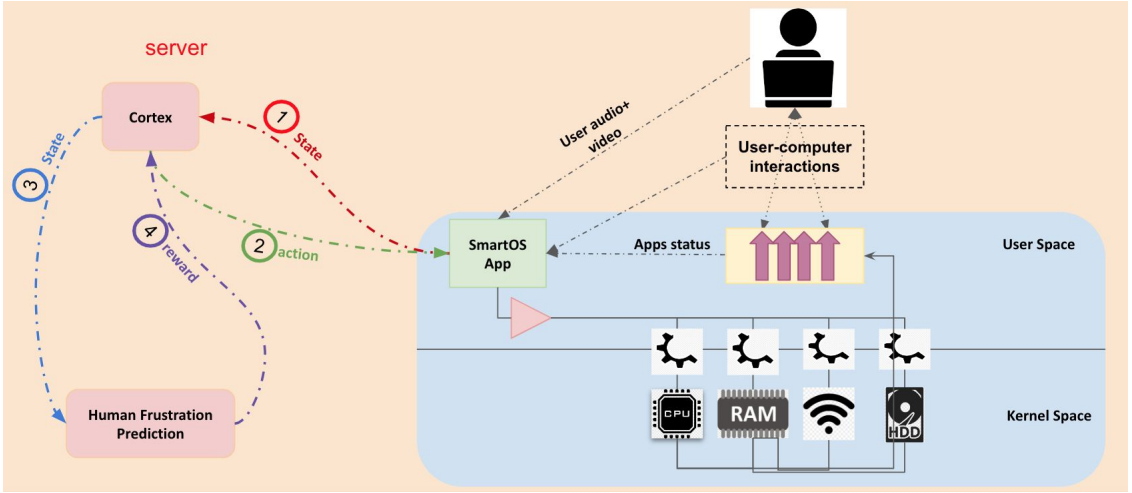
We need to converge faster than our prototype to not add to the user frustration with their computer!

How to converge faster?

- Pre-train smartOS algorithm

Pre-training Methodology:

1. Run the smartOS app on my computer
2. Instead of manual feedback -> let the human frustration model give the Cortex the prediction of my frustration with my computer based on the state as my feedback.



- Nothing should be provided manually
- Speeds the pre-training process
- SmartOS sees more everyday scenarios
- Generalises better in real everyday scenarios

Reinforcement Learning Model Architecture

Continuous State

- User-computer interactions:
  - The mouse cursor's x/y positions, the mouse clicks' x/y positions, the mouse scrolls' x/y positions, and the scrolls' amounts in the x/y coordinate along with timestamps at a millisecond precision rates
  - All the keys clicks and special keys clicks (such as ESC, delete, enter, space, etc.) with timestamps at a millisecond precision rates
- User audio:
  - MFCCS
- User video:
  - Extracted head positions from recorded video frames using CNN face detection models running on the user computer
- Apps' and Comp's statuses:
  - System-wide information such as computation, memory usage, network bandwidth, and input/output bandwidth of the running applications in the computer and the computer
- User Feedback:
  - User reported their frustration with the system through the application

Discrete Action:

- Using same resource allocation interfaces as before

CPU	Memory	Network	Disk I/O
0/1	0/1	0/1	0/1

Discrete Reward

Pre-training:

- 1 - predicted value of the human frustration prediction model based on the next state value

Testing:

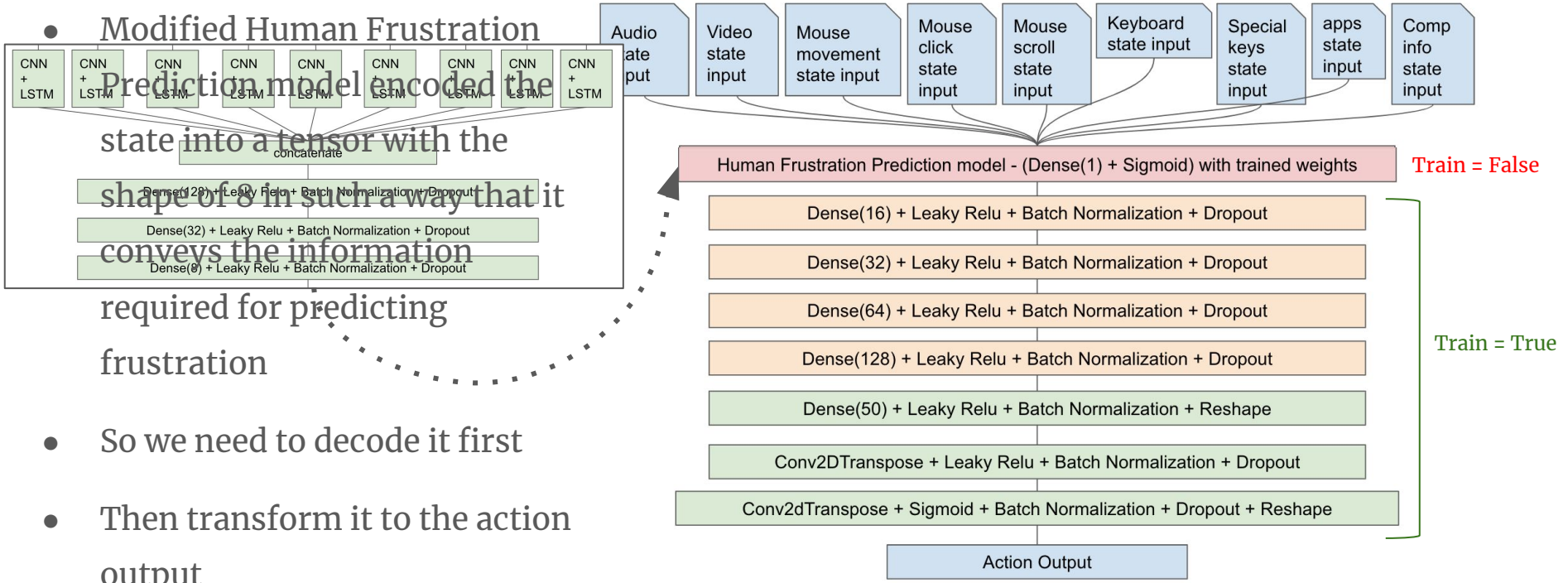
- If the action was in the set of the best actions that resolved the user's frustration -> one
- Otherwise -> zero

A2C (Volodymyr Mnih 2016) Model Architecture

- Actor model
  - State -> Action resulting in the maximum reward based on A2C current knowledge
- Critic model
  - State -> Value of the state based on the insights A2C has gained so far
- State inputs in the actor and critic models = Input in the human frustration prediction machine learning model
- Output of the model was predicting human frustration and reducing human frustration is the final goal of the RL algorithm
- Transfer Learning (Stevo Bozinovski 1976)

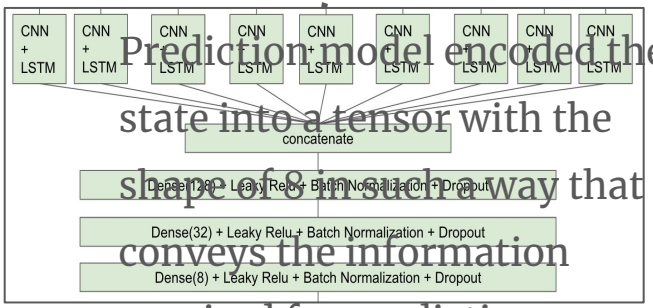
A2C Actor Model Architecture Using Transfer Learning

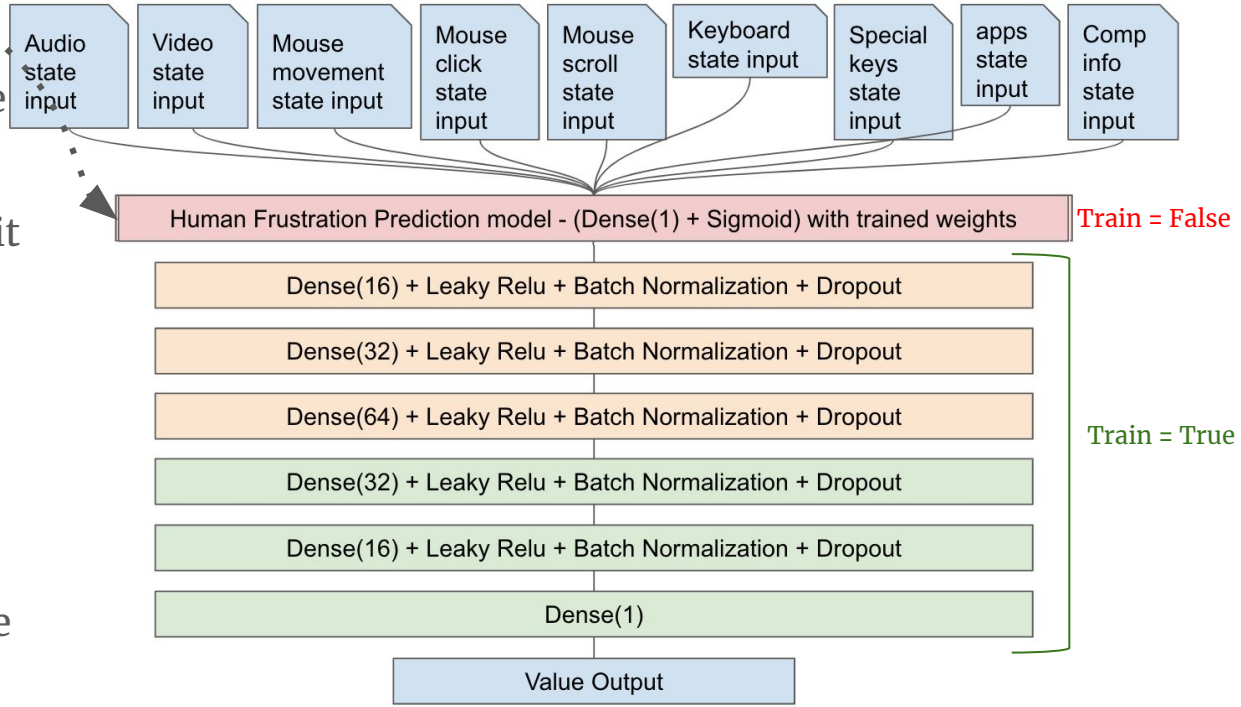
- Modified Human Frustration Prediction model encoded the state into a tensor with the shape of 8 in such a way that it conveys the information required for predicting frustration
- So we need to decode it first
- Then transform it to the action output



A2C Critic Model Architecture Using Transfer Learning

- 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 we need to decode it first
- Then transform it to the value output



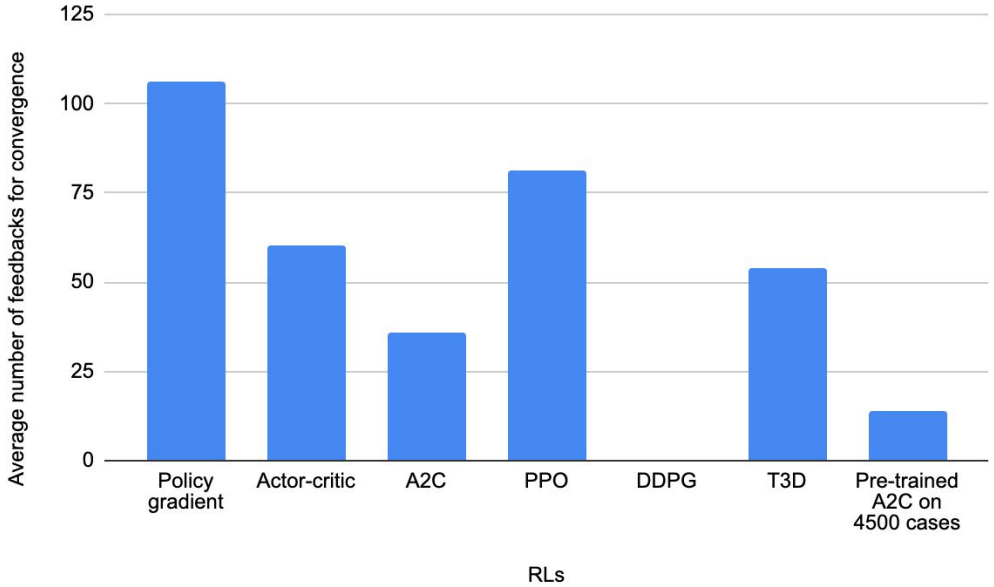


Evaluation

- All the RLs have the same design patterns as A2C -> transfer learning
- Comparison based on the frustration reports in the human study
- Extracted the important app based on the user’s report’s reason.
- Using the computer's state at report’s time -> produced a set of best actions resolving the frustration of the user.
- For convergence, the RL should’ve find an action member of the best set of actions

Evaluation (Cont.)

- 109 reports
- Average convergence between 10 trials for each report
- Evaluation metric for the RL algorithm = Average of all the reports' average convergence times
- Pre-trained A2C after 4500 updates on my computer
- pre-training of the A2C improves convergence by 60.83\%



Conclusion

- Identified the one-size-fits-all problem of operating systems in allocating resources to applications based on user preferences
- Defined the smartOS architecture to fill this need
- Presented a prototype implementation of the smartOS
- Completed a performance analysis of the SmartOS prototype vs. other heuristics
- Demonstrated the Convergence time and overhead of the smartOS prototype implementation
- Developed the SmartOS application for human study and Completed an IRB-approved human study
- Collected 122,577 real-world scenarios from 15 users, with 665 scenarios labeled as frustration

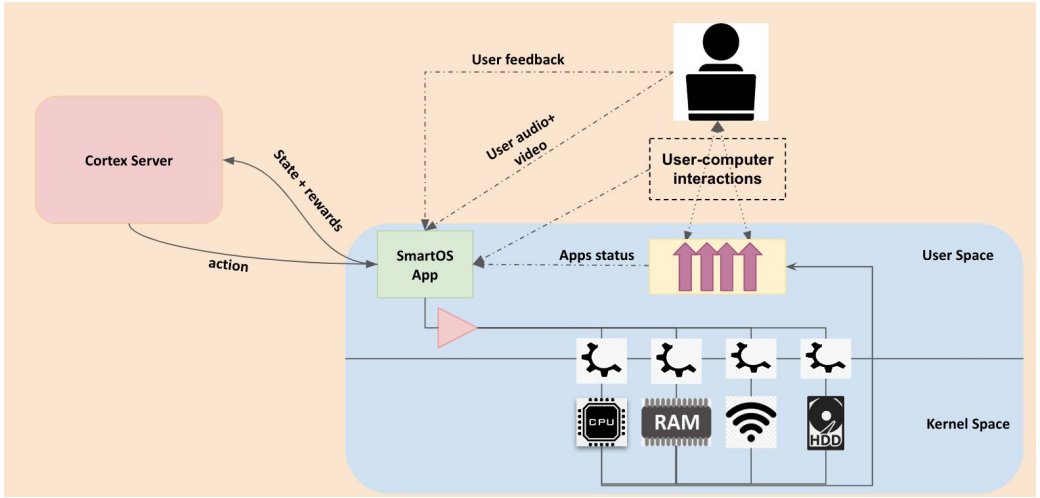
Conclusion

- Developed and trained a semi-supervised human frustration prediction model using GAN
- Compared the semi-supervised human frustration prediction model with other ML models
- Implemented different RLs as the cortex of the SmartOS app to work with complicated everyday scenarios with continuous states
- Compared different RL models' convergence times based on the real-world collected data in the human study
- Pre-trained the RL on my computer, feeding it with reward from the developed human frustration prediction model
- Evaluated the pre-trained RL vs untrained RL convergence times

Future Directions

Close the Loop with an Active SmartOS App

- Active SmartOS App changes the resource allocation policy based on the received action from cortex
- Compare the amount of user feedback (report) with the study done in the previous section using passive SmartOS app



We still need 14.1 rewards to converge on real-world everyday scenarios

Two sources for rewards

1. User Feedback submitted manually
    a. User reports their frustration with the system through the application report button and report form
2. 1 - (The human frustration prediction model's predicted value based on the next state)

- 1 overwrites 2
- Not bothering user for getting manual feedbacks
- Less time for the same amount of rewards
- Can converge fast using average 14.1 rewards from the human frustration prediction model on everyday real-world scenarios

What if The SmartOS App Can Not Converge?

- Default back to Linux CFS

Thank you and Questions

Back Up Slides

Data cleaning and Preprocessing (Cont.)

- Mouse cursor movements
  - Divided mouse movement events into one-second intervals
  - Computed the number of events and the mean and standard deviation of x/y positions of the mouse cursor movement events in each interval
- Mouse clicks
  - Divided mouse click events into one-second intervals.
  - Computed 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

Data cleaning and Preprocessing (cont.)

- Mouse Scrolls
  - Divided mouse scroll events into one-second intervals.
  - Computed the mean and standard deviation of x/y positions and x/y amounts of the mouse scroll events in each interval
- Keyboard and Special Keys Clicks
  - The highest typing speed is a character per 0.1 seconds
  - Divided Keyboard and Special Keys clicks into one-tenth-of-second intervals separately
  - Counted the number of clicks in each interval.

Data cleaning and Preprocessing (cont.)

- Audio Data
  - Computed the mean on axis 0 of the MFCCS
  - Padded it with zeros so that the feature’s length is 864
- Video Data
  - Padded the vector of the top, bottom, left, and right positions of the user head with zeros so that the feature's length is 32

Data cleaning and Preprocessing (cont.)

- Application Data:

  Pre-processed a vector of the top 10 applications in CPU usage, Memory usage, Disk usage and Network usage

- Process ID
- User: root one otherwise zero
- Nice value
- Virtual, residential and shared memory in bytes
- State: The index of the state in the following list - 1
  - Uninterruptible sleep, Idle, Running, Sleeping, Stopped by the job control signal, Stopped by the debugger during trace, Zombie
- CPU and memory usage percentages
- Application command index in the list were used for encoding the commands
- IO priority
- Disk read, write, swap in usages in bytes
- IO usage percentage
- Fg: foreground one otherwise zero
- Sent and received network data in bytes.

Data cleaning and Preprocessing (cont.)

- Computer Total Resource Usage profile:
  - 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 Data Sent and Received in Bytes