

# Bots-R-Us Final Report

Lockheed Martin Advanced Concepts Lab

COM S 402 Computer Science Senior Project

|                  |  |
|------------------|--|
| Adam Riffel      | <a href="mailto:ajriffel@iastate.edu">ajriffel@iastate.edu</a> |
| Carlos Acuna     | <a href="mailto:cacuna@iastate.edu">cacuna@iastate.edu</a>     |
| Corbin Graham    | <a href="mailto:cgraham1@iastate.edu">cgraham1@iastate.edu</a> |
| Jose Medina Mani | <a href="mailto:jomedman@iastate.edu">jomedman@iastate.edu</a> |
| Nhan Tran        | <a href="mailto:nhtran@iastate.edu">nhtran@iastate.edu</a>     |
| Simanta Mitra    | <a href="mailto:smitra@iastate.edu">smitra@iastate.edu</a>     |
| Rayhanul Islam   | <a href="mailto:rayhanul@iastate.edu">rayhanul@iastate.edu</a> |

|  |    |
|--|----|
| Client Letter.....                                 | 6  |
| Requirements .....                                 | 7  |
| 1. Introduction .....                              | 7  |
| 1.1 Purpose .....                                  | 7  |
| 1.2 Scope .....                                    | 7  |
| 1.3 Definitions, Acronyms, and Abbreviations ..... | 7  |
| 1.4 References.....                                | 8  |
| 2. Overall Description.....                        | 9  |
| 2.1 Product Perspective .....                      | 9  |
| 2.1.1 Concept of Operations.....                   | 9  |
| 2.1.2 Major User Interface .....                   | 10 |
| 2.1.2.1 Example Screenshot and Description.....    | 10 |
| 2.1.3 Hardware Interfaces .....                    | 10 |
| 2.2 Product Functions .....                        | 10 |
| 2.2.1 UC-0.....                                    | 11 |
| 2.2.2 UC-1.....                                    | 11 |
| 2.2.3 UC-2.....                                    | 11 |
| 3. Specific Requirements.....                      | 12 |
| 3.1 Features .....                                 | 12 |
| 3.1.1 Feature 1 .....                              | 12 |
| 3.1.2 Feature 2 .....                              | 12 |
| 3.1.3 Feature 3 .....                              | 12 |
| 3.1.4 Feature 4 .....                              | 12 |
| 3.2 Performance Requirements.....                  | 12 |
| 3.3 Design Constraints .....                       | 12 |
| 3.4 Software System Attributes .....               | 12 |
| 3.4.1 Reliability .....                            | 12 |
| 3.4.2 Availability .....                           | 12 |
| 3.4.3 Security .....                               | 13 |

|  |    |
|--|----|
| 3.4.4 Maintainability.....   | 13 |
| 3.4.5 Portability .....  | 13 |
| 3.5 Other Requirements .....                                       | 13 |
| Design.....  | 14 |
| Executive Summary.....   | 14 |
| Development Standards & Practices Used .....                       | 14 |
| Summary of Requirements .....                                      | 14 |
| Applicable Courses from Iowa State University Curriculum.....      | 14 |
| New Skills/Knowledge acquired that was not taught in courses ..... | 14 |
| Introduction.....  | 14 |
| Acknowledgement.....   | 14 |
| Problem and Project Statement .....                                | 14 |
| Requirements.....  | 15 |
| Design Assumptions and Limitations.....                            | 15 |
| Architectural Diagram .....  | 16 |
| Components .....   | 16 |
| 1. Memory Scraping Module.....                                     | 16 |
| 2. Data Processing Module .....                                    | 17 |
| 3. Prediction Algorithm Module .....                               | 17 |
| 3.1 Prediction Model .....   | 17 |
| 3.2 Model Specific Preprocessing .....                             | 17 |
| Super Mario Bros. ....   | 17 |
| Data Decomposition .....   | 17 |
| Design Rationale.....  | 18 |
| Design Issues .....  | 18 |
| 1. Bot Implementation.....   | 18 |
| Description .....  | 18 |
| Factors affecting Issue .....                                      | 18 |
| Alternatives and their pros and cons.....                          | 18 |

|   |    |
|---|----|
| Resolution of Issue .....                 | 18 |
| 2. Real-Time Data Handling .....          | 18 |
| Description .....                         | 18 |
| Factors affecting Issue .....             | 19 |
| Alternatives and their pros and cons..... | 19 |
| Resolution of Issue .....                 | 19 |
| 3. Model Training.....                    | 19 |
| Description .....                         | 19 |
| Factors affecting Issue .....             | 19 |
| Alternatives and their pros and cons..... | 19 |
| Resolution of Issue .....                 | 19 |
| 4. Data Cleaning .....                    | 20 |
| Description .....                         | 20 |
| Factors affecting Issue .....             | 20 |
| Alternatives and their pros and cons..... | 20 |
| Resolution of Issue .....                 | 20 |
| API.....                                  | 20 |
| 1. Memory API.....                        | 20 |
| 2. Data API .....                         | 20 |
| 3. Predict API .....                      | 21 |
| Work Done.....                            | 22 |
| Adam Riffel.....                          | 22 |
| Carlos Acuna .....                        | 22 |
| Corbin Graham .....                       | 22 |
| Jose Medina Mani.....                     | 22 |
| Nhan Tran.....                            | 22 |
| Results.....                              | 24 |
| LSTM Binary Classification.....           | 24 |
| LSTM Autoencoder .....                    | 24 |

Appendix ..... 26

# Client Letter

# Requirements

## 1. Introduction

### 1.1 Purpose

A user (either a player or game developer) can use the software for automated data gathering that is then used by a machine learning algorithm to determine the probability of whether current player is a bot or a real person.

### 1.2 Scope

This software is not intended to be used as anti-cheat software, it provides a probability that a player is a bot and afterwards the user may use this information in whatever way they deem appropriate.

Use Cases:

1. Cheat Detection.
  - a. Admin uses a program to extract information.
2. Training Bot Detection Model.
  - a. End User Trains the Model.
3. Game service changes its data. Program extracts data.

### 1.3 Definitions, Acronyms, and Abbreviations

| Term   | Description  |
|--------|--|
| Bot    | Autonomously controlled agent.   |
| Player | A human playing a game without third-party assistance.   |
| CNN    | Convolutional Neural Network; A deep learning technique that compresses (groups) pixels for faster and more memory-efficient predictions |
| DL     | Deep Learning; A well connected graph of nodes and activation functions that result in a prediction                                      |
| ATT    | Automated Turing Test; A test to determine if someone is a robot or human  |
| ML     | Machine Learning; Using data and neural networks to create predictions   |

|    |   |
|----|---|
| NN | Neural Network; A series of nodes and activation functions modeled after human brains |
|----|---|

## 1.4 References

*A Behavior Analysis-Based Game Bot Detection Approach Considering Various Play Styles*, [MMORPG, Algorithms], <https://arxiv.org/pdf/1509.02458.pdf>.

*Multimodal game bot detection using user behavioral characteristics*, [Pipeline, Results], <https://springerplus.springeropen.com/articles/10.1186/s40064-016-2122-8#:~:text=Server%2Dside%20detection%20methods%20are,method%20for%20detecting%20game%20bots>.

*Bot Detection in Online Games*, [Server-side, Client-side], <https://umm-csci.github.io/senior-seminar/seminars/fall2013/Lee.pdf>.

*How to Compare Machine Learning Models and Algorithms*, [Machine Learning, Compare Model], <https://neptune.ai/blog/how-to-compare-machine-learning-models-and-algorithms>.

*CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images*, [Classification, Machine Learning], <https://doi.org/10.1109/ACCESS.2024.3356122>.

*FakeSpotter: A Simple yet Robust Baseline for Spotting AI-Synthesized Fake Faces*, [Pattern Recognition, Machine Learning], <https://doi.org/10.48550/arXiv.1909.06122>.

*Deep learning and multivariate time series for cheat detection in video games*, [CNN, Multivariate Time Series, Aimbot] <https://link.springer.com/article/10.1007/s10994-021-06055-x>.

*Mario Reinforcement Learning Implementation*, [MARIO, DRL], <https://www.kaggle.com/code/ratthachat/aic502-mario-rl>.

*Classification of Humans and Bots in Two Typical Two-player Computer Games*, [Detection, Turing Test], <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8463277>.

*Turing Test Framework for Cooperative Games*, [Turing Test, Framework, Github], <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9893684>, <https://github.com/bic4907/Multiplayer-TuringTest>.

## 2. Overall Description

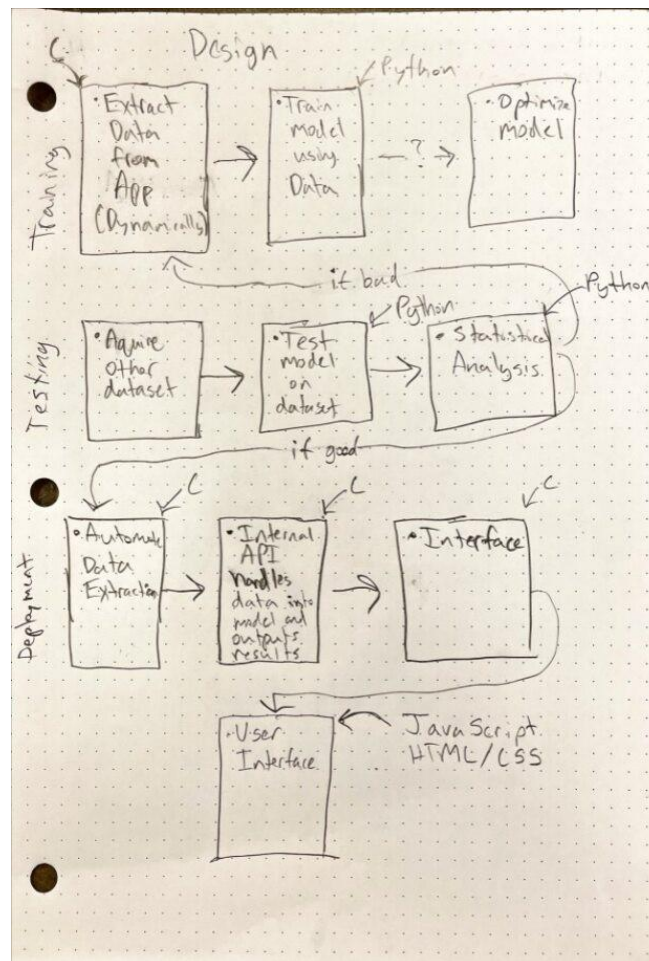
In video games players may download third-party software that hands control to a computer to play the game for them. This often creates problems in being able to determine whether a real person is playing the game or not. Our purpose is to create software that can distinguish between bots and players with a high degree of certainty.

### 2.1 Product Perspective

“reCaptcha,” a tool made by Google to determine whether a website user is a human or a bot. “Super Mario Bros,” a 2D platforming game that was released in the 1980’s.

#### 2.1.1 Concept of Operations

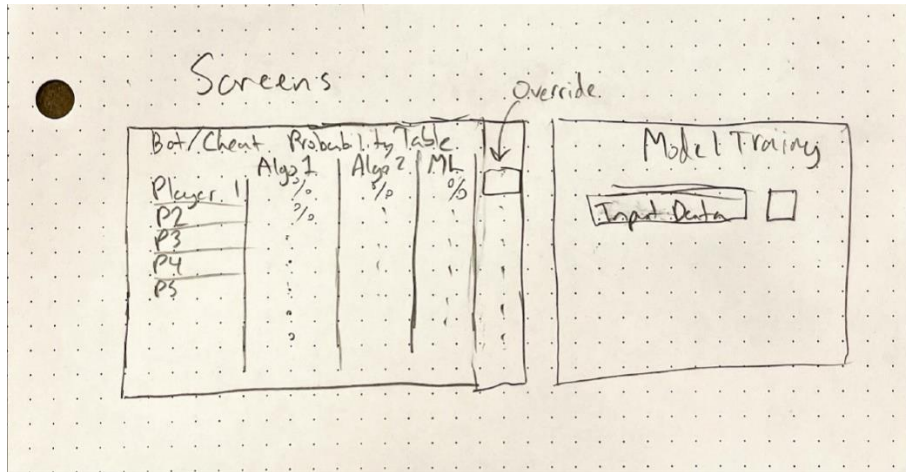
This project will be a web-based application that allows the user to import data for training a model to differentiate bots and players. Game administrators will have access to the model's prediction as the probability that a game agent is a bot. Game administrators can also override the model's prediction, which improves the model by providing an extra data point.



### 2.1.2 Major User Interface

Project is likely to have little to no UI as the intended purpose is to create an API.

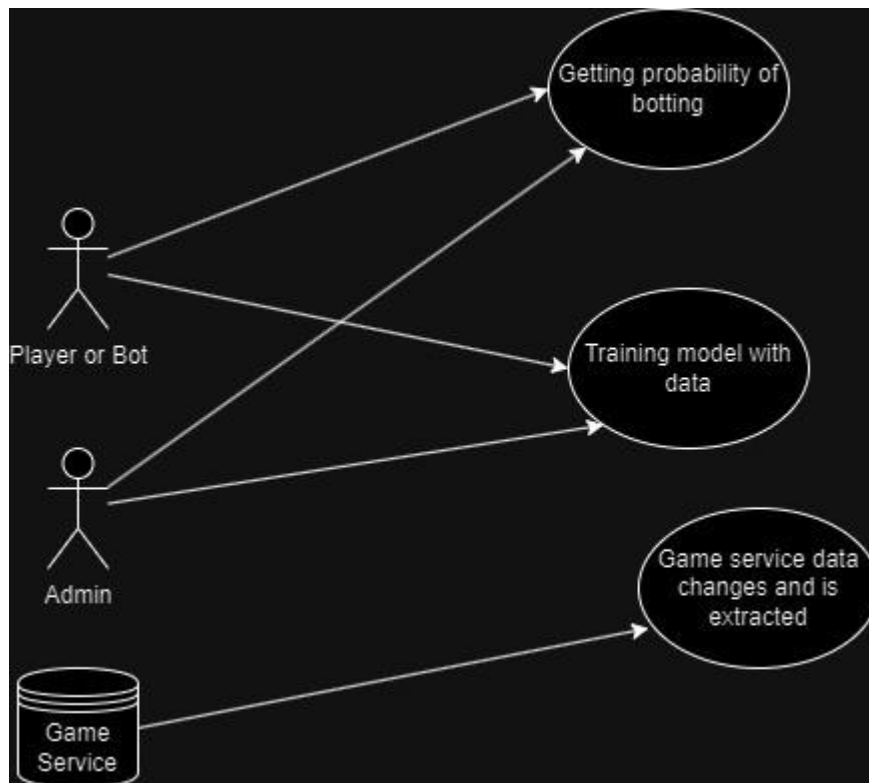
#### 2.1.2.1 Example Screenshot and Description



### 2.1.3 Hardware Interfaces

Any device capable of running Super Mario Bros, C, C++, and Python.

## 2.2 Product Functions



### 2.2.1 UC-0

“Game service data changes and is extracted” (Game Service)

Main Scenario:

1. Game service updates variables for the player’s state.
2. Software makes note of this change and logs it to be analyzed.
3. Data stored is used later to train the model or used to make a prediction.

### 2.2.2 UC-1

“Training model with data” (Player, bot, or admin)

Main Scenario:

1. User indicates to the software a program/source of data to be used for training.
2. Features that are likely predictors are then extracted from the program and split into a test and validation set.
3. Model trains on the test set and prevents overfitting with the validation set.
4. After a series of iterations bias values are linearly fitted to data.

### 2.2.3 UC-2

"Get probability of botting" (Player, bot, or admin)

Main Scenario:

1. Dataset is provided to the software by the user
2. Algorithm is used to analyze the dataset and determine probability of the data coming from a bot.
3. Prediction results are given to the user.

Extensions:

1. Speedrun Validation
  - a. Working under the assumption that the user has selected a model/algorithm that takes in visual/image data, a video could be used as a dataset for the software to analyze.
  - b. A moderator for a website like speedrun.com can use this probability output data to determine if a speedrun is done by a human or not and decide based on that information.
2. Server-side Bot Detection
  - a. An admin can install this software on the server side of the game where all the player’s inputs and actions are processed.

- b. The admin can then use the information provided to decide whether a player should be disciplined or not.

## 3. Specific Requirements

### 3.1 Features

#### 3.1.1 Feature 1

The program's main feature is to predict with high accuracy the likelihood of an actor being human or bot.

#### 3.1.2 Feature 2

A secondary feature is the compatibility with other games/processes in being able to accept input regardless of the game.

#### 3.1.3 Feature 3

Allow detection accuracy feedback from the client side to improve the bot detection algorithm.

#### 3.1.4 Feature 4

The framework is modular and able to swap custom plugins to change learning models, data processing, and memory scraping.

### 3.2 Performance Requirements

The program is required to provide Realtime predictions. This means that the program cannot take multiple cycles to provide a prediction.

### 3.3 Design Constraints

Design constraints will be limited to what provides the greatest prediction accuracy.

### 3.4 Software System Attributes

#### 3.4.1 Reliability

The program should produce consistent, accurate results for an undetermined number of features.

#### 3.4.2 Availability

The program should be operable on client or server-side. It should be fully self-contained and not dependent on third party sources. From this, it should be available 100% of the time.

### *3.4.3 Security*

Encryption of data should be used whenever and wherever possible. But there is no expectation that the data produced by the program is secure, since it is relying on non-secure sources.

### *3.4.4 Maintainability*

The program should be fully pre-trained, but available for finetuning to better suit a given task. Since it will be fully pre-trained, there are no maintenance expectations after launch.

### *3.4.5 Portability*

The program should rely only on open-source frameworks and libraries such that they will continue to be supported on any given system for a limited indefinite time.

## **3.5 Other Requirements**

Requirements will vary depending on the solution platform. Initially, requirements will be limited to those provided above.

# Design

## Executive Summary

### Development Standards & Practices Used

1. Version Control via Git.
2. Agile.

### Summary of Requirements

1. Develop bot detection algorithms.
2. Interface to receive game data and analyze the data for prediction.

### Applicable Courses from Iowa State University Curriculum

1. COMS 327: Advanced Programming Techniques
2. COMS 352: Introduction to Operating Systems
3. COMS 472: Principles of Artificial Intelligence
4. COMS 474: Introduction to Machine Learning

### New Skills/Knowledge acquired that was not taught in courses

1. Linux system process ID handling.
2. Machine learning model training and testing.
  - a. Learning how to use sklearn and pytorch.
3. Data transformation.
  - a. Dimension reduction.
  - b. Principal component analysis

## Introduction

### Acknowledgement

Technical expertise and funding received from employees of Lockheed Martin Advanced Concepts Laboratory regarding machine learning models and practices.

### Problem and Project Statement

“Develop methods and proof of concept for automatically distinguishing bots vs. humans with as much specificity as possible, all the way to identify a specific human if possible.”  
Dan Waitman. Sub-Problem: Game-based automated Turing test.

## Requirements

The program's main feature is to predict with high accuracy the likelihood of an actor being human or bot.

Allow detection accuracy feedback from the client side to improve the bot detection algorithm.

## Design Assumptions and Limitations

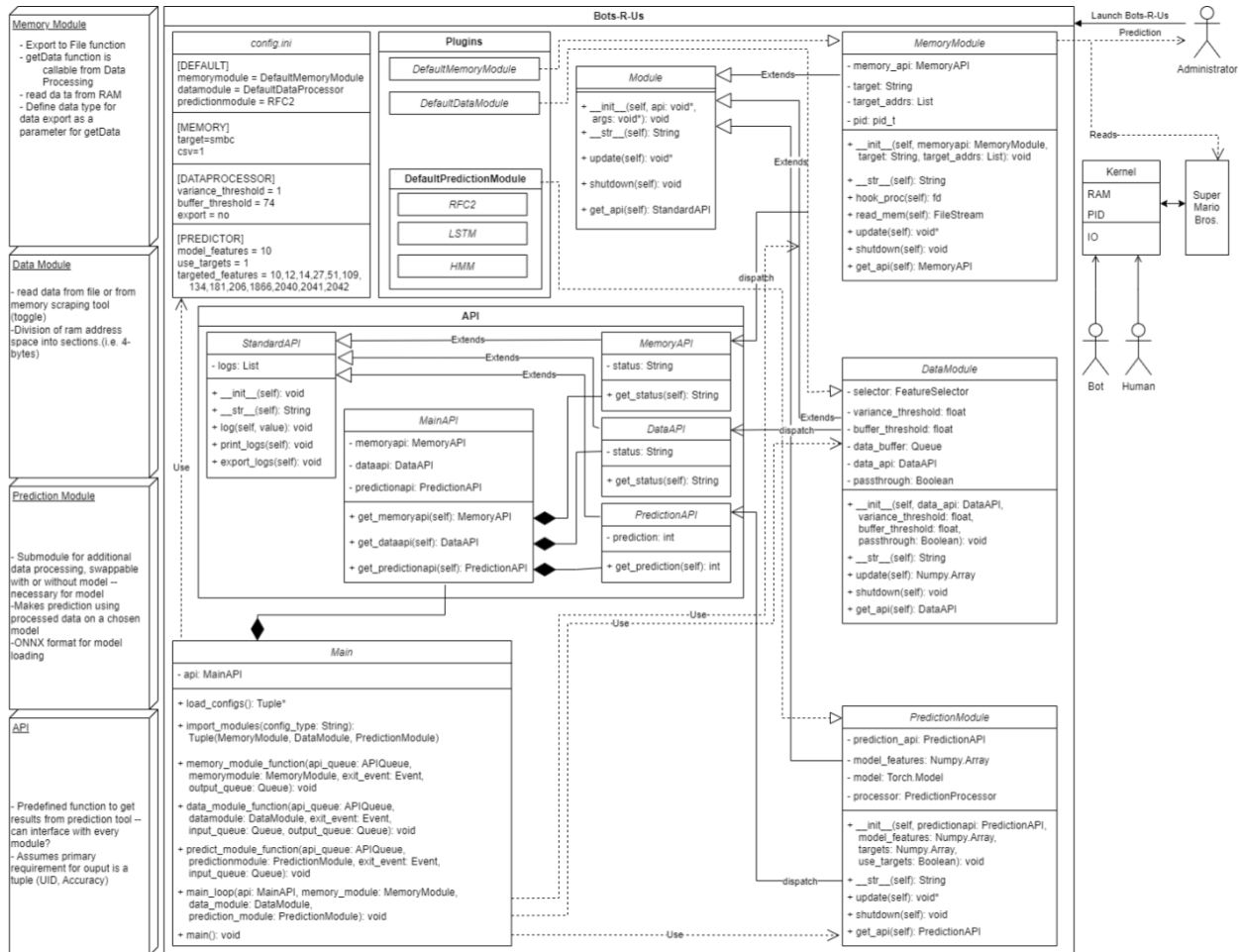
Assumptions:

1. Sufficient storage for bot detection program and its dependencies.
2. Requires a Linux operating system to run the Super Mario Bros. Emulator.

Limitations:

1. Work hours available to work train and refine the models.
2. Funding for more computing power in training the models.

# Architectural Diagram



## Components

This framework would host our three primary components in a nondependent style that allows modification and adaptation as the online environment changes. The three primary components will be demonstrated by three modules: Memory Scraping Module, Data Processing Module, and Prediction Algorithm Module.

### 1. Memory Scraping Module

This module takes input from the command line that is piped into our framework. Through stdin it reads rows of data that are formatted correctly to be passed on to the data processor. Additionally, once it recognizes that there is no more data to be read (the game has ended) it will signal to the rest of the program to begin shutting down.

## 2. Data Processing Module

Receives unprocessed input from the Memory Module and performs general processing to create a standardized output for the Prediction module. Additionally performs feature selection for live training or acts as a passthrough for pre-trained models with targeted features.

## 3. Prediction Algorithm Module

This module takes processed data from the data processing module and makes a prediction based on the model. Posts predictions to the API.

### *3.1 Prediction Model*

Each prediction module comes with a corresponding model that has been previously trained. This allows anyone with a pre-trained model to easily plug it in to the framework with little hassle.

### *3.2 Model Specific Preprocessing*

Some models require further data processing than what is done in the data processor. The user can create a sub module for preprocessing data before it is given to the prediction model.

## Super Mario Bros.

The game is a conversion of a disassembled assembly version of Super Mario Bros. to C++. This project can be seen at <https://github.com/MitchellSternke/SuperMarioBros-C>. C++ is modified to output targeted RAM addresses that described the state of the game. This includes things like the in-game timer, whether a certain button is currently held, whether Mario is in the air, whether Mario is falling, whether Mario has gone in a pipe, Mario's x,y position, and many more. To easily gather samples, the program implementing this module takes an argument for the current user playing and then pipes into a file that is in a "logs" folder.

There is also a Java version of the game used <https://github.com/amidos2006/Mario-AI-Framework/> that we modified to output similar data that can be played by a human or bot to get data that our models were trained on.

## Data Decomposition

The model being used needs to be stored after training on a set of data. The model is to be saved by ONNX which stores the model as an object that can be imported to run a specific model. The dataset of csv files does not need to be stored persistently after the model is trained.

# Design Rationale

## Design Issues

1. Bot Implementation
2. Real-time data handling (Memory race condition)
3. Model training (Transformer)
4. Data cleaning (Too much data)

### 1. Bot Implementation

#### *Description*

Due to running on different game engines, the IEEE bots do not interact with our data extraction framework outside of the box. We need to build a system that allows the IEEE bots to send their inputs to the game we have modified.

#### *Factors affecting Issue*

1. There doesn't exist publicly available data of bots playing the game, meaning we must produce it ourselves.
2. Several bots exist online for the game, but they run on their own client/emulator of the game that is different from ours.
3. IEEE submissions contain several different implementations of the bots within the same client/emulator.

#### *Alternatives and their pros and cons*

1. We could write the bots ourselves; this would take far too long, and it would raise questions over the rigorousness of our research due to a conflict of interest (i.e., we want them to be differentiable from humans.)
2. We could switch to the IEEE client entirely; this would mean that we would have to reimplement our data collection pipeline for human input and would have to re-produce our datasets from scratch.

#### *Resolution of Issue*

We are to write a custom application which communicates input from the IEEE client into the decompiled emulator client.

### 2. Real-Time Data Handling

#### *Description*

When we begin implementing this portion of our program, we foresee an issue arising with how to send newly created data to the models. The game will be writing the data to a CSV file, and we want to periodically send this data to the ML models we have trained so that

the end user can receive real-time predictions. Since we are dealing with shared memory, we need to make sure that the writing of data is not affected by the reading of the data.

#### *Factors affecting Issue*

Our client wants an online bot detection program that can provide a real-time estimate of an agent being human or not. To that end we must implement a streaming feature extraction that feeds into a selection of different models.

#### *Alternatives and their pros and cons*

Because this issue is related to a feature our client explicitly asks for, there is little to say for alternatives, since it can only be achieved in one way.

#### *Resolution of Issue*

There exist libraries that can communicate between the decompiled emulation running in C++ and the machine learning models running in Python. They also take care of doing it live.

### 3. Model Training

#### *Description*

Some of our models take large amounts of time to complete their training. This is most evident with the deep learning models like transformers. Even after reducing the data, the model can take upwards of 5 hours to complete its training on just a single person's samples.

#### *Factors affecting Issue*

Deep learning models are more complex than traditional learning models. That is, in general, they will be more computationally expensive than traditional learning models given the same data set to train on.

#### *Alternatives and their pros and cons*

Discontinue the use of deep learning models, or any model that takes too long to train; this would mean that we would limit ourselves to simpler models, which may or may not be as robust as more complicated solutions. This may lead to more "transparent" solutions to our problem, as deep learning models can lead to obscure solutions.

#### *Resolution of Issue*

Using the university provided HPC (High Performance Computing) clusters to train these more complicated models can reduce the time they need to train.

## 4. Data Cleaning

### *Description*

Due to the time series aspect of the data, a lot of data is created per attempt of Super Mario Bros. Level 1-1.

### *Factors affecting Issue*

We cast a broad net over the entire state of the emulator's RAM for our data; that is, every frame update, we record more than 2,000 features. This has resulted in a large body of raw data that can only be understood as a context-dependent sequence of discrete states representing a continuity.

### *Alternatives and their pros and cons*

1. We could have cast a smaller net over the state of the game to reduce the size of the data sets; we would then have had to make uninformed decisions on which features to include and exclude. Due to the nature of the problem, we did not know which features would strongly correlate human input against autonomous input, so we could have accidentally picked wrongly.
2. We could have quantized each row entry more discretely to reduce the size and complexity of the data; this would have also been an arbitrary decision that would impact how we could use and manipulate the data in the future. If the states were quantized too discretely, then we could lose information even if we still recorded the same 2,000 features.

### *Resolution of Issue*

While the raw data remains large and complex to use, we can use various methods of preprocessing to clean up its representation for use in our models.

## API

The API allows for any client to access the status and predictions from outside the framework while it is running. A client has the ability to read, export, and model data without modifying the framework.

### 1. Memory API

Maintains status of the memory module.

### 2. Data API

Maintains log of API calls.

### 3. Predict API

Maintains prediction object of the currently predicting model.

# Work Done

## Adam Riffel

Worked on memory module in framework, experiments on correlation matrices for features and learning about correlation analysis methods, later experiments in the memory scraper, research/learning on machine learning and machine learning models, image labeling or image detection, data collection, and work on logging functions.

## Carlos Acuna

Worked on the prediction module, early experiments in the memory scraper, fitting the bots to the framework, training of machine learning models, and documentation. Started with none/minimal knowledge on memory manipulation and machine learning models and code implementation. Learned about data processing techniques.

## Corbin Graham

In the first sprint, he researched how the Linux operating system manages program memory and how to use this to grab all the variables from a running game. This research later became an early prototype for the memory scraping module in the final program framework. In the second sprint, he worked on using data collected from the memory scraping research to training machine learning models that could naively predict between a bot and human, and between different humans. In the final sprint, he designed the framework, implemented the core structure, and aided in the development of modules and APIs.

## Jose Medina Mani

Early memory scraping experiments involving input handling; air-time and fall-time data logging from game; experiments with k-means classification and data quantization; data preprocessing research; data module design and implementation; default data plugin variance threshold implementation.

## Nhan Tran

Worked on the prediction modules. Investigated multiple different neural networks to for this problem. Implemented LSTM for binary classification and LSTM Autoencoder to solve this problem. Implemented data cleaning and processing for LSTM. Partially worked on the design of framework. Seldomly provide emotional support.



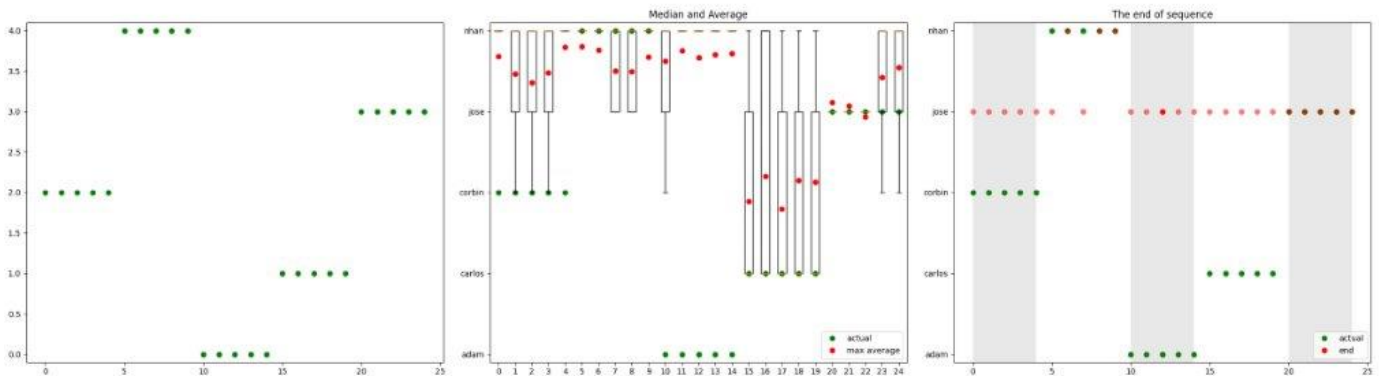
# Results

## LSTM Binary Classification

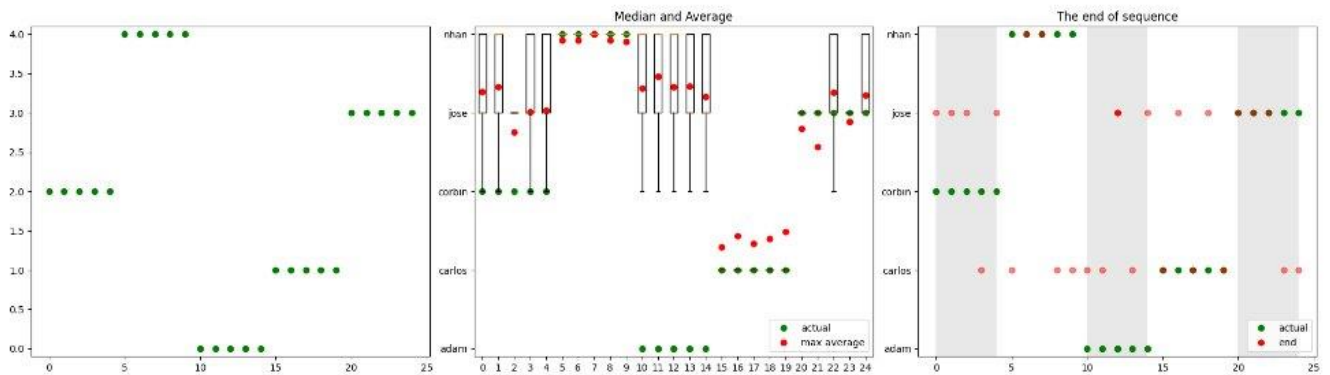
Supervised Learning Method

Result of classification between 5 people. Green is the target. Red is mean prediction. Box plot shows predictions over the gameplay.

- Full Sequence: Trained on the entire sequence of gameplay



- Sequence as Events: Reprocess gameplay sequence down to interesting events. Here, the interesting event indicates the change of inputs.



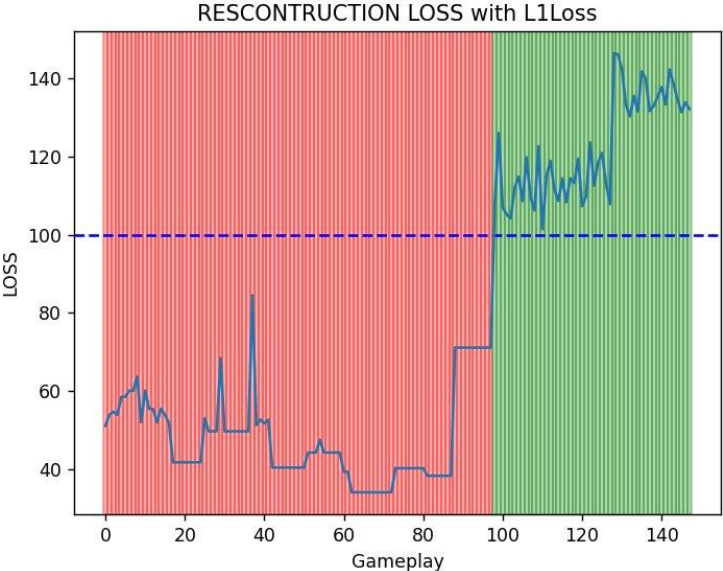
## LSTM Autoencoder

Unsupervised Learning Method

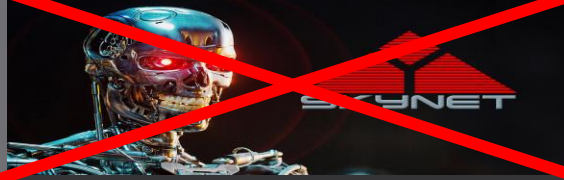
Result of reconstruction of bot gameplay sequences and human gameplay sequences. Red area indicates the bot. Red area indicates the bot sequence reconstruction loss.

Green area indicates the human sequence reconstruction loss. The loss threshold for binary classification between bots and humans is 100.

Trained on a window of sequence of length 50 over an entire gameplay.



# Appendix



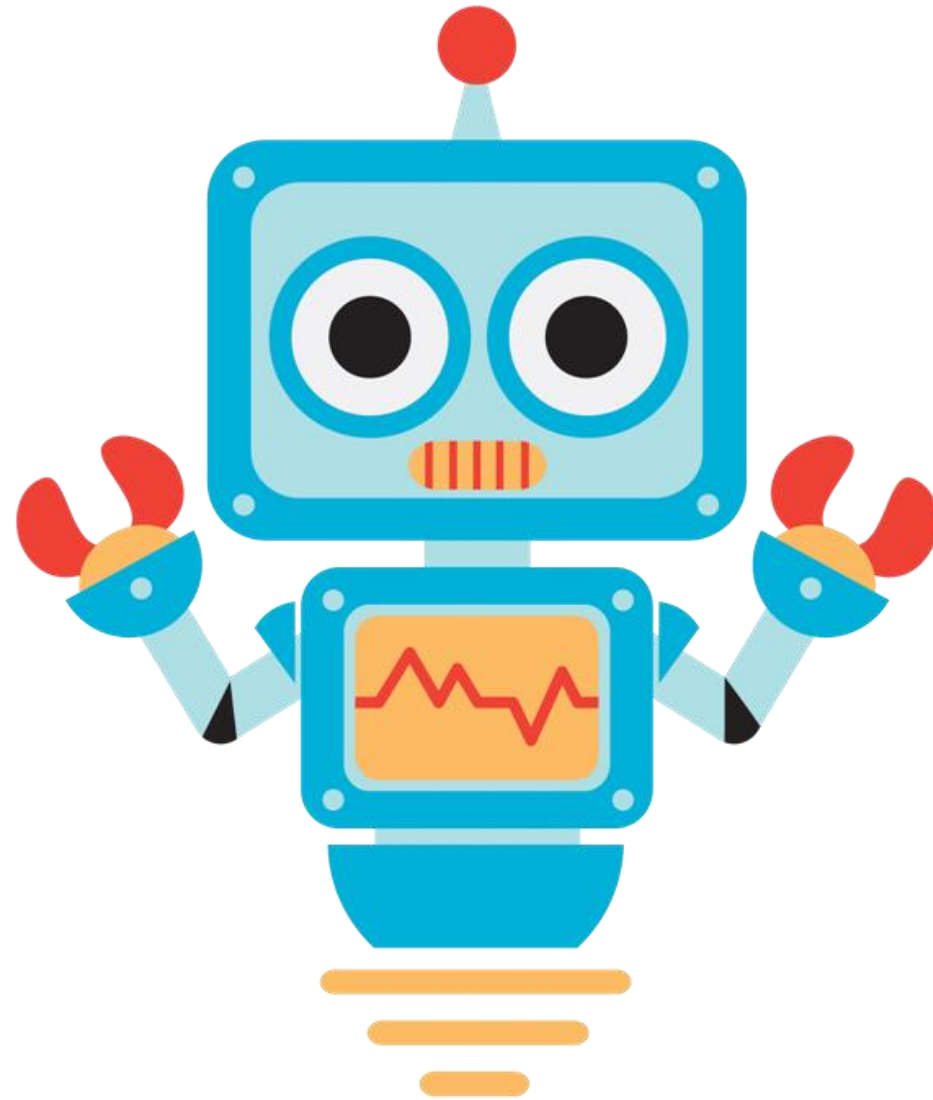
# BotDetect

Team 8

Carlos Acuna, Corbin Graham, Jose Medina Mani, Adam Riffel, Nhan Tran

# Problem

- Is it possible to differentiate between a human and bot playing a game?



# Problem

Challenges:

- Which technique?
- What game?
- Is it generalizable?

Issues:

- Limited data sets
- Limited research
- Limited data storage

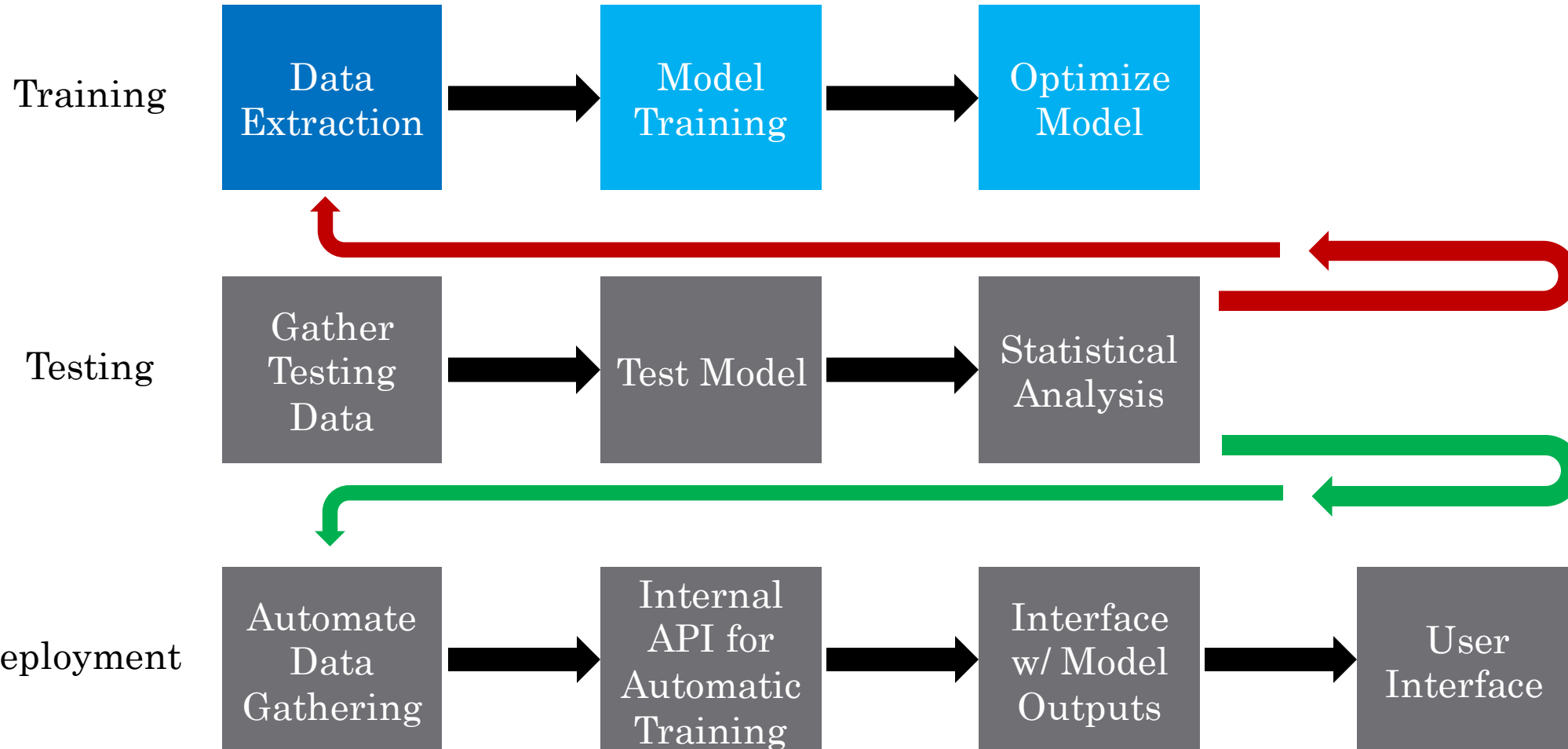
# Project Approach

Naïve  
approach, no  
source code  
modifications

Automated  
pipeline

API, analyzes  
and predicts

# Design Flow



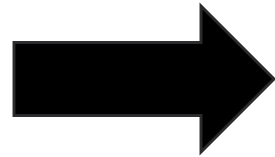
# Tools

- Machine Learning
  - labelImg
  - OpenCV
  - PyTorch
  - Convolutional Neural Networks
- Data Storage
  - No consistent centralized space for data storage.

# Tools

## Data Gathering

labelImg



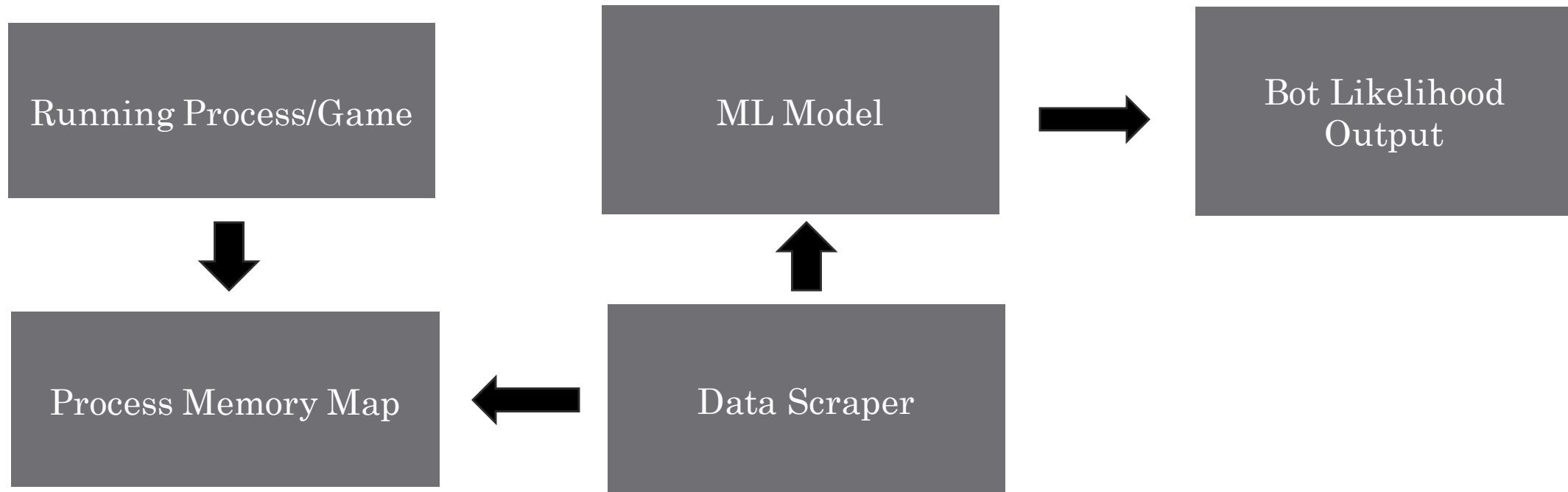
OpenCV

PyTorch



Raw Data:  
Position  
Score  
Level Timer  
Total Play  
Time  
etc.

# Memory Pipeline



### Process Memory Map

|                |    |
|----------------|----|
| 0x7ffeabb9c1a1 | 02 |
| 0x7ffeabb9c1a2 | 01 |
| 0x7ffeabb9c1a3 | 00 |
| .              |    |
| .              |    |
| .              |    |

### Data Scraper

#### Memory Hooking

- Search for particular starting value
- Narrow down addresses by providing updated value

#### Data Processing

- Log important data at specific time intervals

# Memory Pipeline Automation

- For Super Mario Bros.
  - Timer (Continuous Decreasing Value)
    - Assuming no prior knowledge:
      - Make guesses at other important memory addresses.
  - Player Input
    - Likely most important data value.
    - Have automation program send particular inputs before player has control.
      - No need to make guesses at other memory addresses as this is our target.
      - Intrusive

# Tools

- Memory Access/Processing
  - Built-in Python File Processing
  - psutil (Python Library)
  - Scanmem/GameConqueror



# Current progress status

- Memory Scraping Tools
  - Work on several tools to scrape and collect values from active memory
- Interface Scraping Models
  - Work on CNN models to scrape and collect values from active interface
- Basic Data Pipeline
  - Work on synthesizing both techniques to label and record variables automatically

# Remaining work timeline

- Section 1: Data Collection Pipeline
  - Building a robust data collection pipeline to give us access to any variable or feature that may be handy for prediction
- Section 2: Data Storage & Training Pipeline
  - Storing labelled data from bots and humans and presenting it for training
- Section 3: Algorithm Development
  - Training various models
  - Create streamlined algorithm for binary classification between human and bot
- Section 4: Fully Autonomous Pipeline & API Deployment
  - Create a fully automated pipeline with programmable interface

# Challenges – Data Scraping

- Needle in a haystack
  - Heap and stack are huge!
- Where can we infer?
  - What is required for plausible inference?
- Automate! Automate! Automate!

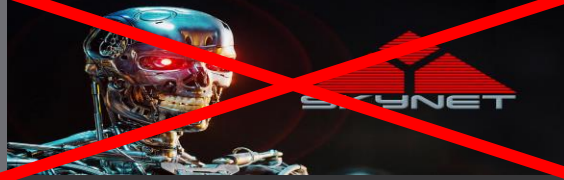
# Challenges – Automation

- Automated Data Gathering Pipelines
  - "By-hand-inference"
- Screen / Display Pipeline
  - Using CNN for location / value tracking
    - Fine-tuning model
    - Training times
    - High error rate
- Memory Scraping Pipeline
  - Heap / Stack Size
    - Lots of data
  - Data type inference

# Demo



```
Windows PowerShell
PS C:\Users\Laptop A\Documents\COLLEGE\ISU\ISU Homework\Spring 2024\Senior Project\project\lm_8\Experiments\image processing object detection\mariomodelsrc> python inference.py
```



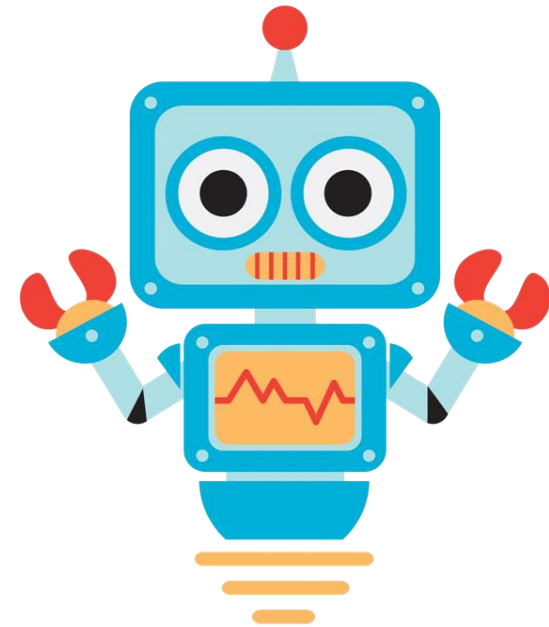
# BotDetect

Team 8 Demo 2: Data Exploration

Carlos Acuna, Corbin Graham, Jose Medina Mani, Adam Riffel, Nhan Tran

# Topic

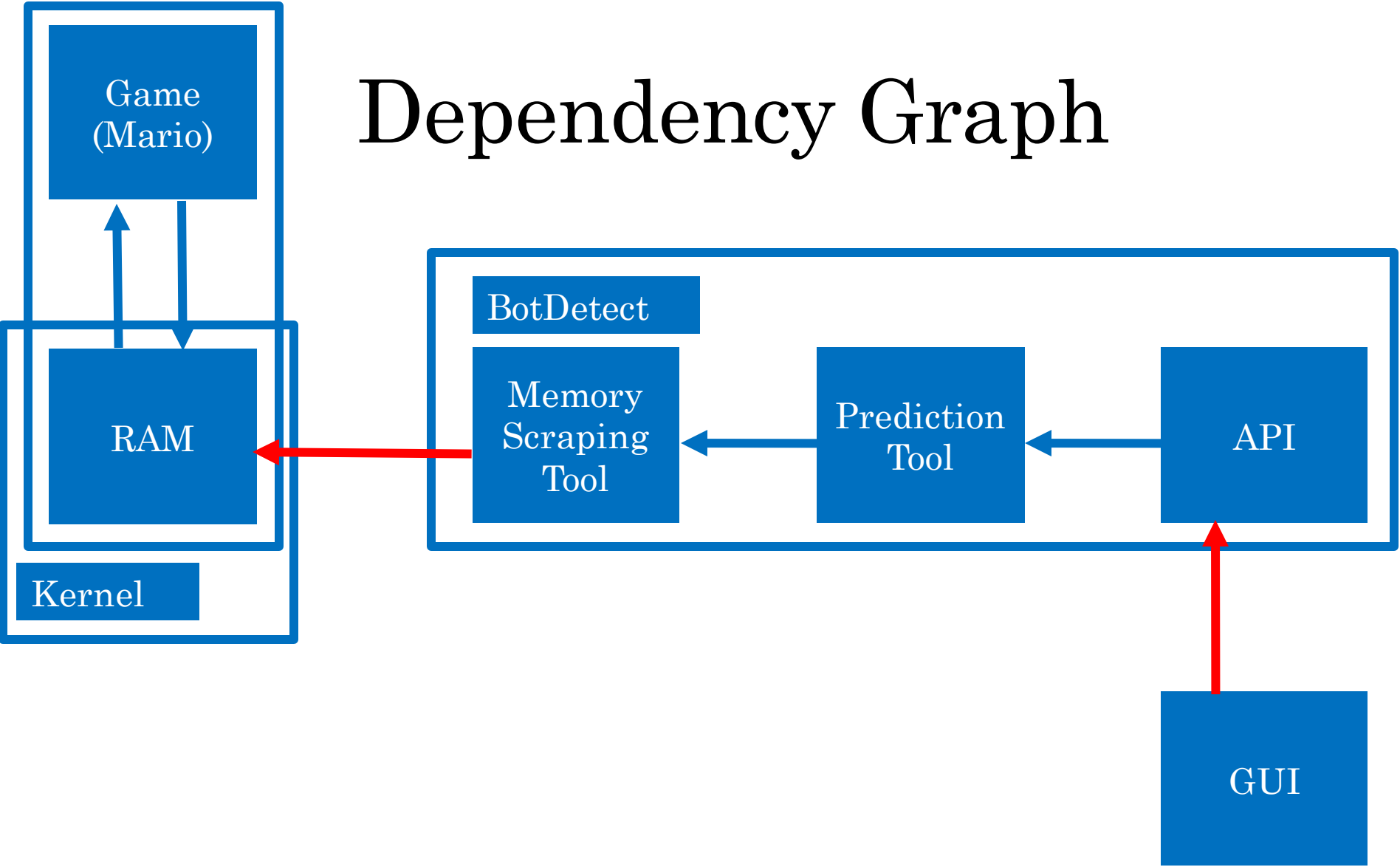
- Primary Research Question
  - Is it possible to differentiate between a human and bot playing a game?
- Secondary Research Questions
  - Can we do this from a naive approach?
  - Can we identify a human or bot in real time?
  - Can we expand this to identifying individual users?
  - Can our solution be generalized to multiple platforms?



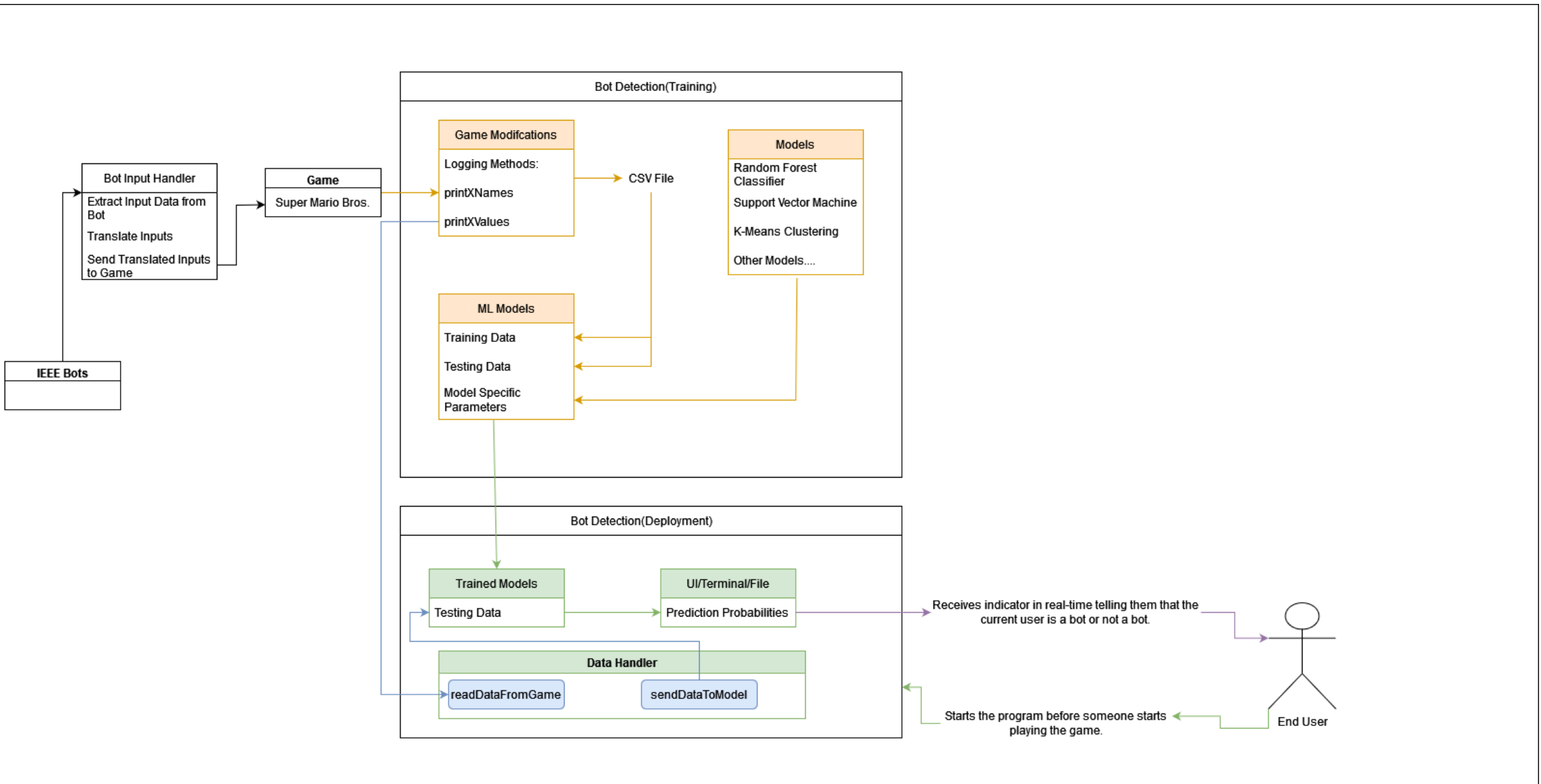




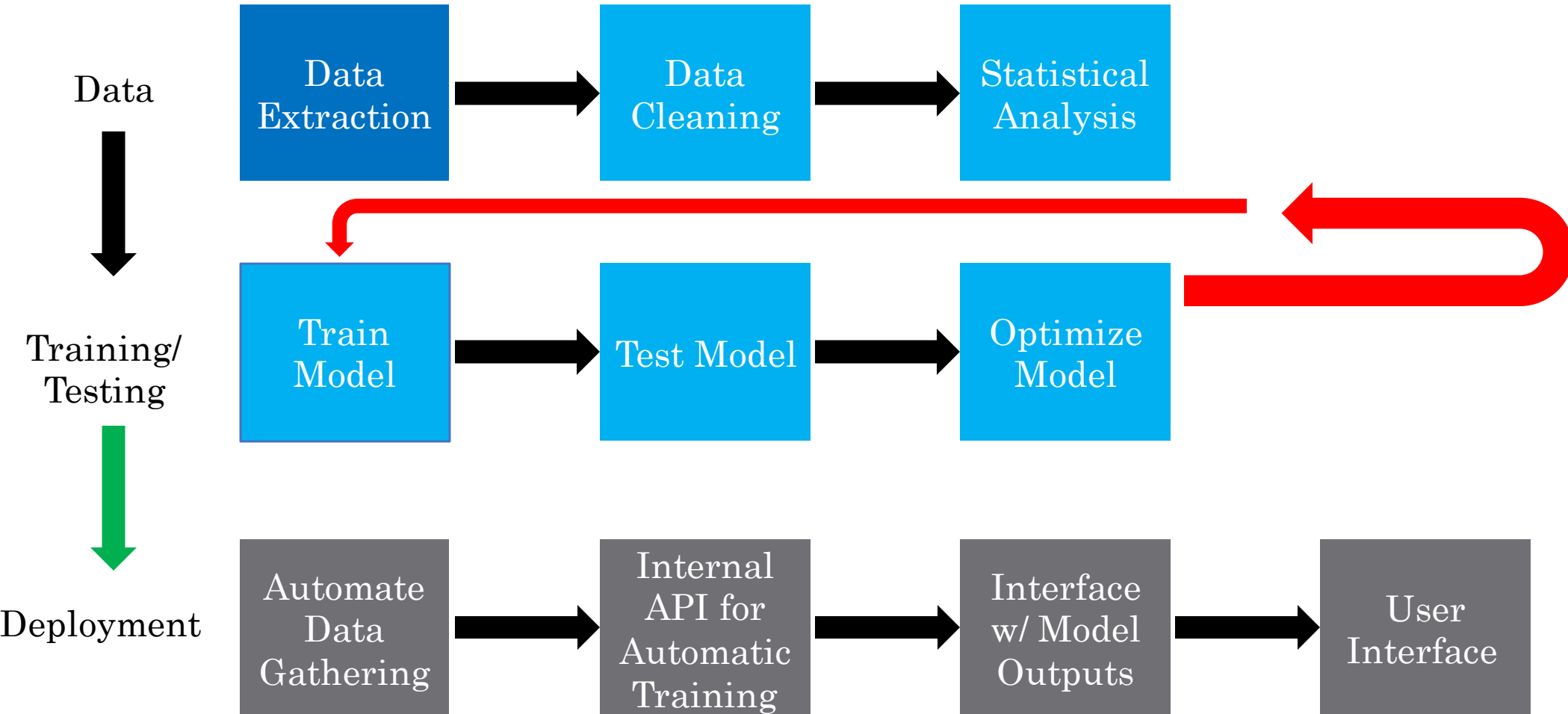
# Dependency Graph



# Interaction Graph



# Implementation Approach



# Current Status

- Implemented Memory Scraping in Host Game
  - Bypassed external memory scraping
    - Kernel encrypts memory (on some systems)
    - Logs RAM (2KB) to .csv file
    - Pre-process some labels
      - Game Time (s)
      - Program Time (ms)
      - Controller Input
      - Input Duration
      - X and Y Position
      - ...
- Implemented Data Cleaning
- Implemented Data Analysis
- Implemented Multiple Prediction Functions
  - Comparing two users until bot is implemented

# Data Cleaning

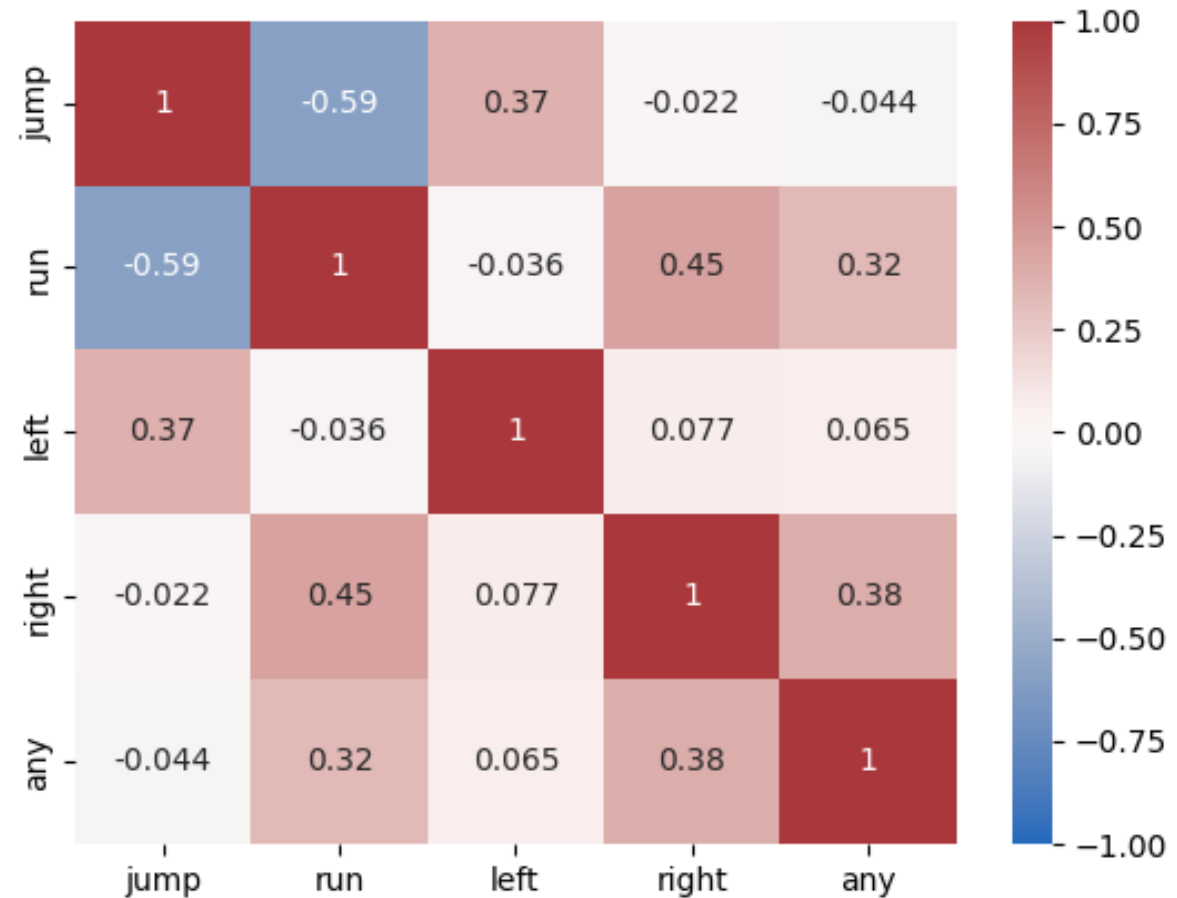
- Remove unnecessary predictors
- Normalize numerical data
  - Different scenarios further normalized data
- Evaluate null values
  - Determine if they should be dropped or modified
- Remove numbers outside standard range

|      | Name | SDL_Ticks | Timer | PlayerState | CombinedButton | A   | B   | Select | Start | Up  | ... | addr_2039 | addr_2040 | addr_2041 | addr_2042 | addr_2043 | addr_2044 | addr_2045 | addr_2046 | addr_2047 | Unnamed: 2071 |       |     |
|------|------|-----------|-------|-------------|----------------|-----|-----|--------|-------|-----|-----|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|---------------|-------|-----|
| 0    | 3    | 74        | 0     | 0           | 0              | 0   | 0   | 0      | 0     | 0   | ... | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 165.0         | NaN   |     |
| 1    | 3    | 91        | 0     | 0           | 0              | 0   | 0   | 0      | 0     | 0   | ... | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0             | 165.0 | NaN |
| 2    | 3    | 107       | 0     | 0           | 0              | 0   | 0   | 0      | 0     | 0   | ... | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0             | 165.0 | NaN |
| 3    | 3    | 123       | 0     | 0           | 0              | 0   | 0   | 0      | 0     | 0   | ... | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0             | 165.0 | NaN |
| 4    | 3    | 140       | 0     | 0           | 0              | 0   | 0   | 0      | 0     | 0   | ... | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0             | 165.0 | NaN |
| ...  | ...  | ...       | ...   | ...         | ...            | ... | ... | ...    | ...   | ... | ... | ...       | ...       | ...       | ...       | ...       | ...       | ...       | ...       | ...       | ...           | ...   | ... |
| 2955 | 3    | 49309     | 219   | 5           | 0              | 0   | 0   | 0      | 0     | 0   | ... | 0         | 2         | 1         | 9         | 0         | 0         | 0         | 0         | 0         | 0             | 165.0 | NaN |
| 2956 | 3    | 49325     | 218   | 5           | 0              | 0   | 0   | 0      | 0     | 0   | ... | 0         | 2         | 1         | 8         | 0         | 0         | 0         | 0         | 0         | 0             | 165.0 | NaN |
| 2957 | 3    | 49341     | 217   | 5           | 0              | 0   | 0   | 0      | 0     | 0   | ... | 0         | 2         | 1         | 7         | 0         | 0         | 0         | 0         | 0         | 0             | 165.0 | NaN |
| 2958 | 3    | 49357     | 216   | 5           | 0              | 0   | 0   | 0      | 0     | 0   | ... | 0         | 2         | 1         | 6         | 0         | 0         | 0         | 0         | 0         | 0             | 165.0 | NaN |
| 2959 | 3    | 49376     | 215   | 5           | 0              | 0   | 0   | 0      | 0     | 0   | ... | 0         | 2         | 1         | 5         | 0         | 0         | 0         | 0         | 0         | 0             | 165.0 | NaN |

```
df.isna().sum()
✓ 0.0s
Name 0
SDL_Ticks 0
Timer 0
PlayerState 0
CombinedButton 0
A 0
B 0
Select 0
Start 0
Up 0
Down 0
Left 0
Right 0
RightDuration 0
LeftDuration 0
JumpDuration 0
RunDuration 0
TotalJumpAirtime 87
TotalFallAirtime 0
POWER_EXIST 0
FACE 0
X 0
Y 0
dtype: int64
```

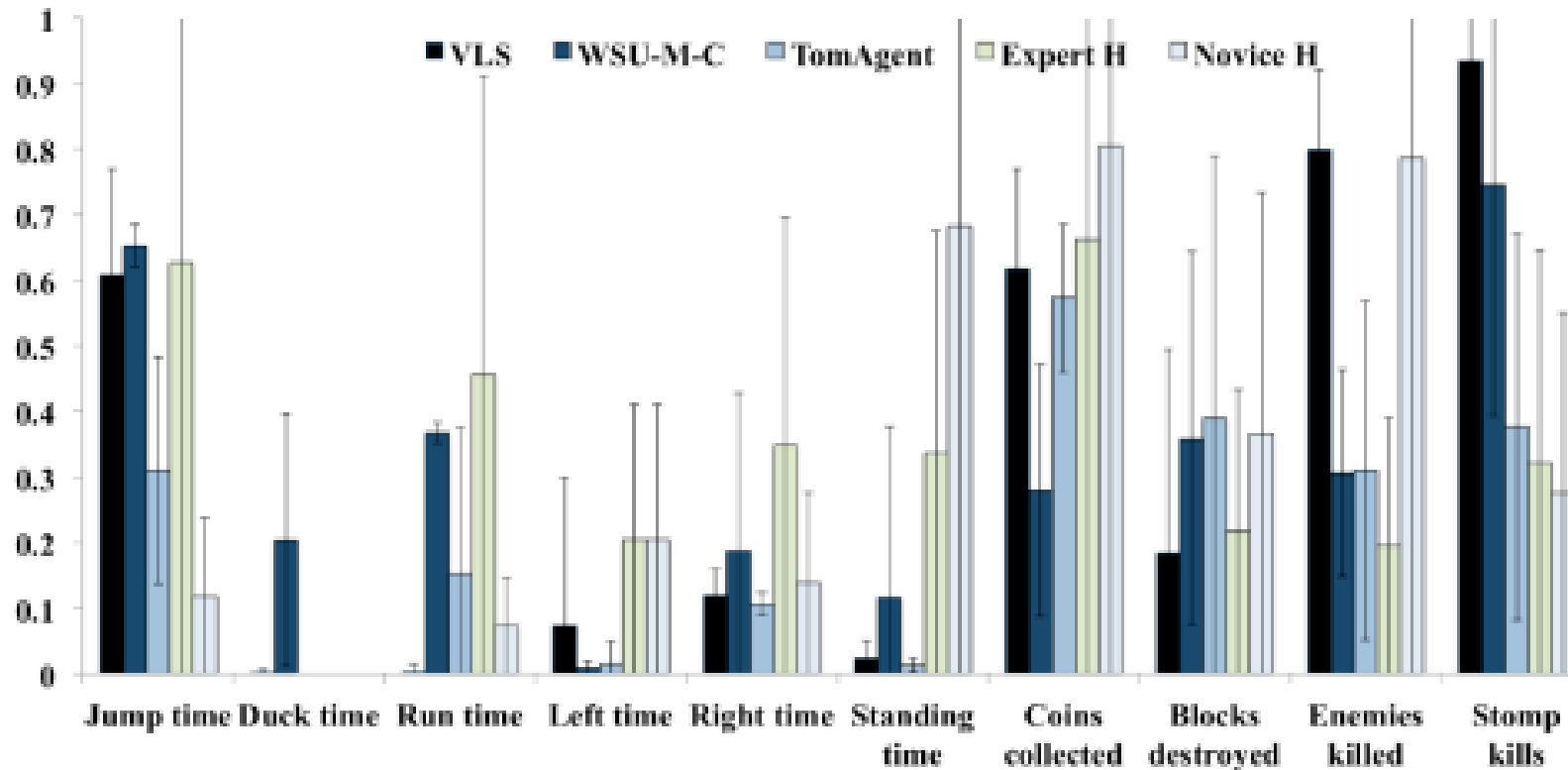
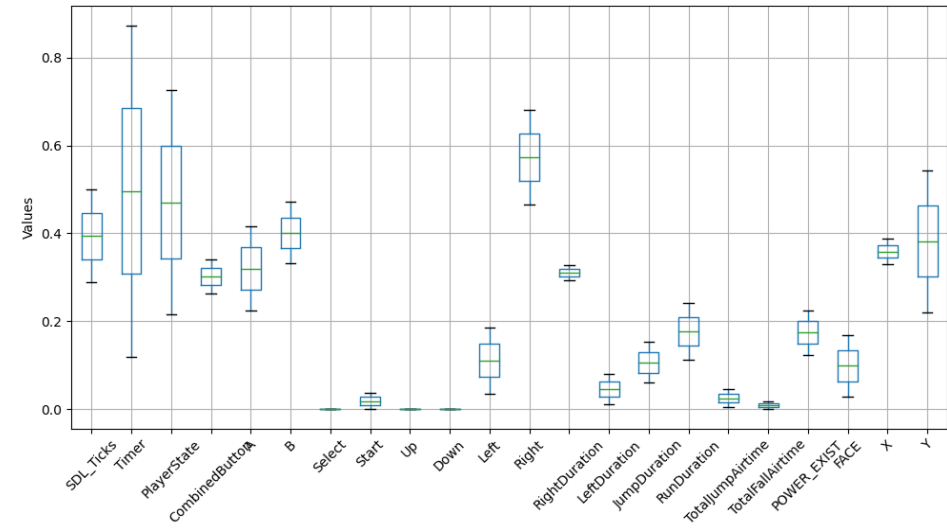
# Correlation Analysis

- Taking averages from various statistics such as how long between inputs.
- Using these averages to try and find a correlation between inputs
- Different correlation methods:
  - Pearson (linear)
  - Kendall
  - Spearman
- Best results, Spearman (depicted in graph)



# Player Analysis

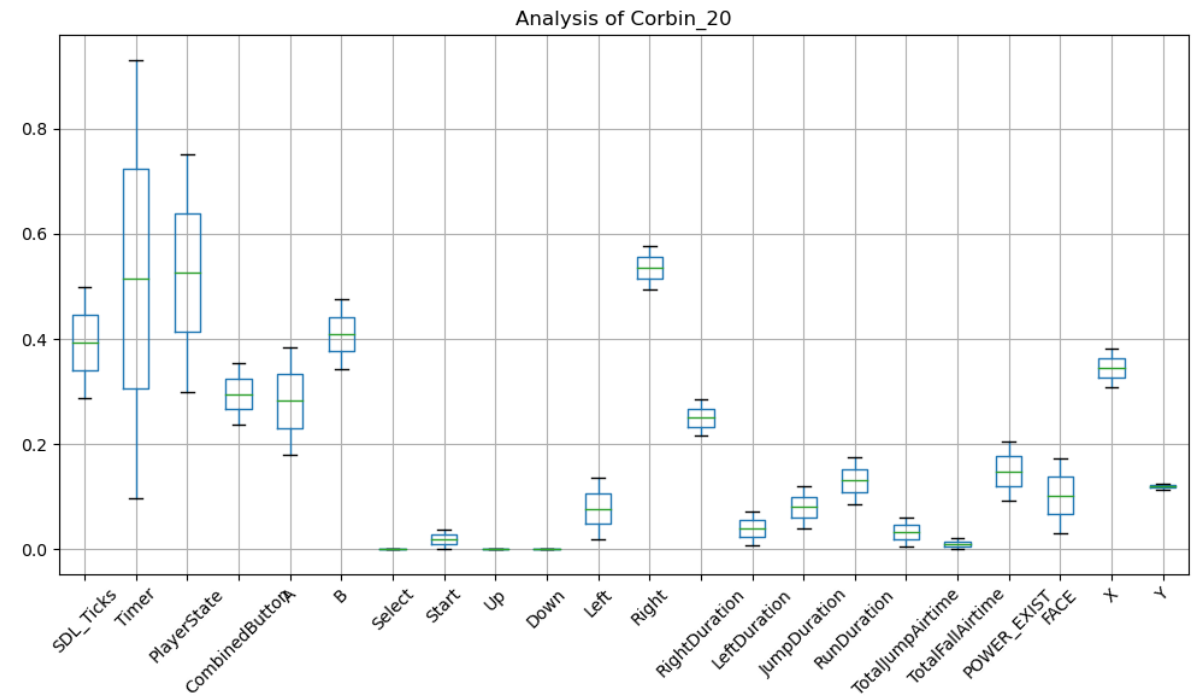
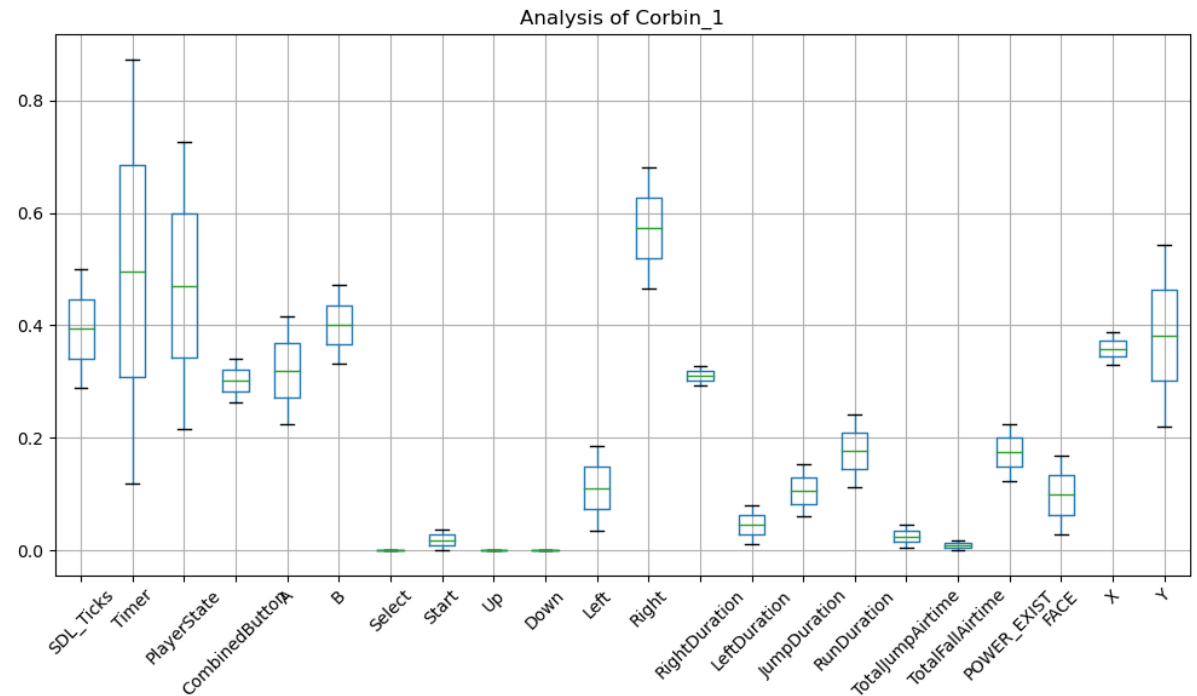
- Modeled results of peer-reviewed paper
- Applied max-min normalization
- Found average and standard deviation for each labeled predictor



N. Shaker *et al.*, "The turing test track of the 2012 Mario AI Championship: Entries and evaluation," *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, Niagara Falls, ON, Canada, 2013, pp. 1-8, doi: 10.1109/CIG.2013.6633634.

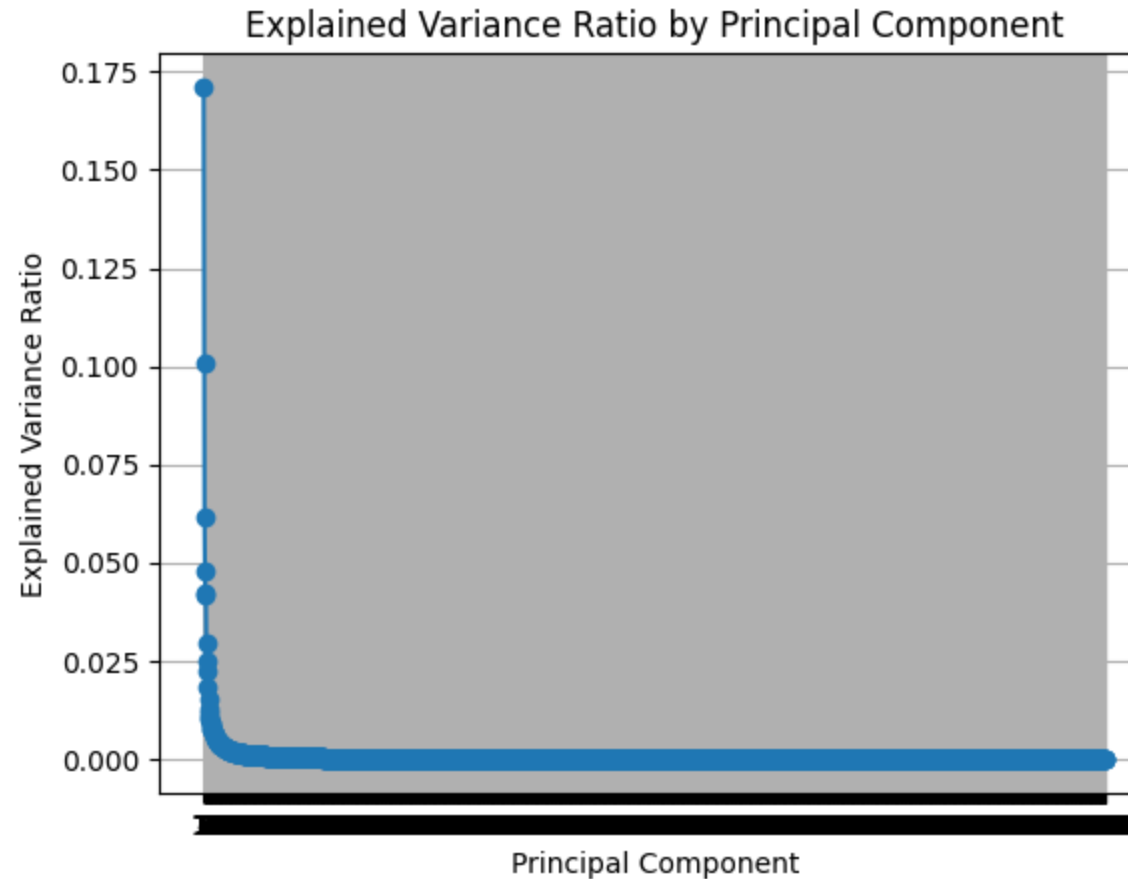
# Improvement Trajectory

- How does the average player improve as they play more?
- Does a non-genetic bot improve in the same way?
- Can we compare the first run and the last run to determine an improvement level?



# Principal Component Analysis (PCA)

- Find the 'elbow'
- Visual analysis indicates our best ratio
- We then reduce the number of dimensions based on this ideal



# Models

Logistic Regression

Random Forest  
Classifier

Support Vector  
Machine

K-Means Clustering

KNN Clustering

LSTM

Hidden Markov  
Models

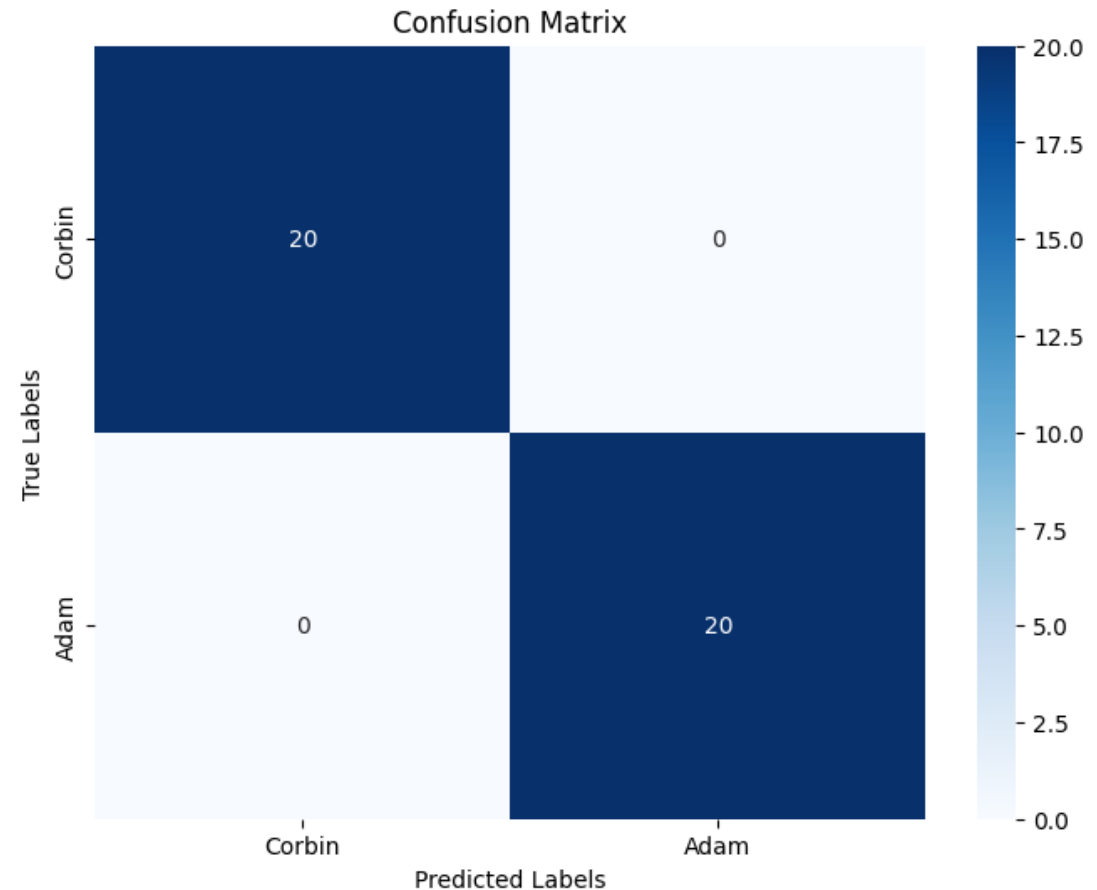
# KNN Clustering / Classification

- Challenges
  - Large range of accuracy when predicting against different users
  - Difficult to adapt
- K=2
  - Binary classification comparing two users
- K=5
  - Cluster All Users

| K=2    | Bot | Adam | Carlos | Corbin | Jose | Nhan |
|--------|-----|------|--------|--------|------|------|
| Bot    | -   |      |        |        |      |      |
| Adam   |     | -    | 100%   | 99%    | 48%  | 52%  |
| Carlos |     | 100% | -      | 100%   | 100% | 100% |
| Corbin |     | 99%  | 100%   | -      | 46%  | 51%  |
| Jose   |     | 48%  | 100%   | 46%    | -    | 48%  |
| Nhan   |     | 52%  | 100%   | 51%    | 48%  | -    |

# Hidden Markov Model

- Each user gets their own HMM which will be adjusted as they play the game to model their improvement trajectory
- Transitions are given a baseline then adjusted from the baseline
- Output: ('1', 82.9877539623103) for the Adam model trained on first run and tested on second run



# Logistic Regression

## Rationale:

- Very simple
- A starting point after collecting data

## Challenges:

- Only works for binary classification.
- Only works on data with linear relationships.

# Logistic Regression w/ Scoring Function

| Samples   | Prediction | Confidence | Samples | Prediction | Confidence |
|-----------|------------|------------|---------|------------|------------|
| Carlos_41 | Jose       | 55.9%      | Jose_41 | Jose       | 74.5%      |
| Carlos_42 | Jose       | 57.0%      | Jose_42 | Jose       | 74.2%      |
| Carlos_43 | Jose       | 57.2%      | Jose_43 | Jose       | 74.8%      |
| Carlos_44 | Jose       | 58.0%      | Jose_44 | Jose       | 75.3%      |
| Carlos_45 | Jose       | 60.0%      | Jose_45 | Jose       | 74.5%      |

# Support Vector Machines

Rationale:

- More robust than logistic regression
- Can choose between linear, polynomial, rbf, or sigmoid kernel.

Challenges:

- Incredibly slow training and testing time.
- For this dataset, does not provide good enough accuracy.
- Had to create a scoring function to reduce dimensions.

# SVM (Polynomial, Degree = 3) w/ Scoring Function

| Samples   | Prediction | Confidence |
|-----------|------------|------------|
| Carlos_41 | Carlos     | 75.4%      |
| Carlos_42 | Carlos     | 77.9%      |
| Carlos_43 | Carlos     | 79.1%      |
| Carlos_44 | Carlos     | 79.6%      |
| Carlos_45 | Carlos     | 76.0%      |

| Samples | Prediction | Confidence |
|---------|------------|------------|
| Jose_41 | Jose       | 57.8%      |
| Jose_42 | Jose       | 55.5%      |
| Jose_43 | Jose       | 53.4%      |
| Jose_44 | Jose       | 55.9%      |
| Jose_45 | Jose       | 53.0%      |

# Random Forest Classifier

Rationale:

- Recommended by client
- Non-Parametric
- Helps identify feature importance
- Faster training time than SVM for the size of the dataset

# Random Forest Classifier (Binary)

Person 1 Sample

| Frame | Prediction                 |
|-------|----------------------------|
| 14842 | [0.65835615<br>0.34164385] |
| 14858 | [0.710278<br>0.289722]     |
| 14875 | [0.96485905<br>0.03514095] |
| ...   |                            |

Person 2 Sample

| Frame | Prediction                 |
|-------|----------------------------|
| 14842 | [0.29875938<br>0.70124062] |
| 14858 | [0.30117425<br>0.69882575] |
| 14875 | [0.43758544<br>0.56241456] |
| ...   |                            |

# Random Forest Classifier (Binary)

| Samples   | Confidence | Samples | Confidence |
|-----------|------------|---------|------------|
| Carlos_41 | 99.2%      | Adam_16 | 80.8%      |
| Carlos_42 | 100.0%     | Adam_17 | 88.0%      |
| Carlos_43 | 99.9%      | Adam_18 | 86.0%      |
| Carlos_44 | 100.0%     | Adam_19 | 75.1%      |
| Carlos_45 | 99.3%      | Adam_20 | 88.9%      |
| Carlos_46 | 99.9%      |         |            |
| Carlos_47 | 99.9%      |         |            |
| Carlos_48 | 99.8%      |         |            |
| Carlos_49 | 98.3%      |         |            |
| Carlos_50 | 99.9%      |         |            |

# Random Forest Classifier (Multi-Class)

Challenges:

- Expanding RFC to multi-class makes it much less accurate.

# Random Forest Classifier (Multi-Class)

| Samples   | Prediction | Confidence |
|-----------|------------|------------|
| Carlos_41 | Carlos     | 65.4%      |
| Carlos_42 | Carlos     | 66.0%      |
| Carlos_43 | Carlos     | 54.2%      |
| Carlos_44 | Carlos     | 68.8%      |
| Carlos_45 | Carlos     | 65.2%      |

| Samples | Prediction | Confidence |
|---------|------------|------------|
| Adam_16 | Nhan       | 61.2%      |
| Adam_17 | Nhan       | 69.0%      |
| Adam_18 | Nhan       | 63.1%      |
| Adam_19 | Nhan       | 51.5%      |
| Adam_20 | Nhan       | 65.7%      |

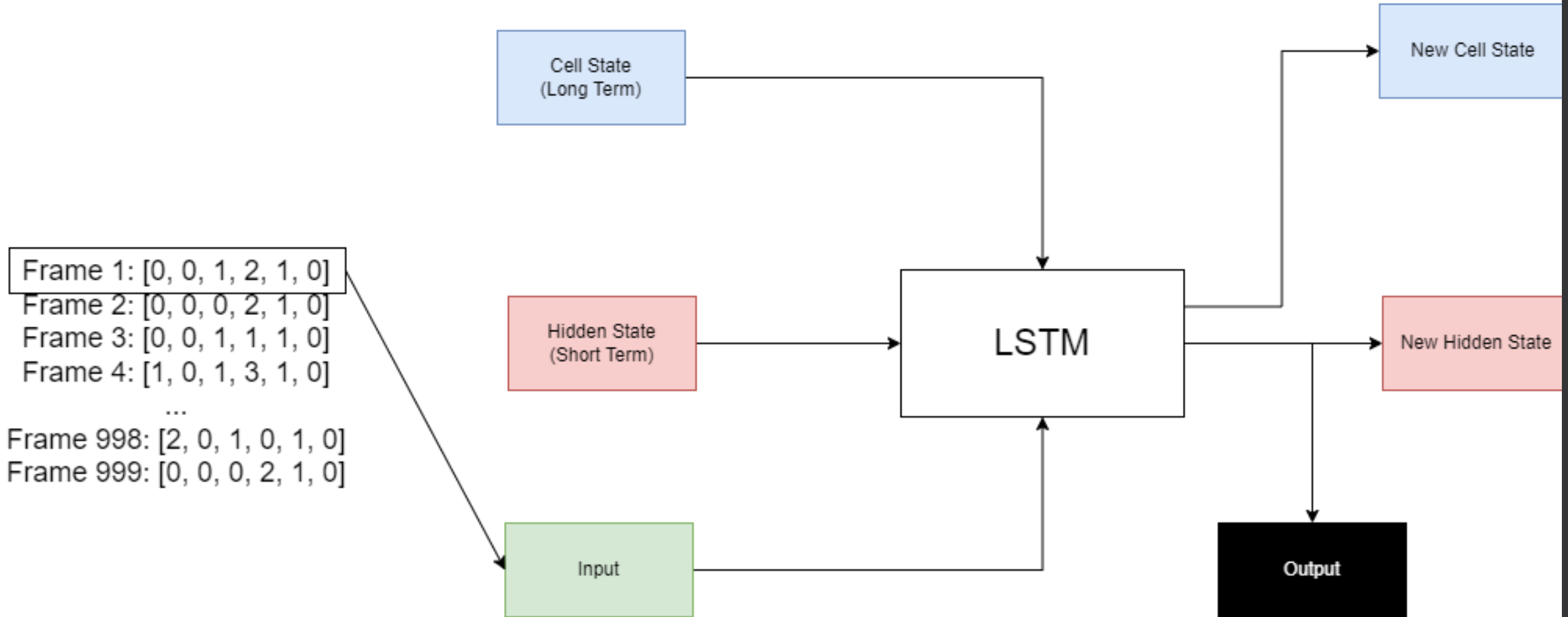
# Long-Short Term Memory

RNN with additional cell states for long term memory

Training on sequence of data one by one

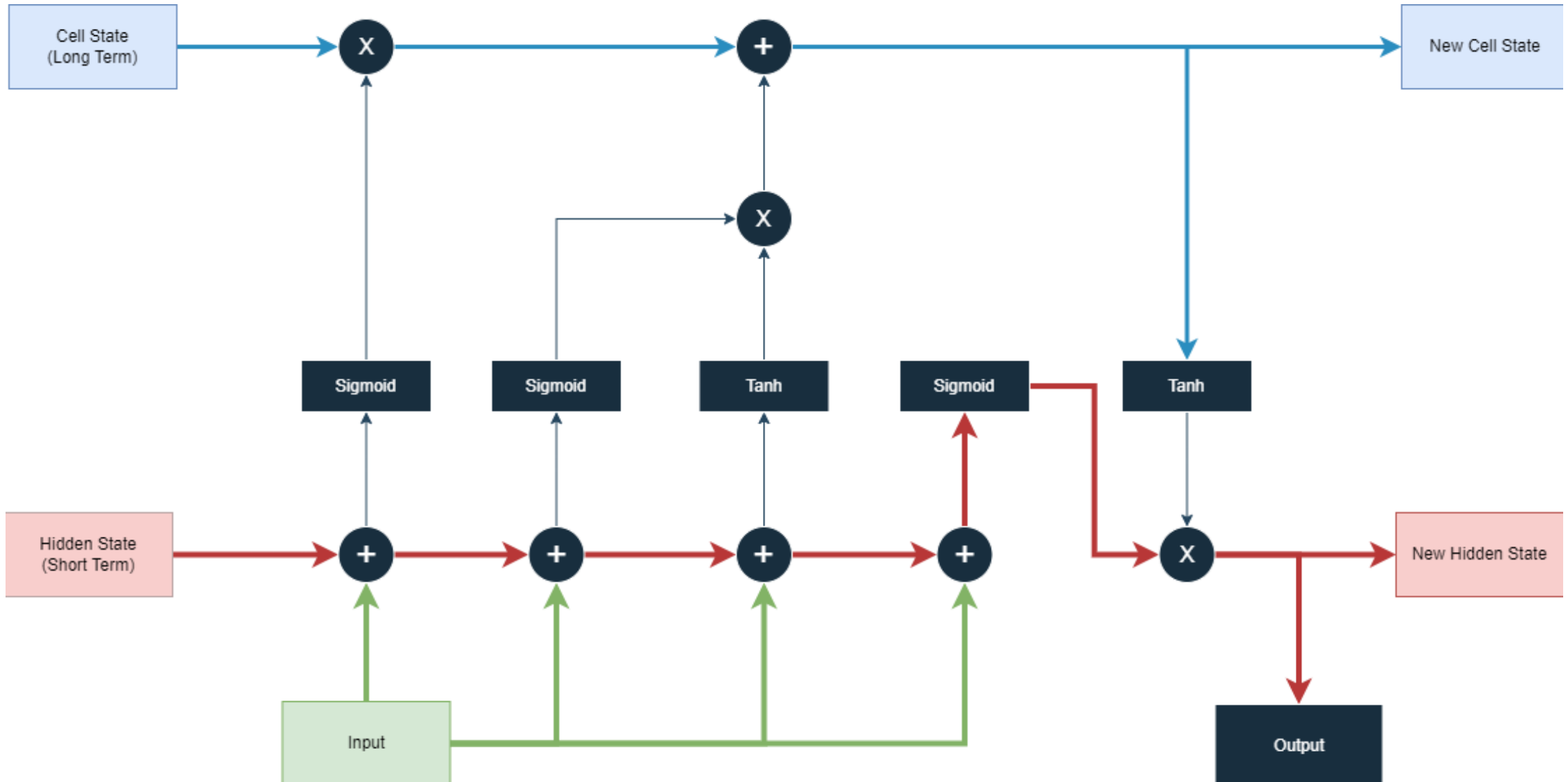
Rationale: Gameplay is a sequence of data

# LSTM Architecture

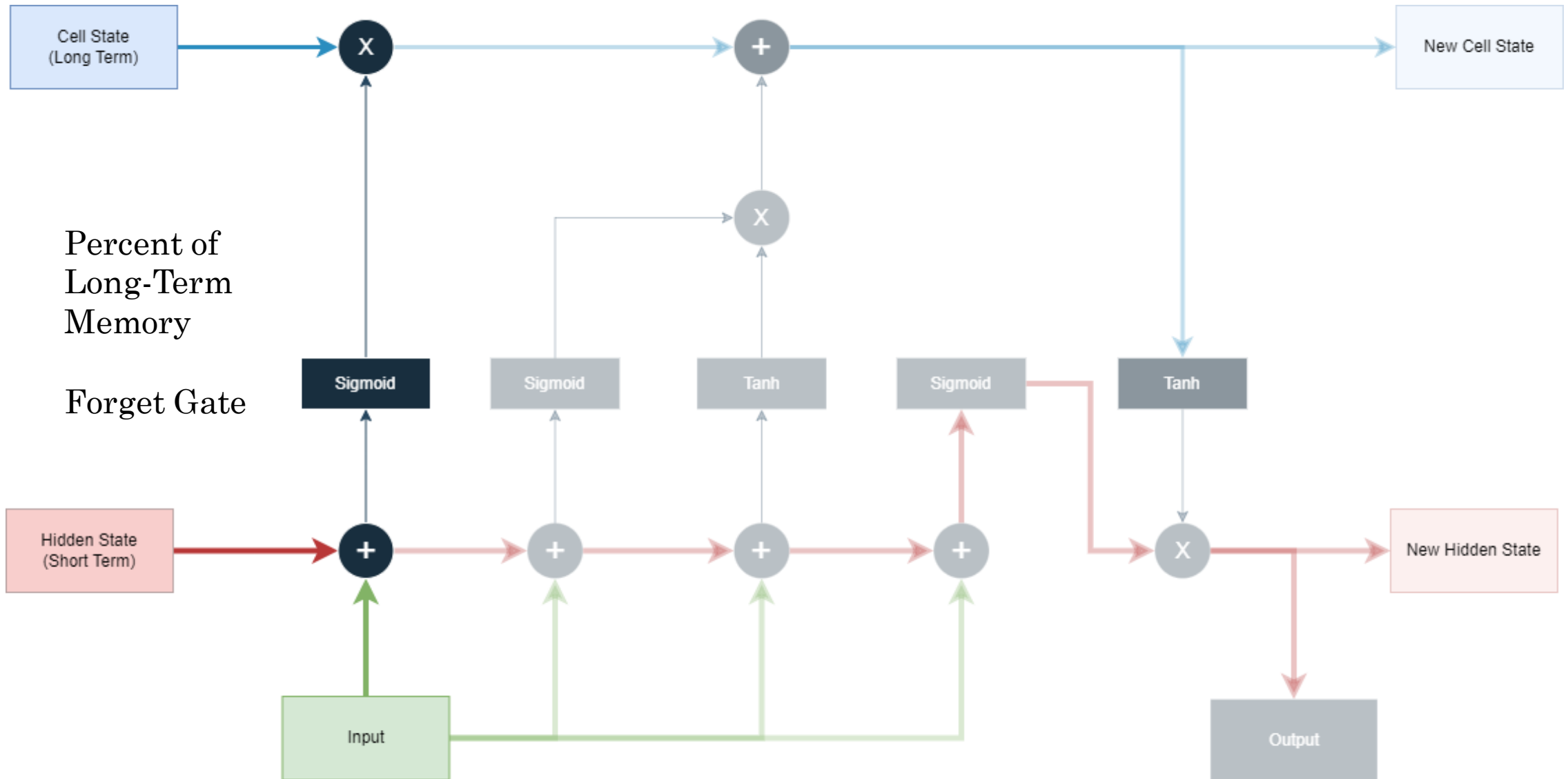


Prediction 1 for Frame 1: 5

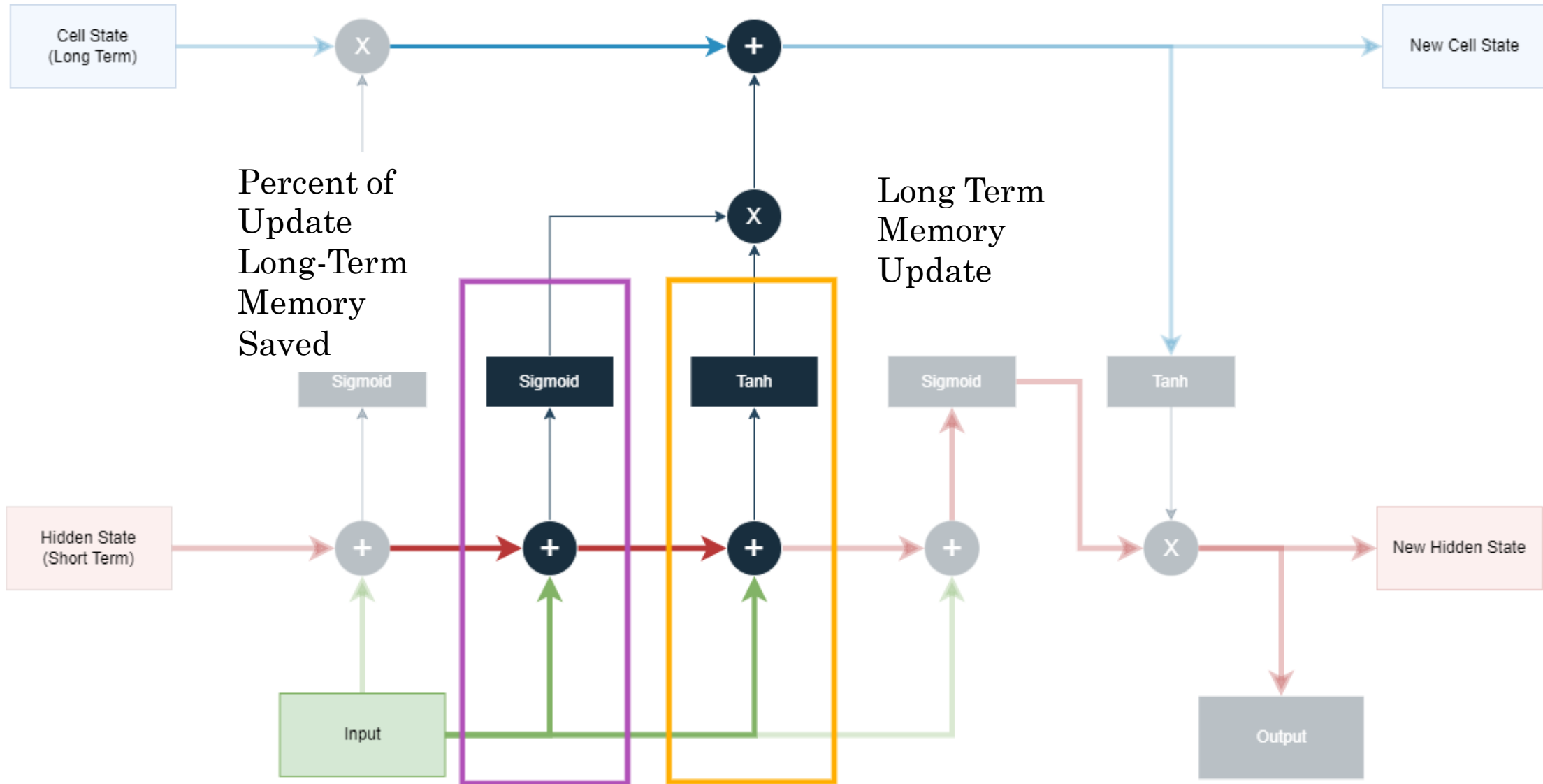
# LSTM Architecture



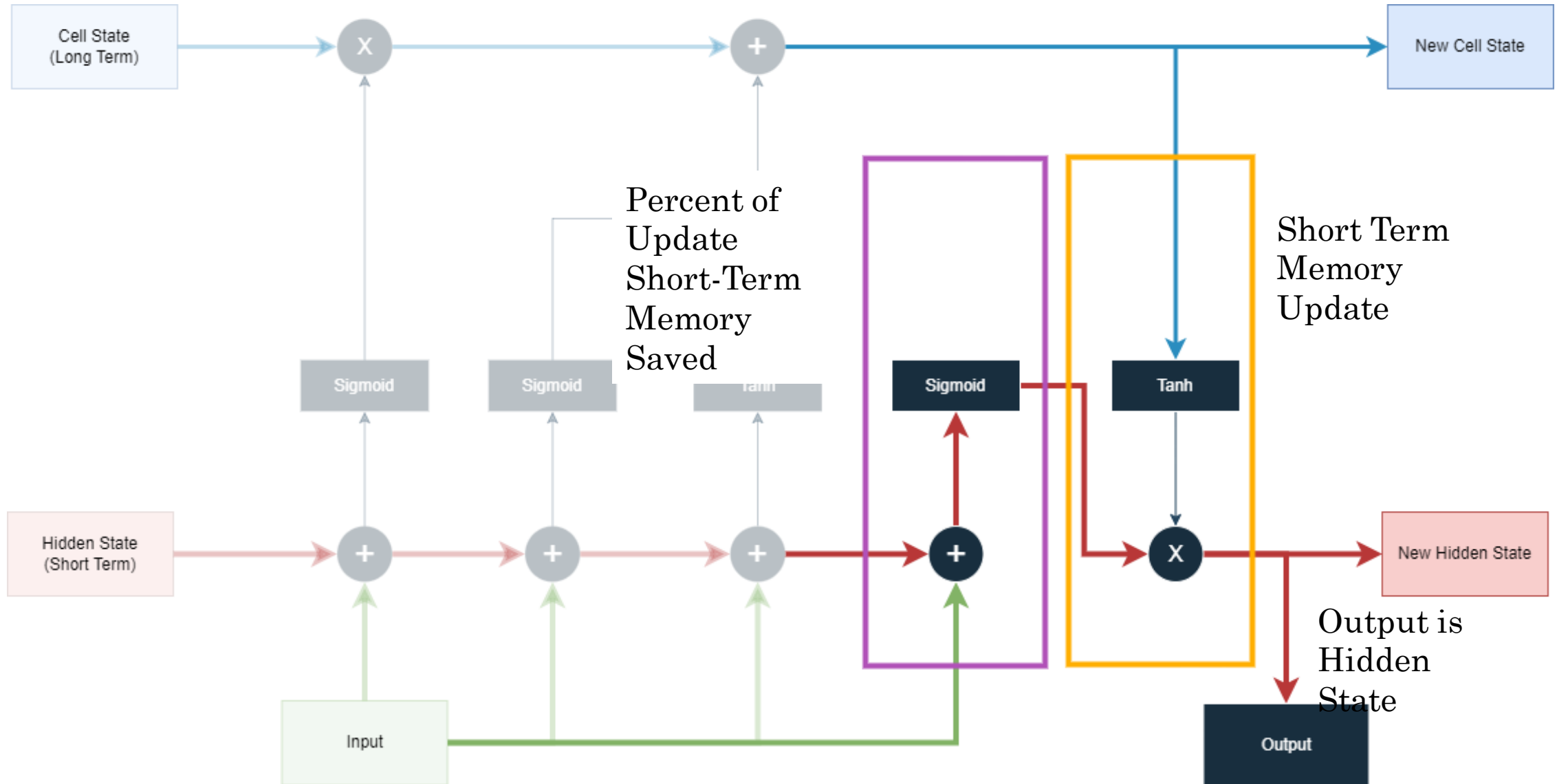
# LSTM Architecture



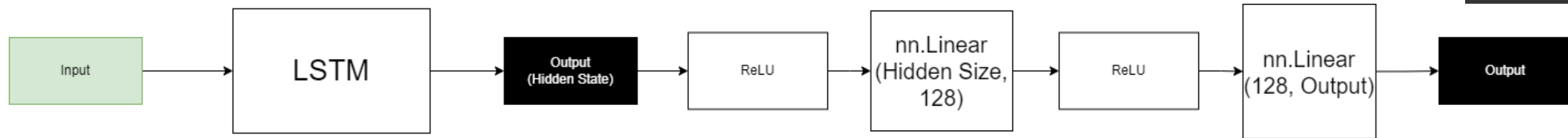
# LSTM Architecture



# LSTM Architecture

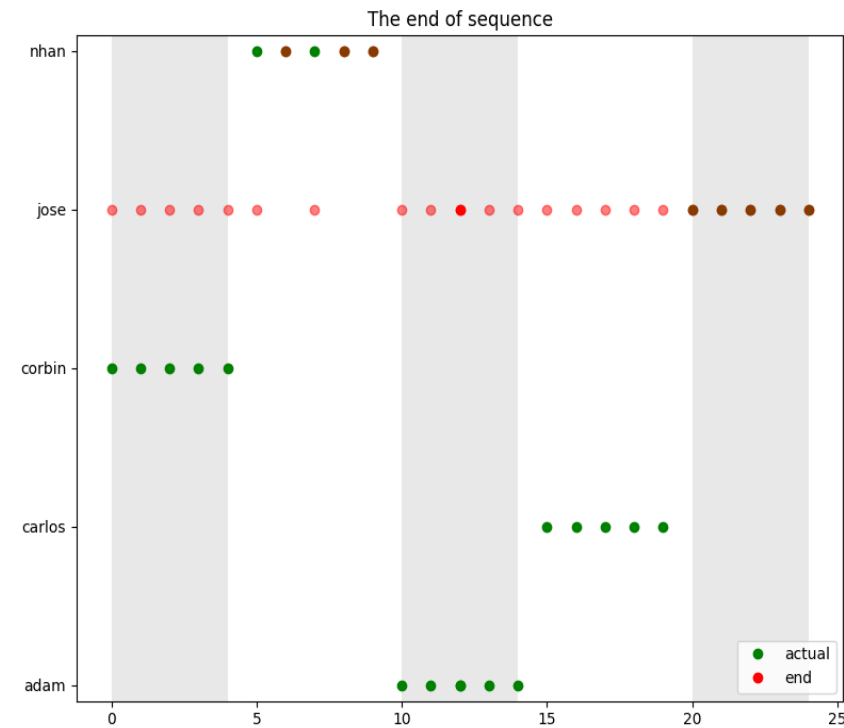
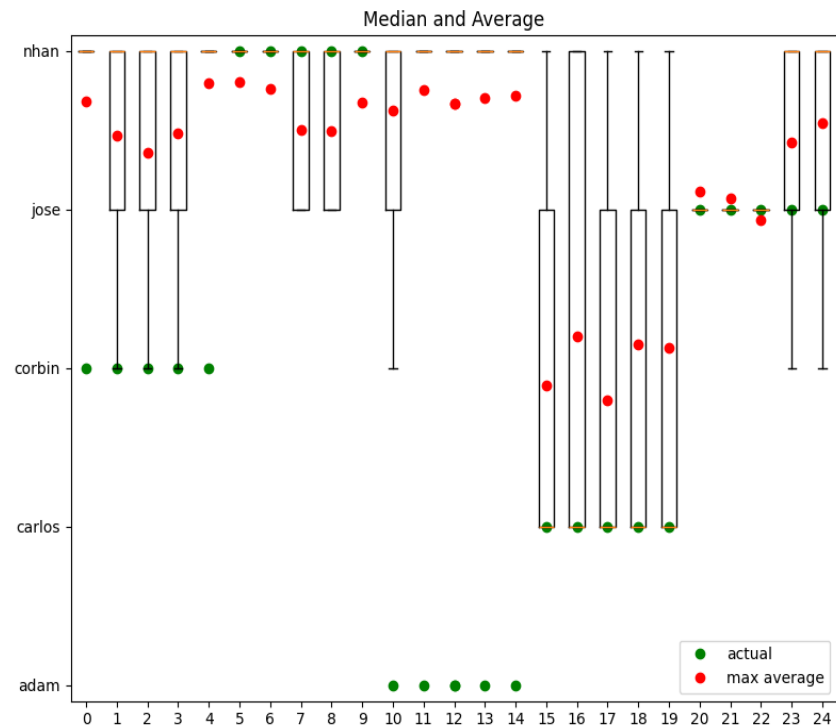
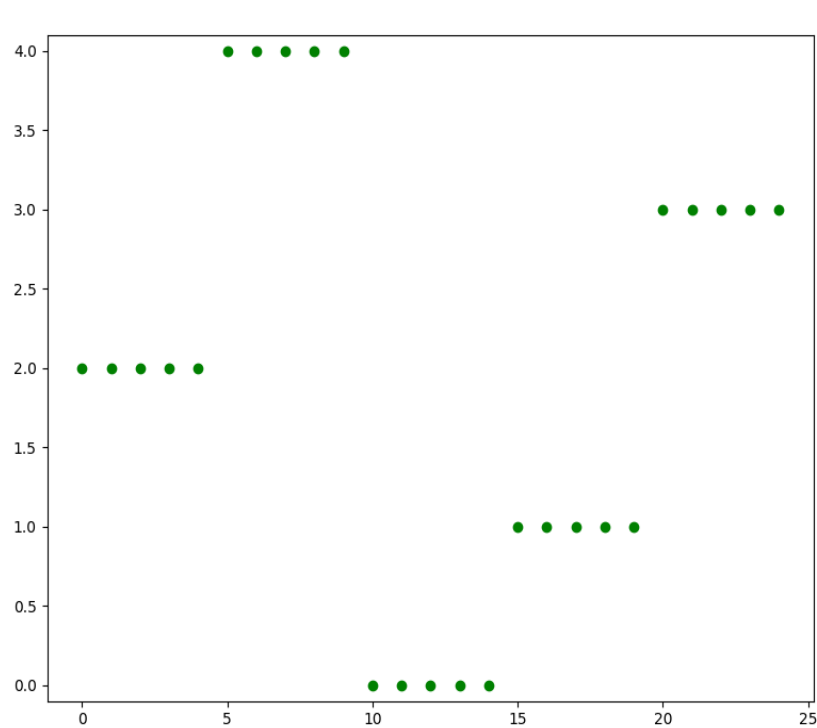


# LSTM Architecture



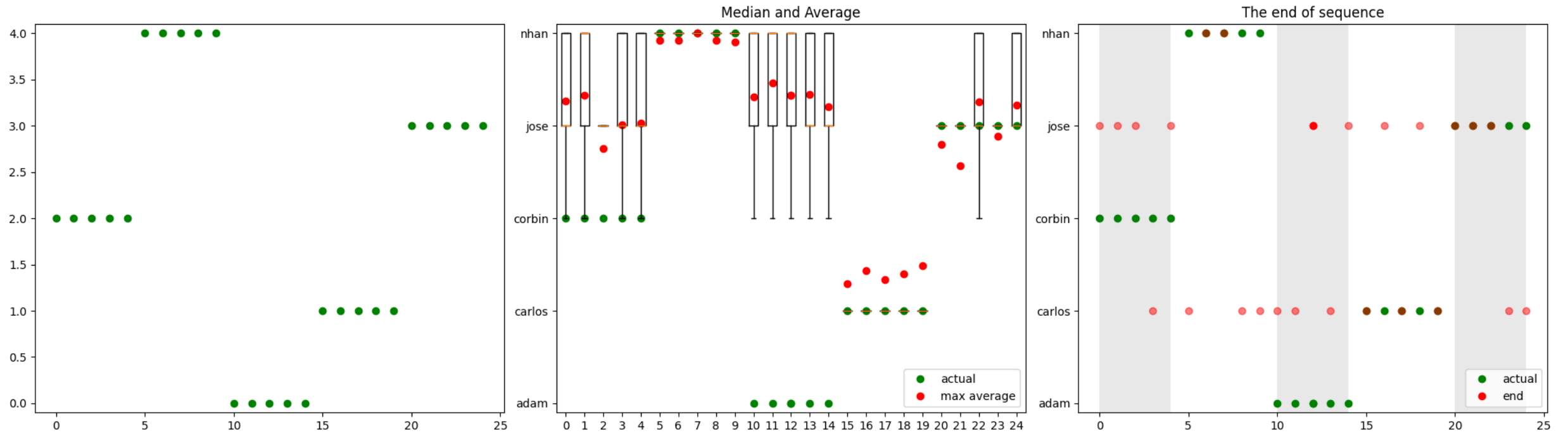
# Long-Short Term Memory Full Sequences

FEATURES: X, Y, RIGHT, LEFT, A, B, PLAYER\_STATE, FACE



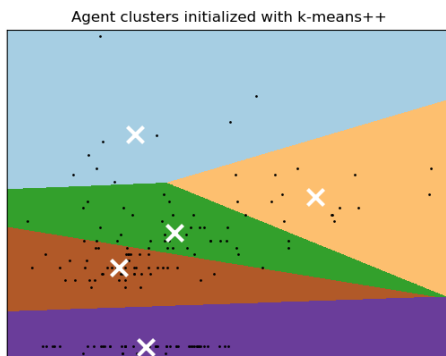
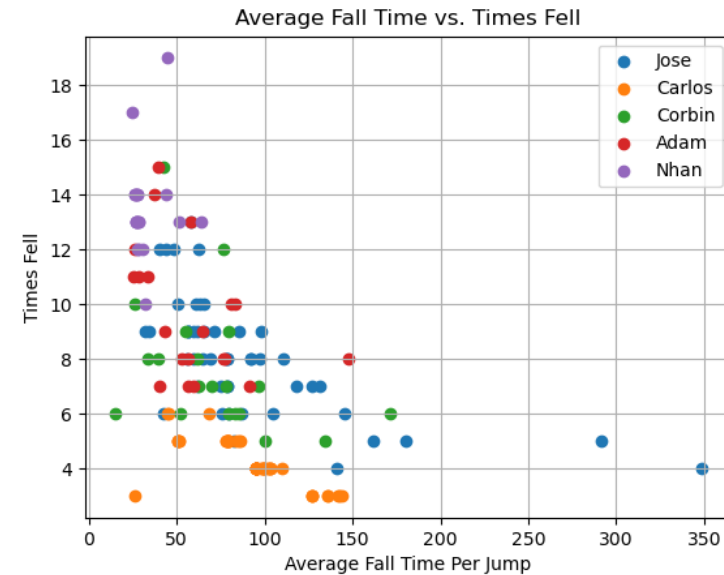
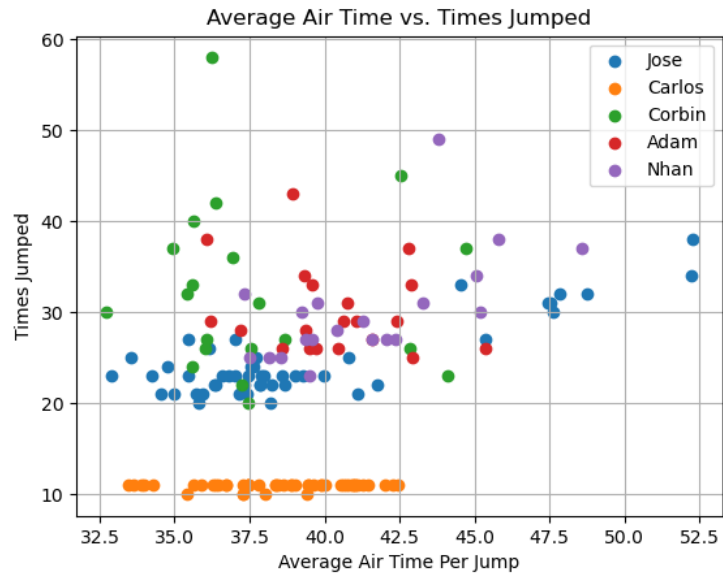
# Long-Short Term Memory Event Sequences

FEATURES: DELTA\_TICKS, X, Y, RIGHT, LEFT, A, B, PLAYER\_STATE, FACE

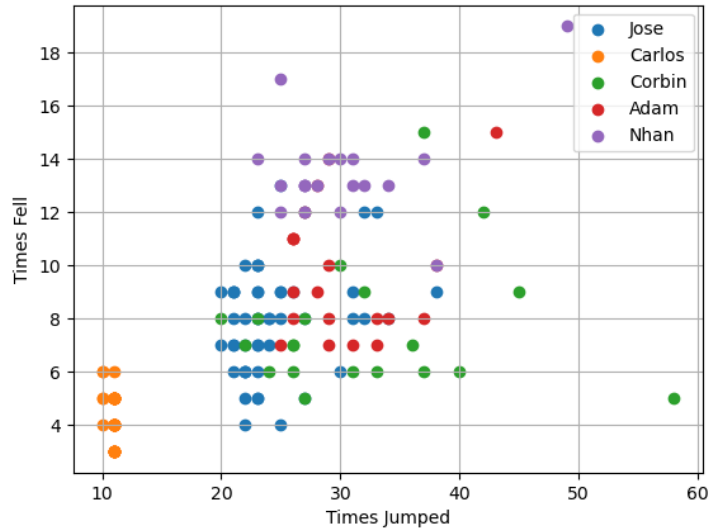


# K-Means Clustering

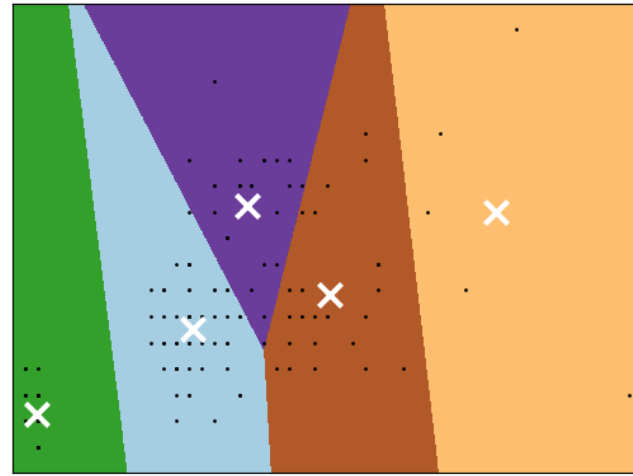
- Limited use unless data is preprocessed and/or feature selected
- Performance measured by similarity to labels (ground truth)



Times Jumped vs. Times Fell

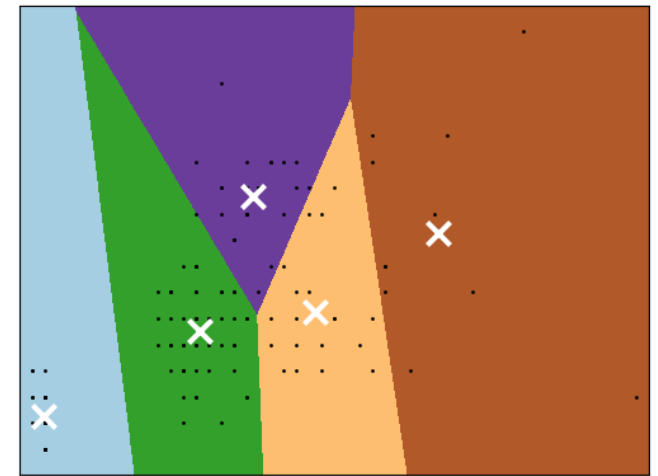


Agent clusters initialized with k-means++



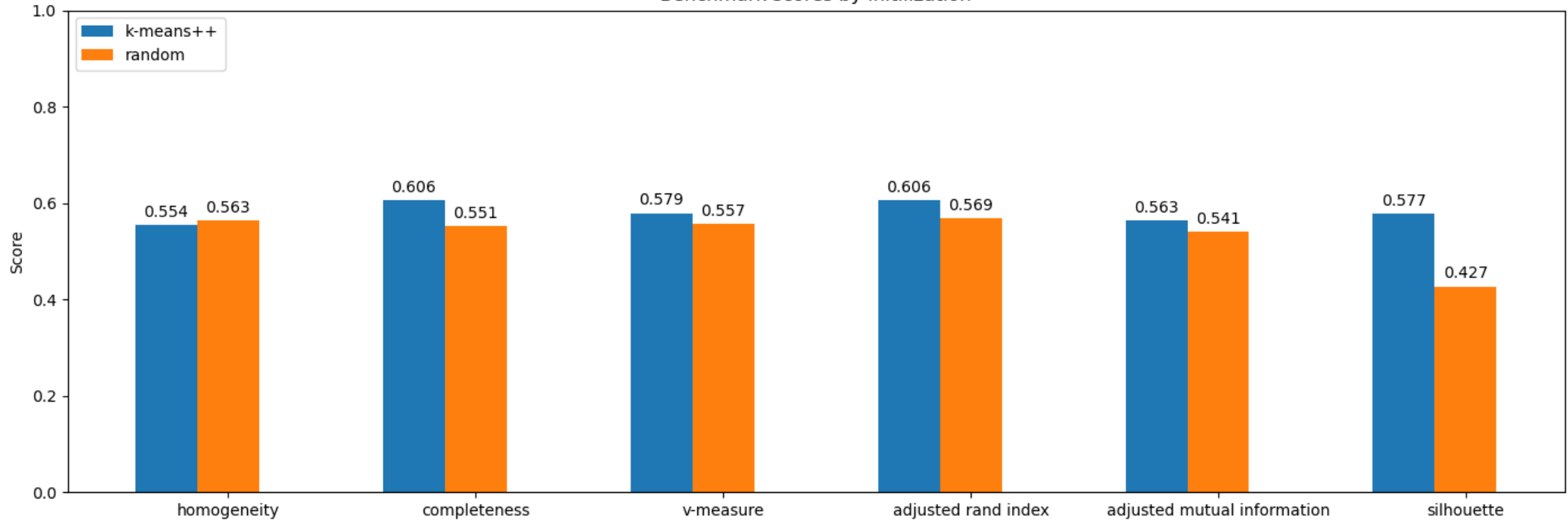
Inertia: 41

Agent clusters initialized with random



Inertia: 42

Benchmark scores by initialization



# Challenges with K-Means

- Arbitrary selection of features
- Information loss
- More agents?
- Different game?

# Challenges with Data

- Had to produce our own data
- Data cleaning/preprocessing needed
- Process of trial and error

# Challenges with Models

- Which model will meet which requirements?
- Model selection (which will be most effective for our data?)
- How to choose proper parameters for models?

# Remaining Work and Timeline

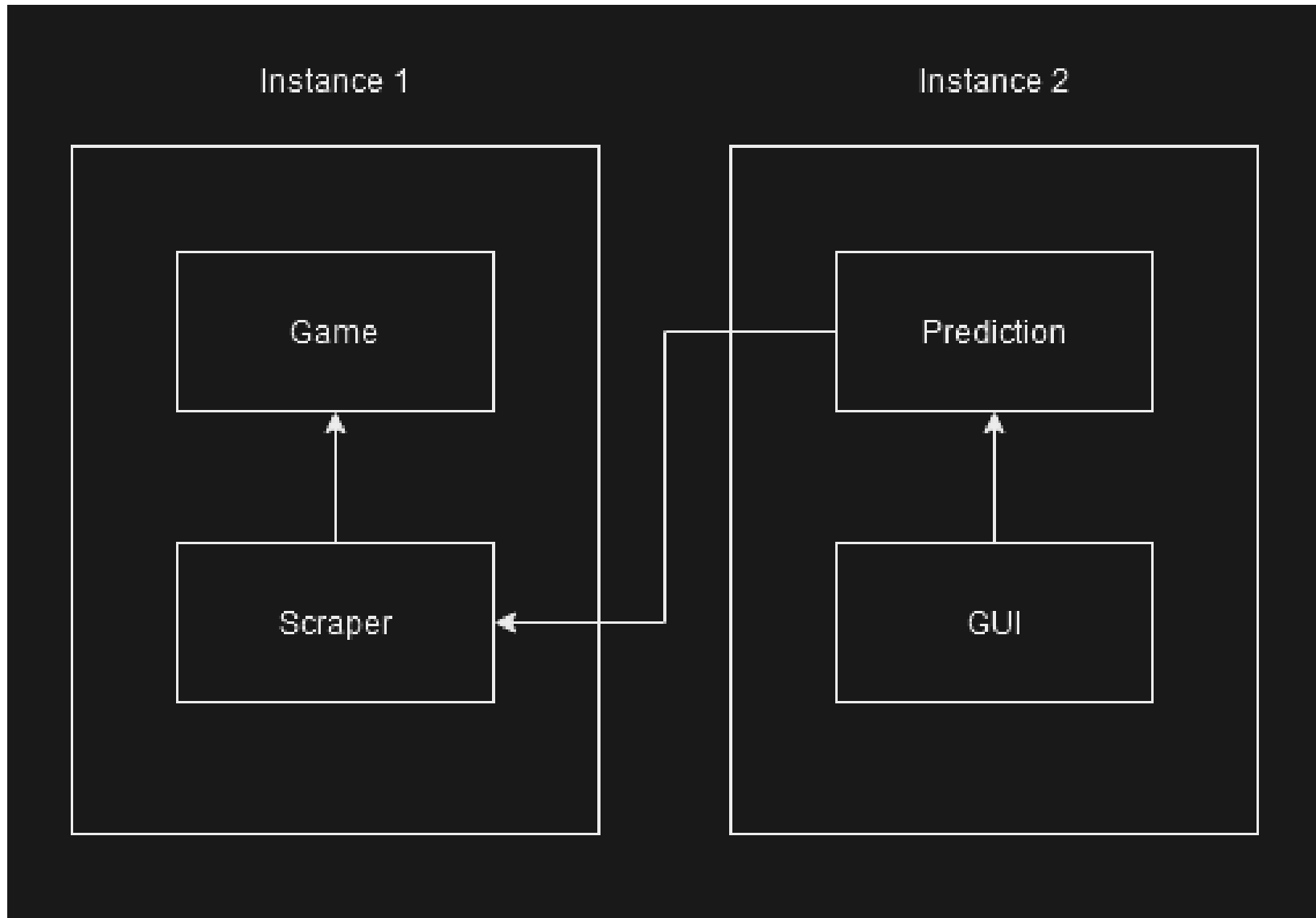
- Collecting bot data from IEEE Mario Turing Test Competition Entries
- Adjusting existing models to interact with the new data
- Creating new models to address several bot play styles
- Generalize findings
- Client-side bot detection in real time

# Demo

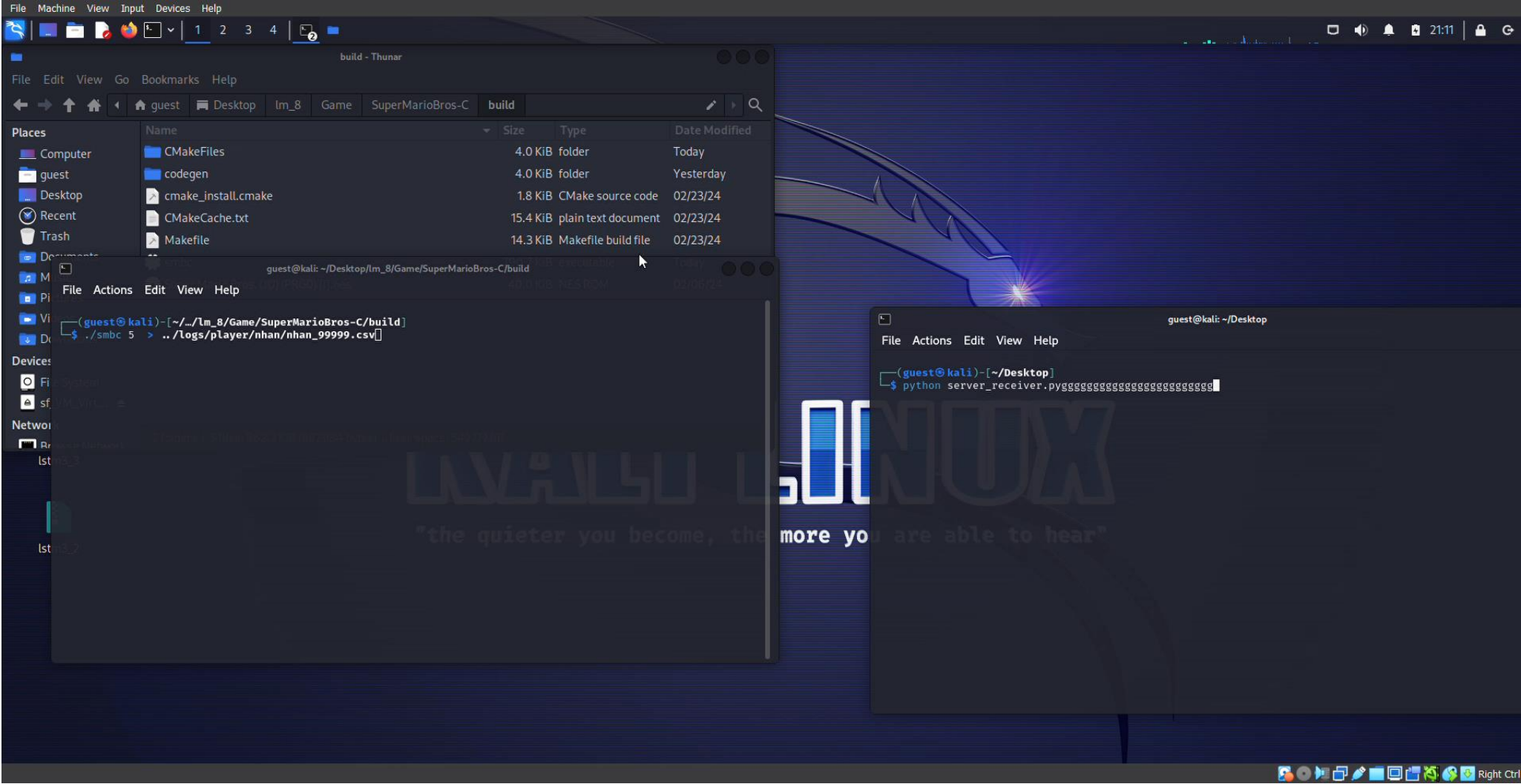
# Current Data to Prediction Pipeline

- Game Data to Prediction
- Socket IO
  - Game: C++
  - Prediction: Python

# Current Data to Prediction Pipeline



# Demo Video





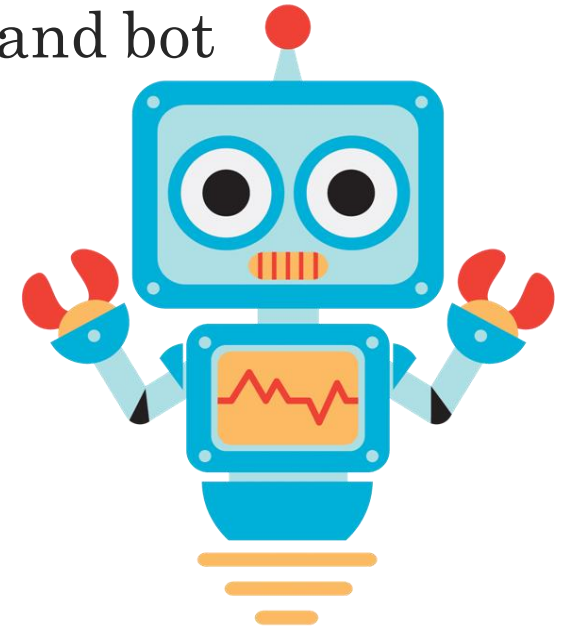
# Bots-R-Us

Team 8 Final Presentation

Adam Riffel, Carlos Acuna, Corbin Graham, Jose Medina Mani, Nhan Tran

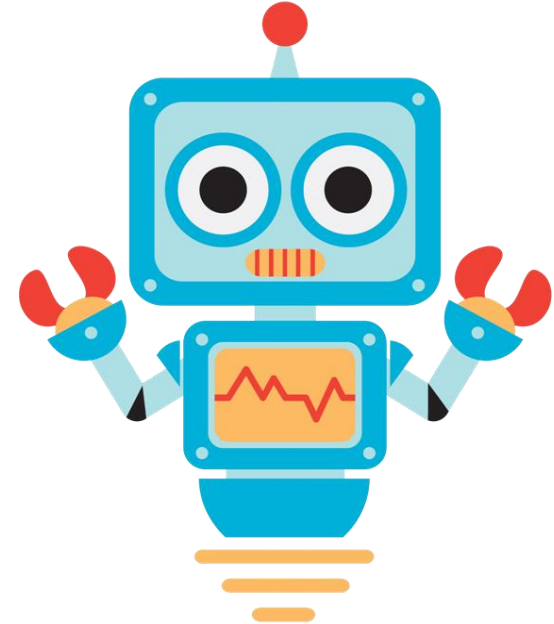
# Problem Description

- Primary Research Question
  - Is it possible to differentiate between a human and bot playing a game?
- Secondary Research Questions
  - Can we do this from a naive approach?
  - Can we identify a human or bot in real time?
  - Can our solution be generalized to multiple platforms?



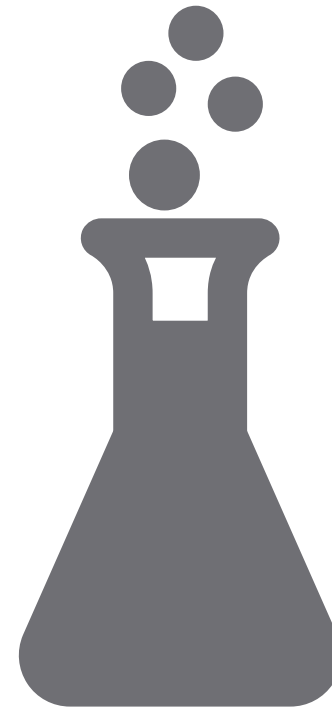
# Approach

- Proved it is possible to differentiate
- Designed framework to address each approach
  - Naïve approach & Calculated Approach
  - Online & Offline Prediction



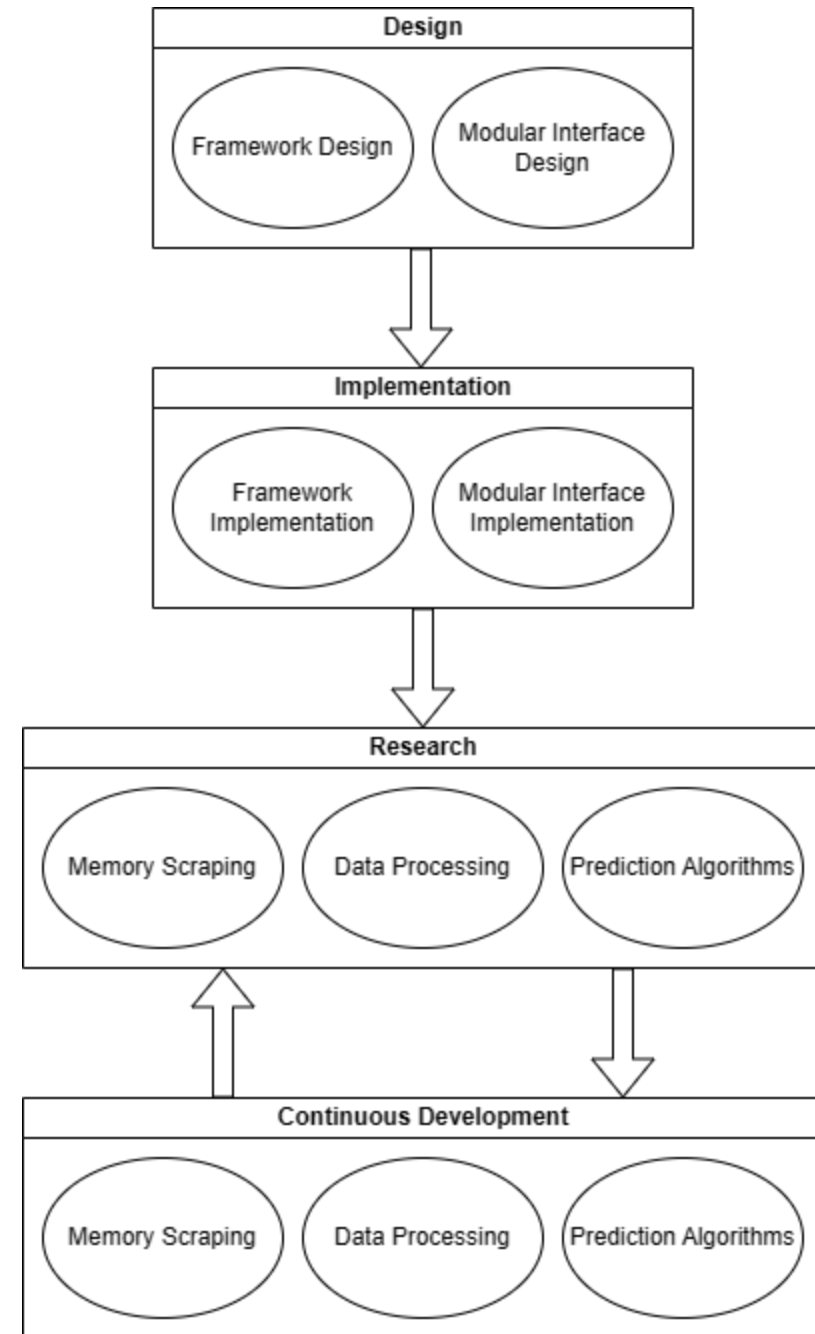
# Our Lifecycle

1. Initial Research
2. Initial Design & Implementation
3. Additional R&D



# Business Implementation Plan

1. Initial Design
2. Initial Implementation
3. Continuous Research
4. Continuous Development



# Research Findings

# Memory Scraping

- Kernel implements methods of abstraction
- Explored function hooking
- Explored system call intercepts
- Explored subprocesses
- Result: Additional Research Required

# Data Exploration

- Explored relevant features
- Explored dimension reduction
- Explored superscoring
- Result: Normalization & Feature Targeting

# Algorithmic Prediction

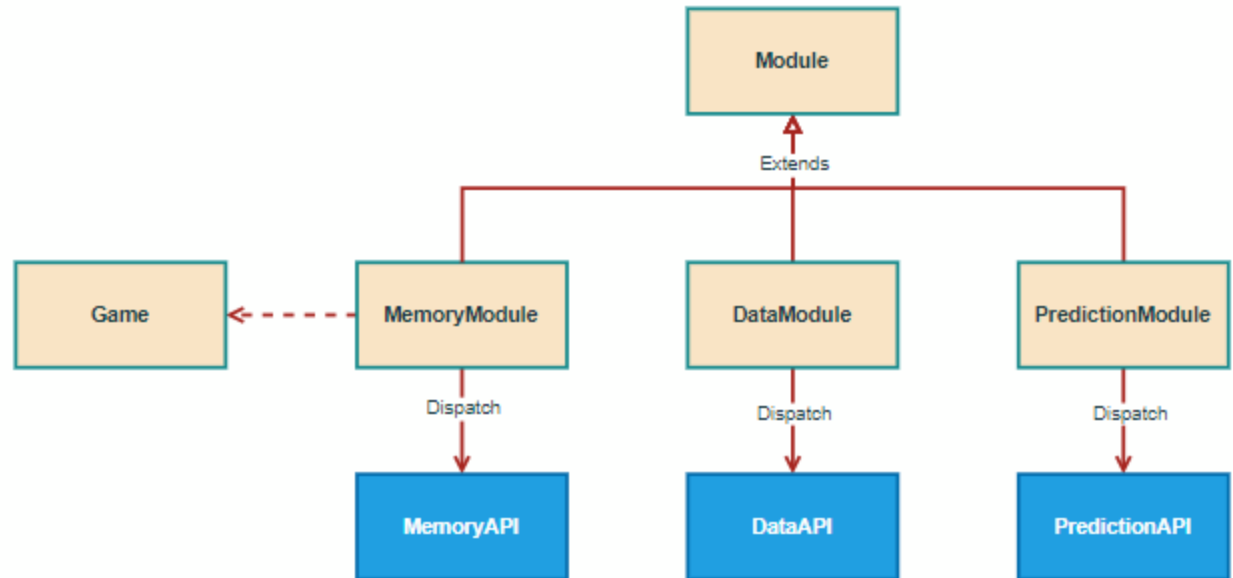
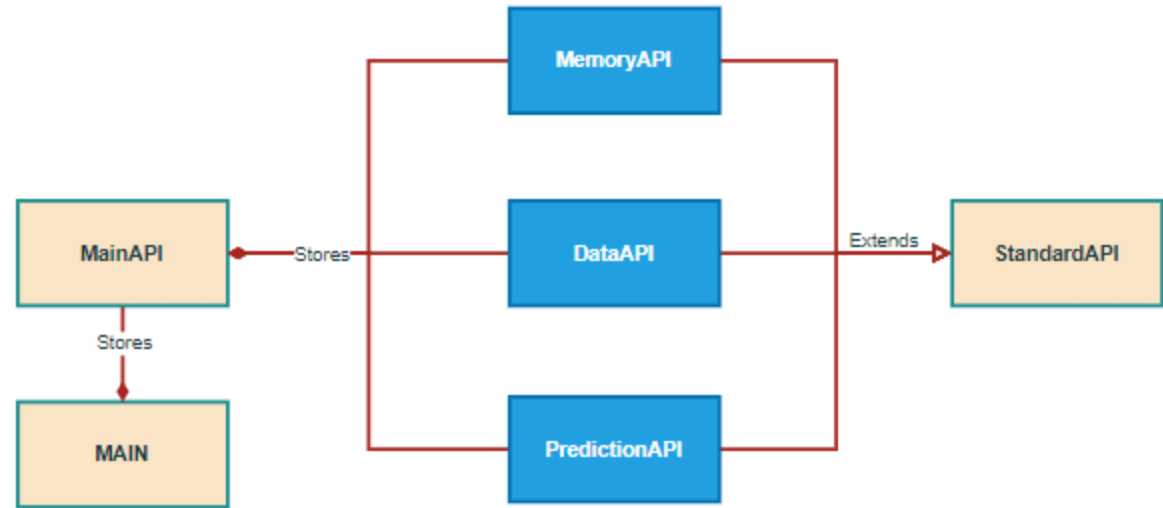
- Explored various machine learning approaches
- Explored score-based algorithms
- Result: LSTM

# Framework Design

# Design Rationale

- Modular design supports multiple approaches
- Swappable interfaces:
  - Allows for client to create and use modules of their own
- API:
  - Allows for client to access the inner-workings
- Multiprocessing supports subprocess-threading

# Simplified Architecture



# Modules

- Memory Module
  - Data collection
- Data Module
  - Data processing
- Prediction Module
  - Data prediction



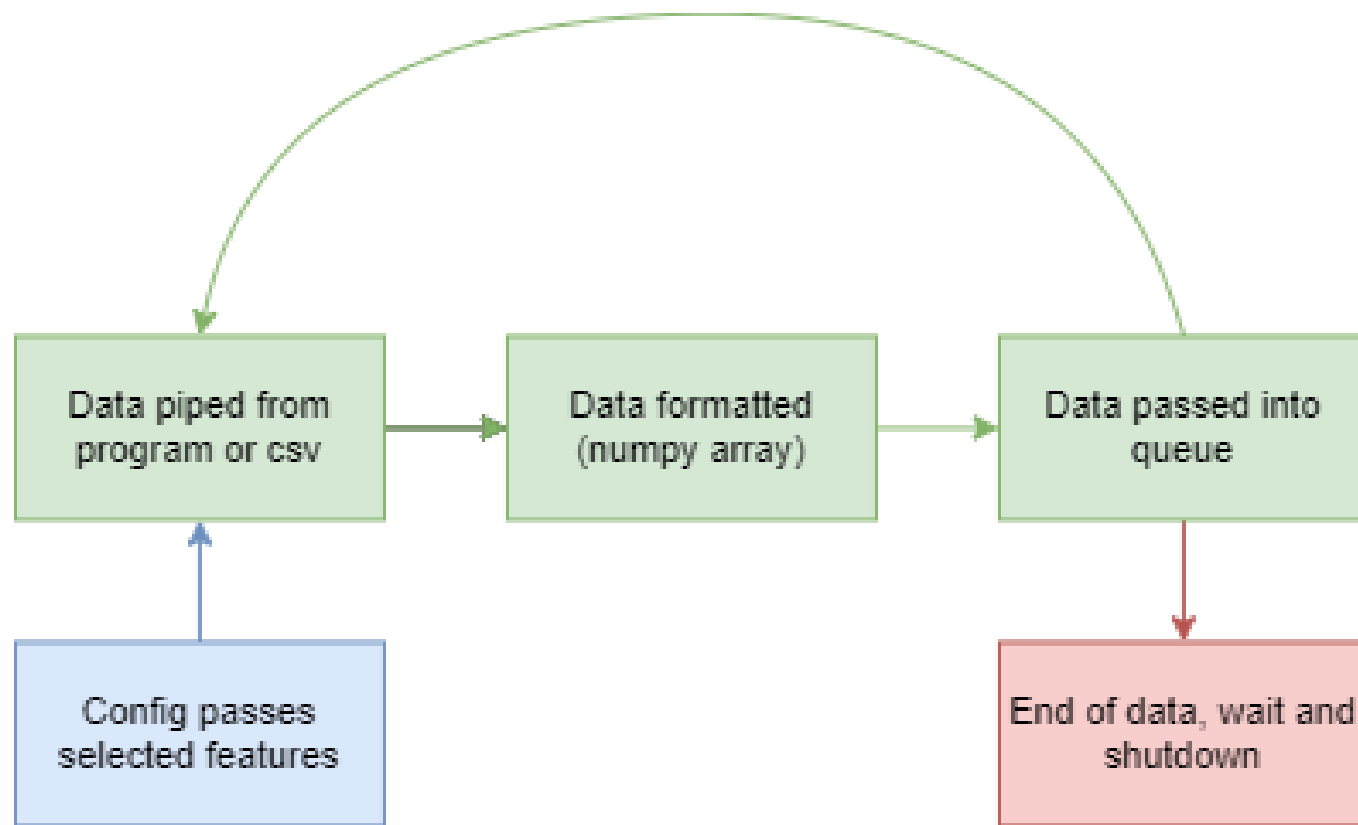
# API

- Standard API
  - Implements logging
- Main API
  - Hosts modular APIs
- Memory API
- Data API
- Prediction API
  - Hosts model API
  - Access prediction

# Memory Module

# Design & Implementation

- Initialize feature selection
- Input data through standard in (stdin)
- Read from stdin to get each row of data
- Format data for data processor
- Pass data into queue for data processor
- At end of data, wait and shutdown



# Rationale

- Bypassing the kernel too much work, read from stdin
- Config settings select features avoid unnecessary data passing
- Queue for multiprocessing
- Data runs out and queue empties to close

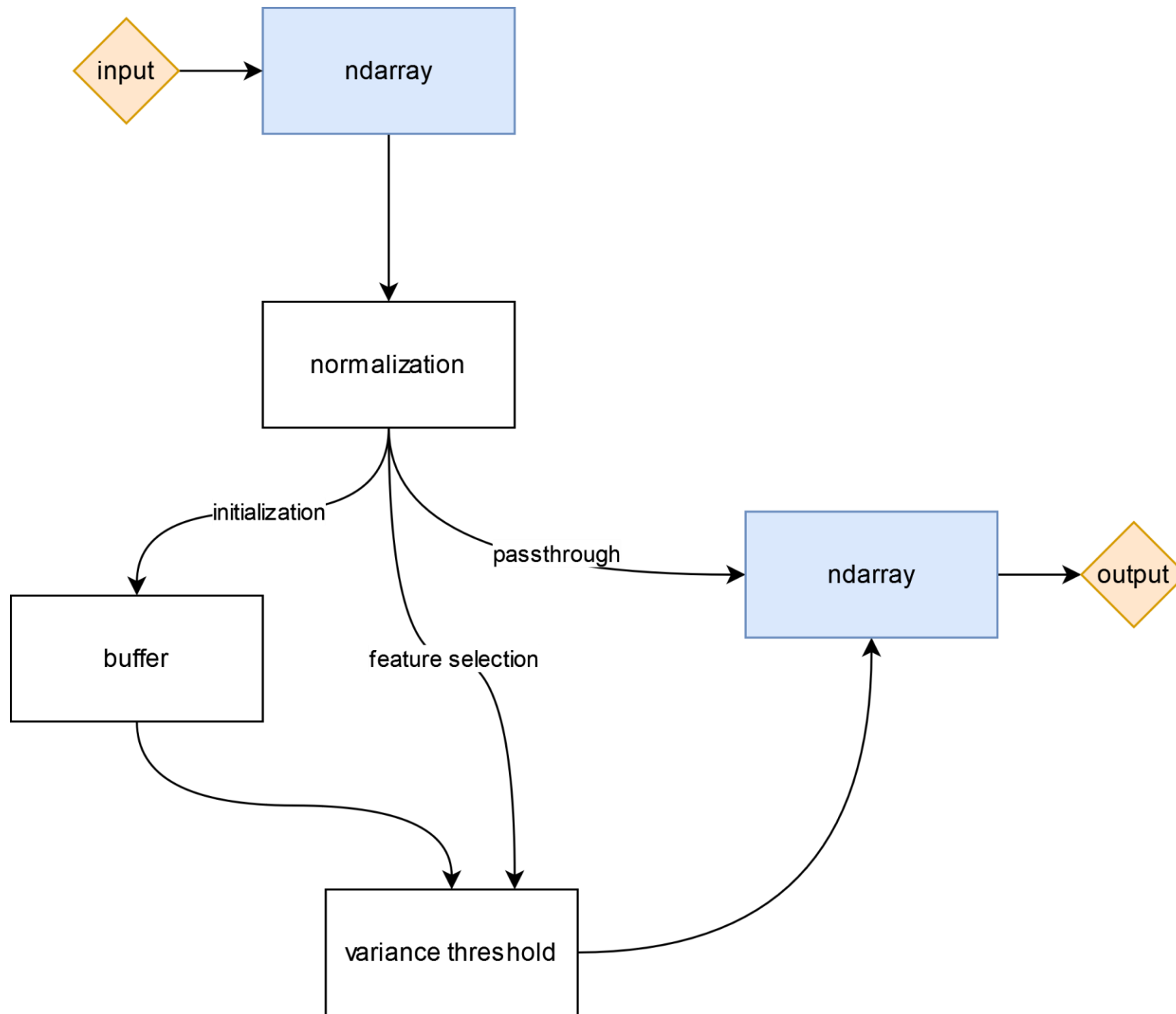
# Data Processing Module

# Rationale

- Data cleaning and normalization
- Feature selection
- Generalize use across different models

# Design & Implementation

- Takes unprocessed input from Memory Module and produces a standard formatting for Prediction Module
- Feature selection for live training, or passthrough for pretrained models with targeted features
- Variance threshold as the default plugin



## ➤ Challenges

- Account for variable input from Memory Module
- Produce standardized output for Prediction Module
- Generalizable feature selection technique across all models

# Prediction Module

# Live Training vs. Pretrained Models

- Not all models support live training easily.
- Pretrained models require less start up time

# Design

- Components
  - Prediction model
    - ONNX format
  - Model specific preprocessing
- Targeted features
  - Utilize domain expert knowledge



ONNX

# Implementation

## Random Forest Classifier

- Preliminary Research
  - Helps identify feature importance
- Model specific preprocessing
  - Data shape fitting



# Challenges

## Random Forest Classifier

- Inaccurate pretrained model
- Sequential data
- Real-time training is difficult

# Overview

## LSTM Autoencoder

- Unsupervised Learning
- Sequence to Sequence
- Encoder encodes input sequence
- Decoder reconstructs input sequence
- Anomaly Detection
- Reconstruction Loss

# Features

## LSTM Autoencoder

**FRAME, X, Y, UP, DOWN, LEFT, RIGHT, SPEED, ONGROUND**

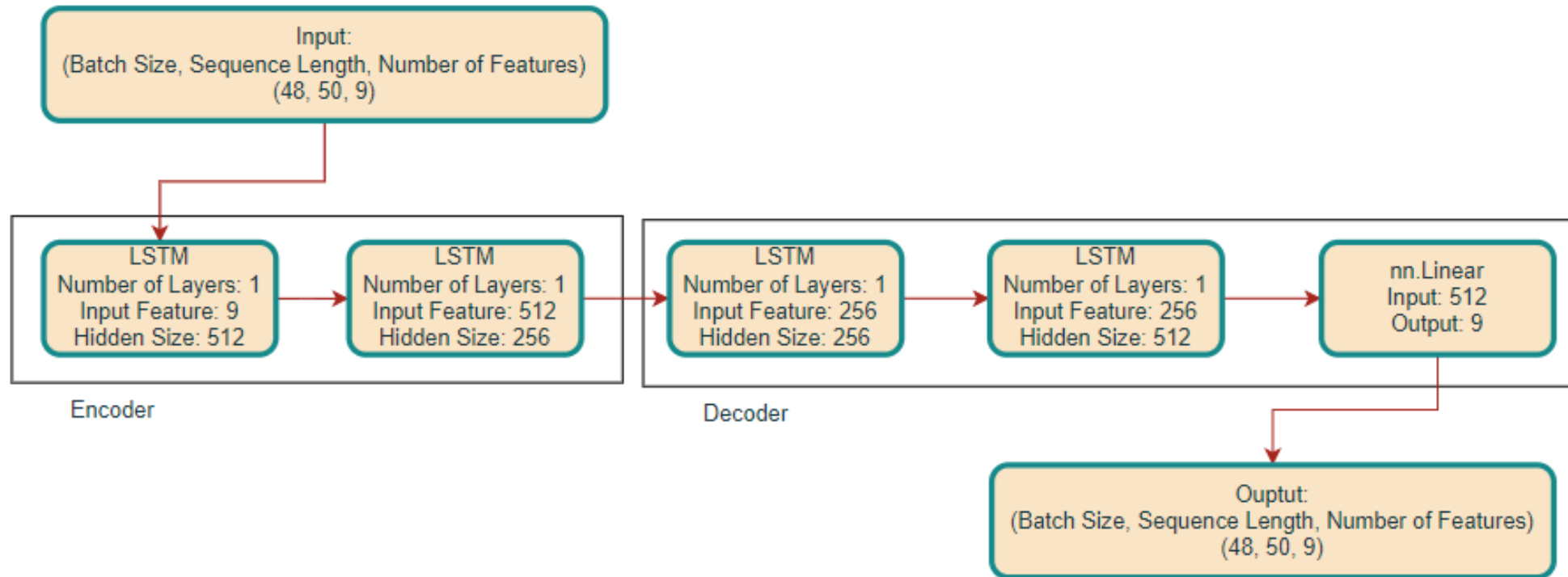
**SEQUENCE LENGTH = 50**

# Implementation

## LSTM Autoencoder

**Human data is varied**  
**Train on bot data**

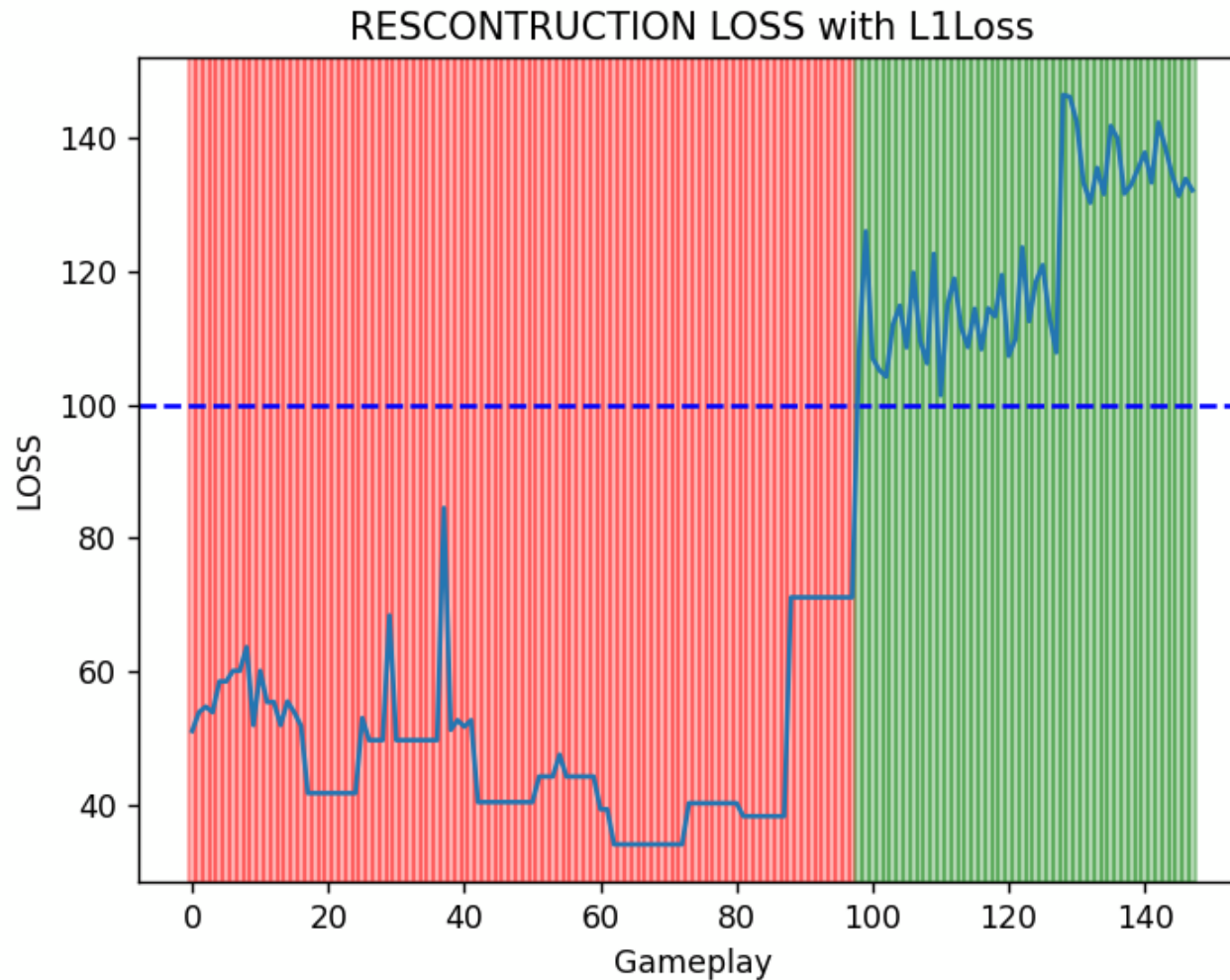
# Implementation LSTM Autoencoder



Loss: Mean Average Error (MAE)

# Result

## LSTM Autoencoder



# Framework Integration

## LSTM Autoencoder

- List of **sequence predictions** over the gameplay
- Threshold = 100
- For each sequence:
  - Reconstruction Loss below threshold: Bot, 1
  - Reconstruction Loss above threshold: Human, 0
- Final prediction: the average of predictions

# Challenges

## LSTM Autoencoder

- Limited variations of bots
- Data overfitting
- Long training time
- Environmental data

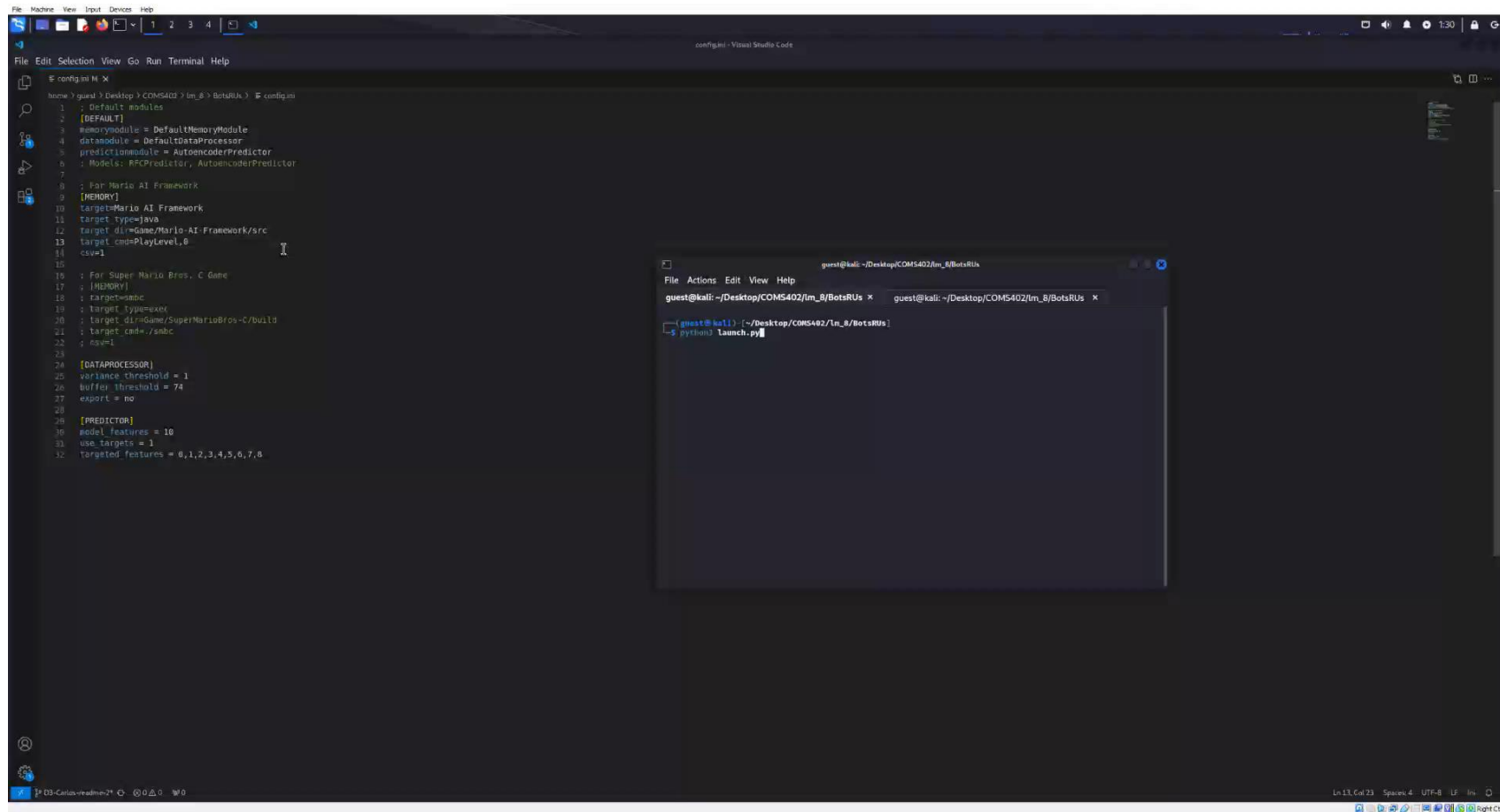
In the future:

- Autoencoder for a feature
- Segment length starting from beginning

# Teamwork Contributions

| Module         | Framework Design                              | Memory Module  | Data Module                | Prediction Module       |
|----------------|---|--|----------------------------|-------------------------|
| Team Members   | Corbin Graham                                 | Adam Riffel  | Jose Medina Mani           | Carlos Acuna, Nhan Tran |
| Research Topic | Memory Scraping                               | Prediction   | Data Exploration           |                         |
| Team Members   | Corbin Graham, Carlos Acuna, Jose Medina Mani | Carlos Acuna, Corbin Graham, Jose Medina Mani, Nhan Tran | Adam Riffel, Corbin Graham |                         |

# Demo (without GUI)



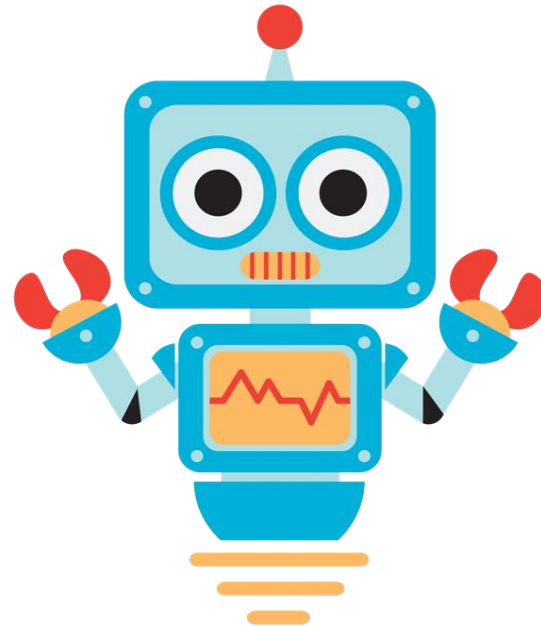
The image shows a screenshot of the Visual Studio Code editor. The main editor area displays a configuration file with the following content:

```
1 : default modules
2 : [DEFAULT]
3 : memorymodule = DefaultMemoryModule
4 : datamodule = DefaultDataProcessor
5 : predictionmodule = AutoencoderPredictor
6 : Models: RFCPredictor, AutoencoderPredictor
7
8 : For Mario AI Framework
9 : [MEMORY]
10 : target=Mario AI Framework
11 : target_type=java
12 : target_dir=Game/Mario AI Framework/src
13 : target_cmd=PlayLevel_0
14 : csv=1
15
16 : For Super Mario Bros. C Game
17 : [MEMORY]
18 : target=smbc
19 : target_type=cxx
20 : target_dir=Game/SuperMarioBros-C/build
21 : target_cmd=./smbc
22 : csv=1
23
24 : [DATAPROCESSOR]
25 : variance_threshold = 1
26 : buffer_threshold = 74
27 : export = no
28
29 : [PREDICTOR]
30 : model_features = 10
31 : use_targets = 1
32 : targeted_features = 6,1,2,3,4,5,6,7,8
```

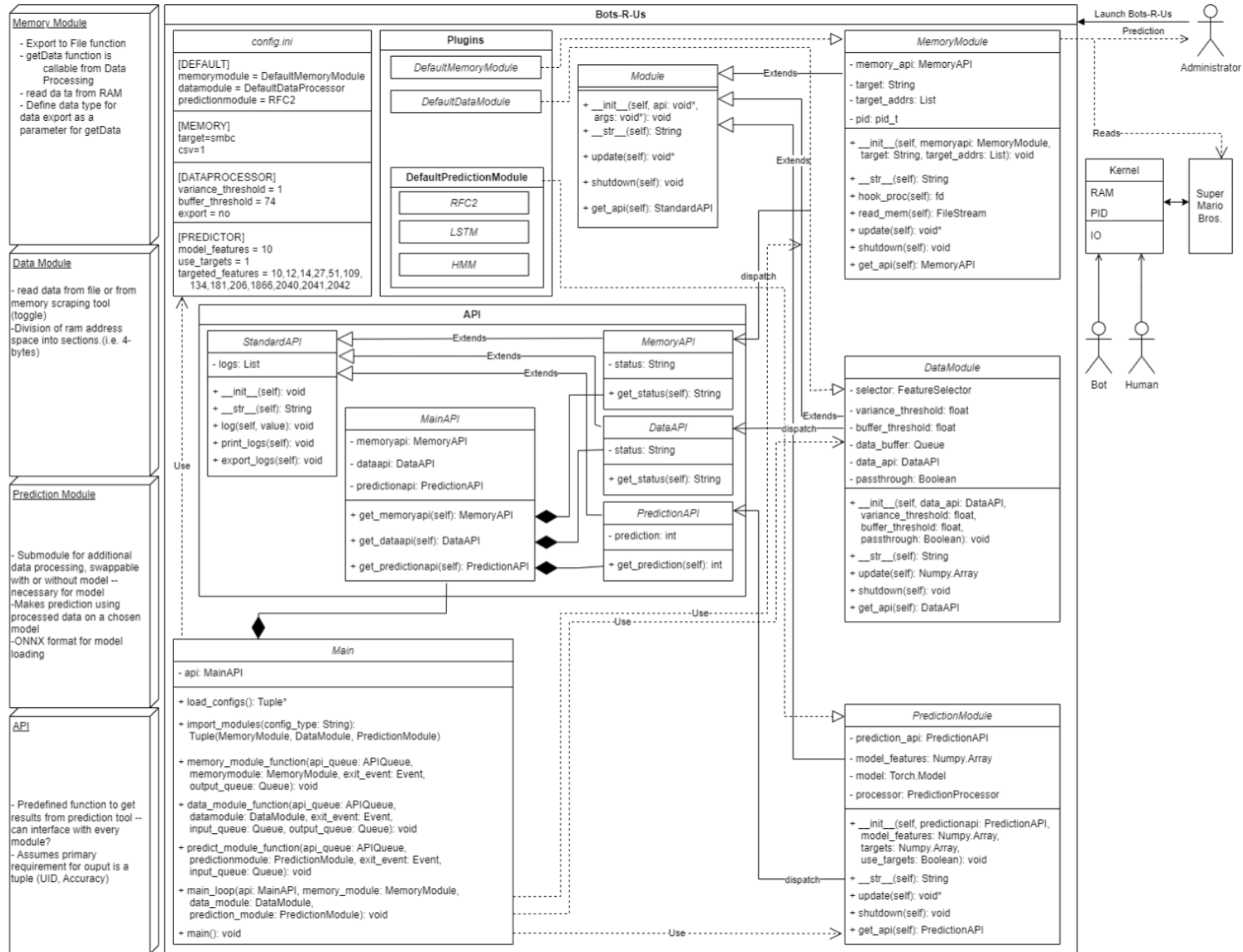
An integrated terminal window is open in the foreground, showing the following command and output:

```
guest@kali: ~/Desktop/COMS402/In_8/BotsRUs
- $ python3 launch.py
```

**Questions?**



# The Architecture



# LSTM Autoencoder prediction code

```
# SEQUENCE PREDICTION
if (reconstruction_loss < threshold):
    predictions.append(1) # BOT
else:
    predictions.append(0) # HUMAN

# AVERAGE PREDICTION OVER GAMEPLAY SO FAR
average_prediction = sum(predictions) / total_number_of_predictions
```