Project DeepLearner

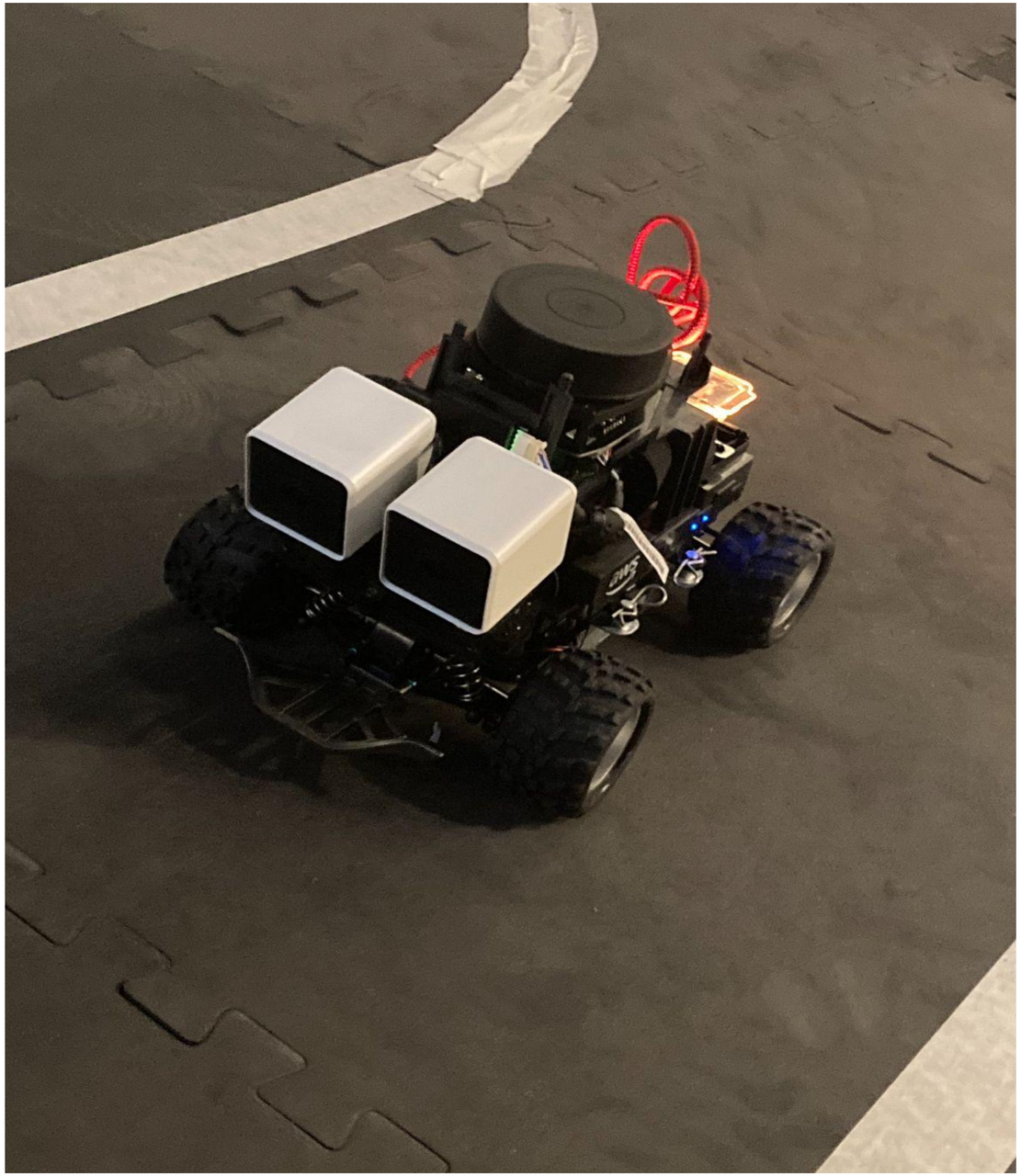
An Interactive Embedded ML Educational Experience

Senior Design May '23 - Team 10

Introduction

Machine Learning for embedded Systems

- Topic for our project: machine learning for embedded systems
- Purpose is expanding ISU curriculum
- Early searches led us to explore the DeepRacer by AWS.
- DeepRacer is a great platform for learning about reinforcement learning without getting bogged down in details.



What is the AWS DeepRacer

An educational tool for machine learning.

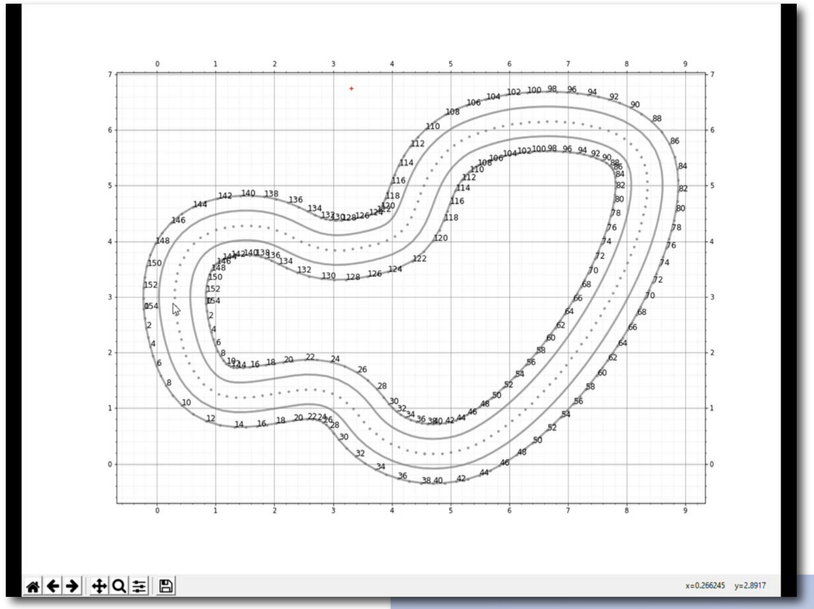
- The DeepRacer is an electric car built specifically for autonomous driving.
- It teaches students about reinforcement learning by letting them easily upload a reward function and train machine learning models.
- Main focus on the DeepRacer League but also allows for custom projects.
- DeepRacer also has a very large, flourishing community with lots of open source software.

Adapting DeepRacer for a classroom

Adapting the platform for an educational environment created a few tasks:

- Develop labs/activities involving machine learning
- Create a platform separate from AWS services for training models on local machines
- Create a user friendly interface

Implementation and Architecture



Simulated DeepRacer Training on the DeepLearner Platform

A Tale of Two Sides

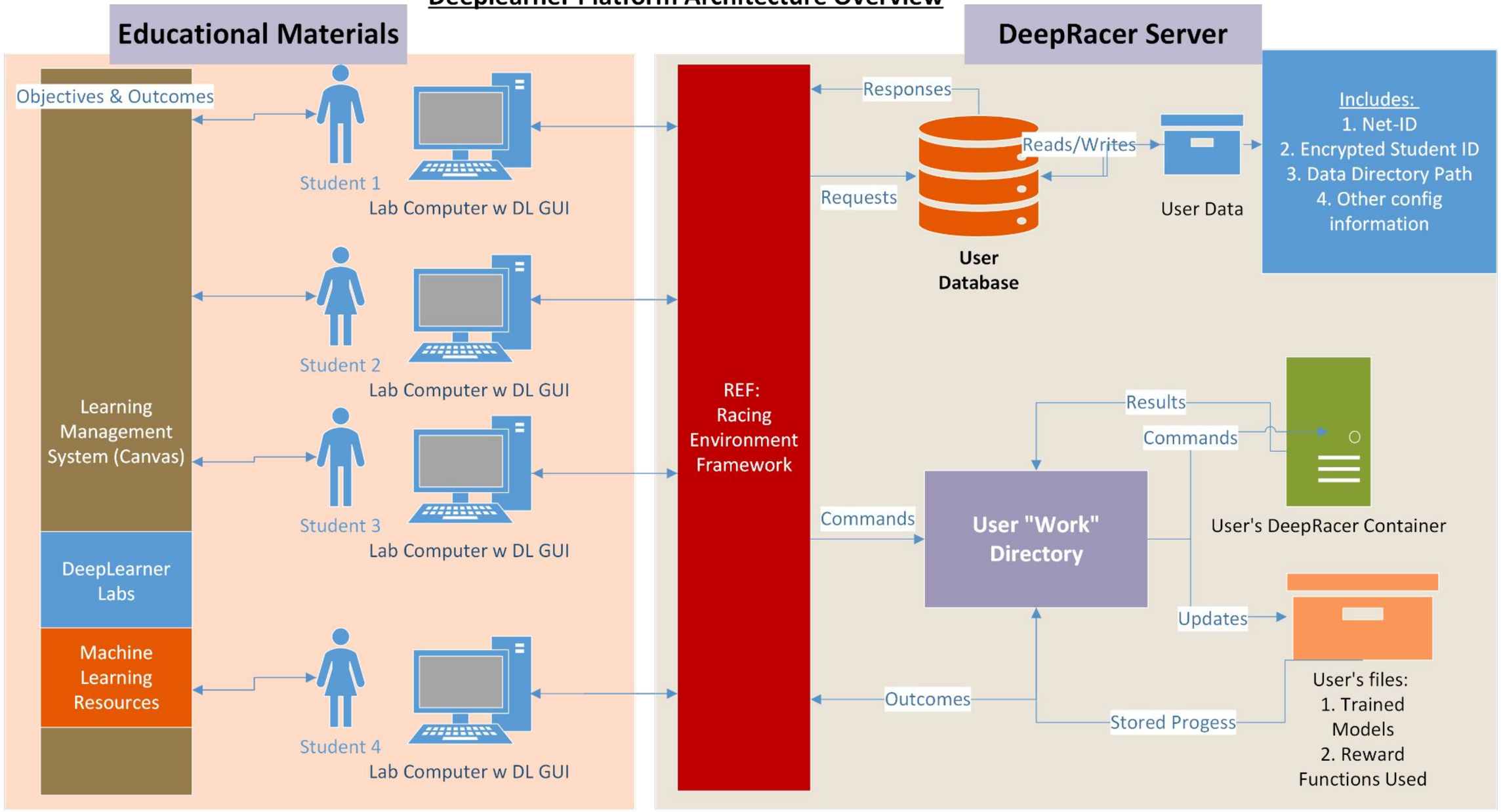
The Education Implementation

- Needs to serve as introduction to ML and related topics for students.
- Various labs centered on our DeepRacer Environment Platform.

The Environment Implementation

- Needs to provide an environment to train, evaluate, and race DeepRacer models.
- A platform for students to implement DeepLearner labs.

Deeplearner Platform Architecture Overview



DeepLearner Labs

Educational Implementation

- Introduce Required ML concepts
  - Reward Function, parameter estimation, etc.
- Familiarize students with the DeepRacer bot
- Create reward function
  - Observe effect of training time
- Encourage additional interaction with Student vs Student race
- Interact with embedded programming

DeepRacer Lab 1/Introduction Document

Introducing the DeepRacer.

Introduction

The AWS DeepRacer is an electric vehicle designed specifically with machine learning capabilities in mind. The car is intended for use on a physical track and relies on its cameras and a network connection to a computer with a dedicated GPU. Despite looking like an RC car, the DeepRacer is intended to run autonomously via a generated algorithm. The algorithm that is used to run autonomously is generated by a special subset of **machine learning** called **reinforcement learning** - more on that later.

DeepRacer vs. CyBot

The DeepRacer differs greatly from the CyBot. Mostly due to the processing power and sensors available to each machine. The DeepRacer relies on dual cameras as well as a lidar sensor and gyroscope. The DeepRacer also has wifi capabilities and a lot more processing power than the CyBot, having an Intel Atom Processor and 4 GB of RAM. The CyBot also has wifi capabilities but only the processing capabilities available to the Tiva C Series Launchpad microcontroller- an ARM Cortex M4 CPU and 256 KB of memory. The CyBot also has light sensors, left and right bump sensors, a ping sensor, and an infrared sensor. The DeepRacer needs a lot more processing power because of how complex its algorithms are. It needs to process data from its sensors and send it over wifi to a more powerful computer for processing. Based on this data it creates a new model for navigating the environment that is stored on the DeepRacer for future decision making.

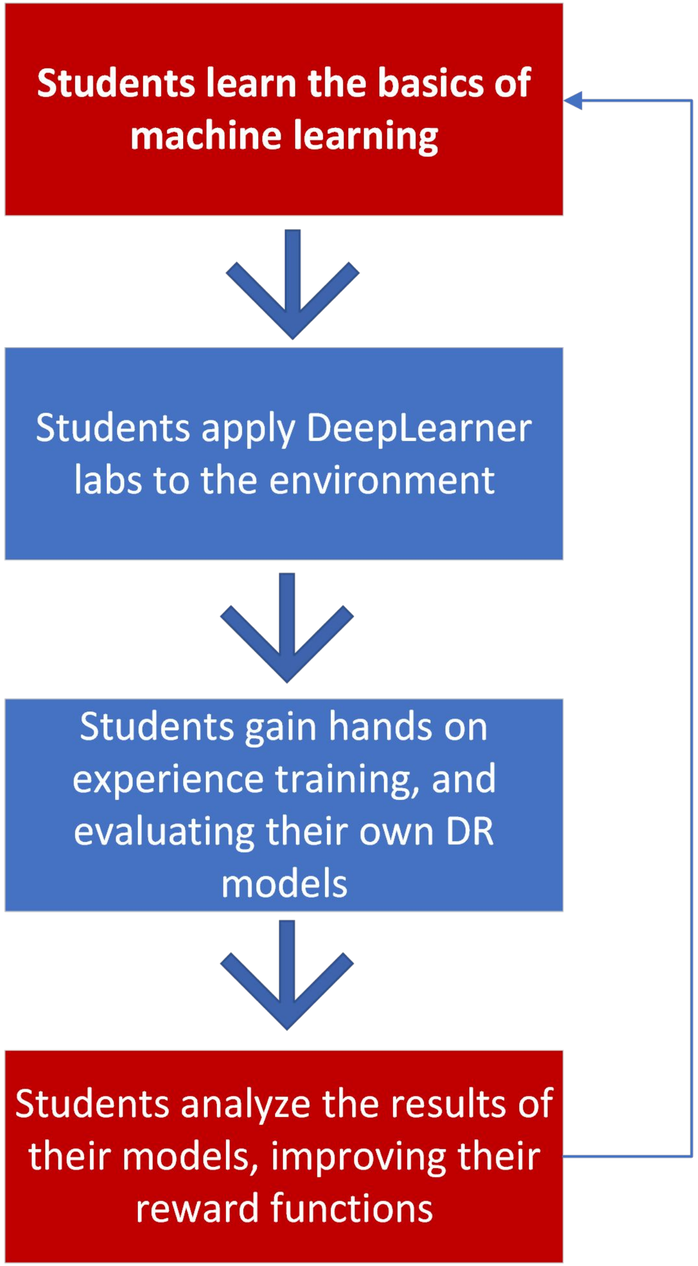
Concept

The DeepLearner Platform

An Interactive, Educational Embedded Machine Learning Platform for Iowa State students based on the AWS DeepRacer. How do we do this without AWS?

Components

- **Front-end GUI:** A graphical user interface to provide a user friendly way for students to interact with the DeepRacer sandbox.
- **DeepRacer Container:** A virtualized environment that contains all necessary DeepRacer components, including the simulation environment and training/evaluation tools.
- **REF (Racing Environment Framework):** An application that handles user requests for training/evaluation. Provides an interface between the GUI and the DeepRacer container.



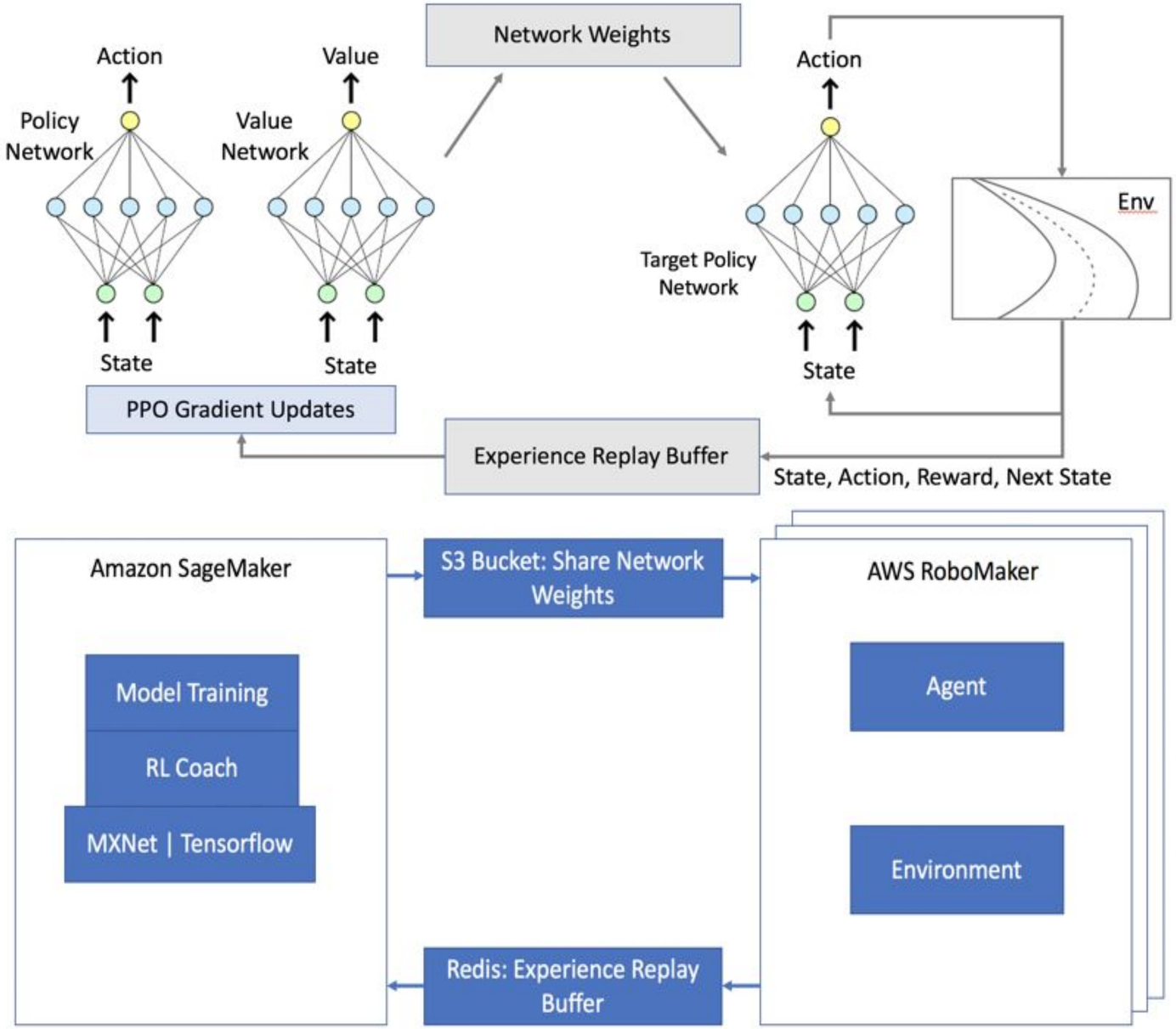
DeepRacer Container

Inspired by AWS's official implementation.

- Combines two open source projects from the DeepRacer Community.
- Scripts are used to connect the two projects, creating an environment.

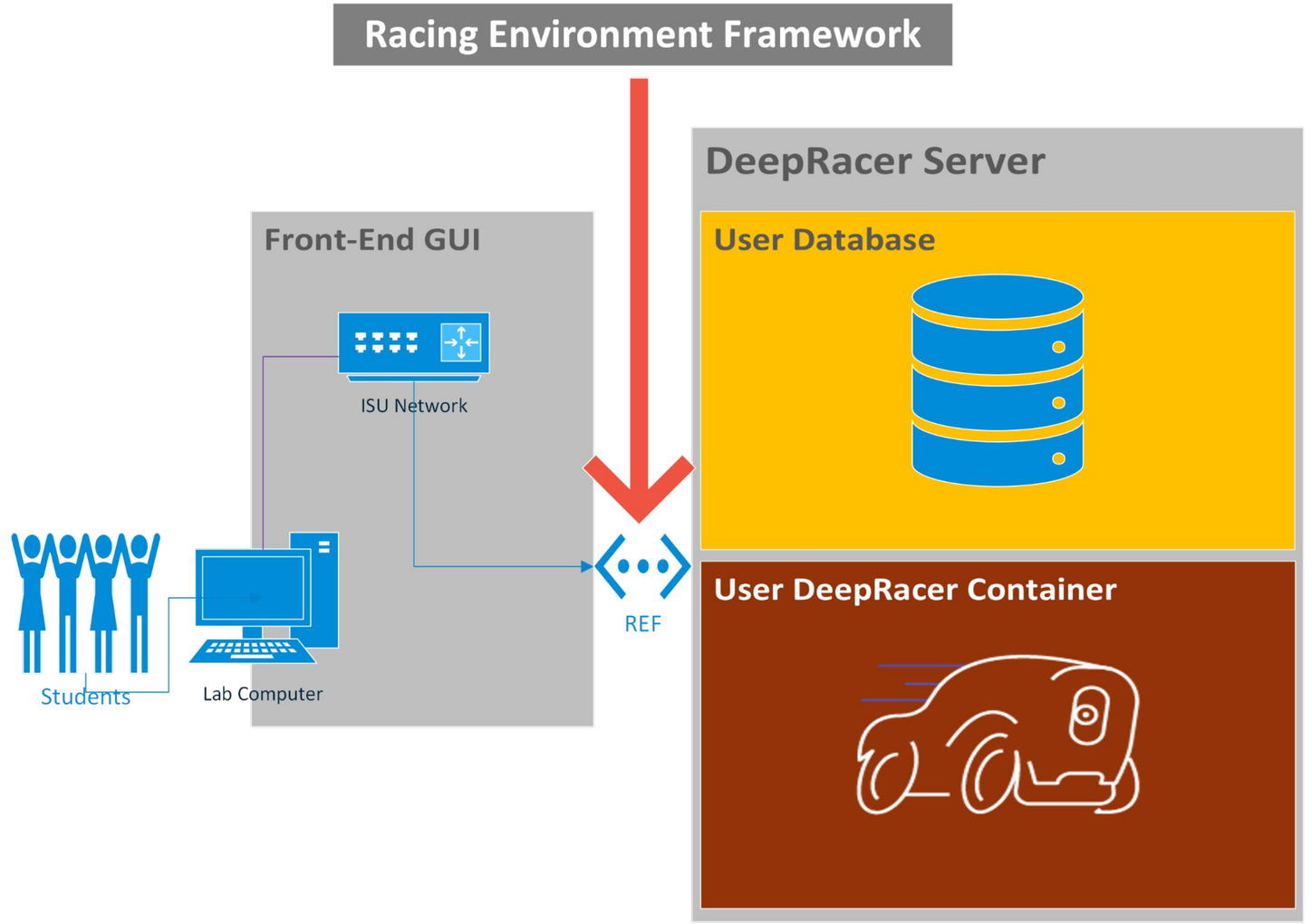
Simulates a machine learning environment.

- The Sagemaker service handles machine learning and training/updating the agent.
- The Robomaker service simulates the environment and the robot.
- Local minio instance provides object storage.



Racing Environment Framework

- The abstraction layer between our DR server and the client (GUI).
- Controls and interfaces to the DeepRacer container.
- Handles client's requests.
- Provides authentication, data access, and ease-of-use for our students.
- Built on the Django Framework.

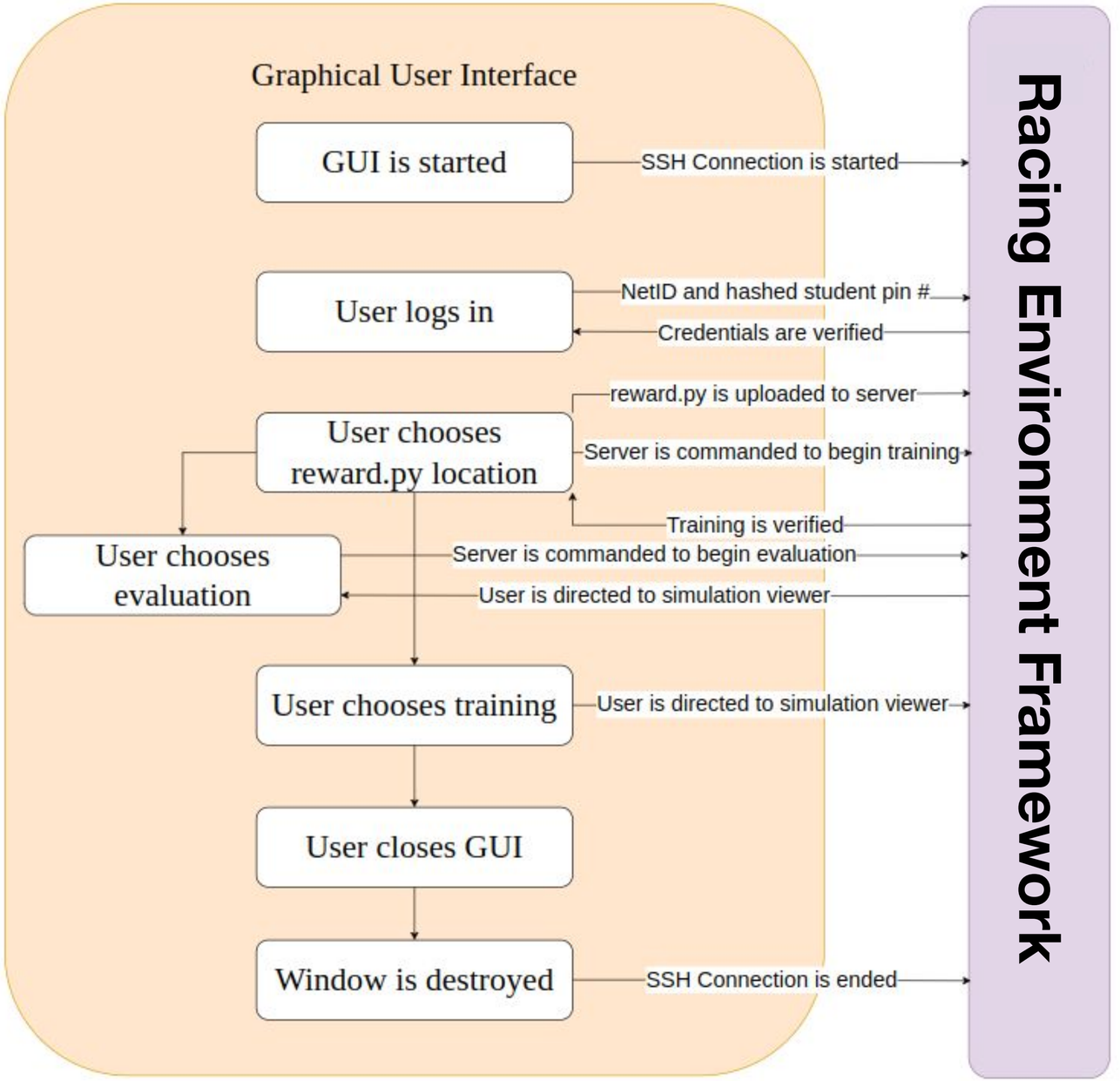


The GUI

The frontend exists for easier use and file transfer

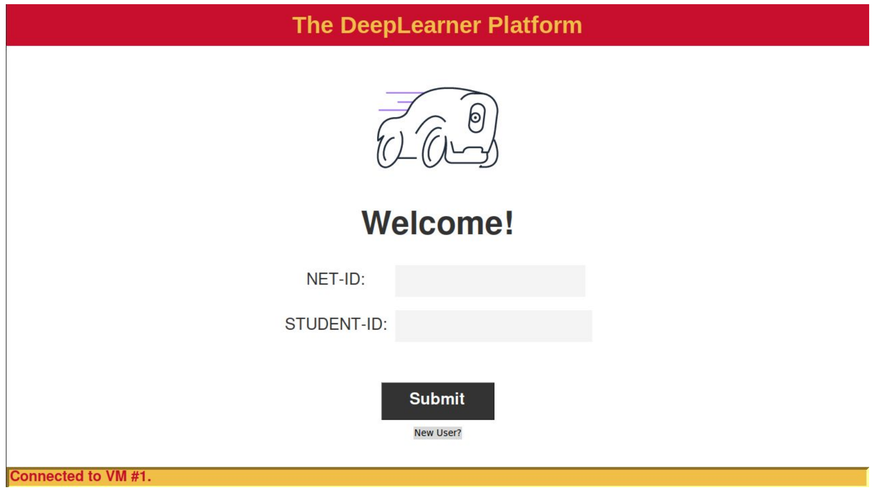
GUI provides services that we enjoyed from AWS Deepracer

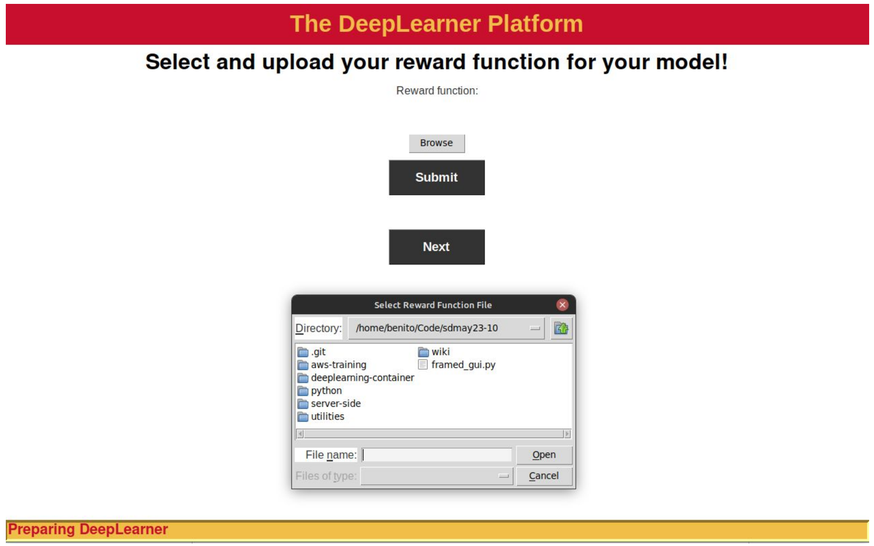
- Ability to train and evaluate DeepRacer models in a simulated environment
- Upload custom reward functions and download trained models

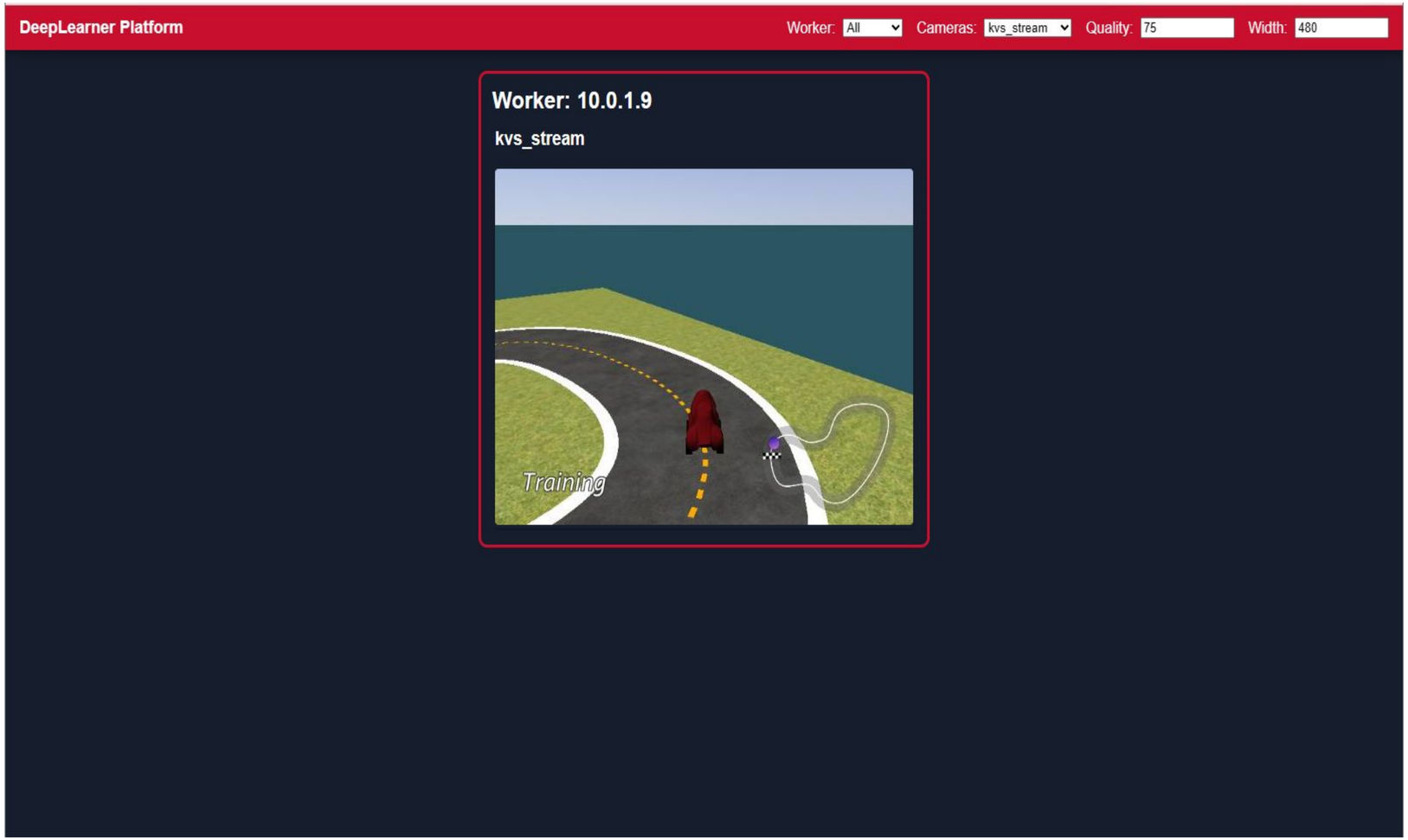


Work Accomplishments

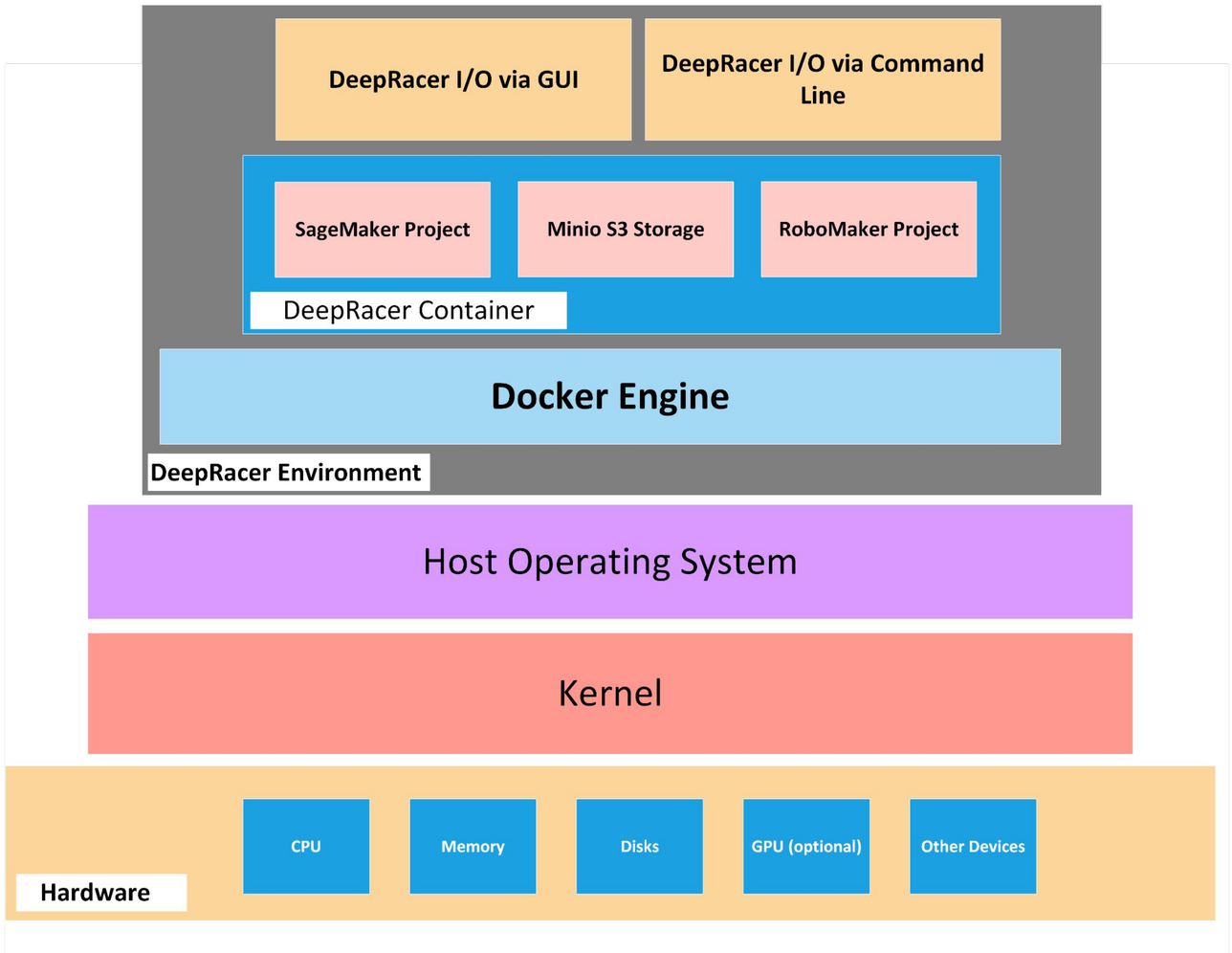
A User-Friendly Experience





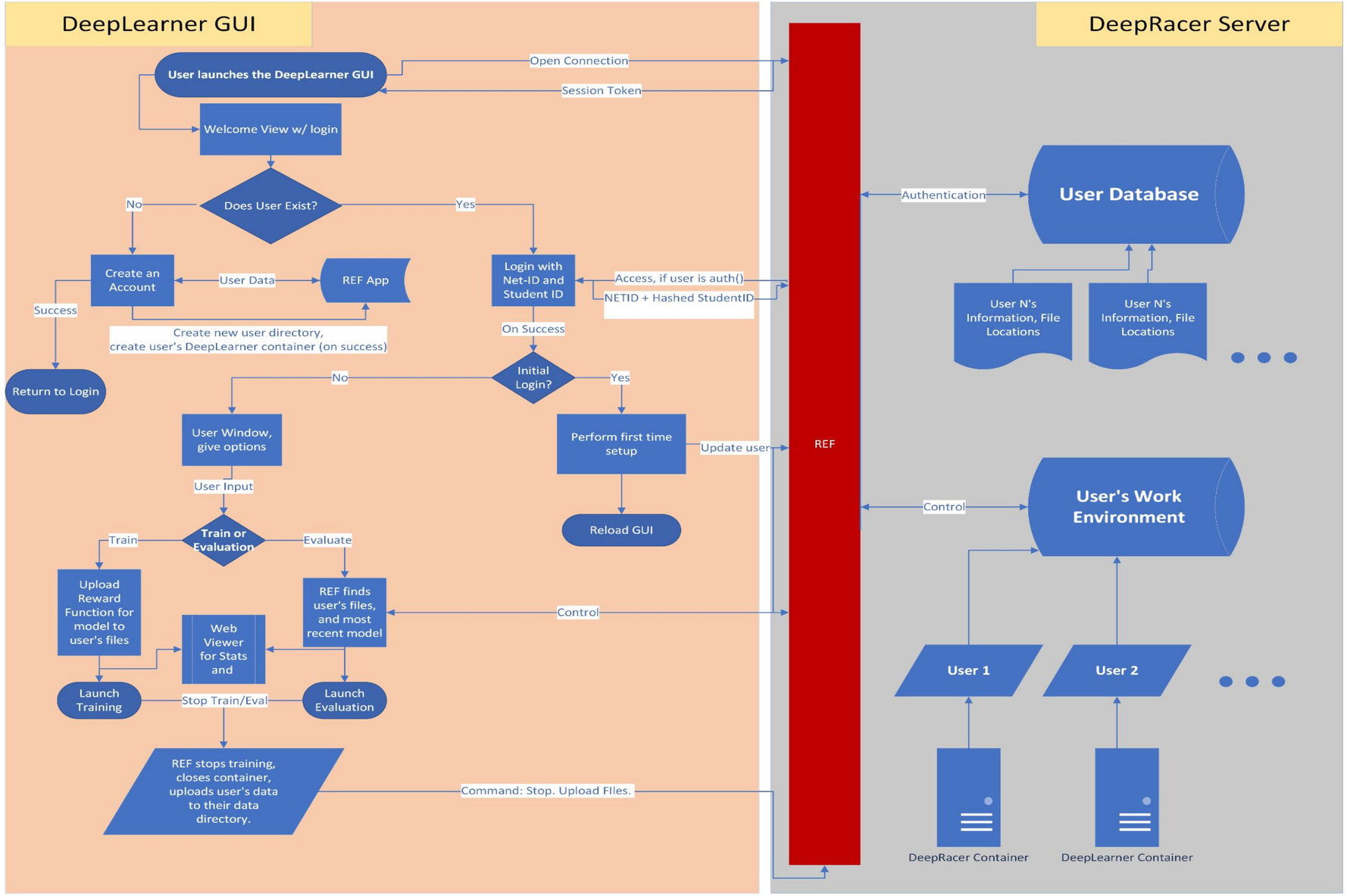


DeepRacer Container



- Houses all components necessary to train, evaluate, and race the DeepRacer virtually.
- Simulates AWS services, locally - avoiding costs.
- Can be installed on a local machine, or on a server.
- Implemented using Docker for dependencies.

Racing Environment Framework



Lab Documents

Lab 1: Introduction to Machine Learning

- Introduce Required ML concepts
  - Reward Function, parameter estimation, etc.
- Familiarize students with the DeepRacer bot
- Create reward function
  - Observe effect of training time
- Encourage additional interaction with Student vs Student race

DeepRacer Lab 1/Introduction Document

Introducing the DeepRacer.

Introduction

The AWS DeepRacer is an electric vehicle designed specifically with machine learning capabilities in mind. The car is intended for use on a physical track and relies on its cameras and a network connection to a computer with a dedicated GPU. Despite looking like an RC car, the DeepRacer is intended to run autonomously via a generated algorithm. The algorithm that is used to run autonomously is generated by a special subset of **machine learning** called **reinforcement learning** - more on that later.

DeepRacer vs. CyBot

The DeepRacer differs greatly from the CyBot. Mostly due to the processing power and sensors available to each machine. The DeepRacer relies on dual cameras as well as a lidar sensor and gyroscope. The DeepRacer also has wifi capabilities and a lot more processing power than the CyBot, having an Intel Atom Processor and 4 GB of RAM. The CyBot also has wifi capabilities but only the processing capabilities available to the Tiva C Series Launchpad microcontroller- an ARM Cortex M4 CPU and 256 KB of memory. The CyBot also has light sensors, left and right bump sensors, a ping sensor, and an infrared sensor. The DeepRacer needs a lot more processing power because of how complex its algorithms are. It needs to process data from its sensors and send it over wifi to a more powerful computer for processing. Based on this data it creates a new model for navigating the environment that is stored on the DeepRacer for future decision making.

Lab Documents

Lab 2: Inside AWS DeepRacer

- Incorporate more embedded ideas
  - PWM.sh file, GPIO pin identification, etc
- Incorporate Amazon-Recommended “Follow the Leader” project
  - Well documented and tested
- Focus more on physical bot

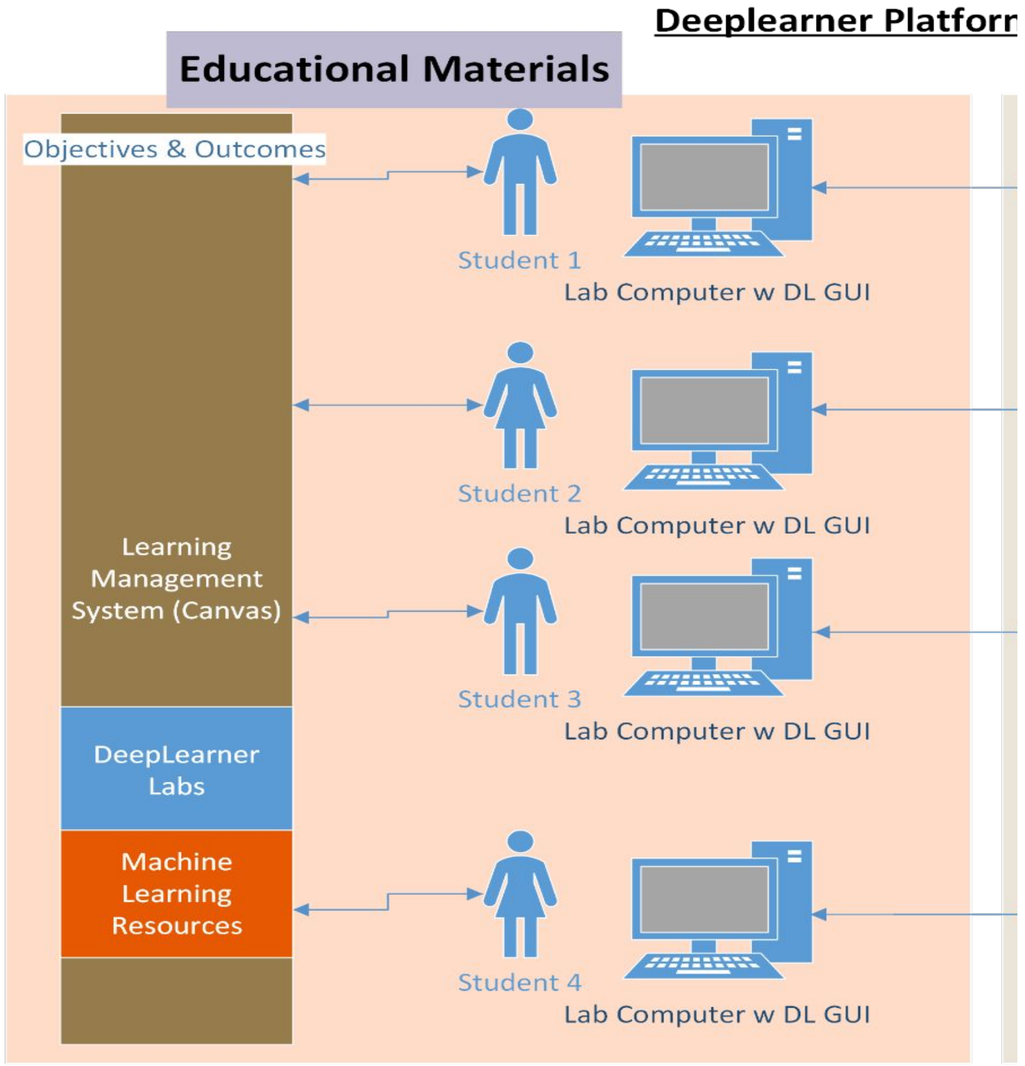
```
Init motor and servo power and duty cycle:
  sudo ./pwm.sh enable

Disable motor and servo power
  sudo ./pwm.sh disable

Motor control Commands:
  sudo ./pwm.sh motor fw             //Forward
  sudo ./pwm.sh motor fw slow        //Slow Forward
  sudo ./pwm.sh motor fw max         //Max Forward
  sudo ./pwm.sh motor stop           //Stop
  sudo ./pwm.sh motor bw             //Backward
  sudo ./pwm.sh motor bw slow         //Slow Backward
  sudo ./pwm.sh motor bw max         //Max Backward
  sudo ./pwm.sh motor [duty_cycle]  //Custom motor duty cycle

Servo control Commands:
  sudo ./pwm.sh servo left            //Turn Left
  sudo ./pwm.sh servo mid             //Middle
  sudo ./pwm.sh servo right           //Turn Right
  sudo ./pwm.sh servo [duty_cycle]    //Custom ESC duty cycle
```

Primary Contributions

Embedded, Education, Research, and Development Team

Jazz & Caleb

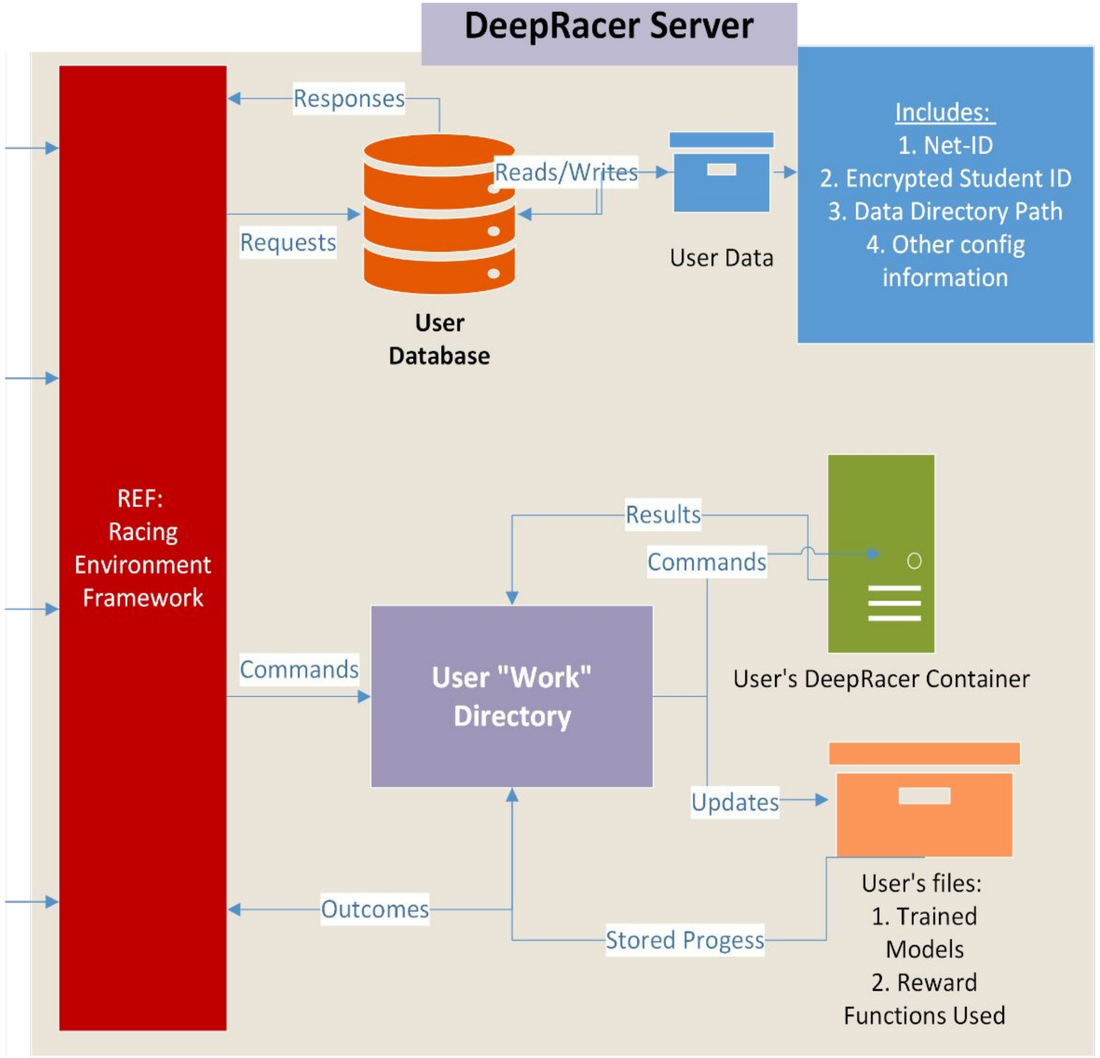
- Prelabs & Labs
- Research materials
  - AWS documentation
  - ROS Documentation
  - Source Code
- Physical Track



Machine Learning, Platform, and Cybersecurity Team

Jose Carlos Garcia & Benito Moeckly

- Front-End GUI
- User Experience
- Racing Environment Framework
- DeepRacer Container



Challenges and Solutions

Educational Challenges

How to keep the information understandable?

- Start basic and give hands on examples
- Provide extra resources for learning

How to keep students engaged and entertained?

- Keep lab goals competitive to encourage competition
- Give clear goals for students to work towards

How do we tie machine learning to embedded systems?

- Explore the inner files of the DeepRacer
- Bridge the gap between what the model does and how the robot uses it

Platform Challenges

Using AWS is expensive and not practical for ~300 students yearly ($42k).

- **Solution:** train our robots locally. Similar solutions, with other purposes have been done in the past [DeepRacer Container].

Students can easily break the newly constructed container.

- **Solution:** Take control out of their hands, give them the same luxuries as AWS would. Also, implement controls to limit access. [front-end GUI and REF].

How do we give students their own environment to work on, independently from other students?

- **Solution:** Give each student their own container, directory, and provide authentication. [REF and base DeepRacer Container]

Challenge: Lack of Access to Underlying Hardware

The DonkeyCar

- Open-source Raspberry Pi Based platform
- Behavioral Cloning vs Reinforcement learning
  - Reinforcement learning more applicable
- Cheaper at MSRP
  - Supply Issues negate
- Well Documented
  - Smaller Community

https://www.donkeycar.com/

Future Work

Custom Robot in our Environment

Create a custom robot from scratch for use in our ML environment.

Create dedicated database for REF

As the platform goes; so will it’s data needs.

Future Labs

There are many additional project ideas that can be implemented in the physical world, as opposed to virtually.

Conclusion

Closing Thoughts

Machine Learning is the future and we need to catch up - Benito Moeckly

Especially with programs like ChatGPT, we need to learn how they work to stay up to date - Caleb DeBoef

There's a lot of room for our application to grow and fulfill the educational needs of students - Jazz Jacobus

Mind the gap -> Bridge the gap

In the information age, we need better accessibility to platforms like the DeepRacer - Jose Carlos Garcia

What better way than for us to go open source, too?

The AWS DeepRacer Robot

On your mark..

Thank you for your time.
At this time we'll take questions!

Supplementary Slides

Benito Moeckly

ML & Cybersecurity Team

- Contributor to GUI design and implementation
  - SSH, file transfer, front-to-backend communication
  - Designed multiple frames
  - Created initial prototype of GUI functionality and aesthetics
- Contributor to lab document and manual
  - Lab 1 explanations of DeepRacer and reinforcement learning
  - User manual
- Helped build track for physical demo of the DeepRacer
- Contributed to REF design
- Contributor to design documents, presentation, and poster

Caleb DeBoef

Electrical Systems

- Helped contribute to Lab documents
  - Specifically prelabs and ML background information
  - Created Prelab.c file to explain parameter estimation
  - Found related research papers to provide further information to students
- Researched DonkeyCar alternative
  - Found which car configuration would work should it be implemented in the future
- Helped order, plan, and construct the physical track
- Contributed to design documents, presentation, and poster

Jose Carlos Garcia

ML & Cybersecurity Team

Designed and implemented DeepRacer container environment.

- Performed research with members of the DeepRacer community.
- Developed scripts, Docker container, and brought together other open source projects to produce our environment.
- Implemented “nuclear-reactor rod” design for students.

Designed and implemented the Racing Environment Framework (REF).

- Sketched design for GUI interaction with the container.
- Initially, developed server app to stop rogue training sessions.
- Designed and developed authentication methods, user access, and sessions.
- Implemented other control methods within the framework, such as initial container setup, etc.

Contributed to the design and development of the front-end GUI.

- Developed current “frame” design of the GUI.
- Interfaced the front-end GUI with the Racing Environment Framework.
- Implemented user-friendly transitions around network related checks, operations, etc.
- Oversaw cybersecurity risks (built-in vs bolted-on).

Jazzlyn Jacobus

Embedded Systems

- Helped contribute to Lab documents
  - Wrote Inside AWS DeepRacer Lab
  - Helped write Intro Lab
  - Researched related materials
- Researched DeepRacer Evo hardware and connections for lab use and created diagram
  - created small bash script to demonstrate embedded programming
- Contributed to design documents, presentation, and poster
- Helped build track for physical demo of the DeepRacer