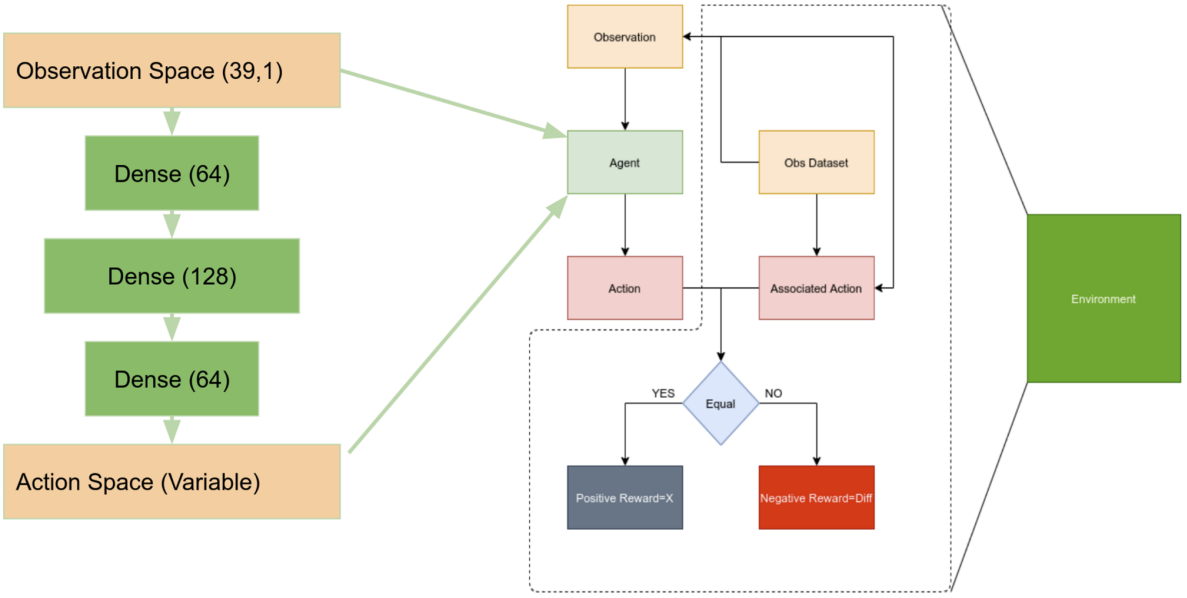
Autonomous Greenhouse Control Descriptions

Team Parshuram: Kaushal, Arpit, Vyoma, Pathik

Objective: The main idea here is how to understand and automatically control the quality of cherry tomatoes by considering different parameters like temperature, air, water supply, light, etc.

Formulating an approach and Solving the base problem



This image represents the approach that we have followed to solve the autonomous greenhouse problem.

We had data of 5 teams that competed in the AGC Challenge. Out of this data, we extracted some features from the greenhouse. (These came from different types of sensors put into the greenhouse such as light, moisture, air etc.). Along with features that were observable we also had the data of the climate settings that these teams used. For example how much water was provided to the plants in the greenhouse or what intensity of white light was kept. This way we created a dataset that has observation and correct actions to formulate the problem in terms of reinforcement learning. (assuming that the settings used by teams are the smart choice). As reinforcement learning deals with an agent exploring an environment and learning from rewards/penalties it receives. We came up with an approach to creating an environment based on the data that we had collected [Shown in the Image].

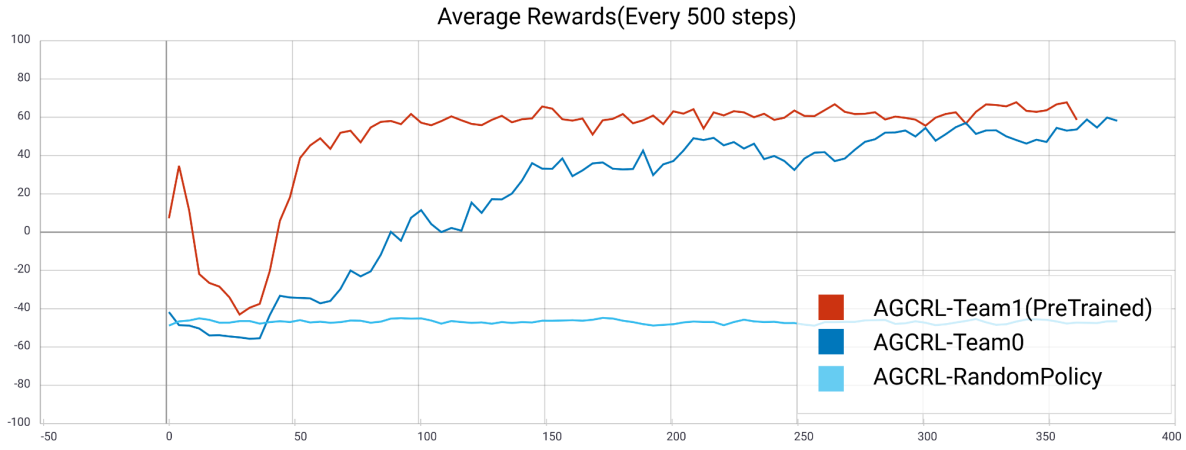
To give some context, We had around 47000 rows of observation data for each team which was collected over a time of 12 weeks every 5 mins. We used this data in the following way:

- When an environment is initialised, it starts with row 1 of team 1.
- This observation is fed to the agent and it predicts something.
- If the predicted action is equal to the action from our dataset, we give the agent a positive reward else we give it a penalty which a difference between the predicted and real value. (This way we ensure that even if the prediction arent exact they get closer to real value.) [This works here as the actions arent independent]
- Each teams data is one episode of this environment. That is
    - An episode contains 47000 steps(One episode is basically training the data on one team's actions)
    - We have 5 teams, so we can train the agent for a total of 5 episodes.

The agent we used here is a simple DQN agent that uses a deep neural network of Dense layers with 3 hidden layers whose specifications have been specified in the diagram. To use this agent we had to discretize our action space.

The above-mentioned approach can be used in any problem where solving the overall problem makes more sense using reinforcement learning but we have data that looks like it could be used in a supervised context.

We used libraries such as Pandas, Numpy, Tensorflow and created and trained the above the shown model. Once we trained this and got some satisfactory results we moved towards creating a complete deployed web service around this.



The above graph shows the results that we got during training our model. We measured the performance of our model by measuring average rewards every 500 steps. As you can see in the graph that when we train the model on Team 1 (index 0) [episode 1] it initially performs similar to a random agent taking actions however with time it learns a proper strategy and then the average rewards gradually increase with time(Dark blue in the graph). When it learns everything that's possible to learn from the data. The average rewards stop increasing. Once the episode gets over and a new episode starts(Red in the graph) the model(which is now trained on team 0's policy/strategy) initially performs badly as the strategy for controlling light used by team 1 is different from what team 0 used. However, The model quickly catches up to the new strategy and the average reward gets to the value it was before and then the model performs well for the rest of the data.

Creating a Web Service around the Problem

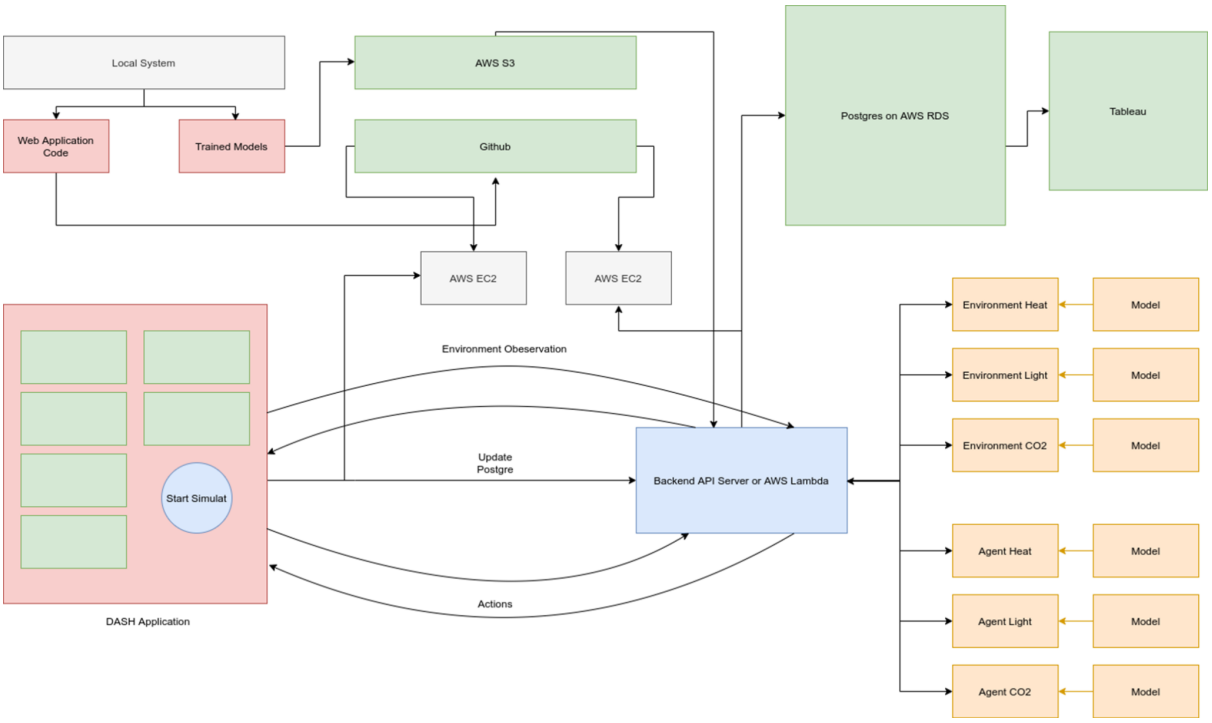
When you think of creating a service for controlling a greenhouse the most usable thing that you can come up with is a Greenhouse Monitoring and Controlling service that lets you visualise the sensor readings coming from the greenhouse and then enables you to take some actions based on these visualisations. As far as creating a web service is considered we can create a dashboard that has graphs to visualise and buttons to manage the greenhouse. In this case, since we had an agent that was going to control the greenhouse.

We built a service that:

- Lets the user start/stop the greenhouse simulation.
- Lets you visualise the environment in the greenhouse
- Lets the user visualize the Actions and Rewards for Random agent vs Train agent.
- Lets the user send the simulation data to a database that can be visualised using third-party tools such as tableau.

We created the following things:

- A backend API (using Fast API) that lets the frontend dashboard control the simulations and fetches observations and actions from our model.
- A frontend Dashboard that lets you visualise data and control the data(using Dash Python).
- A Postgresql database that maintains simulation data so that it can be visualised on third-party tools such as tableau.



Our exact data pipeline:

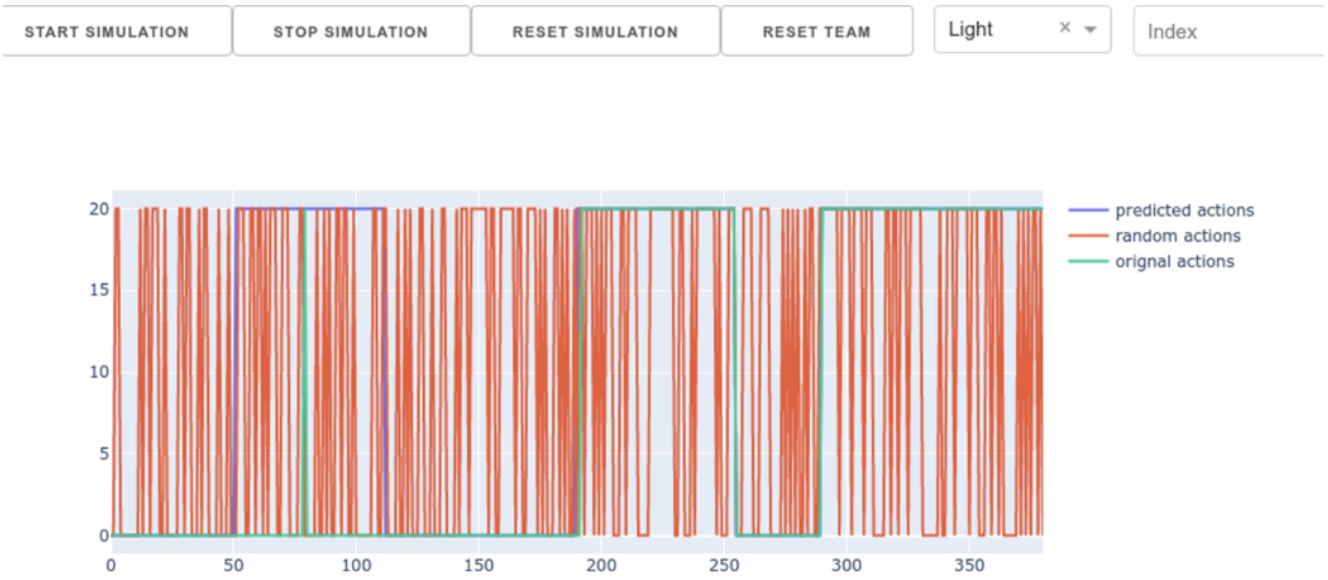
1. When the user visits the front end dashboard. They are shown with simulation controls.
2. Once they start the simulation the frontend starts asking for data from the backend API in the following order.
    a. The first observation from the environment.
    b. The first observation is POST on an endpoint and the API returns with the corresponding action the trained agent takes.
    c. Frontend then POSTs this action to an endpoint that acts as a “step” in the environment the backend returns the next observation and the corresponding rewards for this action. (rewards, actions and observations are plotted in this step)
    d. We go back to step b and POST the previously fetched observation to get the next action and the process continues.
3. Once the user is done analyzing the simulation, they can press the update tableau button and all the simulation data collected is sent to a database deployed to the cloud. This database is then connected to a tableau worksheet that can further be used to analyze data in detail.

Some points of interest about the deployment:

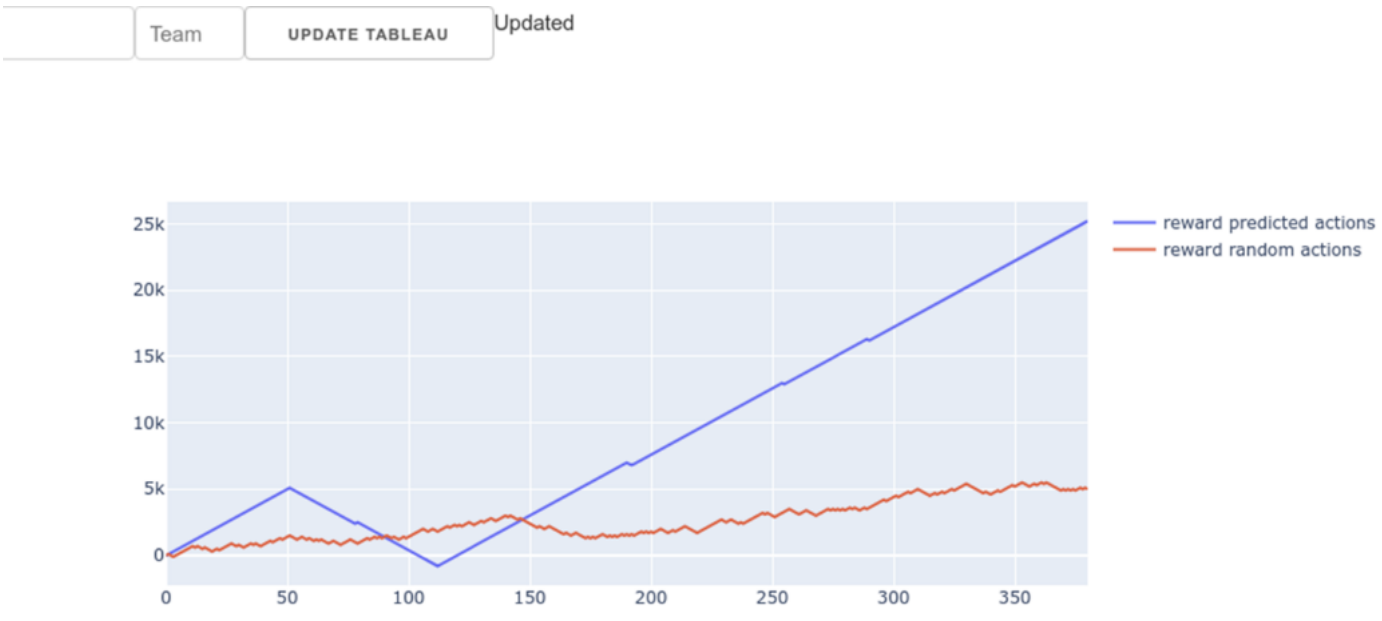
a. Every time the backend API is started it fetches the trained models which have been uploaded to our AWS S3 instance.
b. The backend and frontend had been deployed on free tier AWS EC2 instances and the web service was deployed on a free domain as well.
c. The database was deployed on a free tier AWS RDS free tier Postgres instance and connects to tableau very comfortably.

A look at the UI of our created frontend Dash Application

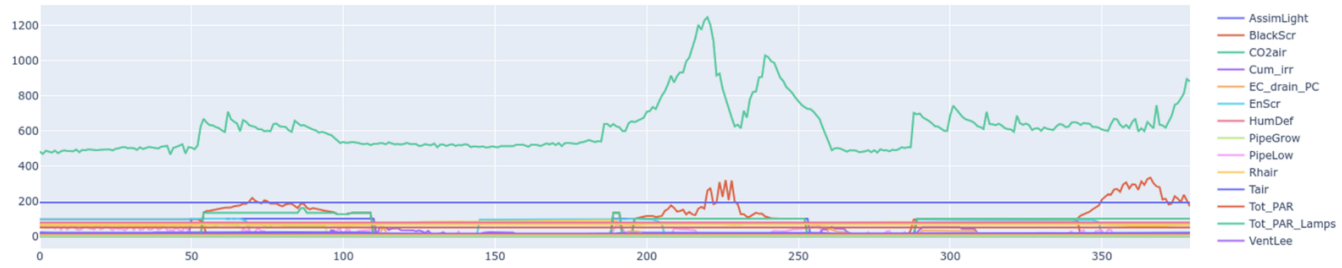
The top right contains buttons to control the simulation and shows an interactive graph that lets you visualise the actions taken by the model, original actions & random actions.



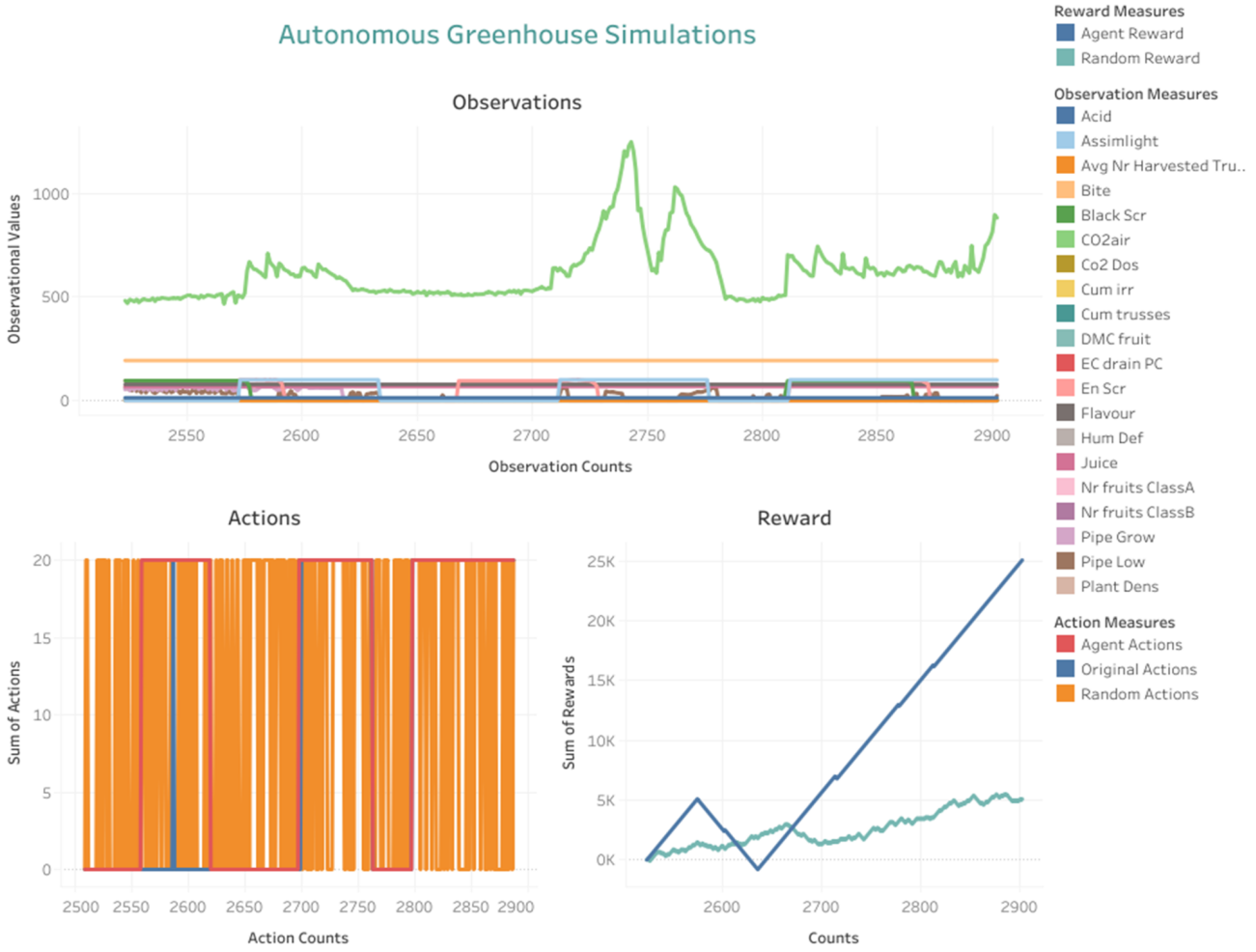
The top left contains the update tableau button and shows an interactive graph that lets you visualise the rewards for predicted actions vs the rewards for random actions.



The bottom part of the dashboard contains a graph that lets you visualise all the observable parameters of the greenhouse simulation. It's is an interactive graph to select what specific parameter plot you want to see as well as zoom in and out on it. (Graphs created using Plotly).



A look at the Tableau worksheet that can visualise this data



The above-shown tableau worksheet has visualisations similar to our frontend dash application. It has 3 main visualisations:

- Actions(Agent, Original and Random)
- Rewards(Agent, Random)
- Observations from the Greenhouse.