Precision Agriculture for Corn using Reinforcement Learning

BY

TAN CARLTON

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2024

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: Precision Agriculture for Corn using Reinforcement Learning

Academic Session: JAN 2024

I TAN CARLTON
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

	Verified by,
[illegible]	[illegible]
(Author's signature)	(Supervisor's signature)
Address:	
10, Jalan Ketumbar,	
Taman Cheras, 56100,	Dr OOI BOON YAIK
Kuala Lumpur	Supervisor's name
Date: 24/04/2024	**Date**: 25/4/2024

Universiti Tunku Abdul Rahman			
Form Title : **Sample of Submission Sheet for FYP/Dissertation/Thesis**			
Form Number: **FM-IAD-004**	Rev No.: **0**	Effective Date: **21 JUNE 2011**	Page No.: **1 of 1**

FACULTY/INSTITUTE* OF Information and Communication Technology

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 24/04/2024

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that ***TAN CARLTON*** (ID No: ***200ACB01512***) has completed this final year project/ dissertation/ thesis* entitled "Precision Agriculture for Corn using Reinforcement Learning" under the supervision of Dr OOI BOON YAIK (Supervisor) from the Department of Computer Science, Faculty/Institute* of Information and Communication Technology , and Tseu Kwan Lee (Co-Supervisor)* from the Department of Computer Science, Faculty/Institute* of Information and Communication Technology.

I understand that University will upload softcopy of my final year project / dissertation/ thesis* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

[illegible]

(*TAN CARLTON*)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**Precision agriculture for corn using Reinforcement Learning**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : [illegible]

Name : TAN CARLTON

Date : 24/4/2024

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Dr Ooi Boon Yaik who has given me this bright opportunity to engage in an AI Smart Farming project. It is my first step to establish a career in AI fields. A million thanks to you.

Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course. I love you Daddy and Mommy.

ABSTRACT

This project introduces RACKY, an innovative and comprehensive API solution that seamlessly integrates the SWAT (Soil and Water Assessment Tool) model into the highly adaptable OpenAI Gym environment, thus creating the powerful simulation framework known as SWATGym. RACKY serves as a versatile interface, facilitating effortless retrieval of detailed corn plant state information based on precise fertilizer or irrigation inputs through its intuitive API endpoints. Beyond data access, RACKY incorporates a sophisticated reinforcement learning agent based on the Proximal Policy Optimization (PPO) algorithm within the SWATGym. This integration empowers users with the capability to input location-specific data alongside plant growth stage parameters, thereby obtaining highly optimized recommendations for fertilizer and irrigation amounts directly from the embedded AI model. RACKY helps people make better farming decisions by showing them how different amounts of fertilizer and water affect plant growth through real-time simulations and detailed analysis. This project aims to make advanced farming information and AI tools accessible to everyone, not just experts. By using RACKY, researchers, farmers, and anyone interested in farming can find ways to grow crops more sustainably and using fewer resources.

Keyword: Precision Agriculture, Smart Farming Cycle, Reinforcement Learning, DDPG, PPO

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Number **Title** **Page**

LIST OF SYMBOLS

π	Policy Network
τ	Update rate

LIST OF ABBREVIATIONS

DQN	Deep Q-Networks
PPO	Proximal Policy Optimization
PCSE	Python Crop Simulation Environment
WOFOST	World Food Studies
DSSAT	Decision Support System for Agrotechnology Transfer
CGMs	Crop Grow Models
TD3	Twin Delayed Deep Deterministic Policy Gradient
SWAT	Soil and Water Assessment Tool
MDP	Markov Decision Process
DDPG	Deep Deterministic Policy Gradient
RNNs	Recurrent Neural Networks
AI	Artificial Intelligence
GDP	Gross Domestic Product
RL	Reinforcement Learning
API	Application Programming Interface
IoT	Internet of Things

Chapter 1:

Introduction

1.1 Problem Statement and Motivation

Developing and optimizing crop growth strategies, such as for corn plants, demands a thorough comprehension of plant growth dynamics and the influence of external factors like fertilizers, irrigation, and weather situation. Conventional approaches to AI model training for crop management involve using physical plants, which is both **time-consuming** and **resource intensive**. Current methods for training AI models in crop management often rely on using physical plants. This process is time-consuming and demands significant resources in terms of space, materials, and maintenance. It limits the scalability of AI-driven solutions.

Furthermore, there is a **lack of user-friendly interfaces** for leveraging AI tools in farming tailored to individual needs. There is a noticeable absence of easy-to-use interfaces that allow individuals with varying levels of technical expertise to harness the power of AI tools in farming. This lack of accessibility hinders widespread adoption and innovation in agricultural practices.

The driving force behind this initiative is **to create a unified, user-friendly interface** that enables anyone, regardless of technical expertise, to access and utilize AI tools for farming. This initiative aims to save time, expedite AI model training, and foster collaboration and knowledge sharing in agriculture.

Additionally, this project encompasses **developing a reinforcement learning agent** capable of interacting with farming environments. This agent empowers users to tackle various farming challenges, such as determining optimal irrigation and fertilizer amount.

By democratizing AI tools for farming, this project seeks to enhance agricultural practices, promote sustainability, and contribute to global food security while mitigating environmental impacts.

1.2 Objectives

Our primary objective is to revolutionize crop management strategies and agricultural practices through the development of an integrated system.

Firstly, the project aims to **create an API** for SWATGym, which will facilitate users' access to and use of the tool for those interested in using AI-driven solutions to optimize crop growth. This API will operate as a doorway for user to run their corn crop simulation based on different weather or location and created their own AI model.

Secondly, the project also focus extends to **building an AI model** that seamlessly integrates with SWATGym, enabling it to interact intelligently with simulated farming environments. This model will utilize the environment and suggest optimal action like amount of fertilizer and irrigation.

Furthermore, we recognize the importance of **accessibility and user-friendliness** in driving widespread adoption of AI tools in farming. To this end, we will develop intuitive and easy-to-use interfaces that leverage the power of the **AI model through the API**. These interfaces will be tailored to accommodate users with varying levels of technical expertise, empowering them to make informed decisions and implement efficient farming practices effortlessly.

In conclusion, this project aims to revolutionize agricultural management by **creating a user-friendly API** for SWATGym. By integrating an AI model with SWATGym, we seek to optimize crop growth strategies by intelligently interacting with farming environments and suggesting optimal actions such as fertilization and irrigation.

1.3 Project Scope and Direction

The project scope encompasses several key elements essential for advancing agricultural management practices using AI-driven solutions. One crucial aspect involves **implementing a reinforcement learning agent**, specifically Proximal Policy Optimization (PPO), to refine and optimize fertilizer and irrigation strategies. This agent will leverage the capabilities of the SWATGym environment, providing a realistic simulation platform for testing and improving farming techniques.

Additionally, a significant focus is on **integrating this AI model and SWATGym simulator into the API infrastructure** using Python Flask. This integration will enable users from various backgrounds, including researchers, developers, and hobbyists, to access the full potential of the reinforcement learning agent and the environment.

1.4 Contributions

The contributions of this project are as below. Firstly, we **provide an AI model** capable of effectively utilizing the SWATGym environment, specifically focusing on optimizing fertilizer and irrigation strategies using Proximal Policy Optimization (PPO) reinforcement learning techniques.

Secondly, our project significantly improves accessibility and usability for users **by developing a user-friendly API** that integrates seamlessly with SWATGym. This integration streamlines access to the AI model and the simulation environment, empowering users such as researchers, developers, and farmers to leverage advanced AI tools without extensive technical expertise.

Furthermore, we **augment the capabilities of develop API by integrating external weather API**. This addition addresses a crucial limitation of the existing SWATGym API, which lacks real-time weather data for accurate simulations. By incorporating external weather data, we enhance the accuracy and relevance of the simulations, enabling users to make informed decisions based on current weather conditions, ultimately improving crop management strategies and productivity.

1.5 Report Organization

This report is organised into 6 chapters: Chapter 1 Introduction, Chapter 2 Literature Review, Chapter 3 System Design, Chapter 4 System Implementation and Testing, Chapter 5 System Outcome and Discussion, Chapter 6 Conclusion.

The **first chapter** is the introduction of this project which includes problem statement, project background and motivation, project scope, project objectives, project contribution and report organisation.

The **second chapter** is the literature review carried out on several existing reinforcement learning environment, smart farming cycle, and existing technique to implementing reinforcement agent.

The **third chapter** is discussing the overall system design of this project. It contains the information of SWATGym, Python Flask, and the methodology of PPO and DDPG Agent that will be implement into this project.

The **fourth chapter** is regarding the details on how to implement the design of the system. It contains with the block diagram and use case diagram to show the interaction of user with the system.

The **fifth chapter** reports are the implementation of the API, AI model and issue and challenge that faces in the implementation.

The **sixth chapter** is the system evaluation which contain the test of the API, result of the AI model, and the project challenge.
The seventh chapter is the conclusion and recommendation of this project.

Chapter 2

Literature Review

2.1 Previous works on reinforcement learning farm

2.1.1 Intelligent Farm Based on Deep Reinforcement Learning for optimal control (Actor Critic)

This paper introduces the concept of smart farming, emphasizing the use of artificial intelligence to optimize agricultural operations. Smart agriculture aims to improve crop yields, reduce costs, and enhance crop quality through automation, data analysis, and resource optimization. The paper proposes a smart farming system that utilizes deep reinforcement learning to make optimal decisions in agricultural settings.
The smart farm had to make decision to growth the crop meanwhile tackle the unpredictable states and uncontrollable factor that will affect the crop. Therefore, the paper had suggested 2 ways to managing the smart farm which is Recurrent Neural Networks (RNNs) and Reinforcement Learning. But we will just focus on the reinforcement learning result. Reinforcement Learning, focusing on the Deep Deterministic Policy Gradient (DDPG) method. In this paper, intelligent agents aim to maximize cumulative rewards by iteratively learning optimal policies. DDPG introduces the actor-critic framework, where the actor (policy) seeks the best actions, and the critic (value function) evaluates these actions' quality. The critic estimates the value of state-action pairs and updates iteratively to reach desired goal values[1]. This reinforcement learning approach presents opportunities for optimal control in complex and dynamic farm environments. Additionally, the study touches on the potential of asynchronous training for multiple agents, offering a means to accelerate learning in intricate systems while acknowledging potential challenges associated with this approach[1].

Below is the experiment that of reinforcement learning work out in the smart farm. The structure of control system is show.

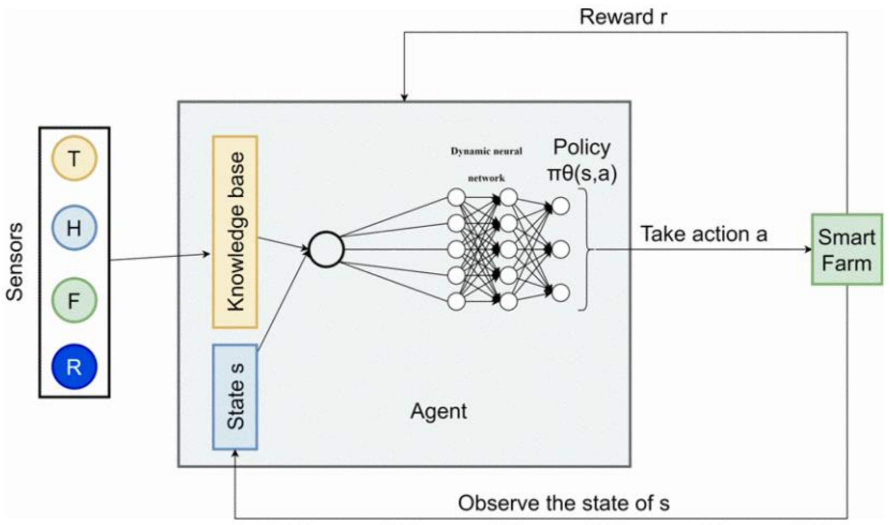


Figure 1: Structure of RL Farm

To optimize crop production and quality, an intelligent agent-based approach that adapts to environmental conditions and controls various aspects of the farm is used. Each smart farm subsystem was controlled by a slave agent, which communicated environmental data to the master agent, enabling precise decision-making[1]. The agents developed strategies based on a knowledge base derived from sensor data and the farm's state. Each agent had its own dynamic neural network with hyperparameters optimized using Bayesian optimization[1]. The key environmental variables considered for optimization were temperature (T), humidity (H), fertilizer (F), and rainfall (R) as shown on the above figure. Rainfall determined crop selection, while other factors were used to optimize crop growth. The fertilizer choice was tailored to the specific soil composition and crop requirements.
The experiment had demonstrated the effectiveness of the RL approach, and 100 versions of the model concurrently, comparing results obtained after hyperparameter optimization.

Iteration	Average Loss	Average Accuracy	Simultaneously
10000	0.83483	74.45%	100
50000	0.53487	85.50%	100
100000	0.41357	93.16%	100

Table 1 Result of Effectiveness of RL approach

These experiments showcased the ability of our model to achieve excellent results in smart farm management, emphasizing the importance of reinforcement learning in enhancing crop production and quality.

2.1.2 A Reinforcement Learning Approach for Smart Farming (MDP, Thomas Sampling, Q-learning)

In this paper had shown the Reinforcement learning algorithm like Markov Decision Process (MDP), Thomas Sampling Algorithm and Q-learning algorithm. The paper shows how the implementation of these algorithm in handling the various control and unpredictable noises in the smart farm.

The Markov Decision Process (MDP), which includes elements such as the states, actions, other probabilities, and reward, and discount factor[2]. The agent's goal is to learn the optimal policy that maximizes the expected cumulative reward. This can be achieved by defining below[2].

$$Q^{\pi}(s,a) = \mathrm{E}_{\pi}\left[R_t | s_t = s, a_t = a\right]$$

Figure 2: Action Value Function

$$V^{\pi}(s) = \mathrm{E}_{\pi}\left[r_{t+1} + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s\right] = \mathrm{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right]$$

Figure 3: Value Function

The Bellman equation will show the relationship between states and enabling the agent to make informed decisions[2].

The Thompson Sampling Algorithm is a strategy for making sequential decisions in situations where there is a need to balance between exploiting known information and exploring to gather new information for better future decision[2]. In the context of the algorithm, the problem is often represented as a multi-armed bandit problem, where the agent must choose among multiple actions, each associated with a reward from a probability distribution[2]. The algorithm can be applied into the farm by helping farmers make decisions about where to place experimental plots for different seeding rates. Each field is characterized by various soil properties, and the algorithm aims to decide where to place experimental plots within the field to optimize crop yield. Each field is divided into smaller areas, and selecting an area for an experimental plot allows the farmer to observe whether it improves yield response[2].

The Q-Learning Algorithm allow the agent to make decisions by estimating the value of states and actions. This method involves assessing rewards or penalties for actions taken at various states and aims to maximize the cumulative return, considering a discount factor γ that influences the trade-off between immediate and future rewards[2]. The algorithm operates by creating and updating a Q-table and learning the optimal strategy over time[2]. The action and states value is update to the q table each time. It adjusts hyperparameters like learning rate, exploration level, and discount factor as it accumulates knowledge, ensuring the agent selects actions that maximize rewards[2].

Both algorithms evaluation is shown below. It had shown how to apply the reinforcement learning into the daily farm task.

In the evaluation of the Multi-Armed Bandit algorithm, randomly generated data was employed to represent observed yields for plots, with values above a threshold considered as "good" as "1" and values below it as "not good" as "0". The algorithm involved creating a DataFrame object with 200 observations, initializing lists for rewards and penalties, and selecting plots based on the highest random beta distribution[2].

In the evaluation of the Q-Learning algorithm, the OpenAI Gym toolkit was employed, utilizing a custom environment called "TruckEnv" to simulate a self-driving delivery truck navigating a field. The algorithm aimed to maximize rewards and minimize penalties over episodes by learning an optimal policy. Hyperparameters such as learning rate, discount factor, and exploration level were set, and performance was evaluated based on the average number of steps, rewards, and penalties per move over episodes. The results demonstrated the agent's learning and improved performance over time[2].

2.1.3 Reinforcement Learning for Sustainable Agriculture

This paper had focus in reinforcement learning, to optimize plant development with respect to key parameters such as yield and environmental impact. It mentions that reinforcement learning to autonomously explore and learn methods for influencing plant development while considering environmental factors like irrigation and nutrient supply[3].

By applying reinforcement learning to agriculture could enable precise control over environmental conditions and resource distribution to maximize yield while minimizing resource usage.

The paper mention that framework that combines physical and modelling components to optimize plant development using reinforcement learning techniques. The core of the system involves an "agent" that learns to control various parameters influencing plant[3]. The used of temperature, humidity, moisture, and specialized chemical sensors, along with elements for heating, cooling, adjustable lighting, water supply, and nutrient delivery able to precise the environment control[3].

Fertilizer management and water management with reinforcement learning is mentioned in this paper. In these scenarios, the reinforcement learning algorithm learns the optimal distribution of fertilizer and water over time, considering environmental conditions[3]. The approach involves conducting numerous parallel experiments to determine the best timing and quantities of resource allocation for maximizing production while minimizing overall resource consumption. To enhance decision-making, sensory data may be augmented with virtual data, such as forecasts of future conditions[3].

The proposed system aims to operate multiple controlled growth chambers in parallel to accumulate sufficient data for effective algorithm learning. Initial experiments could involve the use of Arabidopsis thaliana as a model species due to its small size and rapid life cycle, making it suitable for high-turnover experiments[3]. These initial experiments may focus on nutrient supply, specifically nitrogen and phosphate, with an emphasis on minimizing nitrogen fertilizer use for environmental protection[3]. As the experimental setup stabilizes, the authors plan to scale the approach to monocot species like rice, maize, or wheat, which are more relevant to food security, while still considering model organisms such as Brachypodium distachyon and Setaria viridis for ease of experimentation[3].

This innovative framework demonstrates the potential of reinforcement learning and advanced environmental control to optimize plant development and resource management in agriculture, offering a promising solution to address global food production challenges.

2.2Previous work on reinforcement learning GYM environment

2.2.1 SWATGym

The SWATGym environment is a reinforcement learning (RL) environment based on the Soil and Water Assessment Tool (SWAT) model. It simulates crop growth by considering factors such as nutrient cycling, water availability, and temperature. SWATGym is the first Python-based implementation of SWAT, making it accessible for RL applications. It is built on top of the OpenAI Gym framework, which is widely used for developing RL environments.
Below are the block diagram of SWATGym [4]:

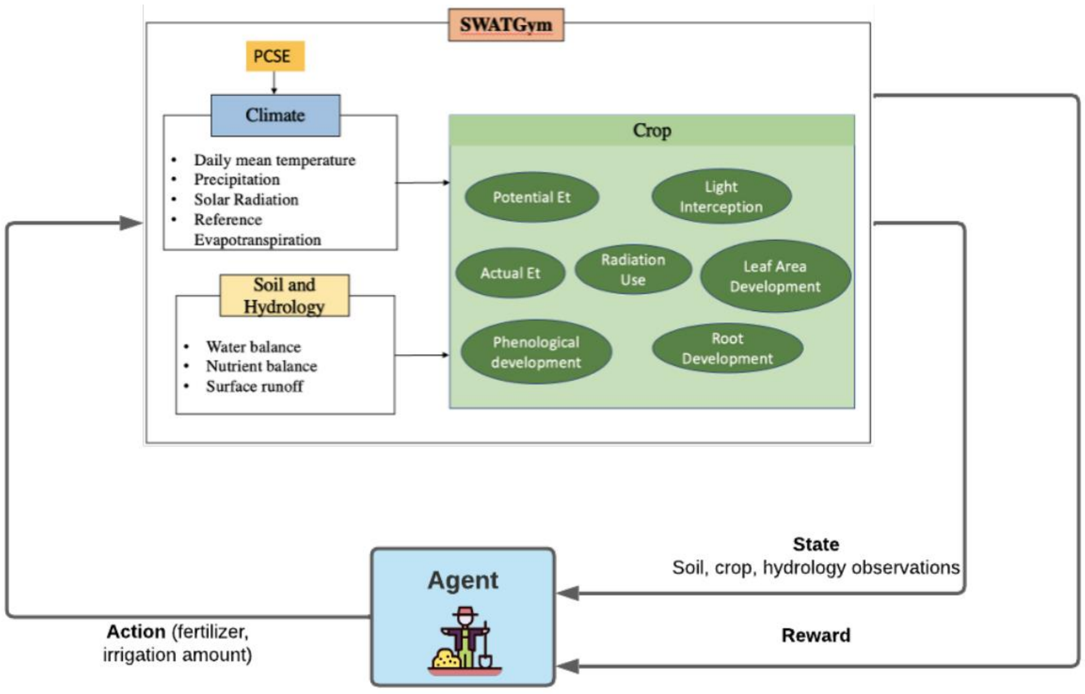


Figure 4: Overview of SWATGym

The main function of SWATGym is to provide a simulation platform for evaluating and comparing different crop management strategies. It allows researchers and practitioners to benchmark multiple strategies simultaneously and at minimal cost. By simulating crop growth from emergence to harvest on a daily basis, SWATGym enables the evaluation of crop management strategies and the development of sustainable agriculture practices.
The SWATGym environment has a continuous state space comprising 14 state variables related to weather, soil, crop, and hydrology dynamics. The details of 14 state are shown below [4]:

Table 2: Table of SWATGym state

Observation	**Unit**
Mean air temperature	°C
Precipitation	mm
Reference Evapotranspiration	mm
Solar Radiation	MJ/mm^2
Mean Vapor Pressure	hPa
Actual Evapotranspiration	mm
Water balance	mm
Daily runoff curve number	-
Leaf area index	-
Nitrogen Uptake	kg/ha
Denitrification	kg/ha
Nitrogen stress factor	-
Temperature stress factor	-
Water stress factor	-

It also has a continuous multidimensional action space, where actions represent the amounts of fertilizer and irrigation applied at each time step. At each time step, the agent selects an action that consists of two components which is the amount of fertilizer (F) and the amount of irrigation (I) to be applied.

The environment produces a reward that characterizes the effects of different choices of actions on crop production. The reward function takes into account crop yield, the cost of applying fertilizer and irrigation, and penalty terms associated with these costs. Below are the reward function formula [4]:

$$r_t = yld_t - \alpha F_t - \beta I_t, \qquad (1)$$

Figure 5: SWATGym reward calculation

where yld is the estimated crop yield on a particular day, Ft represents the amount of fertilizer applied, It represents the amount of irrigation applied, and α and β are penalty terms associated with the estimated cost of applying fertilizer and irrigation respectively[4]. The values of α and β in this case are α = 2.43 and β = 0.16. The goal of the RL agent is to maximize the cumulative rewards over a finite horizon of length T, corresponding to a growing season [4]. Thus, the potential harvest index is shown below[4]:

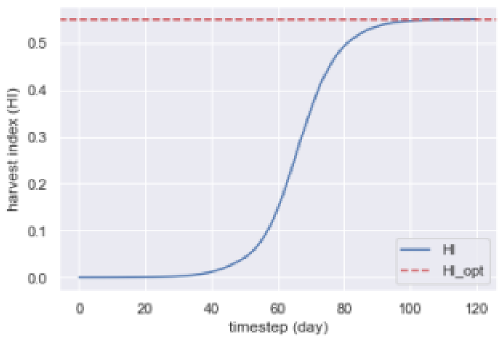


Figure 6: Graph of potential reward of SWATGym

2.2.2 CyclesGym

CyclesGym paper explores the use of reinforcement learning (RL) to design adaptive policies for agricultural systems. This paper shows the related works that use deep RL with sophisticated crop growth models (CGMs) to optimize agricultural management[5]. CGMs are mathematical models that simulate the growth and development of crops over time, taking into account factors such as weather, soil, and management practices. CGMs are used to predict crop yields and optimize management practices. Therefore, CyclesGym is work with the OpenAI gym wrapper around CGMs, such as DSSAT and WOFOST[5]. Below are the overview of interaction of Environment, Management ang Genetic with the crop.

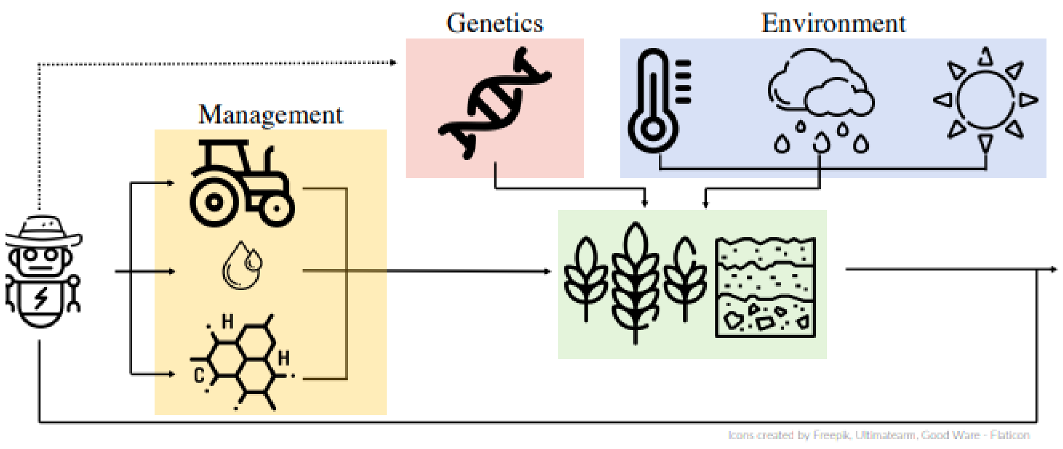


Figure 7: Overview of CycleGym

Besides, CyclesGym is allow to change the year and different crop model to test at daily time steps[5]. Resources information will be collected in each time steps.

CyclesGym interacts with observers, implementers, rewarders, and constrainers that control simulations, parse outputs, and provide interfaces for observations, actions, rewards, and constraints[5]. Users can create custom RL environments by subclassing CyclesEnv and configuring these managers.

This paper had conduct two experiment which are nitrogen (N) application and crop rotation. In the nitrogen application experiments, RL agents, PPO make decisions about the amount of nitrogen to apply to crops on a weekly basis. The goal is to maximize profit per hectare, considering the value of the crop at harvest minus the cost of nitrogen used. The experiments include training RL agents in various environments, including 1, 2, and 5-year settings, and testing their generalization across time, location, and planning horizon. Non-adaptive baselines, including fixed fertilization strategies, are used for comparison[5]. Results show that

RL agents outperform these baselines in terms of profitability, highlighting the potential of RL in smart fertilization practices[5].

For crop planning experiments, RL agents decide which crop to plant and when to plant it within a time window, aiming to maximize profitability without other operations like fertilization or tillage. Training is performed in a specific environment, and generalization is tested across different locations and time horizons. Non-adaptive baselines and fixed crop sequence strategies are used for comparison. RL agents demonstrate reasonable performance in test scenarios, although non-adaptive agents outperform them in some cases[5]. The fixed baselines are generally outperformed by the best-trained RL agents.

2.2.3 CropGym

The CropGym is a reinforcement learning (RL) environment powered with OpenAI framework. CropGym had included with Process-based crop growth models, including APSIM and PCSE for understanding and simulating crop growth dynamics[6]. These models incorporate various biophysical processes, such as light interception, nutrient availability, and water uptake, to predict crop yields accurately. It provides a mechanistic understanding of how crops respond to environmental factors.

The main function of CropGym is optimizing crop fertilization strategies with process-based crop growth models. Below are the description of the environment.

The state space in the CropGym environment operates on a weekly time interval to simulate natural farming practices. The agent's observations include two main components which is the Crop Growth Model and Weather Data[6].

For crop growth model, the agent observes the output variables generated by the LINTUL-3 process-based crop growth model. This model simulates crop growth under nitrogen-limited conditions and is implemented within the Python Crop Simulation Environment (PCSE)[6]. The model parameters have been fine-tuned to accurately represent the growth of winter wheat, making it a suitable choice for this simulation.

For the weather data, past weather conditions will be collected. This data includes default weather variables provided by PCSE, reflecting conditions from the previous week. The weather data is derived from 29 years of historical weather data spanning 9 locations in the Netherlands, sourced from the PowerNASA database in this research[6]. Additionally, the weather data can be changed according to latitude and longitude.

The agent's action space in the CropGym environment is designed to represent different fertilizer application options. The agent can decide on the amount of nitrogen fertilizer to apply per hectare of the crop. The available options are specified as follows[6]:

$$A = \{20\,k\frac{\text{kg}}{\text{ha}} \mid k \in \{0, 1, 2, ..., 6\}\}$$

Figure 8: Action of CropGym

This means the agent can choose to apply no fertilizer (k=0) or select from various discrete dosage levels ranging from 20 kg/ha to 120 kg/ha (k=1 to k=6)[6]. The discretization allows the agent to experiment with different levels of fertilizer application, including options that align with real-world agricultural practices.

The reward system in the CropGym environment is designed to guide the agent's decision-making by incentivizing specific behaviors. The primary objective is to let the agent achieve a large grain yield per hectare. This yield is expressed in kilograms of dry matter. The formular as below[6]:

$$r_t = m_{SO,t} - m_{SO,t-1} - (m^*_{SO,t} - m^*_{SO,t-1}) - \beta m_{fert,t}$$

Figure 9: Reward calculation of CropGym

Some experiment also done in this paper to test the CropGym using PPO agent. The result is quite well. Below are the graph that shows the increase in time also lead to increase in rewards[6].

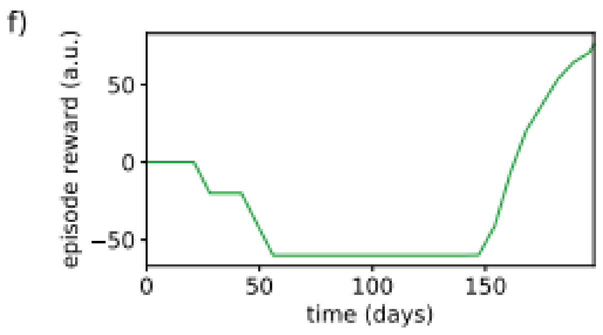


Figure 10: Graph of rewards of CropGym using PPO

2.2.4 FarmGym

Farm-gym is a modular and gamified environment designed for reinforcement learning (RL) research in the context of agricultural decision-making. It provides a platform that similar to actual agricultural farm while maintaining simplicity and controllability[7]. Farm-gym is unique in that it introduces stochasticity, reflecting the inherent uncertainty in farming due to external factors like weather, pests, and complex interactions between entities. Below are the overview of Farm-Gym[7]:

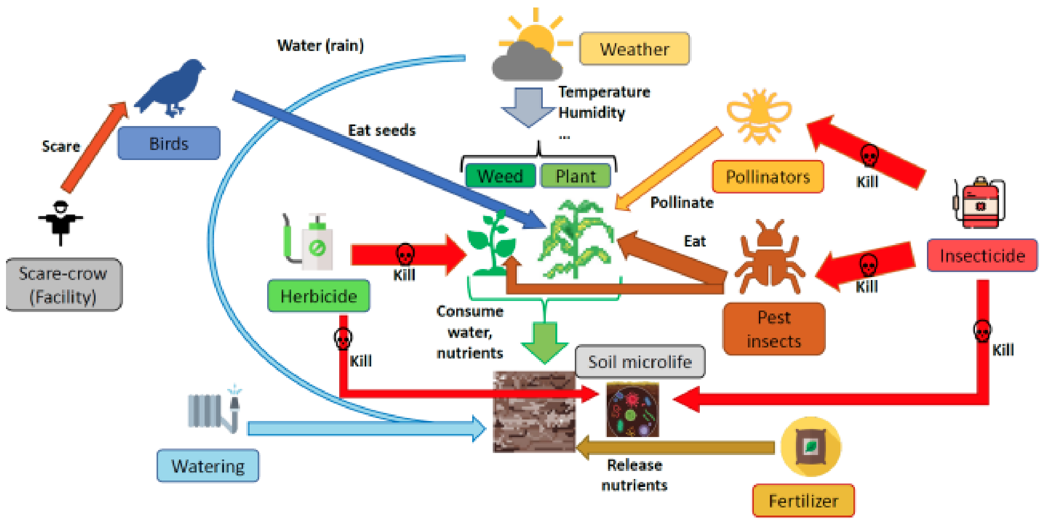


Figure 11: Overview of FarmGym

Farm-gym incorporates a highly stochastic environment by introducing stochastic processes in the dynamics of its entities, including weather and plant growth. It uses a mathematical model based on an exponential general linear function to represent the transitions between states for different entities[7]. This approach allows Farm-gym to create an intrinsically stochastic environment that mimics the uncertain nature of real-world agricultural systems.

Action space of FarmGym is large. In this environment, each entity introduces its own set of actions, some of which can be parameterized[7]. Additionally, Farm-gym allows the agent to review the expansion observation based on each action taken[7]. This expansion results in a space actions per day, compared to the original daily action space. This creates a complex decision-making environment where agents must learn the structure of the action space, understand how actions affect different parts of the system, and optimize their decision-making process[7].

The agent might need to observe the soil's moisture level or the insect population in the field before taking actions as each action may cause affect in the reward counting at the last. This requirement challenges the agent to learn when these additional observations are crucial for enhancing farm management. Essentially, the agent must evaluate its knowledge state, update its values through observations, and make informed decisions[7].

The experiment had done in farm-gym. The cumulative reward obtained during the training of the PPO agent is shown below[7]. The raw reward curve exhibits high variability due to the high changing in the farm-gym environment. To visualize the reward, increase more clearly, smoothing is applied.

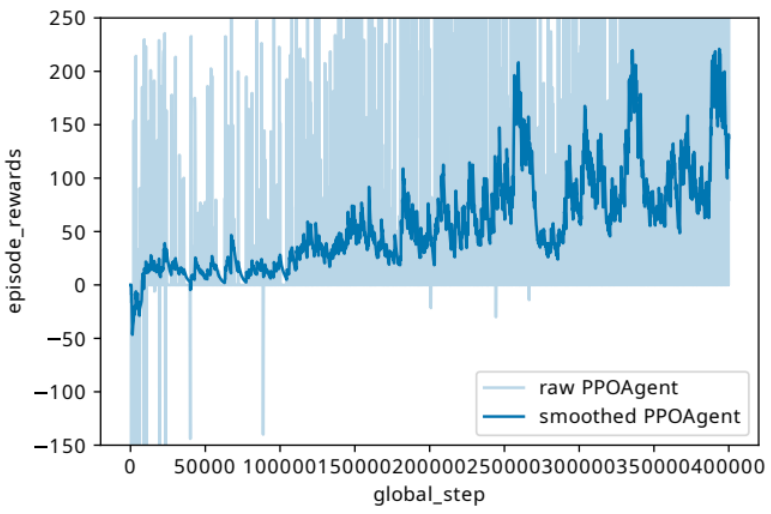


Figure 12: Graph of reward of FarmGym using PPO

2.3 Critical Remarks

In reviewing the previous works related to reinforcement learning (RL) in agricultural environments, several strengths and weaknesses are stated below.

Firstly, the paper “Intelligent Farm Based on Deep Reinforcement Learning” (Actor Critic) paper. The paper introduced the concept of smart farming, highlighting the potential of AI and RL in optimizing agricultural operations. Besides, it also explores the some RL method well-suited for continuous action spaces.

Secondly, the paper “A Reinforcement Learning Approach for Smart Farming” (MDP, Thomas Sampling, Q-learning. The paper explored a variety of RL algorithms, including Markov Decision Process (MDP), Thompson Sampling, and Q-Learning, for smart farming and discusses the application of these algorithms in handling control and unpredictable factors. But this paper does not show the algorithm that would help to handle the continuous control happens in reinforcement learning.

Moreover, the paper “Reinforcement Learning for Sustainable Agriculture”. This paper focuses on using RL to optimize plant development while considering environmental factors like irrigation and nutrient supply. But RL algorithms and their performance are not discussed in detail.

From all the RL Gym that had research above. SWATGym is chosen for further analysis. SWATGym provides a comprehensive RL environment based on the Soil and Water Assessment Tool (SWAT) model, making it highly relevant for crop management and resource optimization. It offers a continuous state space and action space, able to let the user input accurate value in each time steps is very similar with the project scope on optimizing fertilizer and irrigation inputs. But SWATGym it may not address all the complexities of managing unpredictable factors and uncontrollable states in smart agriculture like pest and other change.

Chapter 3

System Methodology/Approach OR System Model

3.1 IoT Smart Farming Cycle

Smart farming represents a revolutionary approach to agriculture and cattle production, leveraging the technologies of the Fourth Industrial Revolution to enhance productivity, minimize resource use, and reduce environmental impact[8]. At the heart of this transformation is the Internet of Things (IoT), a system that collects data from various sources and transmits it over the internet. The IoT-based smart farming cycle consists of several key stages which is observation, diagnostics, decision, and action.

In observation state, sensors will be deployed in agricultural settings and continuously gather data from crops, livestock, or the environment condition that able to capture by sensors[8].

In the diagnostics phase, the data that collected is sent to a cloud hosted IoT platform equipped with predefined decision rules and models. These models, often referred to as "business logic," analyse the data to assess the condition of the monitored objects and identify any issues or requirements[8].

In the decisions stage, the insights will be generated by the IoT platform, users and machine learning-driven components determine whether specific actions are needed for a particular location[8]. These actions can include treatments or interventions.

In the action stage, users implement necessary actions according to the decision that made in previous stage. These actions can involve adjusting farming practices, resource allocation, or other interventions to address identified issues[8]. The cycle then begins to continue following by observation etc until the end.

In this project, diagnostics and decision stage will be implemented. The diagnostics and decision will be implemented by using reinforcement learning. The detail will be show in below section.

3. 2 Reinforcement Learning with Actor Critic Algorithm

In the context of reinforcement learning (RL) and its application in smart farming, let's delve into the concepts of diagnostics and decision-making using reinforcement learning, with a particular focus on the actor-critic algorithm.

3.2.1 Actor Critic

Diagnostics in reinforcement learning refer to the process of assessing the condition or performance of an RL agent as it interacts with its environment. In the provided text, diagnostics involve evaluating the state-value function (Vπ) and the action-value function (Qπ) to estimate the expected returns and the quality of actions taken by the agent[9].

$$V_\pi\ (s) = E[\ R_t \mid s_t\ = s]$$

Figure 13: State-value function

Above is the state-value function which observe the return based on starting state and policy. It show the quality of state based on the specific policy[9].

$$Q_\pi(s,a) = E[\ R_t \mid s_t = s, a_t = a]$$

Figure 14: Action-value function

Above is the action-value function that able to observe the return based on state, action and policy taken. The quality of the action will be show by this function. Diagnostics also involve estimating Qπ, which can be used to make decisions about which actions to take[9].

Decision-making in RL involves selecting actions that maximize the expected return over time. This process will act based on stated policy. Two fundamental concepts related to decision-making in RL are policy and actor critic algorithms.

The policy is the behaviour function that dictates how the RL agent selects actions in response to observed states[9]. In the context of smart farming, the policy could determine actions like when to irrigate crops or which treatment to apply to livestock. It's a crucial component of the RL agent.

The actor-critic algorithm is a popular approach in RL that combines two key components which are actor and critic.

The actor represents the policy function π. It will act based on the state that observe[9]. The actor could decide actions like adjusting the action value based on the environment requirement.

The critic will review the action and provides feedback on how good or bad the actor's actions are in a given state[9].

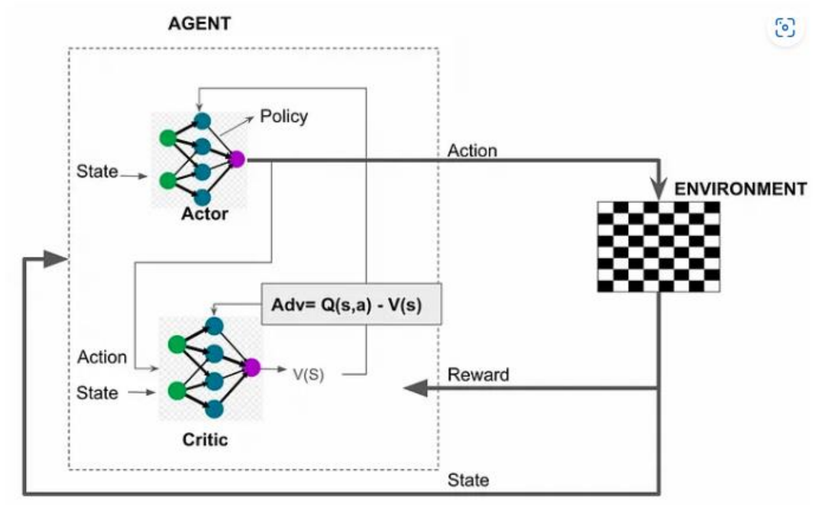
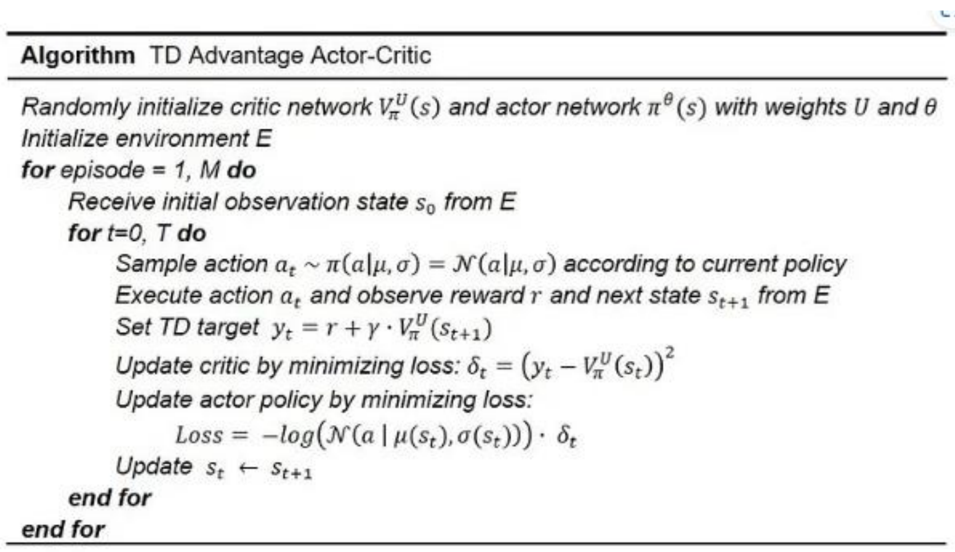


Figure 15: Overview of actor-critic structure

The actor-critic algorithm is uses of both critic and actor to improve decision-making in RL[9]. Firstly, the actor will select action. Next, the critic will review the action by calculate the return. Then, the critic provides feedback to the actor about the quality of the chosen actions. Lastly, the actor changes the policy to achieve higher return from critic feedback. Below are the pseudocode that collected from the internet[9].



```
Algorithm TD Advantage Actor-Critic
Randomly initialize critic network $V_\pi^U(s)$ and actor network $\pi^\theta(s)$ with weights $U$ and $\theta$
Initialize environment E
for episode = 1, M do
    Receive initial observation state $s_0$ from E
    for t=0, T do
        Sample action $a_t \sim \pi(a|\mu,\sigma) = \mathcal{N}(a|\mu,\sigma)$ according to current policy
        Execute action $a_t$ and observe reward $r$ and next state $s_{t+1}$ from E
        Set TD target $y_t = r + \gamma \cdot V_\pi^U(s_{t+1})$
        Update critic by minimizing loss: $\delta_t = \left(y_t - V_\pi^U(s_t)\right)^2$
        Update actor policy by minimizing loss:
            $Loss = -log\left(\mathcal{N}(a \mid \mu(s_t), \sigma(s_t))\right) \cdot \delta_t$
        Update $s_t \leftarrow s_{t+1}$
    end for
end for
```

Figure 16: Pseudocode of Actor-Critic

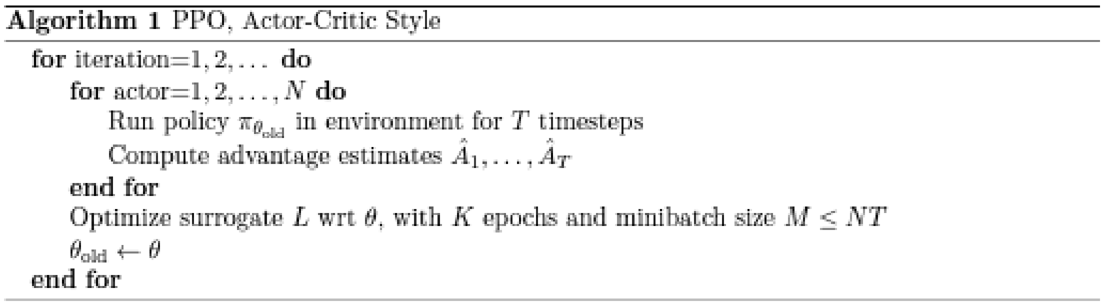
The actor-critic algorithm helps in achieving better convergence and more stable learning compared to using only value-based or policy-based methods. In IoT-based smart farming, this actor-critic framework can be applied to optimize farming decisions. Therefore, the theory of actor critic will be applied in this project. The actor will be deciding the amount of fertilizer and irrigation to apply to crops, while the critic assesses the impact of these decisions on crop yield and resource usage. Over time, the actor adapts its policy to make more informed and effective decisions, leading to improved farming outcomes.

3.3 PPO Algorithm

The Proximal Policy Optimization (PPO) algorithm builds upon policy gradient methods and trust region optimization techniques. It begins by estimating the policy gradient using stochastic gradient ascent, where the objective is to maximize the expected reward while updating the policy parameters.

PPO introduces a clipped surrogate objective function that addresses the limitations of previous methods by preventing excessively large policy updates. This clipped objective combines the benefits of trust region methods, which ensure stable policy updates, with the efficiency of policy gradient methods. Additionally, PPO incorporates an adaptive KL penalty coefficient to regulate the KL divergence between the old and updated policies, further enhancing stability during training[10].

In practical terms, the PPO algorithm involves collecting data by running the old policy in the environment, estimating advantages for each timestep, optimizing the surrogate loss (either clipped or with a KL penalty) using minibatch SGD, and then updating the policy parameters based on the optimization results[10]. Below are the steps to implement a PPO Agent.



```
Algorithm 1 PPO, Actor-Critic Style
for iteration=$1, 2, \ldots$ do
    for actor=$1, 2, \ldots, N$ do
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for
```

Figure 17: Pseudocode of PPO Algorith with Actor Critic Architecture

Based on the PPO algorithm pseudocode above, it operates through a series of iterative steps that ensure stable and efficient learning in reinforcement learning tasks. First, previous state and action from the replay buffer is collected by executing the old policy in the environment, generating experiences used for training.

Next, advantage estimates are computed to gauge the effectiveness of actions taken under the current policy[10]. Subsequently, the surrogate loss function, either the clipped surrogate objective or the KL-penalized objective, is optimized using minibatch stochastic gradient descent (SGD)[10]. This optimization process fine-tunes the policy parameters based on the observed advantages, striking a balance between exploration and exploitation. Finally, the updated policy parameters are applied to the agent, allowing it to interact further with the environment and repeat the learning process[10]. This iterative approach, coupled with the clipped surrogate objective and adaptive KL penalty coefficient, underpins PPO's ability to achieve robust and scalable reinforcement learning outcomes across a variety of environments and tasks.

Chapter 4

System Design

4. 1 RACKY API Design

4. 1. 1 Flowchart of RACKY API

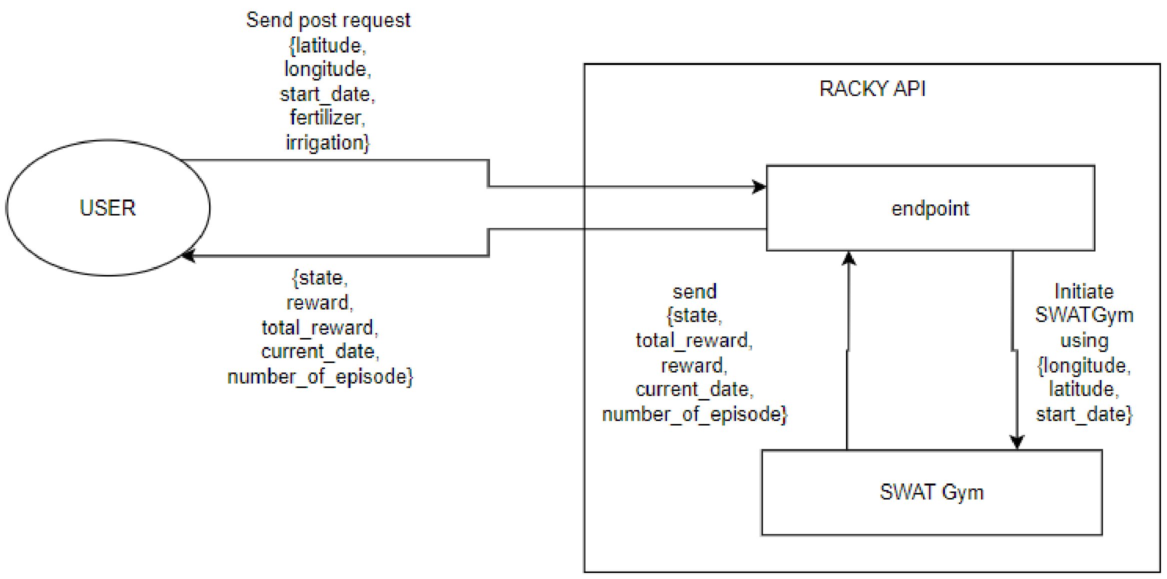

Send post request {latitude, longitude, start_date, fertilizer, irrigation}
USER
{state, reward, total_reward, current_date, number_of_episode}
RACKY API
endpoint
send {state, total_reward, reward, current_date, number_of_episode}
Initiate SWATGym using {longitude, latitude, start_date}
SWAT Gym

Figure 18: Flowchart of RACKY API

Above is the use case diagram of the RACKY API. Firstly, users are able to input information such as latitude, longitude, start_date, fertilizer (kg/ha), and irrigation (mm). The latitude, longitude, and start_date will be used to initialize the SWATGym to the specified location and date. With the initialization of the gym environment, the simulation will start at the specified location and return its weather information. The gym environment will then process the input of fertilizer and irrigation amount and return the state information to the endpoint. The endpoint will in turn return the reward, total_reward, current_date, number_of_episodes, and state information to the user. This process can be repeated up to 120 episodes. Once 120 episodes are reached, the total_reward will be returned to the user and the simulation will be reset.

4. 2 RACKY PPO agent API Design

4. 2. 1 Flowchart of RACKY PPO agent API

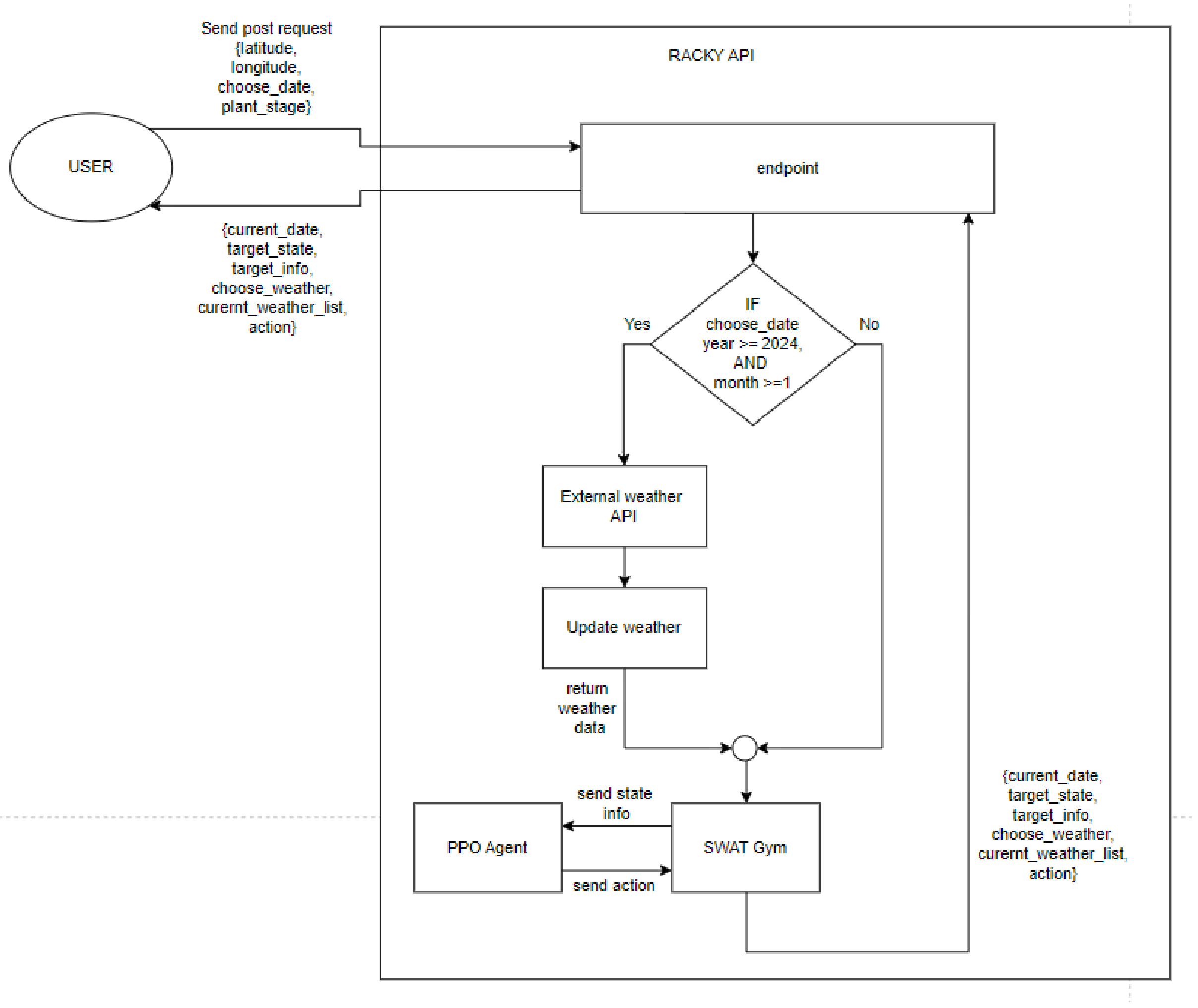

Send post request {latitude, longitude, choose_date, plant_stage}
USER
{current_date, target_state, target_info, choose_weather, curernt_weather_list, action}
RACKY API
endpoint
IF choose_date year >= 2024, AND month >=1
Yes
No
External weather API
Update weather
return weather data
send state info
send action
PPO Agent
SWAT Gym
{current_date, target_state, target_info, choose_weather, curernt_weather_list, action}

Figure 19: Flowchart of RACKY PPO Agent API

Above is the use case diagram of the RACKY PPO Agent API. Firstly, users are able to input information such as latitude, longitude, choose_date, and plant_stage. The latitude and longitude will be used to initialize the SWATGym to the specified location. The choose_date will be used to check if the year is greater than or equal to 2024 AND the month is greater than or equal to 1; if true, access the external weather API to return weather data; otherwise, use the original weather data as the gym environment does not provide weather data after the year 2024. Once the weather data is chosen, the PPO agent will receive state information from the initialized environment, and then the PPO Agent will return an action to the gym. The information regarding action, current_date, state info, current_weather, and choose_weather_list will be returned to the endpoint, and the endpoint will send it back to the user.

4.2.2 Flowchart of PPO Agent and SWATGym

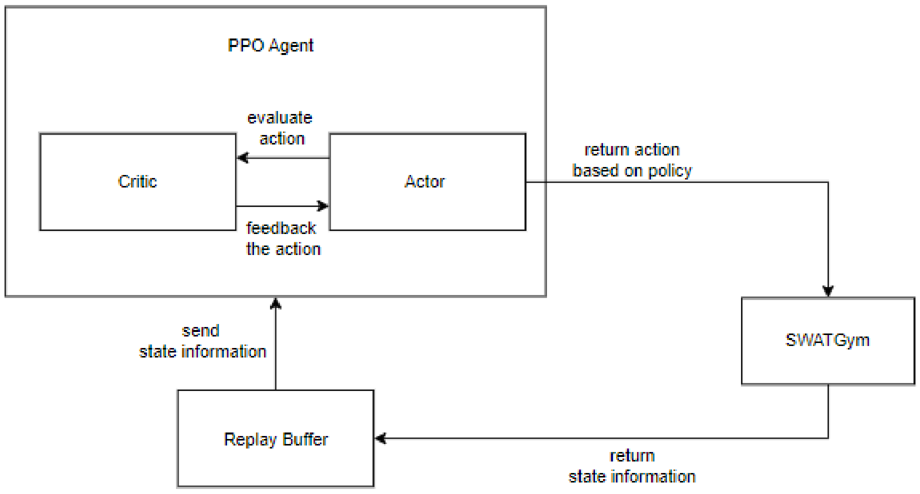


Figure 20: Flowchart of PPO Agent and SWATGym

The flow chart shows the integration of the Actor-Critic architecture within the Proximal Policy Optimization (PPO) agent, a reinforcement learning algorithm suited for environments with continuous action spaces. The Actor-Critic architecture consists of two main components, the Actor Network, and the Critic Network. The Actor Network learns the policy by mapping states to actions, while the Critic Network evaluates actions taken by the actor by estimating the value function. This dual-component setup allows for more efficient learning and decision-making. The flowchart show that the actor explores the environment and selects actions based on policy, while the critic provides feedback on action quality to guide policy improvements.

The training loop in the flow chart showcases how the PPO agent interacts with the SWATGym and collecting experiences and storing them in a replay buffer. Through periodic updates to its policy based on experiences sampled from the replay buffer, the PPO agent refines its decision-making processes. By leveraging the Actor-Critic architecture within the PPO agent, the flow chart highlights a structured approach to learning complex policies and making informed decisions in dynamic and challenging environments.

Chapter 5

System Implementation

5.1 Software Setup

5.1.1 Jupyter Notebook Setup & Laptop Specification

Jupyter notebook setup

Creating a new environment in anaconda navigator with Python 3.9. Next, install required library to enable using SWATGym in Anaconda console using conda command line. The libraries that required are notebook (Jupyter Notebook), NumPy, PCSE, PyTorch, Gym, Pandas, Matplotlib, Python Flask and Flask Cors.

Laptop Specification

The hardware involved in this project is a computer. A computer is used to develop user applications and reinforce learning AI system.

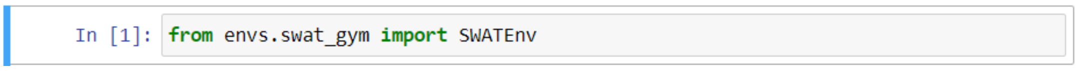
Table 3: Specification of Laptop

Description	Specifications
Model	MSI GF63
Processor	Intel Core i5-9300H
Operating System	Windows 11
Graphic	NVIDIA GeForce GTX 1650 4GB GDDR6
Memory	16GB DDR4 RAM
Storage	1TB SATA HDD + 230GB NVMe SSD

5.1.2 SWATGym Setup

The SWATGym simulator from Madondo, Azmat, et al is installed from the GitHub link[4]: GitHub - IBM/SWATgym: SWATgym: a reinforcement learning environment for crop management.

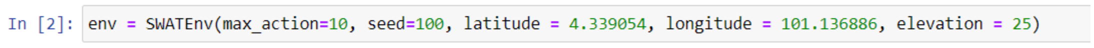
Then, import the SWATGym into the notebook:



```
In [1]: from envs.swat_gym import SWATEnv
```

Figure 21: Import SWAT code

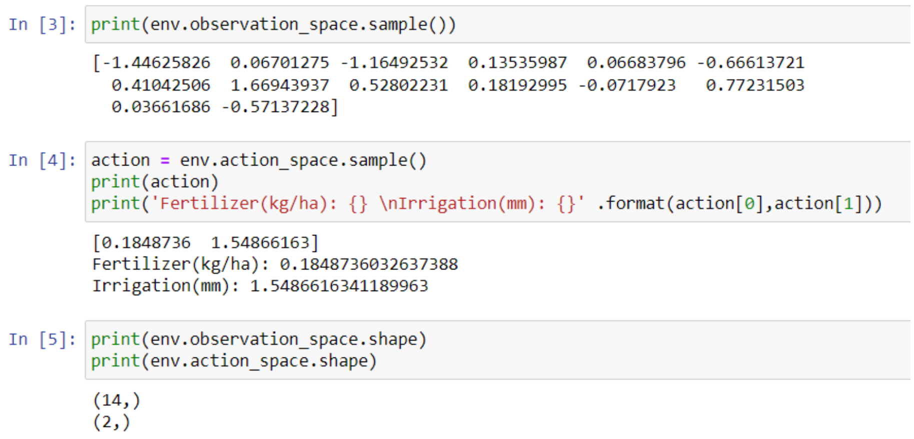
After import the SWATGym, the initialize of SWATGym is shown below:



```
In [2]: env = SWATEnv(max_action=10, seed=100, latitude = 4.339054, longitude = 101.136886, elevation = 25)
```

Figure 22: Define SWATEnv

In the initialization, max_action is the maximum value that the action space can be input into the simulator. Seed is used to provide consistency in later experiment. Latitude and longitude are set to the UTAR FICT area, and the elevation is set to 25.



```
In [3]: print(env.observation_space.sample())

        [-1.44625826  0.06701275 -1.16492532  0.13535987  0.06683796 -0.66613721
          0.41042506  1.66943937  0.52802231  0.18192995 -0.0717923   0.77231503
          0.03661686 -0.57137228]

In [4]: action = env.action_space.sample()
        print(action)
        print('Fertilizer(kg/ha): {} \nIrrigation(mm): {}' .format(action[0],action[1]))

        [0.1848736  1.54866163]
        Fertilizer(kg/ha): 0.1848736032637388
        Irrigation(mm): 1.5486616341189963

In [5]: print(env.observation_space.shape)
        print(env.action_space.shape)

        (14,)
        (2,)
```

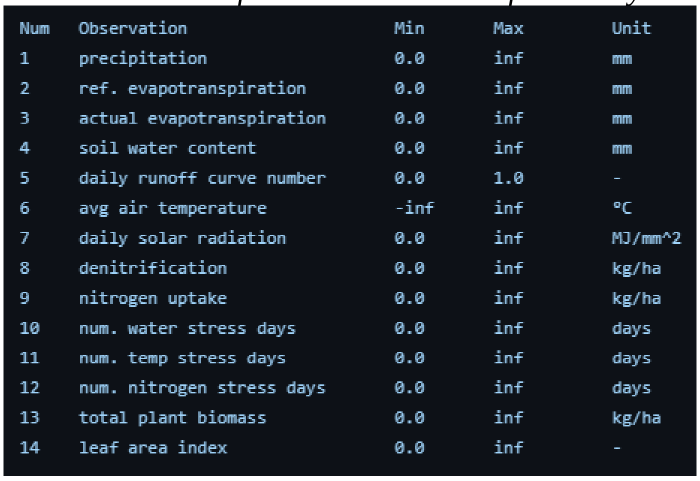
Figure 23: Example input of SWATEnv

The sample and shape are further determined.

The action space shape is 2 and continuous. The first element in the action space is the amount of fertilizer and calculated in kg/ha. The second element in the action space is the amount of irrigation and calculate in mm. Both of them are with the minimum of 0 and he maximum of max_action explained above.

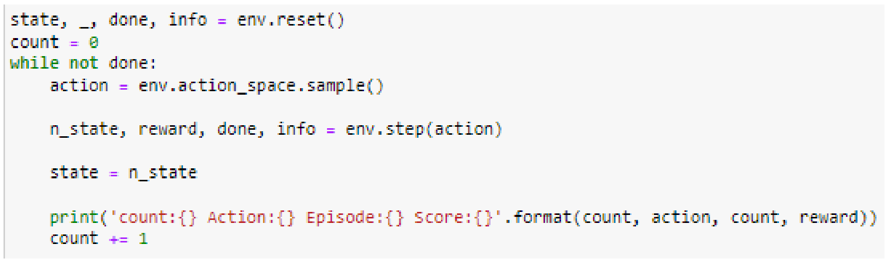
Besides, the observation space shape is 14 and continuous. The detail of the observation space is below[4]:

Table : Detail of Observation State of SWATGym

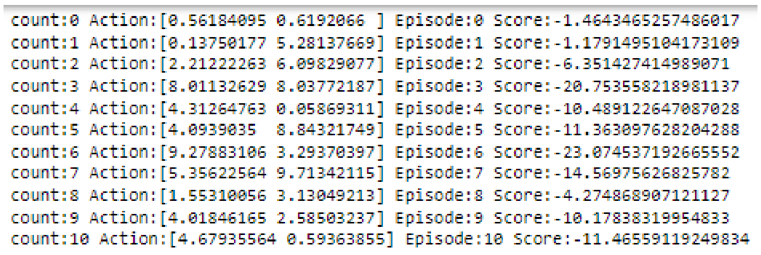


Num	Observation	Min	Max	Unit
1	precipitation	0.0	inf	mm
2	ref. evapotranspiration	0.0	inf	mm
3	actual evapotranspiration	0.0	inf	mm
4	soil water content	0.0	inf	mm
5	daily runoff curve number	0.0	1.0	-
6	avg air temperature	-inf	inf	°C
7	daily solar radiation	0.0	inf	MJ/mm^2
8	denitrification	0.0	inf	kg/ha
9	nitrogen uptake	0.0	inf	kg/ha
10	num. water stress days	0.0	inf	days
11	num. temp stress days	0.0	inf	days
12	num. nitrogen stress days	0.0	inf	days
13	total plant biomass	0.0	inf	kg/ha
14	leaf area index	0.0	inf	-

5.1.2.1 SWATGym Testing



```
state, _, done, info = env.reset()
count = 0
while not done:
    action = env.action_space.sample()

    n_state, reward, done, info = env.step(action)

    state = n_state

    print('count:{} Action:{} Episode:{} Score:{}'.format(count, action, count, reward))
    count += 1
```



```
count:0 Action:[0.56184095 0.6192066 ] Episode:0 Score:-1.4643465257486017
count:1 Action:[0.13750177 5.28137669] Episode:1 Score:-1.1791495104173109
count:2 Action:[2.21222263 6.09829077] Episode:2 Score:-6.351427414989071
count:3 Action:[8.01132629 8.03772187] Episode:3 Score:-20.753558218981137
count:4 Action:[4.31264763 0.05869311] Episode:4 Score:-10.489122647087028
count:5 Action:[4.0939035  8.84321749] Episode:5 Score:-11.363097628204288
count:6 Action:[9.27883106 3.29370397] Episode:6 Score:-23.074537192665552
count:7 Action:[5.35622564 9.71342115] Episode:7 Score:-14.56975626825782
count:8 Action:[1.55310056 3.13049213] Episode:8 Score:-4.274868907121127
count:9 Action:[4.01846165 2.58503237] Episode:9 Score:-10.17838319954833
count:10 Action:[4.67935564 0.59363855] Episode:10 Score:-11.46559119249834
```

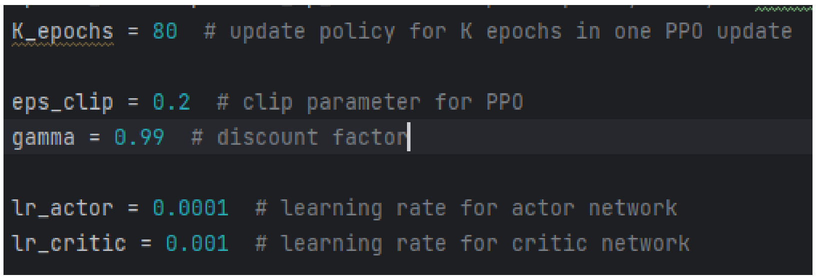
Figure 24: Random Test SWATGym

Random test is conducted to test SWATGym simulator. Random action is input into the simulator and the reward for each episode is show. Each episode contain 120 days in the simulator.

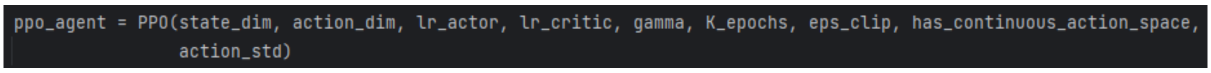
5.1.3 PPO Setup

Below are the pseudocode of PPO Agent that referred on Barhate and Nikhil's pytorch_minimal_ppo published on GitHub [11].

```
class RolloutBuffer:
    //act as memory to store all the previous state and action information

class ActorCritic:
    //actor and critic networks
    //evaluate action and state


class PPO:
    Initialize:
        Initialize buffer, policy, old policy, optimizer, and loss function

    Method set_action_std(new_action_std):
        Set the action standard deviation for continuous action space in policy
and old policy

    Method decay_action_std(action_std_decay_rate, min_action_std):
        Decrease action standard deviation with decay rate until minimum value

    Method select_action(state):
        Choose action based on the current policy and add data to the buffer

    Method update():
        Compute discounted rewards, normalize rewards, and calculate advantages
        Update policy for multiple epochs using PPO loss
        Copy updated policy weights to the old policy
        Clear buffer data
```



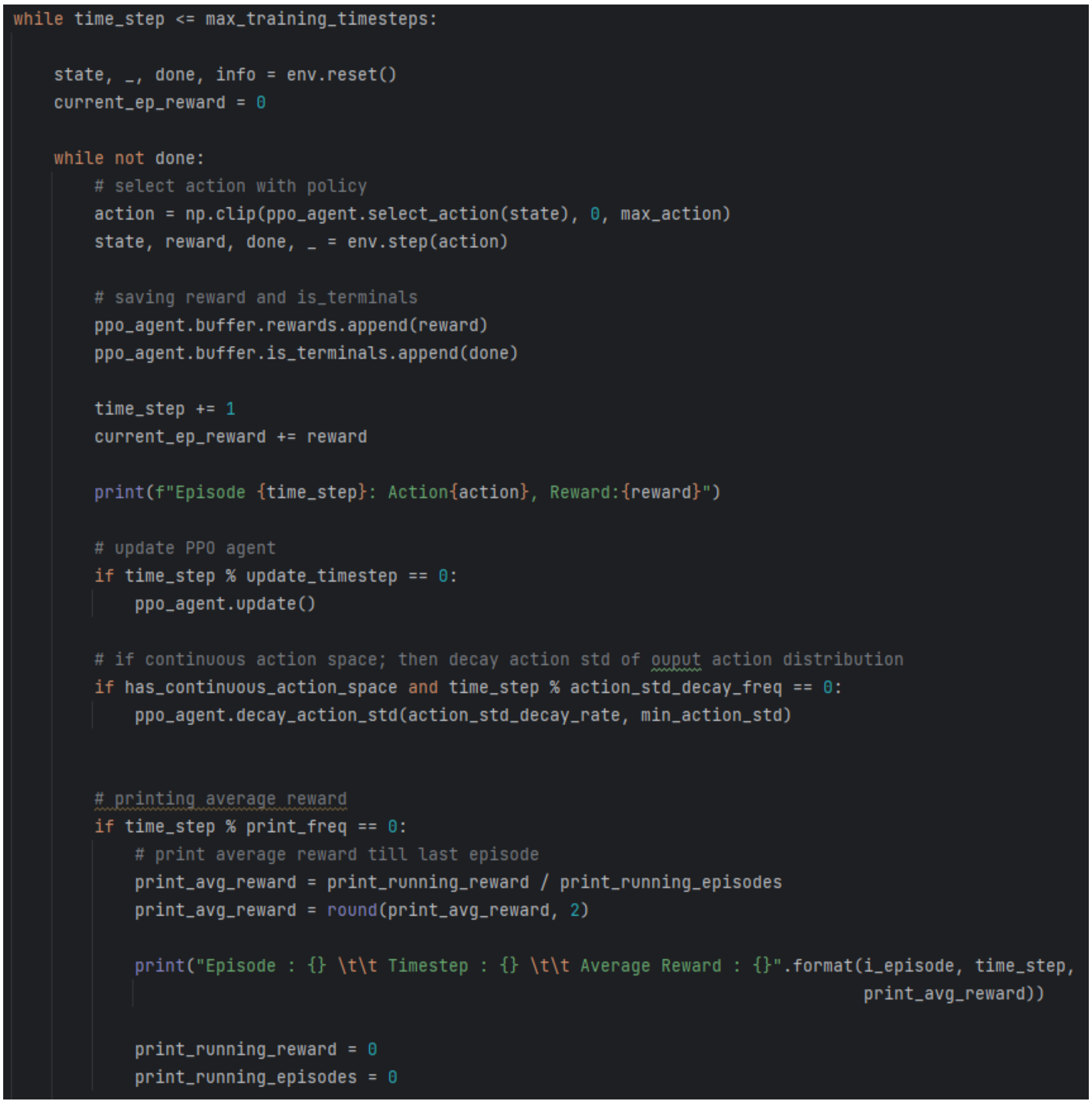
```
K_epochs = 80  # update policy for K epochs in one PPO update

eps_clip = 0.2  # clip parameter for PPO
gamma = 0.99  # discount factor

lr_actor = 0.0001  # learning rate for actor network
lr_critic = 0.001  # learning rate for critic network
```



```
ppo_agent = PPO(state_dim, action_dim, lr_actor, lr_critic, gamma, K_epochs, eps_clip, has_continuous_action_space,
                action_std)
```

Figure 25: Initialize for PPO Agent

Above picture show on the initialize of PPO Agent. The K_epochs parameter dictates the number of iterations the policy network undergoes during a single PPO update, impacting the depth of policy optimization. eps_clip serves to limit policy changes within each update, promoting stability and preventing drastic policy shifts with a value set at 0.2. The gamma parameter, set to 0.99, determines the importance of future rewards relative to immediate ones, influencing the agent's long-term planning abilities. Learning rates, lr_actor is 0.0001 and lr_critic is 0.001 for the actor and critic networks respectively guide the pace of learning for policy and value.



```
while time_step <= max_training_timesteps:

    state, _, done, info = env.reset()
    current_ep_reward = 0

    while not done:
        # select action with policy
        action = np.clip(ppo_agent.select_action(state), 0, max_action)
        state, reward, done, _ = env.step(action)

        # saving reward and is_terminals
        ppo_agent.buffer.rewards.append(reward)
        ppo_agent.buffer.is_terminals.append(done)

        time_step += 1
        current_ep_reward += reward

        print(f"Episode {time_step}: Action{action}, Reward:{reward}")

        # update PPO agent
        if time_step % update_timestep == 0:
            ppo_agent.update()

        # if continuous action space; then decay action std of ouput action distribution
        if has_continuous_action_space and time_step % action_std_decay_freq == 0:
            ppo_agent.decay_action_std(action_std_decay_rate, min_action_std)


        # printing average reward
        if time_step % print_freq == 0:
            # print average reward till last episode
            print_avg_reward = print_running_reward / print_running_episodes
            print_avg_reward = round(print_avg_reward, 2)

            print("Episode : {} \t\t Timestep : {} \t\t Average Reward : {}".format(i_episode, time_step,
                                                                                    print_avg_reward))

            print_running_reward = 0
            print_running_episodes = 0
```

Figure 26: training loop for PPO Agent

After initializing the PPO Agent, the provided code is used for training the agent. Figure 27 illustrates the training results, showing that after about 175k episodes of training, the PPO Agent's maximum reward plateaued at over 7000 rewards.

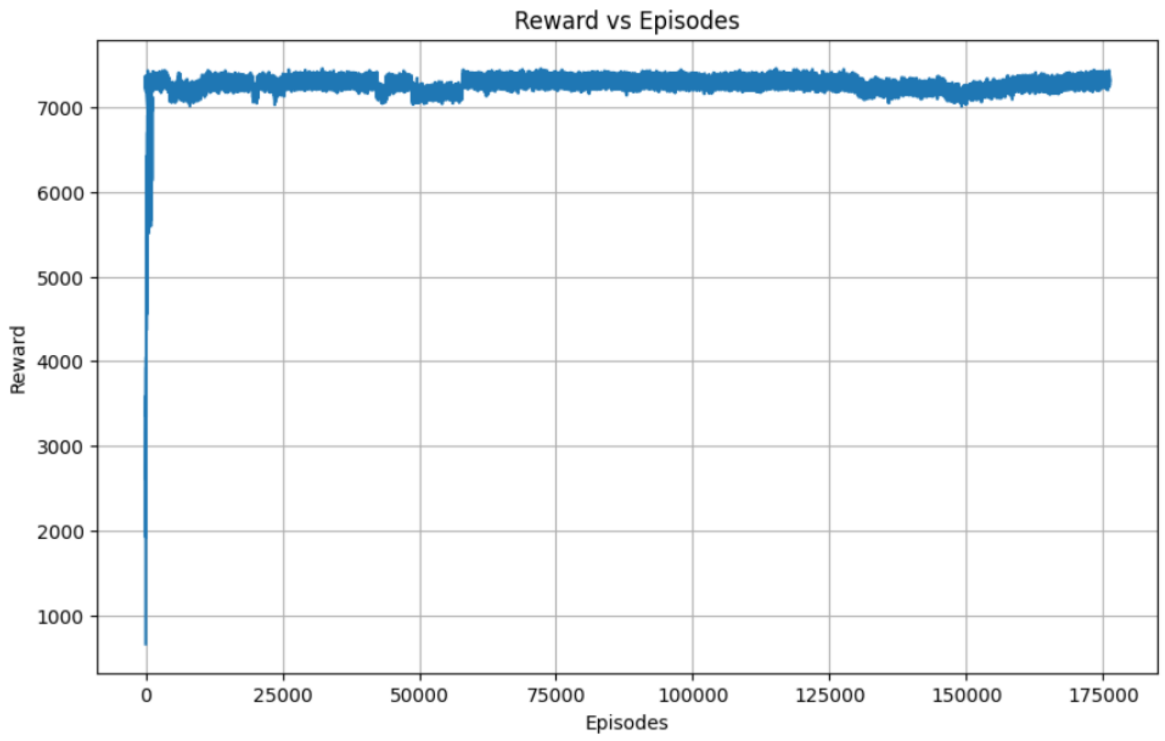


Figure 27: Reward vs episode

5.2 System Operation

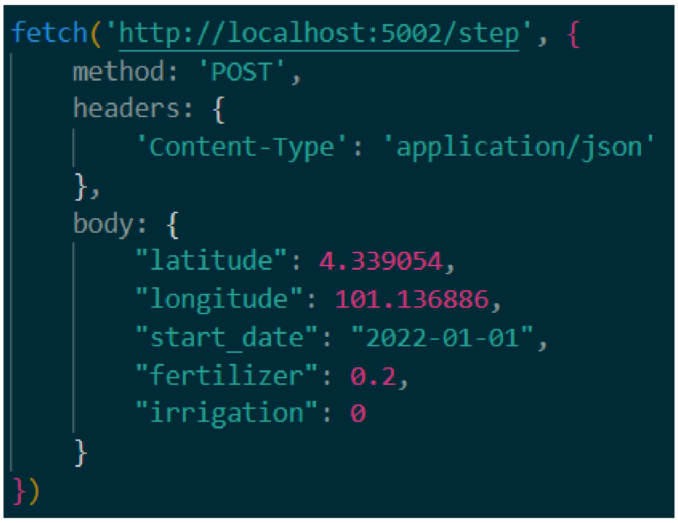
5.2.1 RACKY API

```
app = Flask(__name__)
CORS(app)
env = None  # Initialize as None initially
data_sent_counter = 0
total_reward = 0.0  # Initialize total reward

def initialize_env(latitude, longitude, start_date, seed=100):
    global env, total_reward, data_sent_counter
    env = SWATEnv(max_action=50, seed=seed, latitude=latitude, longitude=longitude, elevation=25)
    env.start_date = start_date
    env.end_date = env.start_date + timedelta(days=120)
    state, _, done, info = env.reset(seed=seed)  # Reset with the specified seed
    total_reward = 0.0  # Reset total reward
    data_sent_counter = 0
    return state, done

@app.route('/step', methods=['POST'])
def step():
    global state, done, env, data_sent_counter, total_reward

    if env is None:
        # Environment not initialized, so initialize it
        latitude = float(request.json['latitude'])
        longitude = float(request.json['longitude'])
        start_date = request.json['start_date']
        start_date = datetime.strptime(start_date, "%Y-%m-%d")
        seed = int(request.json.get('seed', 100))

        state, done = initialize_env(latitude, longitude, start_date, seed)
        return jsonify({"state": state, "reward": 0.0, "total_reward": total_reward, "done": done,
"data_sent_counter": data_sent_counter})

    if done:
        env = None
        return jsonify({"message": "Episode terminated. Reset environment to continue.",
"total_reward": total_reward, "data_sent_counter": data_sent_counter})
```

```
    fertilizer = int(request.json['fertilizer'])
    irrigation = int(request.json['irrigation'])
    action = (fertilizer, irrigation)

    n_state, reward, done, info = env.step(action)
    state = n_state
    total_reward += reward  # Accumulate total reward

    data_sent_counter += 1  # Increment data counter
    return jsonify({"state": state, "current_date": env.current_date, "reward": reward,
"total_reward": total_reward, "done": done, "data_sent_counter": data_sent_counter})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5002, debug=True)
```

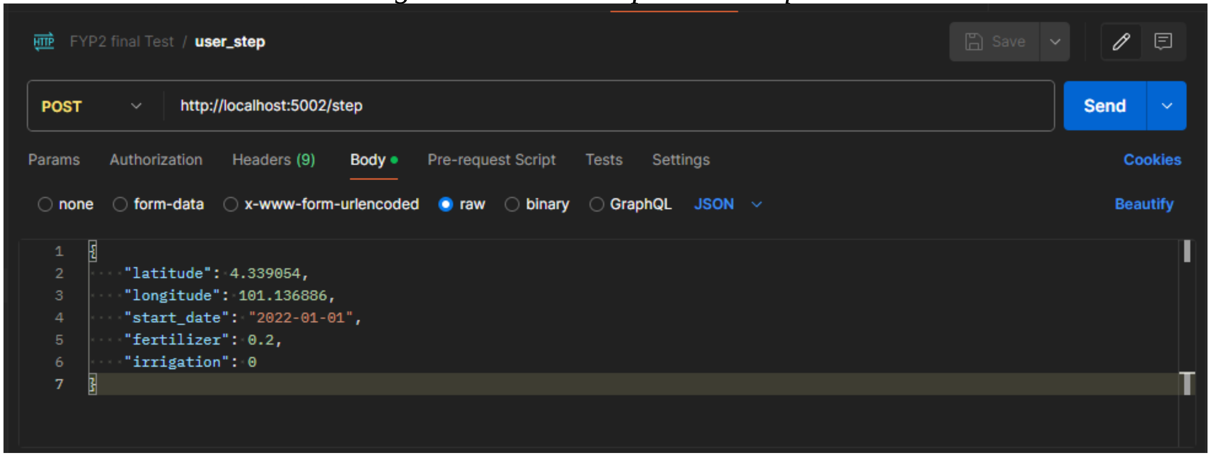
The code above demonstrates the implementation of the RACKY API. To execute the code, simply click 'run'. The API will be hosted on port 5002 on the localhost.

To access to the RACKY API user need to use post method. Example of POST method to call the API is as below:



```
fetch('http://localhost:5002/step', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: {
        "latitude": 4.339054,
        "longitude": 101.136886,
        "start_date": "2022-01-01",
        "fertilizer": 0.2,
        "irrigation": 0
    }
})
```

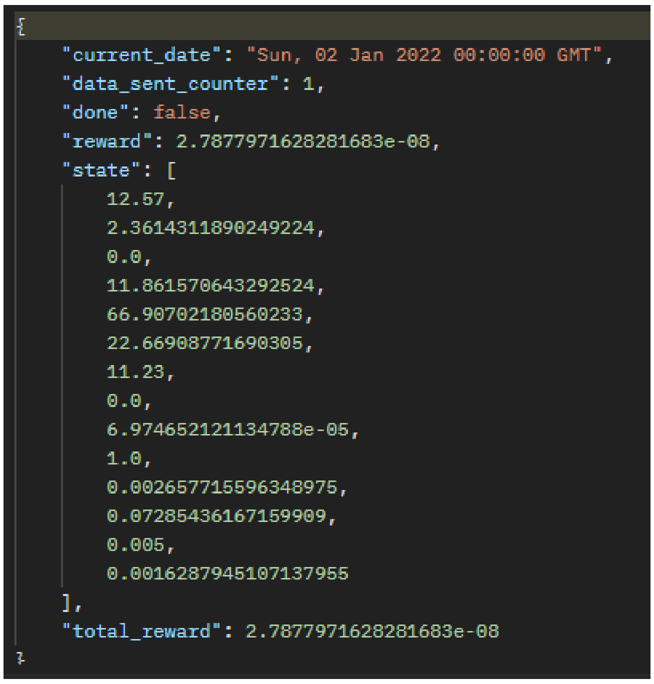
Figure 28: Java Script POST request



```
{
    "latitude": 4.339054,
    "longitude": 101.136886,
    "start_date": "2022-01-01",
    "fertilizer": 0.2,
    "irrigation": 0
}
```

Figure 29: Postman POST request

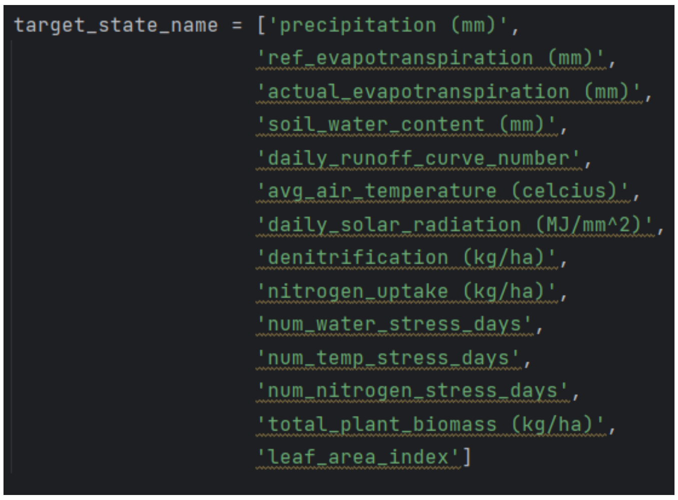
Above figure 28 and figure 29 had shown the way to call the API by using Java Script and Postman. After successfully access to the API, the API will start to process the input json information and return the result as below:



```
{
    "current_date": "Sun, 02 Jan 2022 00:00:00 GMT",
    "data_sent_counter": 1,
    "done": false,
    "reward": 2.7877971628281683e-08,
    "state": [
        12.57,
        2.3614311890249224,
        0.0,
        11.861570643292524,
        66.90702180560233,
        22.66908771690305,
        11.23,
        0.0,
        6.974652121134788e-05,
        1.0,
        0.002657715596348975,
        0.07285436167159909,
        0.005,
        0.0016287945107137955
    ],
    "total_reward": 2.7877971628281683e-08
}
```

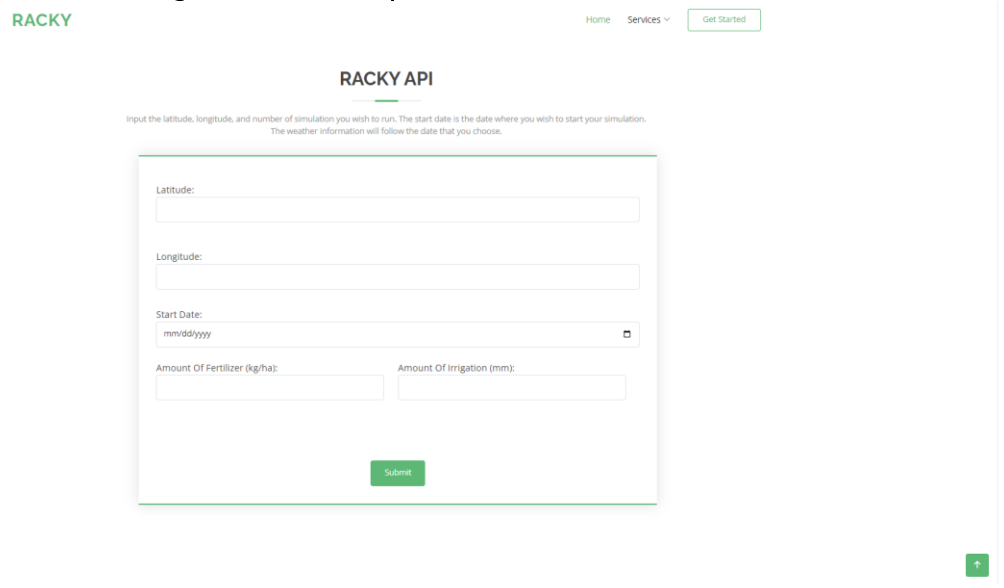
Figure 30: Output of RACKY API

The current_date is the date on which the simulator is running. The data_send_counter represents the days the user has been running the simulation. The 'done' variable is used to check if the simulation is completed. The reward indicates the current reward for the user's action. The state represents the observation state returned by the SWATGym, containing crop information and weather data. Lastly, the total_reward is the accumulated reward for the simulation. Once the simulation has run to completion, it will be initialized based on the user's input again, and a new simulation will start. The specific name of each value is shown in Figure 31. The showcase of RACKY API show in the Figure 32 below.



```
target_state_name = ['precipitation (mm)',
                     'ref_evapotranspiration (mm)',
                     'actual_evapotranspiration (mm)',
                     'soil_water_content (mm)',
                     'daily_runoff_curve_number',
                     'avg_air_temperature (celcius)',
                     'daily_solar_radiation (MJ/mm^2)',
                     'denitrification (kg/ha)',
                     'nitrogen_uptake (kg/ha)',
                     'num_water_stress_days',
                     'num_temp_stress_days',
                     'num_nitrogen_stress_days',
                     'total_plant_biomass (kg/ha)',
                     'leaf_area_index']
```

Figure 31: Name of each return values



RACKY

Home Services Get Started

RACKY API

Input the latitude, longitude, and number of simulation you wish to run. The start date is the date where you wish to start your simulation. The weather information will follow the date that you choose.

Latitude:

Longitude:

Start Date: mm/dd/yyyy

Amount Of Fertilizer (kg/ha):

Amount Of Irrigation (mm):

Submit

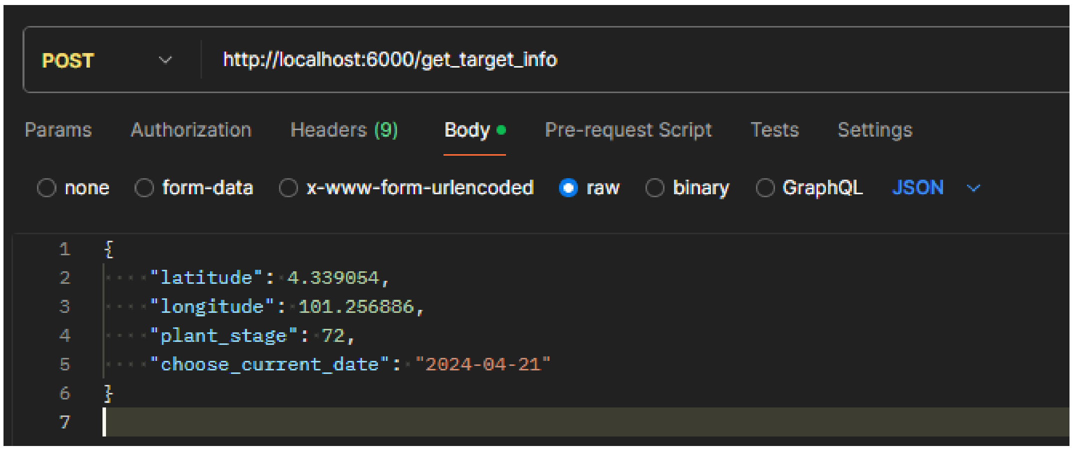
Figure 32: Showcase for RACKY API

5.2.2 RACKY API PPO Agent

```
app = Flask(__name__)
CORS(app)

def get_weather_data(latitude, longitude, date):
    url = f"https://archive-api.open-
meteo.com/v1/archive?latitude={latitude}&longitude={longitude}&start_date={date}&end_date={date}&hou
rly=temperature_2m,precipitation,relative_humidity_2m,soil_temperature_0_to_7cm,soil_moisture_0_to_7
cm&daily=temperature_2m_mean,precipitation_sum,shortwave_radiation_sum,et0_fao_evapotranspiration&ti
mezone=Asia%2FSingapore"

    response = requests.get(url)
    data = response.json()
    return data

def calculate_vapor_pressure(temperature_C, relative_humidity):
    # Constants for Magnus-Tetens formula
    A = 17.27
    B = 237.7  # Temperature offset
    # Convert temperature to Kelvin
    temperature_K = temperature_C + 273.15
    # Calculate saturation vapor pressure (SVP) in kPa
    svp = 6.112 * 10**((A * temperature_C) / (B + temperature_C))
    # Calculate actual vapor pressure (AVP) in kPa
    avp_kpa = relative_humidity * svp
    # Convert AVP from kPa to hPa (divide by 10)
    avp_hpa = avp_kpa / 10
    return avp_hpa

def calculate_soil_evaporation(soil_temp_C, soil_moisture_m3m3, et0_mm_day):
    # Convert ET₀ from mm/day to m/day
    et0_m_day = et0_mm_day * 0.001

    # Assume average values for Crop Coefficient (Kc) and Soil Moisture Stress Factor (Fs)
    kc = 1  # Crop Coefficient
    fs = 0.8  # Soil Moisture Stress Factor

    # Calculate Soil Evaporation
    soil_evaporation_m_day = et0_m_day * kc * fs * soil_moisture_m3m3

    # Convert soil evaporation from m/day to mm/day
    soil_evaporation_mm_day = soil_evaporation_m_day * 1000

    return soil_evaporation_mm_day

def check_thresholds(weather_list, current_list, thresholds):
    if len(weather_list) != len(current_list) or len(weather_list) != len(thresholds):
        raise ValueError("Lists must have the same length.")

    for i in range(len(weather_list)):
        if abs(weather_list[i] - current_list[i]) > thresholds[i]:
            #print(f"wrong: {i}, value:{abs(weather_list[i] - current_list[i])}")
            return False

    return True

def copy_state_info(longitude, latitude, plant_stage, start_date, end_date, target_current_date):
    env = SWATEnv(max_action=10, latitude = latitude, longitude = longitude, elevation = 25)
    env.start_date = start_date
    env.end_date = end_date
    state, _, done, info = env.reset(seed=100)
    target_state_info = []
    target_info = []

    while not done:
        current_date = env.current_date
        action = env.action_space.sample()
        next_state, reward, done, info = env.step(action)

        if current_date == target_current_date:
            target_state_info.append(state)
            target_info.append(info)
            break

        state = next_state

    env.close()

    return target_state_info, target_info

def get_same_weather_date(longitude, latitude, choose_current_date):

    weather_data = get_weather_data(latitude, longitude, choose_current_date)

    humidity_mean = np.mean(weather_data['hourly']['relative_humidity_2m'])/100
    soil_temp_mean = np.mean(weather_data['hourly']['soil_temperature_0_to_7cm'])
    soil_mois_mean = np.mean(weather_data['hourly']['soil_moisture_0_to_7cm'])

    temperature_mean = weather_data['daily']['temperature_2m_mean'][0]
    precipitation_sum_daily = weather_data['daily']['precipitation_sum'][0]
    shortwave_radiation_sum_daily = weather_data['daily']['shortwave_radiation_sum'][0]
    et0_fao_evapotranspiration_daily = weather_data['daily']['et0_fao_evapotranspiration'][0]
    vapor_pressure = calculate_vapor_pressure(temperature_mean, humidity_mean)
    soil_evaporation = calculate_soil_evaporation(soil_temp_mean, soil_mois_mean,
et0_fao_evapotranspiration_daily)
```

```
    current_list = [temperature_mean, shortwave_radiation_sum_daily, vapor_pressure,
precipitation_sum_daily, et0_fao_evapotranspiration_daily, soil_evaporation]
    thresholds = [3.5, 3.5, 3.5, 3.5, 3.5, 3.5]

    start_date_find = datetime.strptime('20230924', '%Y%m%d')
    date_list = []
    weather_accepted = []
    count_threshold = 1

    while len(date_list) < count_threshold:
        env = SWATEnv(max_action=10, latitude = latitude, longitude = longitude, elevation = 25)
        env.start_date = start_date_find
        env.end_date = start_date_find + timedelta(days=120)

        state, _, done, info = env.reset(seed=100)
        episode_reward = 0
        count = 0

        while not done:
            current_date = env.current_date
            weather_list = [env.avg_temp, env.solar_rad, env.avg_vapor_pressure,
                            env.precip, env.ref_et, env.soil_evap]

            action = env.action_space.sample()
            next_state, reward, done, info = env.step(action)

            # Compare weather_list and current_list
            all_within_threshold = check_thresholds(weather_list, current_list, thresholds)
            if all_within_threshold:
                date_list.append(current_date.strftime('%Y%m%d'))
                weather_accepted.append(weather_list)
                #print(f'found date: {current_date}')

            episode_reward += reward
            state = next_state
            count += 1

        # Move to the next start date
        start_date_find = start_date_find - timedelta(days=121)

    env.close()

    return date_list, weather_accepted, current_list

def find_stage_index(input_num, stage_ranges):
    for idx, stage_range in enumerate(stage_ranges):
        start, end = stage_range
        if start <= input_num <= end:
            return idx
    return 5

@app.route('/api/get_target_info', methods=['POST'])
def get_target_info():
    data = request.get_json()
    latitude = data.get('latitude')
    longitude = data.get('longitude')
    plant_stage = data.get('plant_stage')
    choose_current_date = data.get('choose_current_date')

    date_list, weather_list, current_waether_list = get_same_weather_date(longitude, latitude,
choose_current_date)
    choose_date = date_list[0]
    choose_weather = weather_list[0]

    stage_ranges = [(0, 14), (15, 29), (30, 44), (45, 59), (60, 74), (75, 89), (90, 104), (105,
119)]
    stage_midpoints = [7, 22, 37, 52, 67, 82, 97, 112]

    target_current_date = datetime.strptime(choose_date, '%Y%m%d')
    # index = find_stage_index(plant_stage, stage_ranges)
    # current_stage = stage_midpoints[index]

    target_start_date = target_current_date - timedelta(days=plant_stage)
    target_end_date = target_start_date + timedelta(days=120)

    target_state, target_info = copy_state_info(longitude, latitude, plant_stage, target_start_date,
target_end_date, target_current_date)


####################################################################################################
#####
    env_name = "SWATEnv"
    max_action = 5
    elevation = 25
    env = SWATEnv(max_action=max_action, latitude=latitude, longitude=longitude,
elevation=elevation)

    state_dim = env.observation_space.shape[0]
    action_dim = env.action_space.shape[0]
    lr_actor = 0.0001
    lr_critic = 0.001
    gamma = 0.99
    K_epochs = 80
    eps_clip = 0.2
    action_std = 0.6
    has_continuous_action_space = True

    ppo_agent = PPO(state_dim, action_dim, lr_actor, lr_critic, gamma, K_epochs, eps_clip,
has_continuous_action_space,
                    action_std)

    random_seed = 0
    num_load = 1

    directory = "RACKY_API/SWAT_gym" + '/'
    checkpoint_path_load = directory + "PPO_{}_{}_{}.pth".format(env_name, random_seed, num_load)
    print("loading network from : " + checkpoint_path_load)

    ppo_agent.load(checkpoint_path_load)


####################################################################################################
########################

    state, _, done, info = env.reset(seed=100)

    while not done:
        state = target_state[0]
        info = target_info[0]
        action = np.clip(ppo_agent.select_action(state), 0, max_action)
        state, reward, done, _ = env.step(action)

        done = True

        if done:
            break

    # clear buffer
    ppo_agent.buffer.clear()
    env.close()

    # Create JSON response
    response_data = {
        'target_current_date': target_current_date,
        'target_state': target_state,
        'target_info': target_info,

        'choose_weather': choose_weather,
        'current_weather_list': current_waether_list,

        'action': action.tolist()
    }

    return jsonify(response_data)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Above are the implementation of RACKY PPO Agent API. To access the RACKY PPO Agent API user need to use POST method with body information as below. The latitude and longitude are used to define the current location and fetch the current weather for the simulation. The plant_stage is the date where the plant grows till, the maximum number for plant_stage is 120.

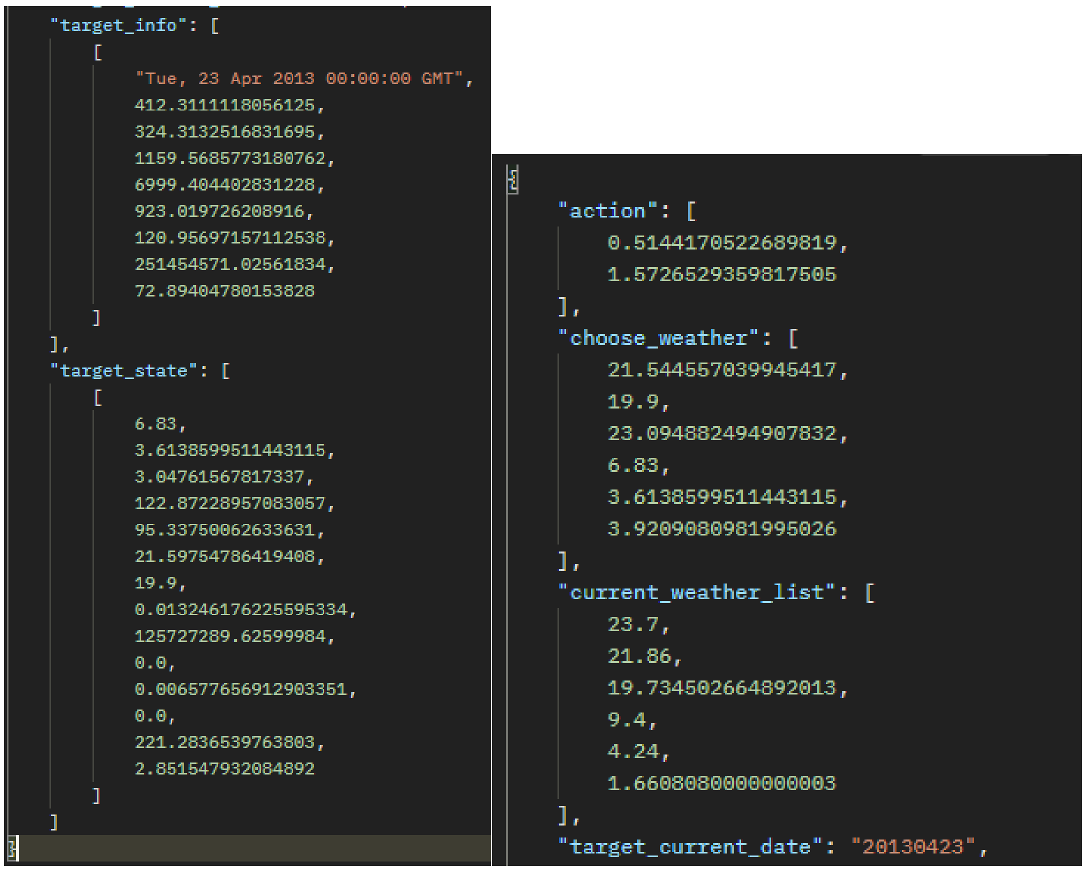

POST http://localhost:6000/get_target_info

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{
    "latitude": 4.339054,
    "longitude": 101.256886,
    "plant_stage": 72,
    "choose_current_date": "2024-04-21"
}
```

Figure 33: POST request for RACKY PPO Agent API

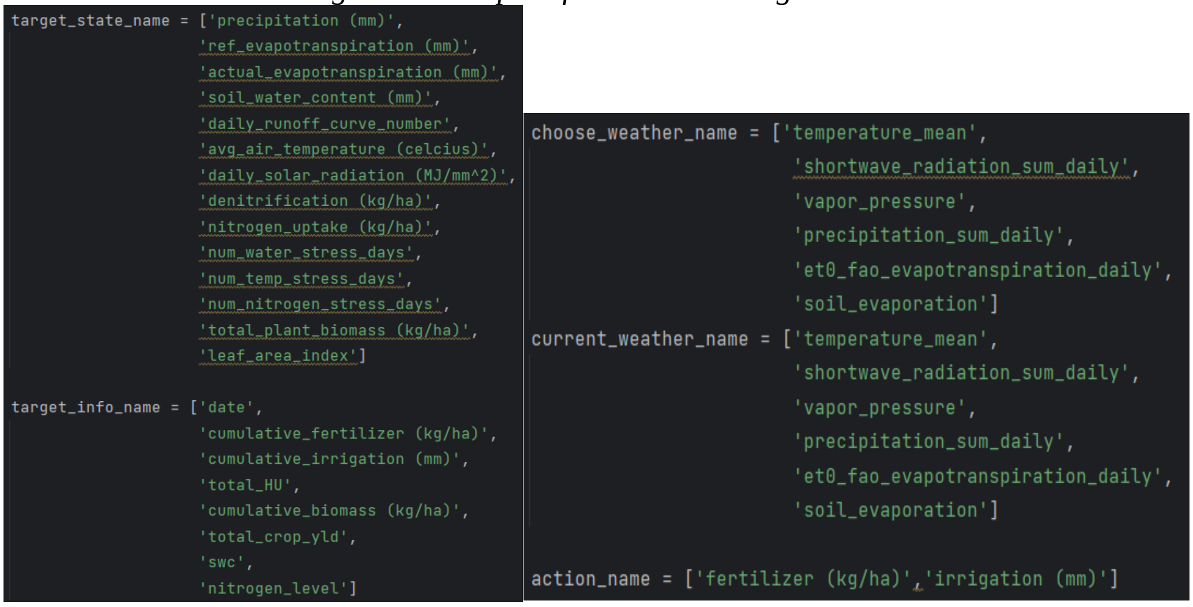
After sending the POST request to the RACKY PPO Agent API, it will provide the following output in Figure 34. The suggested applied amount of fertilizer and irrigation to the crop will be included in the response. The `current_weather_list` contains information about the current weather conditions. To assess the plant's condition accurately, a matching weather condition will be selected from SWATGym for that specific date. The resulting weather data will be stored in `choose_weather`, plant information in `target_state`, and additional details in `target_info`. The name of each value return is shown in Figure 35. For example, action [0] is fertilizer (kg/ha), action[1] is irrigation (mm). Lastly, the showcase for RACKY PPO Agent API is show in the Figure 36 below.



```
    "target_info": [
        [
            "Tue, 23 Apr 2013 00:00:00 GMT",
            412.3111118056125,
            324.3132516831695,
            1159.5685773180762,
            6999.404402831228,
            923.019726208916,
            120.95697157112538,
            251454571.02561834,
            72.89404780153828
        ]
    ],
    "target_state": [
        [
            6.83,
            3.6138599511443115,
            3.04761567817337,
            122.87228957083057,
            95.33750062633631,
            21.59754786419408,
            19.9,
            0.013246176225595334,
            125727289.62599984,
            0.0,
            0.006577656912903351,
            0.0,
            221.2836539763803,
            2.851547932084892
        ]
    ]
}
```

```
{
    "action": [
        0.5144170522689819,
        1.5726529359817505
    ],
    "choose_weather": [
        21.544557039945417,
        19.9,
        23.094882494907832,
        6.83,
        3.6138599511443115,
        3.9209080981995026
    ],
    "current_weather_list": [
        23.7,
        21.86,
        19.734502664892013,
        9.4,
        4.24,
        1.6608080000000003
    ],
    "target_current_date": "20130423",
```

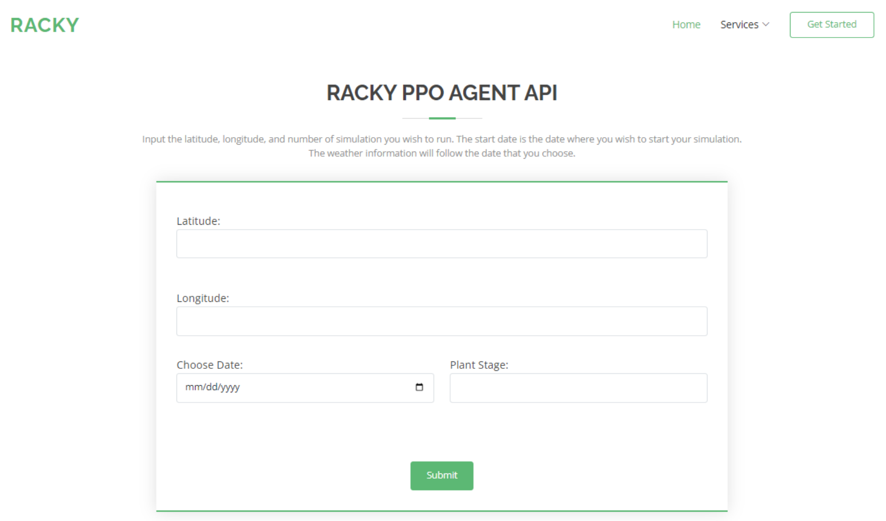
Figure 34: Output of RACKY PPO Agent API



```
target_state_name = ['precipitation (mm)',
                     'ref_evapotranspiration (mm)',
                     'actual_evapotranspiration (mm)',
                     'soil_water_content (mm)',
                     'daily_runoff_curve_number',
                     'avg_air_temperature (celcius)',
                     'daily_solar_radiation (MJ/mm^2)',
                     'denitrification (kg/ha)',
                     'nitrogen_uptake (kg/ha)',
                     'num_water_stress_days',
                     'num_temp_stress_days',
                     'num_nitrogen_stress_days',
                     'total_plant_biomass (kg/ha)',
                     'leaf_area_index']

target_info_name = ['date',
                     'cumulative_fertilizer (kg/ha)',
                     'cumulative_irrigation (mm)',
                     'total_HU',
                     'cumulative_biomass (kg/ha)',
                     'total_crop_yld',
                     'swc',
                     'nitrogen_level']
```

```
choose_weather_name = ['temperature_mean',
                       'shortwave_radiation_sum_daily',
                       'vapor_pressure',
                       'precipitation_sum_daily',
                       'et0_fao_evapotranspiration_daily',
                       'soil_evaporation']
current_weather_name = ['temperature_mean',
                       'shortwave_radiation_sum_daily',
                       'vapor_pressure',
                       'precipitation_sum_daily',
                       'et0_fao_evapotranspiration_daily',
                       'soil_evaporation']


action_name = ['fertilizer (kg/ha)','irrigation (mm)']
```

Figure 35: Name of each return value RACKY PPO Agent API


RACKY

Home Services Get Started

RACKY PPO AGENT API

Input the latitude, longitude, and number of simulation you wish to run. The start date is the date where you wish to start your simulation. The weather information will follow the date that you choose.

Latitude:

Longitude:

Choose Date: mm/dd/yyyy

Plant Stage:

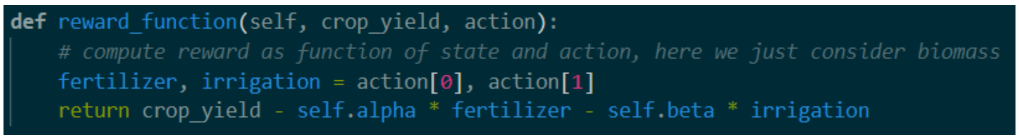
Submit

Figure 36: Showcase of RACKY PPO Agent API

5.3 *Issue and Challenge*

5.3.1 The model return action (0, 0)

After completing the model training, certain actions return as (0,0) due to the reward function within the SWATGym environment. Below is the code for the reward function.



```
def reward_function(self, crop_yield, action):
    # compute reward as function of state and action, here we just consider biomass
    fertilizer, irrigation = action[0], action[1]
    return crop_yield - self.alpha * fertilizer - self.beta * irrigation
```

Figure 37: SWATGym reward function

The model identifies actions as detrimental to crop yield, resulting in reduced rewards. To enhance the training process and ensure model reliability, I implemented a function to penalize crop yield when action (0, 0) is detected. This function is temporary and is removed from SWATGym after training completion.

Despite modifying the training process for the PPO agent, actions are still prevalent in the early plant stage and are less observed in later timesteps. This is because actions can quickly lead to negative rewards due to the sigmoid-like distribution of crop yield rewards shown in the figure below, depicting potential rewards across timesteps.

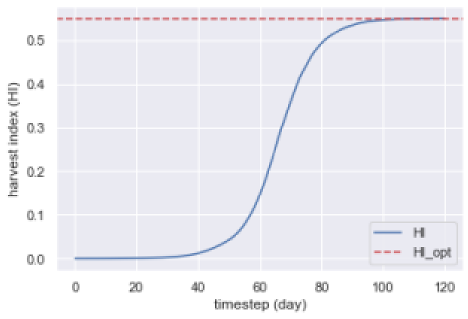
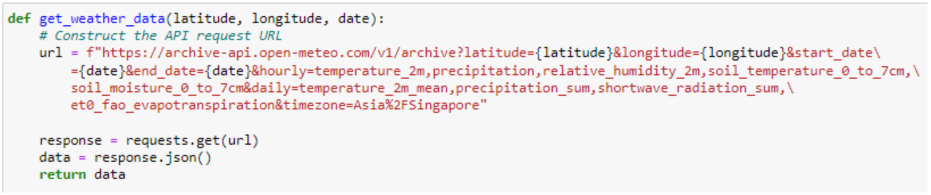


Figure 38: SWATGym potential reward in each timestep

5.3.2 Weather data is not provided

To obtain accurate weather information, SWATGym utilizes an external weather API as the current weather API it relies on is not updated to the current date. The code below demonstrates how to call the weather API, which is sourced from Zippenfenig's Open-Meteo.com Weather API [12].



```
def get_weather_data(latitude, longitude, date):
    # Construct the API request URL
    url = f"https://archive-api.open-meteo.com/v1/archive?latitude={latitude}&longitude={longitude}&start_date\
        ={date}&end_date={date}&hourly=temperature_2m,precipitation,relative_humidity_2m,soil_temperature_0_to_7cm,\
        soil_moisture_0_to_7cm&daily=temperature_2m_mean,precipitation_sum,shortwave_radiation_sum,\
        et0_fao_evapotranspiration&timezone=Asia%2FSingapore"

    response = requests.get(url)
    data = response.json()
    return data
```

Figure 39:External weather API

5.4 Remarks

The RACKY API, implemented using Python Flask on port 5002, can be accessed via POST requests with longitude, latitude, start date, fertilizer and irrigation to retrieve output such as current date, reward, observation state, and total reward. This API resets the simulation after completion, initializing based on user input for a new simulation.

On the other hand, the RACKY PPO Agent API, implemented using Python Flask on port 5000, also accessed through POST requests with latitude, longitude, plant stage, and current date information then provides recommendations for fertilizer and irrigation amounts, current weather conditions, selected weather data, plant information, and additional details.

However, challenges such as **actions returning as (0,0)** due to reward function limitations in SWATGym have been addressed through temporary penalty functions during training to penalize such actions. Furthermore, **weather data is sourced from an external API** due to limitations in the current weather API used by SWATGym, ensuring accurate and up-to-date information for realistic simulation environments. These APIs serve as powerful tools for agricultural optimization and decision-making processes, bridging the gap between simulation and real-world agricultural practices by providing actionable insights based on environmental and crop-specific data.

Chapter 6

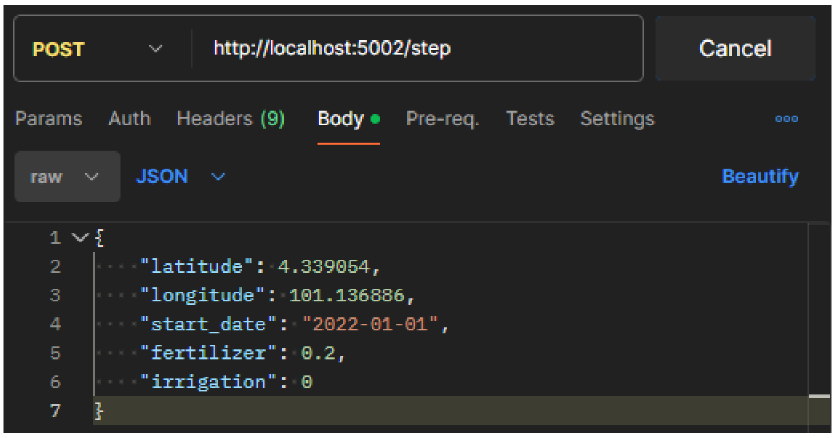
System Evaluation

6.1 System Testing

6.1.1 Test RACKY API

The test will be tested on 2 different locations in Perak which is Ipoh and Kampar with detail of latitude and longitude as below.

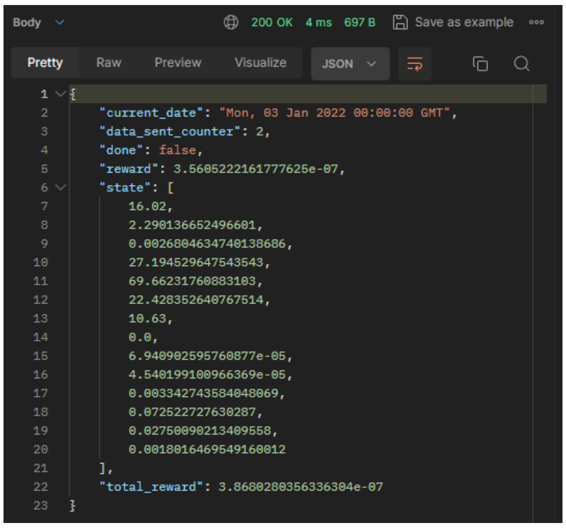
Kampar (4.339054 101.136886) (2022-01-01)



```
{
    "latitude": 4.339054,
    "longitude": 101.136886,
    "start_date": "2022-01-01",
    "fertilizer": 0.2,
    "irrigation": 0
}
```

Figure 40: POST request for RACKY API Kampar test

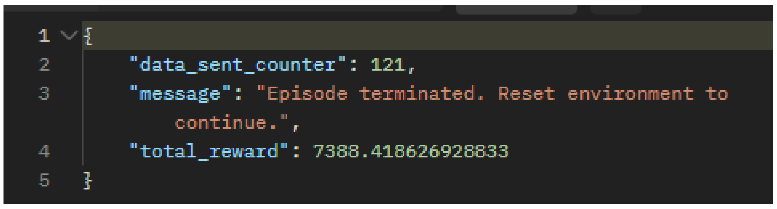
Request as above is send to the RACKY API as Figure 40 above.



```
{
    "current_date": "Mon, 03 Jan 2022 00:00:00 GMT",
    "data_sent_counter": 2,
    "done": false,
    "reward": 3.5605222161777625e-07,
    "state": [
        16.02,
        2.290136652496601,
        0.0026804634740138686,
        27.194529647543543,
        69.66231760883103,
        22.428352640767514,
        10.63,
        0.0,
        6.940902595760877e-05,
        4.540199100966369e-05,
        0.003342743584048069,
        0.072522727630287,
        0.02750090213409558,
        0.0018016469549160012
    ],
    "total_reward": 3.8680280356336304e-07
}
```

Figure 41: output result for RACKY API Kampar test

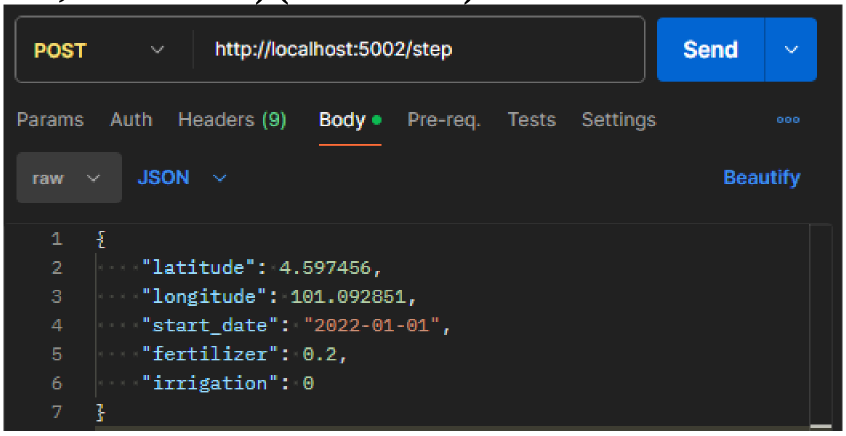
After few requests keep sending to the API, example of output will be shown like above Figure. Once the simulation end, it shows the total reward like in figure 42 below.



```
{
    "data_sent_counter": 121,
    "message": "Episode terminated. Reset environment to
        continue.",
    "total_reward": 7388.418626928833
}
```

Figure 42: simulation termination for RACKY API Kampar test

Ipoh (4.597456, 101.092851) (2022-01-01)

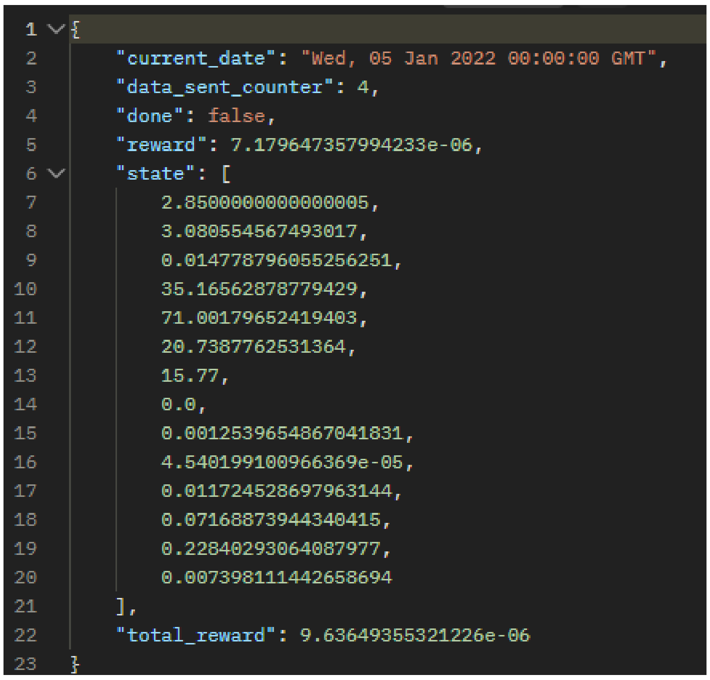


POST http://localhost:5002/step

```
{
    "latitude": 4.597456,
    "longitude": 101.092851,
    "start_date": "2022-01-01",
    "fertilizer": 0.2,
    "irrigation": 0
}
```

Figure 43: POST request for RACKY API Ipoh test

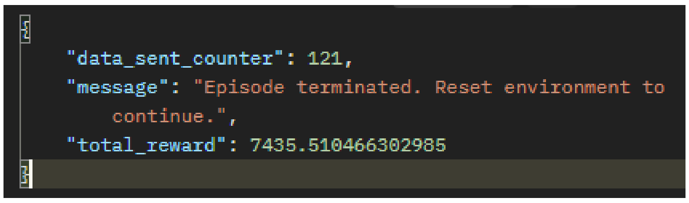
Request as above is send to the RACKY API.



```
{
    "current_date": "Wed, 05 Jan 2022 00:00:00 GMT",
    "data_sent_counter": 4,
    "done": false,
    "reward": 7.179647357994233e-06,
    "state": [
        2.8500000000000005,
        3.080554567493017,
        0.014778796055256251,
        35.16562878779429,
        71.00179652419403,
        20.7387762531364,
        15.77,
        0.0,
        0.0012539654867041831,
        4.540199100966369e-05,
        0.011724528697963144,
        0.07168873944340415,
        0.22840293064087977,
        0.007398111442658694
    ],
    "total_reward": 9.63649355321226e-06
}
```

Figure 44: output result for RACKY API Ipoh test

After few requests keep sending to the API, example of output will be shown like above Figure 44. Once the simulation end, it shows the total reward like in Figure 45 below.



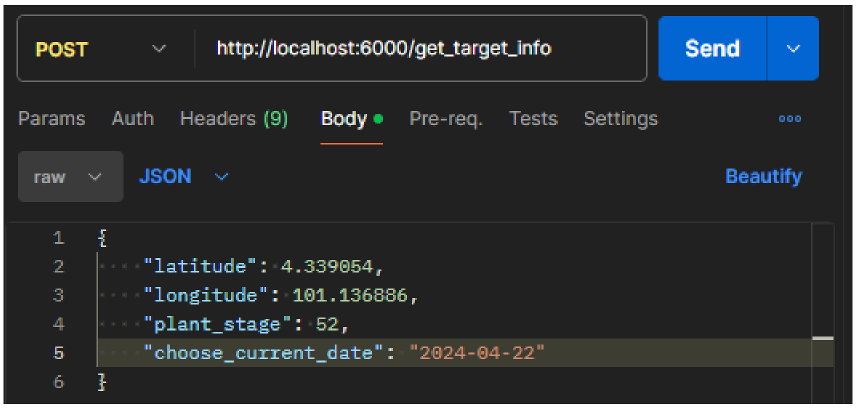
```
{
    "data_sent_counter": 121,
    "message": "Episode terminated. Reset environment to
        continue.",
    "total_reward": 7435.510466302985
}
```

Figure 45: simulation termination for RACKY API Ipoh test

6.1.2 Test RACKY PPO agent API

The following tests evaluate the RACKY API using various plant stages and dates, including the current date and other specified dates in Kampar (latitude 4.339054, longitude 101.136886).

Tested on current date (2024-04-22) with plant_stage of 52

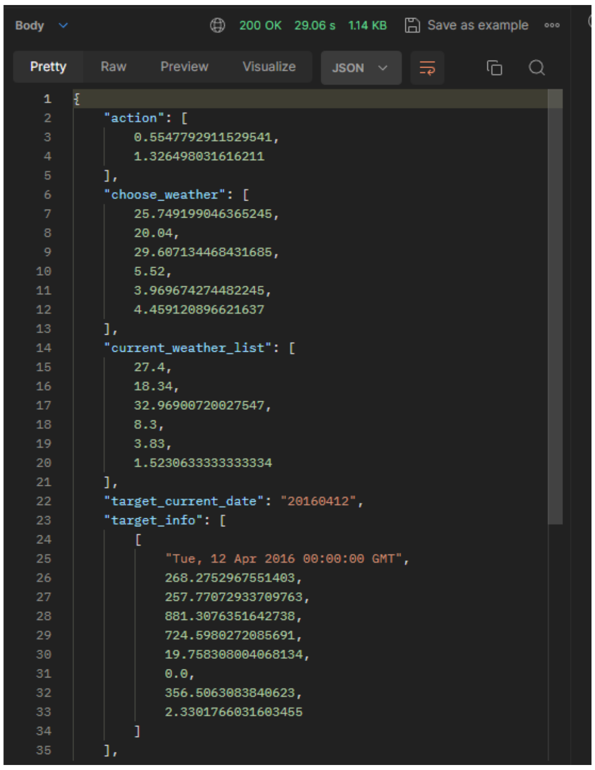


```
http://localhost:6000/get_target_info
```

```
{
    "latitude": 4.339054,
    "longitude": 101.136886,
    "plant_stage": 52,
    "choose_current_date": "2024-04-22"
}
```

Figure 46: POST request for RACKY PPO Agent API test 1

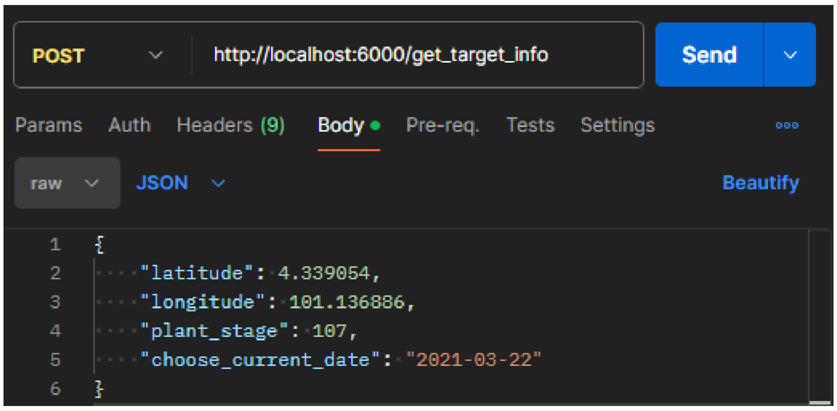
Above Figure 46 is the POST request to test the PPO Agent. The output of the API is shown in below Figure 47.



```
{
    "action": [
        0.5547792911529541,
        1.326498031616211
    ],
    "choose_weather": [
        25.749199046365245,
        20.04,
        29.607134468431685,
        5.52,
        3.969674274482245,
        4.459120896621637
    ],
    "current_weather_list": [
        27.4,
        18.34,
        32.96900720027547,
        8.3,
        3.83,
        1.5230633333333334
    ],
    "target_current_date": "20160412",
    "target_info": [
        [
            "Tue, 12 Apr 2016 00:00:00 GMT",
            268.2752967551403,
            257.77072933709763,
            881.3076351642738,
            724.5980272085691,
            19.758308004068134,
            0.0,
            356.5063083840623,
            2.3301766031603455
        ]
    ],
```

Figure 47: output for RACKY PPO Agent API test 1

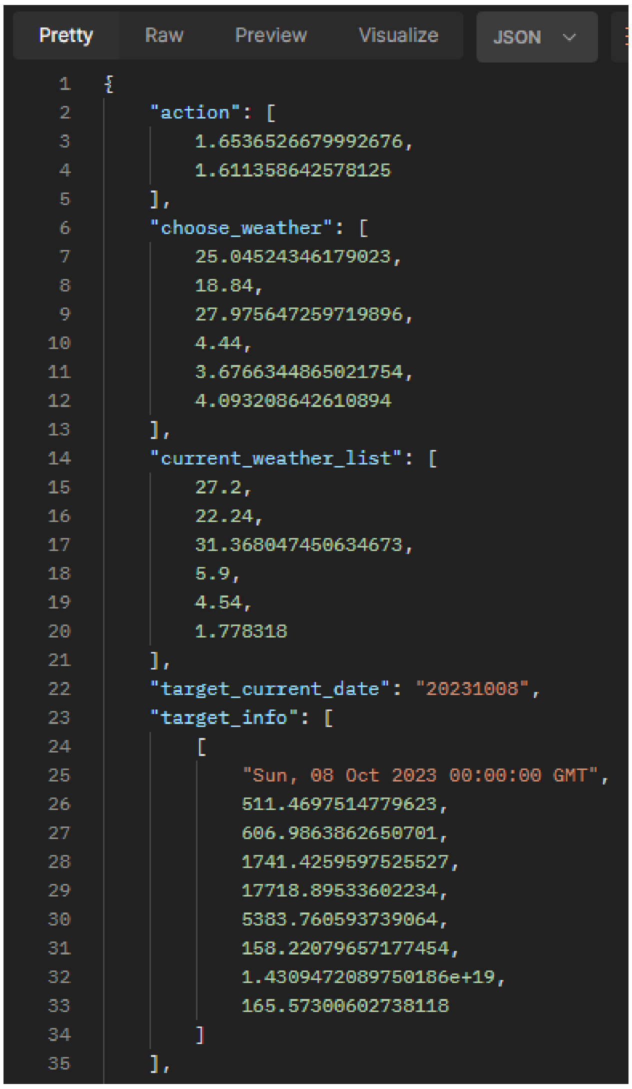
Tested on current date (2021-03-22) with plant_stage of 107



```
{
    "latitude": 4.339054,
    "longitude": 101.136886,
    "plant_stage": 107,
    "choose_current_date": "2021-03-22"
}
```

Figure 48: POST request for RACKY PPO Agent API test 2

Above Figure 48 is the POST request to retrieve the information of action from the API. Below Figure 49 are the output of the test.



```
{
    "action": [
        1.6536526679992676,
        1.611358642578125
    ],
    "choose_weather": [
        25.04524346179023,
        18.84,
        27.975647259719896,
        4.44,
        3.6766344865021754,
        4.093208642610894
    ],
    "current_weather_list": [
        27.2,
        22.24,
        31.368047450634673,
        5.9,
        4.54,
        1.778318
    ],
    "target_current_date": "20231008",
    "target_info": [
        [
            "Sun, 08 Oct 2023 00:00:00 GMT",
            511.4697514779623,
            606.9863862650701,
            1741.4259597525527,
            17718.89533602234,
            5383.760593739064,
            158.22079657177454,
            1.4309472089750186e+19,
            165.57300602738118
        ]
    ],
```

Figure 49: output for RACKY PPO Agent API test 2

6.2 *Testing Result*

6.2.1 RACKY PPO Agent evaluation

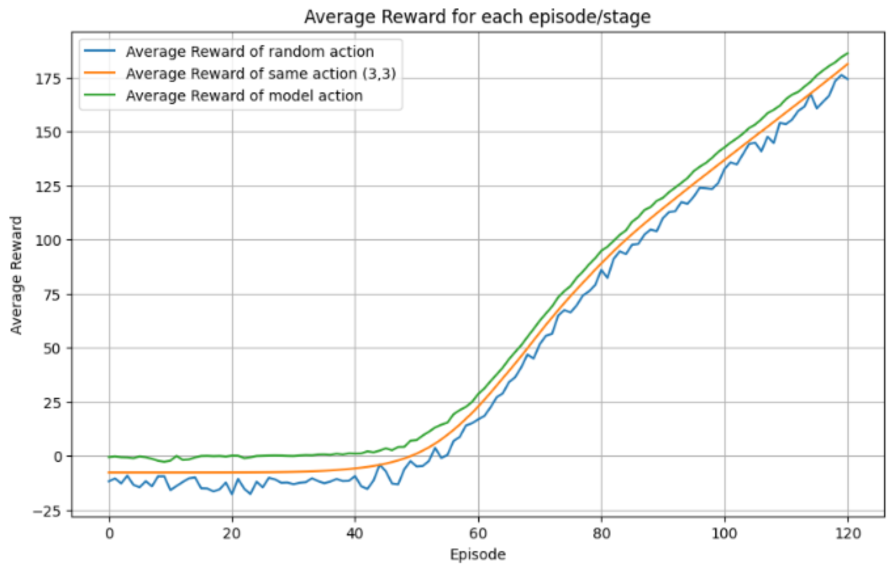


Figure 50: Graph of comparison of the average reward for each epoch

The graph above Figure 50 clearly illustrates that the average reward per timestep of the PPO Agent outperforms both the fixed action graph (3,3) and the random action graph.

To provide a more comprehensive view of the PPO agent's performance, the total reward is calculated over 20 epochs and depicted in the graph below Figure 51. It is evident that the model consistently achieves a total reward exceeding 7000, unlike the other graphs. This confirms the effectiveness of the PPO agent within the SWATGym environment.

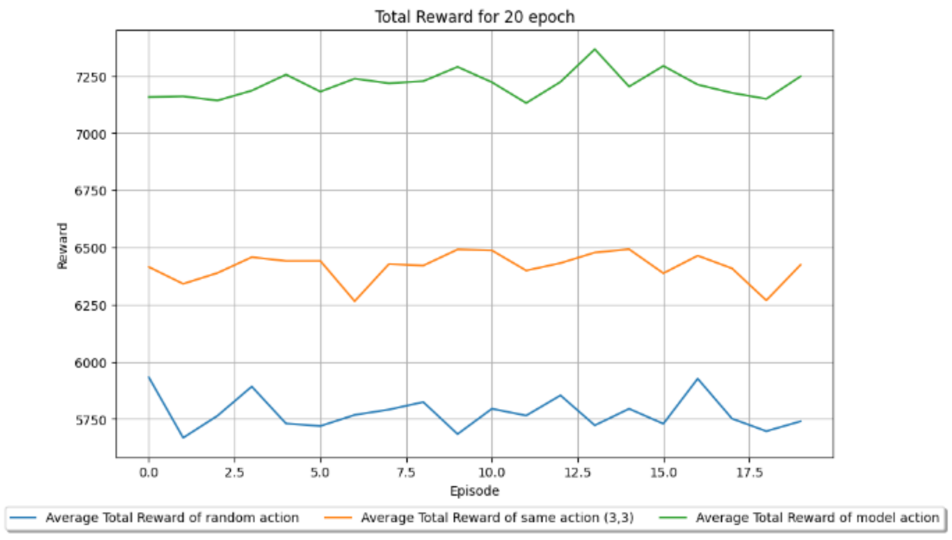


Figure 51: Graph of Comparison of the Total reward

6.3 Project Challenge

Lack of Initial Knowledge

Starting the project without a deep understanding of reinforcement learning posed a significant challenge. It required investing time and effort to grasp concepts and techniques, especially for environments that accept continuous states and actions where popular agents like DQN are not directly applicable.

Difficulty Implementing Agents

Implementing agents like DQN for continuous states and actions was challenging, leading to exploration of more advanced methods like Actor-Critic architectures, which demanded substantial learning time.

Issues with DDPG Agent

Encountering persistent issues with the DDPG agent returning actions of (0,0) led to exploring alternative agents like PPO, which required additional learning and experimentation to enhance training by adjusting reward functions.

Resource-Intensive Training

Training both PPO and DDPG agents was resource-intensive and time-consuming. However, saving and reloading models allowed for incremental training with more epochs until achieving satisfactory rewards.

Weather Data Integration

Integrating the system with SWATGym revealed limitations in receiving current weather data. Leveraging an external weather API was a fortunate discovery, providing accurate weather information crucial for the project's accuracy and reliability.

6.4 Objective Evaluation

The project has successfully achieved its objectives as evidenced by the comprehensive system testing conducted on the RACKY API and the integrated RACKY PPO Agent API within the SWATGym environment.

The RACKY API, designed to facilitate corn crop simulations based on user inputs such as location and date, demonstrated its functionality through tests in locations like Kampar and Ipoh, providing simulation outputs such as total rewards.

Concurrently, the integration and testing of the RACKY PPO Agent API showcased the AI model's capability to suggest optimal actions and achieve rewards, as validated by graphical evaluations comparing its performance against fixed and random actions. Graphs depicting average rewards per timestep and total rewards over epochs visually confirmed the effectiveness of the PPO agent in optimizing crop management strategies within the SWATGym framework.

Thus, the above evidence had shown the project had fully meeting the project's objective.

6.5 Concluding Remark

The system evaluation and testing conducted on the RACKY API and RACKY PPO Agent API within the SWATGym environment have provided compelling evidence of the project's success in meeting its objectives.

The thorough testing in locations like Kampar and Ipoh, along with evaluations using various plant stages and dates, demonstrates the **functionality and effectiveness of both APIs in simulating crop growth scenarios and suggesting optimal actions**. Besides, The graphical evaluations further confirm the **superiority of the PPO agent in optimizing crop management strategies**.

Overall, the project has not only met but exceeded expectations, showcasing the potential of AI-driven solutions in revolutionizing agricultural practices and crop management strategies.

Chapter 7

Conclusion & Recommendation

Conclusion

The project embarked on a journey to revolutionize agricultural practices through the **integration of AI-driven solutions with farming environments**. By addressing key challenges such as limited accessibility to AI tools and the need for optimized crop management strategies, significant strides were made towards democratizing advanced technologies in farming.

The **development of a user-friendly API for SWATGym** opens doors for a wide range of users, from researchers to farmers, to harness the power of AI models in optimizing crop growth. Leveraging reinforcement learning techniques such as **Proximal Policy Optimization (PPO)**, the project successfully created an AI agent capable of intelligently interacting with farming environments, suggesting optimal actions like fertilizer and irrigation amounts.

Moreover, the **integration of an external weather API enriched the simulation environment**, providing real-time weather data crucial for accurate and relevant simulations. This addition not only improved the accuracy of the AI model's recommendations but also facilitated informed decision-making based on current weather conditions.

Recommendation

Consider **importing a more reliable and up-to-date weather API** to address the delays experienced with the current weather API. Timely and accurate weather information is crucial for effective simulations and decision-making in agricultural management. By ensuring the availability of real-time weather data, users can make more informed decisions regarding crop management strategies.

Expand the range of plant types within the environment beyond solely focusing on corn. Incorporating a variety of plant types caters to a broader user base, including farmers and researchers working with different crops. This diversification not only increases the platform's utility but also promotes innovation and exploration of optimized growth strategies for various agricultural products.

Conduct thorough **testing and experimentation with a range of reinforcement learning agents** to enhance the platform's functionality. Offering users a selection of agents enables them to choose models that align best with their specific needs and preferences. This approach not only improves user satisfaction but also provides valuable insights into the performance and applicability of different AI models across diverse plant types and environmental conditions.

REFERENCES

[1] H. M. Yassine, K. Roufaida, V. P. Shkodyrev, M. Abdelhak, L. Zarour, and R. Khaled, "Intelligent Farm Based on Deep Reinforcement Learning for optimal control," *2022 International Symposium on iNnovative Informatics of Biskra, ISNIB 2022*, Dec. 2022, doi: 10.1109/ISNIB57382.2022.10076088.

[2] G. Ene, "A Reinforcement Learning Approach for Smart Farming," *Database Systems Journal*, vol. X/2019, p. 3, 2019, Accessed: Sep. 07, 2023. [Online]. Available: https://dbjournal.ro/archive/30/30_1.pdf

[3] J. Binas, L. Luginbuehl, and Y. Bengio, "Reinforcement Learning for Sustainable Agriculture," 2019, Accessed: Sep. 07, 2023. [Online]. Available: https://s3.us-east-1.amazonaws.com/climate-change-ai/papers/icml2019/32/paper.pdf

[4] M. Madondo *et al.*, "A SWAT-based Reinforcement Learning Framework for Crop Management," 2023, Accessed: Sep. 07, 2023. [Online]. Available: www.aaai.org

[5] M. Turchetta ETH Zurich Zurich *et al.*, "Learning Long-Term Crop Management Strategies with CyclesGym," Sep. 2022, Accessed: Sep. 07, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/4a22ceafe2dd6e0d32df1f7c0a69ab68-Paper-Datasets_and_Benchmarks.pdf

[6] H. Overweg, H. N. C. Berghuijs, and I. N. Athanasiadis, "CROPGYM: A REINFORCEMENT LEARNING ENVIRON-MENT FOR CROP MANAGEMENT," Apr. 2021, Accessed: Sep. 07, 2023. [Online]. Available: https://github.com/BigDataWUR/crop-gym

[7] O.-A. Maillard, T. Mathieu, and D. Basú Basú, "Farm-gym: A modular reinforcement learning platform for stochastic agronomic games," Jan. 2023, Accessed: Sep. 07, 2023. [Online]. Available: https://openreview.net/pdf?id=Ev4NOEeTYL

[8] "Smart Farming Cycle and Benefits." Accessed: Sep. 07, 2023. [Online]. Available: https://bsb-smartfarming.com/tpost/b3b1igjma1-smart-farming-cycle-and-benefits

[9] "RL introduction: simple actor-critic for continuous actions | by Andy Steinbach | Medium." Accessed: Sep. 07, 2023. [Online]. Available: https://medium.com/@asteinbach/rl-introduction-simple-actor-critic-for-continuous-actions-4e22afb712

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Jul. 2017, Accessed: Apr. 24, 2024. [Online]. Available: http://arxiv.org/abs/1707.06347

[11] N. Barhate, “Minimal PyTorch Implementation of Proximal Policy Optimization,” GitHub. Accessed: Apr. 24, 2024. [Online]. Available: https://github.com/nikhilbarhate99/PPO-PyTorch

[12] P. Zippenfenig, “Open-Meteo.com Weather API.” doi: 10.5281/zenodo.7970649.

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 2**
Student Name & ID: Tan Carlton 2001512	
Supervisor: Dr Ooi Boon Yaik	
Project Title: Precision Agriculture for corn using Reinforcement Learning	

1. WORK DONE

- **Training on the DDPG Agent that use in Project 1.**

2. WORK TO BE DONE

- **Find the cause that cause DDPG Agent to return Action (0, 0)**
- **Find another Agent to replace DDPG Agent**
- **Test integration of API with SWATGym and Agent.**

3. PROBLEMS ENCOUNTERED

- **DDPG Agent return action (0, 0) for all the timestep in each training epoch.**

4. SELF EVALUATION OF THE PROGRESS

- **Do not familiar to the architecture of DDPG Agent. Training of DDPG Agent took a lot of time.**

[illegible] [illegible]

Supervisor's signature Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 4**
Student Name & ID: Tan Carlton 2001512	
Supervisor: Dr Ooi Boon Yaik	
Project Title: Precision Agriculture for corn using Reinforcement Learning	

1. WORK DONE

- **Research on new Agent, PPO**
- **Tested PPO Agent on the SWATGym**
- **Changed training reward function while training the Agent**

2. WORK TO BE DONE

- **Find new resource to overcome no weather data issue**
- **Integrate the PPO model to the API**

3. PROBLEMS ENCOUNTERED

- **Simulation cannot proceed with current date.**

4. SELF EVALUATION OF THE PROGRESS

- **After the modification in reward function, the PPO Agent able to return better results**

[illegible]
Supervisor's signature

[illegible]
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 6**
Student Name & ID: Tan Carlton 2001512	
Supervisor: Dr Ooi Boon Yaik	
Project Title: Precision Agriculture for corn using Reinforcement Learning	

1. WORK DONE

- **Train PPO Agent**
- **Write API for SWATGym simulator**

2. WORK TO BE DONE

- **Find external API**

3. PROBLEMS ENCOUNTERED

- **PPO take a lot of time to train**

4. SELF EVALUATION OF THE PROGRESS

- **A lot of time is needed to train PPO Agent, but manage to study some ways to implement API using Python Flask while waiting for the result.**

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 8**
Student Name & ID: Tan Carlton 2001512	
Supervisor: Dr Ooi Boon Yaik	
Project Title: Precision Agriculture for corn using Reinforcement Learning	

1. WORK DONE - **Found new weather API to replace existing weather data in SWATGym**
2. WORK TO BE DONE - **Integrating the new weather API with PPO Agent for user that looking for current action recommendation.**
3. PROBLEMS ENCOUNTERED - **The API still have some delay for weather data for current date, and some information that is required is not provided.**
4. SELF EVALUATION OF THE PROGRESS - **Take a lot of time to understand the weather information in SWATGym and find the information that is needed to replace.**

Type text h

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 10**
Student Name & ID: Tan Carlton 2001512	
Supervisor: Dr Ooi Boon Yaik	
Project Title: Precision Agriculture for corn using Reinforcement Learning	

1. WORK DONE

- **Integrating the external weather API to the fully trained RACKY API PPO Agent**
- **Calculated all the weather information that is not provided by external weather API**

2. WORK TO BE DONE

- **Prepare showcase for both RACKY API and RACKY PPO Agent API**

3. PROBLEMS ENCOUNTERED

- **Some weather information is not provided by the new external weather API and there are some delays for some data**

4. SELF EVALUATION OF THE PROGRESS

- **Found alternative to replace the current weather information.**

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	**Study week no.: 12**
Student Name & ID: Tan Carlton 2001512	
Supervisor: Dr Ooi Boon Yaik	
Project Title: Precision Agriculture for corn using Reinforcement Learning	

1. WORK DONE

- Finalizing the report
- Posted the RACKY API and RACKY PPO Agent API to the website

2. WORK TO BE DONE

- **Complete the FYP report and poster**

3. PROBLEMS ENCOUNTERED

4. SELF EVALUATION OF THE PROGRESS

- **The project objective is achieved and prepare to send the report for marking.**

[illegible]

Supervisor's signature

[illegible]

Student's signature

POSTER

Introduction

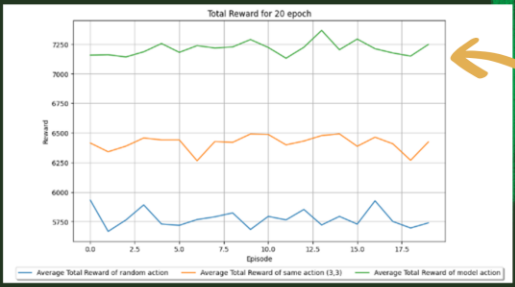
This study introduces RACKY API, aprecision agricultural system for Corn aiming to **reduce the time and resources of generating AI-model** AND **improve the accessbility** to the simulator by integrating API

Objective

- **Reduce time and resource to train an AI model**
- **Develop user-friendly API for simulator SWATGym.**
- **Allow user to interact with PPO model API**

Project Scope

- Implement **PPO** (Proximal Policy Optimization) Agent to utilize the simulator (SWATGym)
- Integrating the simulator and the PPO Agent with external weather API into **RACKY API**



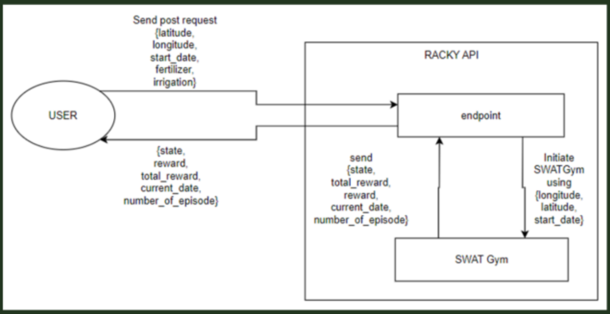
PRECISION AGRICULTURE FOR CORN USING RACKY API

By Tan Carlton

Supervisor: Dr Ooi Boon Yaik

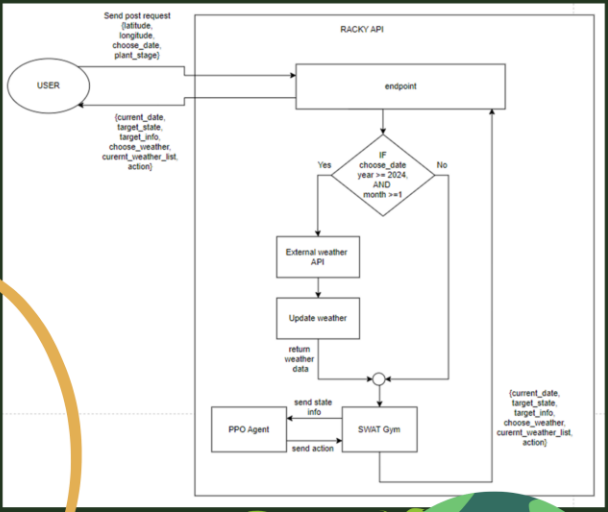
Design of System

Below are the overview of RACKY API :



Send post request {latitude, longitude, start_date, fertilizer, irrigation}
USER
RACKY API
endpoint
{state, reward, total_reward, current_date, number_of_episode}
send {state, total_reward, reward, current_date, number_of_episode}
Initiate SWATGym using {longitude, latitude, start_date}
SWAT Gym

Below are the overview of RACKY PPO Agent API :



Send post request {latitude, longitude, choose_date, plant_stage}
USER
RACKY API
endpoint
{current_date, target_state, target_info, choose_weather, curernt_weather_list, action}
IF choose_date year >= 2024, AND month >=1
Yes
No
External weather API
Update weather
return weather data
send state info
send action
PPO Agent
SWAT Gym
{current_date, target_state, target_info, choose_weather, curernt_weather_list, action}

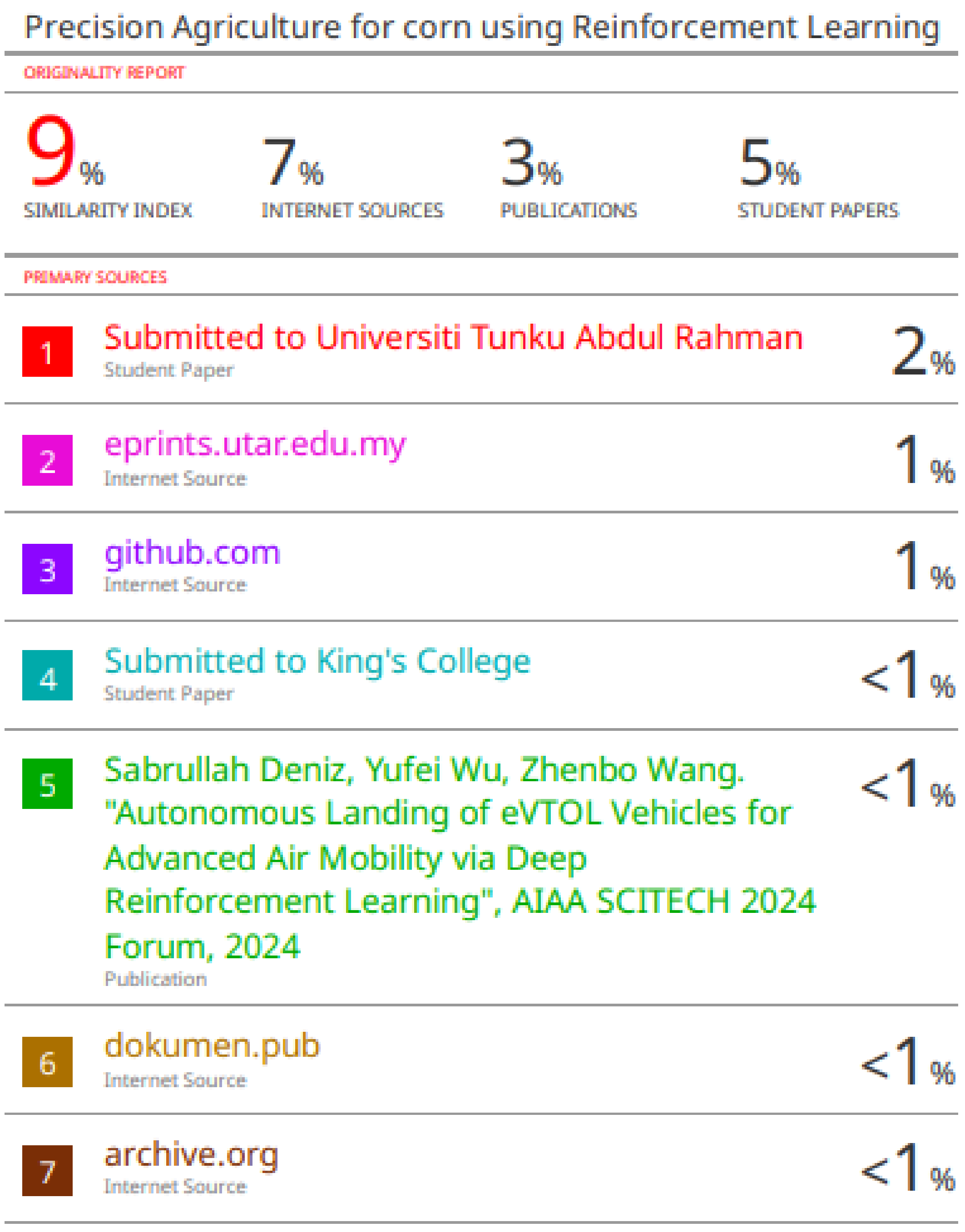
Conclusion

The green plotted graph are the result of the PPO Agent which had achive high and consistent reward

RACKY API aimed to **revolutionize agricultural practices** by **integrating AI-driven solutions** with farming environments.
Developing a user-friendly API for SWATGym enabled diverse users to utilize AI models for crop growth optimization, leveraging **reinforcement learning techniques like Proximal Policy Optimization (PPO)**.
The project **successfully created an AI agent** capable of intelligent interaction in farming environments, suggesting optimal actions such as fertilizer and irrigation amounts. **Integrating an external weather API enriched simulations with real-time data, enhancing accuracy and decision-making**.

UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

Plagiarism Check Result



Precision Agriculture for corn using Reinforcement Learning

ORIGINALITY REPORT

9%	7%	3%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Universiti Tunku Abdul Rahman Student Paper	2%
2	eprints.utar.edu.my Internet Source	1%
3	github.com Internet Source	1%
4	Submitted to King's College Student Paper	<1%
5	Sabrullah Deniz, Yufei Wu, Zhenbo Wang. "Autonomous Landing of eVTOL Vehicles for Advanced Air Mobility via Deep Reinforcement Learning", AIAA SCITECH 2024 Forum, 2024 Publication	<1%
6	dokumen.pub Internet Source	<1%
7	archive.org Internet Source	<1%

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	TAN CARLTON
ID Number(s)	20ACB01512
Programme / Course	Bachelor of Computer Science
Title of Final Year Project	Precision Agriculture for corn using Reinforcement Learning

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 9 % **Similarity by source** Internet Sources: 7 % Publications: 3 % Student Papers: 5 %	
Number of individual sources listed of more than 3% similarity: 0	
Parameters of originality required and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

[illegible]

Signature of Supervisor	Signature of Co-Supervisor
Name: Dr. Ooi Boon Yaik	Name:
Date: 25/4/2024	Date:

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB01512
Student Name	Tan Carlton
Supervisor Name	Dr Ooi Boon Yaik

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
/	Title Page
/	Signed Report Status Declaration Form
/	Signed FYP Thesis Submission Form
/	Signed form of the Declaration of Originality
/	Acknowledgement
/	Abstract
/	Table of Contents
/	List of Figures (if applicable)
/	List of Tables (if applicable)
/	List of Symbols (if applicable)
/	List of Abbreviations (if applicable)
/	Chapters / Content
/	Bibliography (or References)
/	All references in bibliography are cited in the thesis, especially in the chapter of literature review
	Appendices (if applicable)
/	Weekly Log
/	Poster
/	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
/	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

[illegible]

(Signature of Student)
Date: 24/04/2024