**Comparative analysis of Reinforcement learning agents for portfolio management**

*Report submitted to the Amrita Vishwa Vidyapeetham, Coimbatore as the requirement for the course*

**21AIE311: Reinforcement Learning­­­­**

*Submitted by*

## Batch ATeam-14

**MAY 2024**



**SCHOOL OF ARTIFICIAL INTELLIGENCE**

## COIMBATORE, TAMIL NADU, INDIA – 641112



**SCHOOL OF ARTIFICIAL INTELLIGENCE**

## COIMBATORE, TAMIL NADU, INDIA – 641112

# Bonafide Certificate

This is to certify that the report titled “**Portfolio Management using A2C, DDPG and PPO”** submitted as arequirement for the course, **21AIE311: Reinforcement Learning** for B.Tech. Computer Science and Engineering (Artificial Intelligence) program , is a bonafide record of the work done by **Team -14** during the academic year 2023-2024, in the School of Arti, under my supervision.

## Signature of Project Supervisor :

## Name with Affiliation :

## Date :

Project *Viva-vo*ce held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Examiner 1 Examiner 2



**SCHOOL OF ARTIFICIAL INTELLIGENCE**

## COIMBATORE, TAMIL NADU, INDIA – 641112

# Declaration

We declare that the report titled “**Portfolio Management using A2C, DDPG and PPO**” submitted by us is an original work done by **Team14** under the guidance of **Dr. Palmani Duraisamy, Associate Professor, School of Artificial Intelligence, Amrita Vishwa Vidyapeetham, Coimbatore** during the sixth semester of the academic year 2023-24, in the **School of Artificial Intelligence**. The work is original and wherever we have used materials from other sources, we have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associate-ship, fellowship or other similar title to any candidate of any University.

Batch A Team-14

## 1.Akshayaa BK [ CB.EN.U4AIE21002]

## 2. Mummidi Devi Siva Rama Saran [ CB.EN. U4AIE21034]

## 3. Poornima N [ CB.EN.U4AIE21042]

## 4. Raghavendra [ CB.EN. U4AIE21049]

**Abstract**

In the realm of financial portfolio management, employing advanced machine learning techniques has shown significant promise in optimizing investment strategies. This project leverages Actor-Critic Deep Reinforcement Learning (DRL) algorithms, specifically Advantage Actor-Critic (A2C), Deep Deterministic Policy Gradient (DDPG),Twin Delay Deep Deterministic Policy Gradient(TD3), and Proximal Policy Optimization (PPO), to enhance decision-making processes in portfolio management. By integrating these sophisticated algorithms, the project aims to dynamically adjust portfolio allocations to maximize returns while minimizing risks in a constantly fluctuating market environment.

The first algorithm, A2C, combines the benefits of both policy-based and value-based methods, providing a balanced approach to learning and decision making. A2C enhances the stability and performance of traditional actor-critic methods by synchronizing multiple actor-learners, which in turn accelerates the learning process. This algorithm's capability to handle continuous action spaces makes it particularly suitable for financial applications where portfolio weights need to be adjusted precisely and continuously.

DDPG, the second algorithm implemented, is designed for environments with continuous action spaces. It extends the capabilities of traditional actor-critic models by incorporating a deterministic policy gradient, which simplifies the policy optimization process. DDPG's robustness in handling high-dimensional action spaces allows it to efficiently manage a diverse set of financial instruments within a portfolio, ensuring optimal asset allocation and improved overall performance.

Lastly, PPO is employed for its robustness and reliability in training stability. PPO strikes a balance between exploration and exploitation by maintaining a clipped objective function, which prevents drastic updates and ensures smoother training. This algorithm's ability to maintain a stable learning process while achieving high performance makes it an ideal choice for financial markets, where stability and consistency are paramount. This project seeks to develop a sophisticated portfolio management system capable of adapting to market changes and achieving superior financial outcomes.

Signature of the Guide

Name : Dr. Palmani Duraisamy

**Table of Contents**

Bonafide Certificate

Declaration

Acknowledgements

Abstract

1. Introduction
   1. Portfolio 7
   2. Reinforcement Learning 7
   3. Elements of RL 8
   4. RL Algorithm 9
   5. Model based reinforcement learning 10

1.5.1Model learning 10

* + 1. Decision making
  1. Model free reinforcement learning 10
     1. Value based methods 11
     2. Policy based methods 11

1. Objectives ` 12

1. Methodology 12

3.1Data from yahoo finance 13 3.2 Building an environment 13

* 1. Initialising 3 agents- DDPG,A2C,PPO 14
     1. A2C 14
     2. DDPG 15
     3. PPO 19

* 1. Train,validate and test 22
     1. Train 22
     2. Validate 23
     3. Test 23
  2. Evaluate the agent’s performance

1. Results 24
2. References 29

**INTRODUCTION**

The financial industry has always been at the forefront of adopting new technologies to enhance investment strategies and optimize portfolio management. With the advent of machine learning and artificial intelligence, the field of financial portfolio management has seen a paradigm shift. Among the various techniques available, Deep Reinforcement Learning (DRL) has emerged as a particularly promising approach due to its ability to learn and adapt to complex and dynamic environments. This project focuses on utilizing Actor-Critic DRL algorithms, namely Advantage Actor-Critic (A2C), Deep Deterministic Policy Gradient (DDPG), and Proximal Policy Optimization (PPO), to develop an advanced portfolio management system.

Reinforcement Learning (RL) is a branch of machine learning where agents learn to make decisions by interacting with an environment to maximize cumulative rewards. Traditional RL algorithms, while effective in some domains, often struggle with the high-dimensional and continuous action spaces characteristic of financial markets. Actor-Critic methods, which combine the benefits of value-based and policy-based approaches, have shown great potential in overcoming these challenges. Specifically, they separate the learning of the policy (actor) from the value function (critic), allowing for more efficient and stable learning.

The A2C algorithm enhances traditional actor-critic methods by synchronizing multiple actors, which can significantly speed up the learning process. Each actor interacts with its own copy of the environment, and their experiences are used to update a shared critic network. This approach not only accelerates learning but also provides a more robust estimate of the value function, leading to better policy updates. In the context of portfolio management, A2C can dynamically adjust asset allocations to optimize returns while managing risk.

DDPG is another powerful actor-critic algorithm designed for continuous action spaces. It combines the deterministic policy gradient with the power of deep learning to handle high-dimensional state and action spaces effectively. DDPG employs an actor network to propose actions and a critic network to evaluate them. This separation allows DDPG to optimize policies in a more structured manner, making it well-suited for financial applications where precise adjustments to portfolio weights are crucial. By leveraging DDPG, this project aims to create a system that can efficiently manage a diverse set of financial instruments.

In portfolio management, TD3 can be used to optimize the allocation of assets in a portfolio over time. By modeling the portfolio as a continuous control problem, TD3 can learn a policy that dynamically adjusts the asset allocation based on market conditions and desired objectives. This approach can lead to improved portfolio performance and risk management compared to traditional static allocation strategies.

PPO, on the other hand, is renowned for its training stability and reliability. It introduces a novel objective function that strikes a balance between exploring new strategies and exploiting known ones by maintaining a clipped surrogate objective. This clipping mechanism prevents excessive updates, ensuring that the learning process remains stable and consistent. Stability is a critical factor in financial markets, where abrupt changes can lead to significant losses. PPO's ability to maintain smooth training trajectories makes it an ideal choice for developing robust portfolio management strategies.

This system will be tested and validated using historical market data to ensure its practical applicability and performance. The ultimate goal is to provide a sophisticated tool that can assist investors in making more informed and strategic decisions in the ever-evolving financial landscape.

### 1.1 Portfolio

A financial portfolio is a collection of assets and investments that an individual or institution holds. These assets can include stocks, bonds, mutual funds, exchange-traded funds (ETFs), real estate, commodities, and other financial instruments. The primary objective of a portfolio is to allocate investments in a manner that meets the investor’s goals, whether they are for capital appreciation, income generation, risk management, or a combination of these. Effective portfolio management involves making strategic decisions about asset selection, diversification, risk assessment, and ongoing adjustment to align with the investor's objectives and market conditions.

Investment objectives serve as guiding principles for portfolio construction. Clearly defining investment objectives is critical for building a portfolio tailored to meet specific financial goals. Whether the goal is capital appreciation, income generation, wealth preservation, or a combination thereof, having a clear understanding of objectives helps in making informed investment decisions and selecting appropriate investment strategies.

Asset allocation plays a significant role in determining portfolio performance and risk. Asset allocation refers to the distribution of investments across various asset classes, such as stocks, bonds, cash equivalents, and alternative investments. The optimal asset allocation depends on factors like risk tolerance, investment objectives, and time horizon. Maintaining a balanced asset allocation helps in achieving long-term financial goals while managing risk effectively.

While it's impossible to predict market movements with certainty, monitoring economic indicators, geopolitical events, and monetary policy decisions can provide valuable insights. This information helps investors make informed decisions and adjust their portfolios accordingly to capitalize on opportunities and mitigate risks.

Portfolio management is an ongoing process that requires regular monitoring and, if necessary, rebalancing. Market fluctuations and changes in personal circumstances may necessitate adjustments to the portfolio's asset allocation. Rebalancing involves buying or selling assets to restore the portfolio's target asset allocation, ensuring that it remains aligned with the investor's goals and risk tolerance. By considering these key points before investing in a portfolio, investors can make informed decisions that align with their financial objectives and risk preferences, laying the groundwork for long-term financial success.

**1.2 Reinforcement learning**

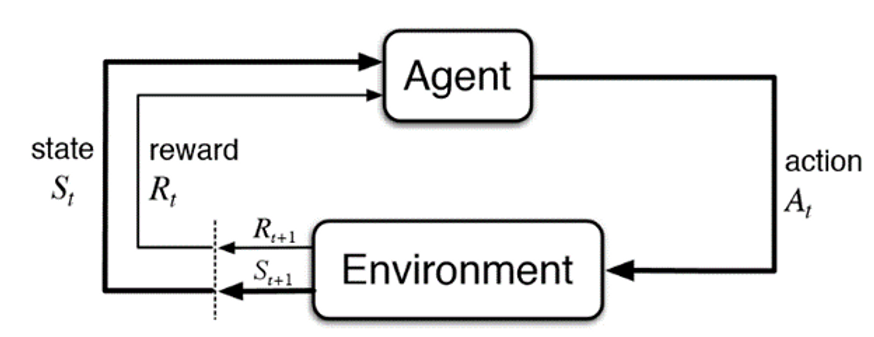
Reinforcement Learning (RL) is a powerful machine learning paradigm that enables agents to learn optimal decision-making strategies through interaction with an environment. At its core, RL is inspired by behavioral psychology principles, emphasizing how agents can learn from the consequences of their actions. The RL framework consists of an agent, which is the decision-maker or learner, and an environment, which represents the external system with which the agent interacts. Through a process of trial and error, the agent learns to take actions that maximize cumulative rewards over time. This trial-and-error learning process is fundamental to RL, as it allows agents to explore different strategies, learn from their experiences, and adapt their behavior to achieve desired outcomes.

In RL scenarios, the interaction between the agent and the environment unfolds over discrete time steps. At each time step, the agent observes the current state of the environment, selects an action based on its policy (a strategy for decision-making), and receives feedback in the form of a reward signal. The goal of the agent is to learn a policy that maximizes the cumulative reward it receives over time. This involves a balance between exploration, where the agent tries new actions to discover potentially better strategies, and exploitation, where the agent leverages known strategies to maximize rewards. By iteratively adjusting its policy based on feedback from the environment, the agent gradually improves its decision-making abilities and learns to make optimal choices in different situations.

RL problems are often formalized as Markov Decision Processes (MDPs), which provide a mathematical framework for modeling sequential decision-making under uncertainty. MDPs consist of states, actions, transition probabilities, and rewards, which capture the dynamics of the environment and the consequences of the agent's actions. States represent different configurations of the environment, actions are the decisions available to the agent, transition probabilities specify the likelihood of moving from one state to another based on the chosen action, and rewards provide feedback on the desirability of the agent's actions. By solving the MDP associated with a particular RL problem, agents can learn optimal policies that guide their behavior towards achieving specific objectives within the environment.

**1.3 Elements of RL**

Reinforcement Learning (RL) encompasses several key elements that define its framework and operation:



Agent: The agent is the entity responsible for making decisions and taking actions within the environment. It is the learner in the RL framework and seeks to maximize cumulative rewards over time by selecting actions that lead to desirable outcomes. The agent's decision-making process is guided by its policy, which maps states to actions based on the expected rewards.

Environment: The environment represents the external system with which the agent interacts. It is dynamic and uncertain, providing feedback to the agent based on its actions. The environment's state changes in response to the agent's actions, influencing future states and rewards. The agent's goal is to learn an optimal policy for interacting with the environment to achieve specified objectives.

State: A state is a specific configuration or snapshot of the environment at a given time. It captures all relevant information necessary for decision-making at that moment. States can be discrete or continuous and provide the context for the agent to assess its current situation and select appropriate actions.

Action: An action is a decision or choice made by the agent at a particular state. Actions can be discrete, where the agent chooses from a finite set of options, or continuous, where actions are selected from a continuous range. The agent's goal is to learn which actions to take in different states to maximize cumulative rewards over time.

Reward: A reward is feedback provided by the environment to the agent after it takes an action in a particular state. It indicates the immediate benefit or consequence of the agent's action and influences its future behavior. Rewards can be positive, negative, or zero, reflecting the desirability or undesirability of the agent's actions. The agent's objective is to learn a policy that maximizes cumulative rewards over time.

Policy: The policy defines the agent's strategy or behavior for selecting actions in different states. It maps states to actions and determines the agent's decision-making process. Policies can be deterministic, where the same action is always chosen for a given state, or stochastic, where actions are selected probabilistically based on the policy's parameters. The agent's goal is to learn an optimal policy that maximizes cumulative rewards over time.

Value Function: The value function estimates the expected cumulative reward that the agent can achieve from a given state or state-action pair. It quantifies the desirability of different states or actions and provides guidance for the agent's decision-making process. The value function helps the agent evaluate the long-term consequences of its actions and select actions that lead to higher rewards.

These elements form the foundation of the Reinforcement Learning framework and define the interactions between the agent and the environment. By learning from experiences and feedback, the agent can adapt its behavior over time to achieve specified objectives within the environment. RL algorithms leverage these elements to develop intelligent decision-making strategies across a wide range of applications and domains.

**1.4 RL Algorithm**

Reinforcement Learning (RL) algorithms work by enabling an agent to learn optimal decision-making strategies through interaction with an environment. RL algorithms follow a common iterative process:

1. The RL process begins with initializing the agent's policy, value function, or other relevant parameters. This step sets the starting point for the learning process.
2. The agent observes the current state of the environment. States represent different configurations or snapshots of the environment at a given time and contain all relevant information necessary for decision-making.
3. Based on its current policy, the agent selects an action to take in the observed state. The policy defines the agent's strategy for decision-making and maps states to actions.
4. The agent executes the selected action, causing the environment to transition to a new state. This action may result in a change in the environment's state and potentially in receiving a reward from the environment.
5. After taking an action, the agent receives feedback from the environment in the form of a reward signal. Rewards indicate the immediate benefit or consequence of the agent's action and influence its future behavior.
6. Using the observed state, action, reward, and resulting next state, the agent updates its policy, value function, or other relevant parameters. This update is based on the observed outcomes and aims to improve the agent's decision-making capabilities.
7. Steps 2-6 are repeated iteratively as the agent continues to interact with the environment, observe states, take actions, receive rewards, and update its policy or value function. Over time, the agent learns from its experiences and improves its decision-making strategies.
8. The RL process continues until the agent's policy or value function converges to an optimal or near-optimal solution. Convergence occurs when the agent's behavior stabilizes, and further updates yield minimal improvements in performance.

Throughout this iterative process, RL algorithms aim to maximize cumulative rewards over time by learning optimal policies or value functions. By exploring different actions, observing outcomes, and adjusting behavior based on feedback, RL algorithms enable agents to adapt to changing environments, optimize their strategies, and achieve specified objectives.

**1.5 Model-based Reinforcement Learning:-**

Model-based Reinforcement Learning (RL) is an approach that involves learning an explicit model of the environment's dynamics and using this model to make decisions. Unlike model-free RL algorithms, which directly learn from experience without explicitly modeling the environment, model-based RL algorithms aim to understand how the environment behaves and use this understanding to plan future actions more effectively.

In model-based RL, the learning process typically consists of two main components: model learning and decision making.

1.5.1Model Learning:

Model learning involves estimating or learning a representation of the environment's dynamics. This includes understanding how the environment transitions from one state to another in response to the agent's actions and how rewards are obtained.The model can take various forms, such as a transition model (predicting the next state given the current state and action), a reward model (predicting the reward obtained from a given state-action pair), or a combined dynamics model (predicting both state transitions and rewards).

Model learning can be achieved through various techniques, including supervised learning, dynamic programming, or more sophisticated approaches such as neural network-based models.

1.5.2 Decision Making:

Once a model of the environment is learned, the agent can use this model to simulate future trajectories and plan actions accordingly.Decision-making strategies in model-based RL often involve using the learned model to perform lookahead search or optimization to find the sequence of actions that maximizes expected cumulative rewards.

Planning algorithms, such as dynamic programming, Monte Carlo tree search, or model-predictive control, can be employed to generate action sequences by simulating possible future trajectories using the learned model and selecting actions that lead to desirable outcomes.

**1.6 Model-free Reinforcement Learning:-**

Model-free Reinforcement Learning (RL) is an approach where an agent learns to make decisions without explicitly modeling the dynamics of the environment. Unlike model-based RL, which involves learning an explicit representation of the environment's dynamics, model-free RL algorithms directly learn from experience, typically through trial and error.

In model-free RL, the agent's learning process focuses on estimating the value function or policy directly from interactions with the environment. This involves iteratively updating the agent's estimates based on observed states, actions, and rewards without explicitly modeling how the environment transitions from one state to another.There are two main types of model-free RL algorithms: value-based methods and policy-based methods.

1.6.1 Value-based methods:-

Value-based methods are a category of reinforcement learning algorithms that focus on estimating the value function, which quantifies the expected cumulative reward an agent can obtain from a given state or state-action pair. These methods aim to learn the optimal value function that maximizes cumulative rewards over time, without explicitly modeling the dynamics of the environment.

The core idea behind value-based methods is to iteratively update estimates of the value function based on observed states, actions, and rewards. Value-based methods offer several advantages, including simplicity, scalability to high-dimensional state spaces, and ease of implementation. However, they also have limitations, such as difficulty in handling continuous action spaces and lack of explicit policy representation. Despite these limitations, value-based methods remain a popular choice for solving reinforcement learning problems, particularly in domains where discrete actions and state representations are feasible.

1.6.2 Policy-based methods:-

Policy-based methods in reinforcement learning (RL) focus on directly parameterizing and optimizing the agent's policy, which defines the probability distribution over actions given states. Unlike value-based methods that estimate the value function, policy-based methods aim to find the optimal policy by maximizing the expected cumulative rewards.

Policy-based methods offer several advantages over value-based methods, including the ability to handle continuous action spaces, explicit policy representation, and stability in training. They are well-suited for problems with high-dimensional action spaces and noisy or stochastic environments. However, policy-based methods can suffer from high variance in gradients, which can lead to slower convergence and unstable training. Despite these challenges, policy-based methods remain a powerful approach for solving a wide range of reinforcement learning problems, particularly in domains where explicit policy representation is desirable.

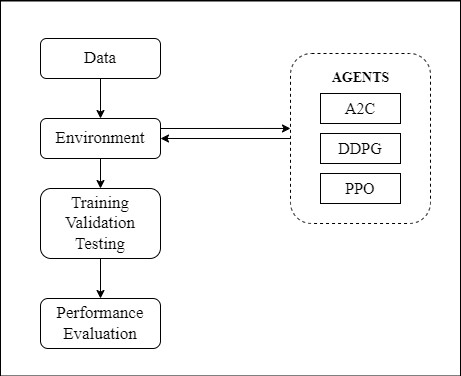
**2 OBJECTIVES**

The primary objective of this project is to optimize portfolio allocation using Actor-Critic Deep minimizing risks. Through continuous learning and adaptation, the system seeks to identify and capitalize on profitable investment opportunities while mitigating potential losses.

Implement A2C, DDPG, and PPO algorithms to improve the precision and efficiency of decision-making processes in portfolio management, leveraging their respective strengths in handling continuous action spaces and high-dimensional environments. Develop a robust portfolio management system that maintains stability and consistency in learning and decision making, crucial for long-term success in financial markets.

KEY TASKS:-

1. Implement Actor-Critic Deep Reinforcement Learning algorithms (A2C, DDPG, PPO) for dynamic portfolio allocation.
2. Evaluate algorithm performance in optimizing returns and minimizing risks in fluctuating market conditions.
3. **METHODOLOGY**



**3.1 Loading Data**

Data is fetched from the yahoo finance website for the asked dates given the tickers file . These are then stored as separate csv files which are then used by the agents while interacting with the environment.

**3.2 Building an environment**

The custom PortfolioEnv environment is tailored for implementing Actor-Critic Deep Reinforcement Learning algorithms in portfolio management. It facilitates dynamic portfolio allocation and trading decision-making processes by providing a flexible framework for interacting with financial market data.

Purpose and Functionality: The PortfolioEnv environment serves as a simulation platform where reinforcement learning agents can learn optimal portfolio management strategies. It allows agents to observe market conditions, take actions such as buying or selling assets, and receive feedback in the form of rewards based on their actions' performance. The environment encapsulates key functionalities required for portfolio management, including tracking portfolio performance, executing trades, and managing available funds.

State Representation

The Portfolio environment state representations is a rich array of technical indicators, including moving averages (MA20, MA50, MA200), average true range (ATR), and volume. This enhanced representation empowers agents to discern more nuanced market signals, facilitating more informed decision-making and adaptive portfolio management strategies.

Action Interpretation

Within the Portfolio environment, actions agents can opt to directly adjust portfolio allocations, a strategy referred to as portfolio interpretation.

Market Simulation

A core feature of the **Portfolio** environment is its ability to simulate market behavior using historical data for multiple assets. Agents interact with this simulated market environment by making decisions on portfolio allocations or transactions. As agents execute actions, the environment dynamically updates asset prices to reflect the impact of their decisions on the market. This dynamic interaction enables agents to learn and adapt to changing market conditions, honing their ability to navigate complex investment landscapes and optimize portfolio performance over time.

Reward Calculation

In the environment, agent performance is evaluated based on the change in portfolio value over time. Rewards are calculated as agents aim to maximize portfolio performance by making decisions that lead to favorable outcomes, such as maximizing returns while minimizing risks. This reward scheme incentivizes agents to learn effective investment strategies and adapt their behavior to achieve optimal portfolio outcomes in the simulated market environment, fostering continuous improvement and adaptive decision-making strategies.

Overall, this environment offers a sophisticated and customizable platform for training and evaluating portfolio management strategies. By providing diverse state representations, action interpretations, and realistic market simulations, the environment empowers agents to learn from historical data and optimize portfolio performance in dynamic market conditions.

**3.3 Initialising 4 agents- DDPG,A2C,PPO and TD3**

**A2C: Advantage Actor-Critic**

Overview

Advantage Actor-Critic (A2C) is a popular deep reinforcement learning algorithm that combines elements of both policy-based and value-based methods. A2C belongs to the broader class of actor-critic algorithms, where an actor network learns a policy (actions) and a critic network evaluates the value of state-action pairs. The key innovation of A2C lies in its use of an advantage function to estimate the relative advantage of taking a particular action over others, leading to more efficient and stable learning.

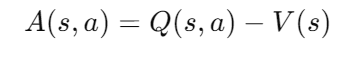
Actor-Critic Architecture

The Advantage Actor-Critic (A2C) algorithm is a variant of the actor-critic architecture, which combines elements of both policy-based and value-based methods in reinforcement learning. This architecture consists of two main components: the actor network and the critic network, each serving a distinct role in the learning process.

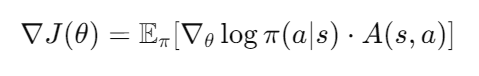
Actor Network**:** The actor network parameterizes a policy 𝜋(𝑎∣𝑠), where given a state 𝑠s, it outputs a probability distribution over possible actions 𝑎. Essentially, the actor learns to predict the probability of selecting each action based on the current state. By exploring the environment and observing rewards, the actor network adjusts its parameters to maximize expected returns over time.

Critic Network: The critic network estimates the value function 𝑉(𝑠), which represents the expected return or cumulative reward from being in state 𝑠s onwards. The critic evaluates the goodness of states and helps the actor network make better decisions by providing feedback on the expected value of taking certain actions in different states. By learning the value function, the critic network guides the actor towards actions that lead to higher cumulative rewards.

Advantage Function

The advantage function, denoted by *A*(*s*,*a*), quantifies the advantage of taking action *𝑎* in state *𝑠s* compared to the average action value. It is calculated as the difference between the state-action value *Q*(*s*,*a*) and the value function *V*(*s*). Intuitively, the advantage function tells the agent how much better (or worse) an action is compared to the average action in a given state. By incorporating the advantage function into the policy gradient updates, A2C encourages the agent to prioritize actions that lead to better-than-average returns, thereby enhancing learning efficiency and stability.

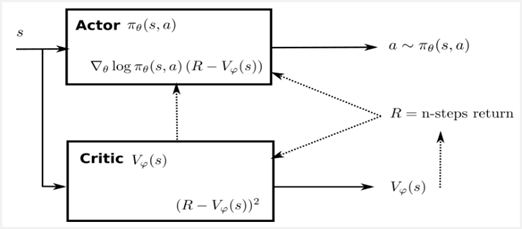
Policy Gradient Updates

A2C utilizes policy gradient methods to update the parameters of the actor network in the direction that maximizes expected returns. The policy gradient is computed as:

This equation represents the gradient of the expected return with respect to the actor network parameters. By leveraging this gradient, A2C adjusts the actor's parameters to increase the likelihood of actions that yield higher-than-average returns while decreasing the likelihood of actions with lower returns. This iterative process helps the agent learn a policy that maximizes cumulative rewards over time.

Parallel Advantage

A notable feature of A2C is its ability to leverage parallel environments to generate multiple trajectories concurrently. This parallelization enables more efficient data collection and utilization, leading to faster and more stable learning. By gathering experiences from multiple trajectories in parallel, A2C reduces the variance of the gradient estimates and accelerates convergence. This capability makes A2C well-suited for tackling large-scale reinforcement learning tasks, where extensive exploration and data collection are necessary for effective policy learning.



**Deep Deterministic Policy Gradient (DDPG) Algorithm**

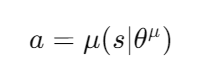
Overview:

Deep Deterministic Policy Gradient (DDPG) is a model-free, off-policy reinforcement learning algorithm designed to handle continuous action spaces in environments with high-dimensional state spaces. DDPG is an extension of the actor-critic framework, combining the advantages of both value-based and policy-based methods. It employs deep neural networks to approximate both the actor (policy) and critic (action-value) functions, enabling it to learn complex, high-dimensional policies for continuous control tasks.

Actor-Critic Architecture:

DDPG utilizes an actor-critic architecture, where the actor learns a deterministic policy *𝜇(𝑠)*, mapping states to specific actions, and the critic evaluates the action-value function *𝑄(𝑠,𝑎)*, estimating the expected return of taking action *𝑎* in state s. The actor network outputs continuous action values directly, allowing for deterministic policy updates, while the critic network estimates the value of the state-action pairs.

Actor Network: The actor network in the Deep Deterministic Policy Gradient (DDPG) algorithm parameterizes the policy function *𝜇(𝑠∣𝜃𝜇),* where *𝜃𝜇* represents the parameters of the actor network. Given a state *𝑠s*, the actor network outputs a deterministic action a,



which represents the action to be taken in that state. This deterministic policy allows for direct action selection without the need for exploration strategies such as epsilon-greedy or softmax.

The actor network is typically implemented using a deep neural network (DNN), such as a feedforward neural network or a convolutional neural network (CNN). The input to the actor network is the state s, and the output is the action a . The network architecture can vary depending on the complexity of the environment and the action space. Common activation functions used in the actor network include ReLU, tanh, or sigmoid functions.

During training, the actor network parameters *𝜃𝜇* are updated to maximize the expected return by following the policy gradient ascent method. The gradient of the expected return with respect to the actor network parameters is estimated using the chain rule, allowing the actor network to learn an optimal policy that maximizes cumulative rewards over time.

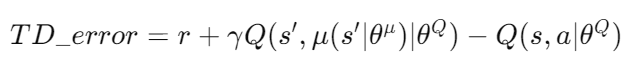
Critic Network: The critic network in DDPG estimates the action-value function *(𝑠,𝑎∣𝜃𝑄*), where *𝜃Q* represents the parameters of the critic network. The action-value function predicts the expected return when taking action a *in state s* . In other words, it estimates the long-term value of selecting a particular action in a given state.

Similar to the actor network, the critic network is typically implemented using a DNN. The input to the critic network is the state sand the action a, and the output is the estimated action value *(𝑠,𝑎∣𝜃𝑄)*. The network architecture may involve concatenating the state and action as inputs or using a separate pathway for each. Common activation functions used in the critic network include ReLU or linear functions.

During training, the critic network parameters *𝜃Q* are updated to minimize the temporal difference (TD) error, which measures the discrepancy between the predicted action value and the target value. This is typically done using methods such as gradient descent or stochastic gradient descent, where the target value is computed using the Bellman equation or its variants.

Bellman Equation:

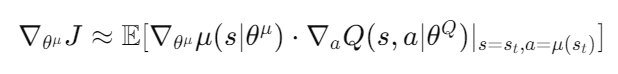
The critic network is trained to minimize the temporal difference (TD) error, which is computed using the Bellman equation:



where r *is the immediate reward received after taking action 𝑎a* in state s, *𝑠′* is the next state, *𝛾* is the discount factor, and *𝑄(𝑠′,𝜇(𝑠′∣𝜃𝜇)∣𝜃𝑄)* is the target value estimated by the critic for the next state-action pair.

Policy Gradient Updates:

The actor network in DDPG is updated using the policy gradient ascent method. This involves estimating the gradient of the expected return with respect to the actor network parameters. By maximizing this gradient, the actor network learns to improve its policy towards actions that lead to higher cumulative rewards. The gradient is typically estimated using the chain rule, which involves computing the gradient of the action probabilities with respect to the actor network parameters and scaling it by the advantage function.



Experience Replay:

DDPG employs an experience replay buffer to enhance learning stability and efficiency. The replay buffer stores past experiences, typically in the form of tuples containing the state, action, reward, and next state. During training, samples are randomly drawn from the replay buffer and used to update the actor and critic networks. This process helps to decorrelate the data and break temporal correlations, preventing the networks from overfitting to recent experiences and leading to more robust learning.

Target Networks:

To further stabilize training and improve convergence, DDPG utilizes target networks. These are copies of the actor and critic networks that are periodically updated using a soft target update mechanism. Instead of updating the target networks with the exact parameters of the main networks at each iteration, the target networks are updated gradually by averaging their parameters with those of the main networks. This soft target update mechanism helps to reduce the variance of the Q-value and policy estimates, leading to smoother and more stable learning. By using target networks, DDPG is able to mitigate the issue of overestimation bias and achieve better performance in training.

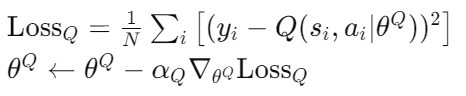
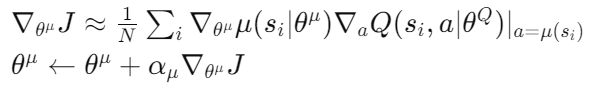
**Algorithm :-**

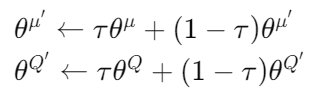
1. Initialize actor network *𝜇* with random weights *𝜃𝜇*, critic network *Q* with random weights *𝜃Q*, target networks *𝜇*′ and *Q*′ with weights and , respectively.. 2.Initialize replay buffer *D*.

3. For each episode:

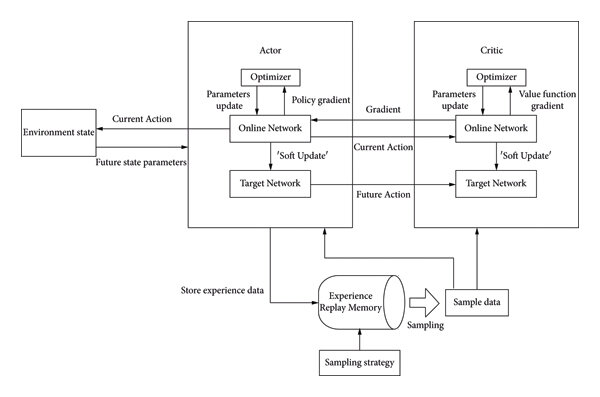
* Initialize a random process for action exploration (e.g., Ornstein-Uhlenbeck process).
* Receive initial observation *s*1 .
* For each time step t in the episode:
* Select action according to the current policy and exploration noise.



* Execute action *at* and observe reward *𝑟t* and next state st+1 .
* Store transition *(𝑠, 𝑎 t , 𝑟t ,𝑠𝑡+1)*in replay buffer *D*.
* Sample a random minibatch of transitions *(si , 𝑎i , 𝑟i , 𝑠𝑖+1)* from replay buffer *D*.
* Compute target Q-value
* Update critic network by minimizing the TD error: 
* Update actor network using the sampled policy gradient:
* Update target networks with a soft update:



4. Repeat until convergence.



DDPG is a powerful algorithm for solving continuous control problems in reinforcement learning. By combining deep neural networks with deterministic policies and off-policy learning, DDPG can effectively learn complex policies for a wide range of continuous control tasks. Its ability to handle high-dimensional state and action spaces makes it a versatile and widely used algorithm in the field of reinforcement learning.

### Proximal Policy Optimization (PPO) Algorithm

#### Overview

Proximal Policy Optimization (PPO) is a popular reinforcement learning algorithm designed to balance the trade-off between exploration and exploitation while ensuring stable and efficient policy updates. PPO is an on-policy method that improves upon the vanilla policy gradient methods by incorporating mechanisms to prevent large, destabilizing policy updates. It has been widely adopted for various complex environments, particularly those with high-dimensional state and action spaces.

#### Policy and Value Networks

PPO uses two neural networks: the policy network (actor) and the value network (critic).

### Policy Network (Actor)

The policy network, or actor, in Proximal Policy Optimization (PPO) is responsible for determining the actions that an agent should take in a given state to maximize its long-term rewards. This network parameterizes a stochastic policy *𝜋𝜃(𝑎∣𝑠)*, where *𝜃* represents the network's parameters. Given a state , the policy network outputs a probability distribution over possible actions *𝑎* . This distribution reflects the likelihood of taking each action given the current state, thus allowing the agent to explore different actions while learning which ones yield the highest rewards.

The stochastic nature of the policy network is crucial for exploration in reinforcement learning. By sampling actions from a probability distribution rather than selecting the highest-valued action deterministically, the agent can discover potentially better strategies that might be missed otherwise. The policy network is typically implemented using deep neural networks (DNNs), which can effectively handle high-dimensional state spaces and complex environments. The input to the policy network is the current state *𝑠s*, and the output is a probability distribution over actions. Common activation functions in the hidden layers include ReLU (Rectified Linear Unit) or tanh, while the output layer often uses a softmax function to ensure the output is a valid probability distribution.

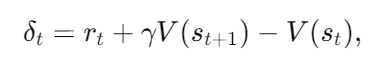
The primary goal of the policy network is to maximize the expected cumulative reward, also known as the expected return. This is achieved by adjusting the network parameters *𝜃* through gradient ascent. The policy gradient, which is the gradient of the expected return with respect to the policy parameters, guides these updates. By following the policy gradient, the network learns to increase the probability of actions that lead to higher returns and decrease the probability of less favorable actions. This iterative process allows the policy network to progressively improve its action-selection strategy.

### Value Network (Critic)

The value network, or critic, in PPO estimates the state-value function *𝑉𝜙(𝑠)*, where *𝜙* are the parameters of the value network. The state-value function *𝑉(𝑠)* represents the expected return, or cumulative reward, from state *𝑠* onwards, given that the agent follows the current policy. This estimation helps the actor network by providing feedback on how good it is to be in a particular state, essentially serving as a baseline to measure the advantage of actions taken.

The value network is also typically implemented using deep neural networks, with the state  as input and the estimated value *𝑉(𝑠)*  as output. The network architecture can vary depending on the complexity of the task, but common activation functions in the hidden layers include ReLU or linear functions. The value network aims to approximate the true value function as accurately as possible to reduce the variance of the policy gradient estimates, making the learning process more stable and efficient.

During training, the parameters of the value network *𝜙* are updated to minimize the error between the predicted value and the actual return observed. This error, known as the temporal difference (TD) error, is given by:

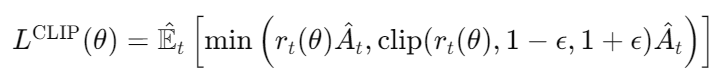


where *𝑟t*  is the reward received after taking an action in state *𝑠t* , *𝑠𝑡+1*  is the subsequent state, and *𝛾* is the discount factor. The value network is trained using gradient descent to minimize the mean squared error between the predicted values and the observed returns. By minimizing this error, the value network provides more accurate estimates of the expected returns, which in turn helps the policy network make better decisions.

Together, the policy and value networks form the backbone of the PPO algorithm. The policy network focuses on selecting actions that maximize rewards, while the value network ensures these actions are evaluated accurately, providing a robust and stable learning framework for continuous and high-dimensional control tasks.

#### Policy Update Mechanism

PPO introduces a novel objective function to ensure stable policy updates. The main idea is to clip the probability ratio between the old and new policies to prevent large updates that can lead to instability. The PPO objective function is defined as:

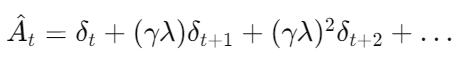
where:

* is the probability ratio of the new and old policies.
* *Ât* is the advantage estimate at time step *𝑡*.
* *𝜖* is a hyperparameter that controls the clipping range.

The clipping function ensures that the policy update does not deviate too much from the old policy.

#### Advantage Estimation

The advantage function Ât  is used to measure the relative value of taking action    
*𝑎𝑡* in state *𝑠𝑡*  compared to the average action. It is typically computed using Generalized Advantage Estimation (GAE):



where *𝛿𝑡* is the temporal difference (TD) error:



with *𝛾* being the discount factor and *𝜆* being the GAE parameter.

ALGORITHM:-

1.Initialize the policy network *𝜋𝜃*  with random weights *𝜃*.

2.Initialize the value network *𝑉𝜙* with random weights *𝜙*.

3.Initialize the old policy network *𝜋𝜃old* with *𝜃old←𝜃*.

4.Repeat for each iteration:

Collect Experience:

* + Run the policy *𝜋* in the environment for *𝑇* timesteps to collect trajectories {(*st* ,*at* ,*rt* ,*st*+1 )}.
  + Compute returns *𝑅𝑡* and advantage estimates *Ât* for each timestep.

Policy Update:

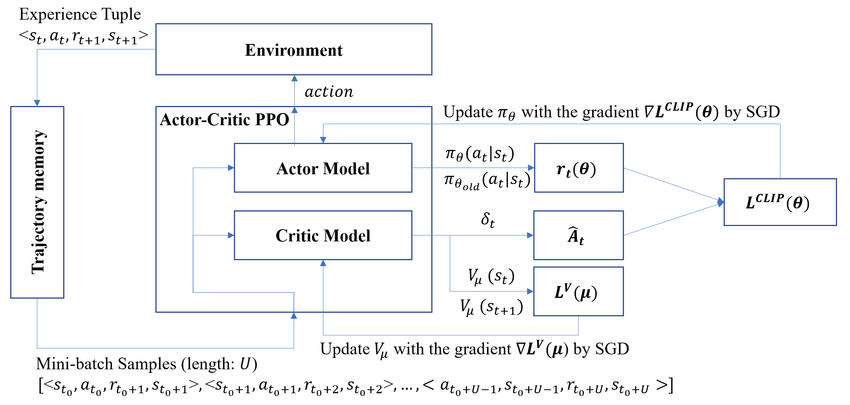
* + For each epoch:
    - Sample a minibatch of transitions from the collected trajectories.
    - Compute the PPO objective *𝐿CLIP(𝜃)* and update *𝜃* using stochastic gradient ascent.

Value Update:

* + For each epoch:
    - Sample a minibatch of transitions *{(st , 𝑅𝑡)}* from the collected trajectories.
    - Compute the value loss *𝐿𝑉(𝜙)* and update *𝜙* using gradient descent.

Update the Old Policy:

* + After the policy update, set *𝜃old←𝜃*

5.Repeat until convergence.

**TWIN DELAYED DEEP DETERMINISTIC POLICY GRADIENT (TD3) ALGORITHM**

#### Overview

Twin Delayed Deep Deterministic Policy Gradient (TD3) is a reinforcement learning algorithm designed for continuous control tasks, including portfolio management. TD3 enhances the original DDPG algorithm by using twin Critic networks to estimate Q-values, which mitigates overestimation bias. This approach improves stability and performance, making TD3 suitable for complex environments with high-dimensional state and action spaces.

#### Policy and Value Networks

In TD3, the policy network (Actor) determines actions to maximize long-term rewards, parameterized by a stochastic policy 𝜋(𝑎∣𝑠). The Actor network outputs a probability distribution over actions, allowing for exploration. The value network (Critic) estimates the state-value function 𝑉(𝑠), providing feedback on state goodness and serving as a baseline for action advantage estimation.

#### Policy Update Mechanism

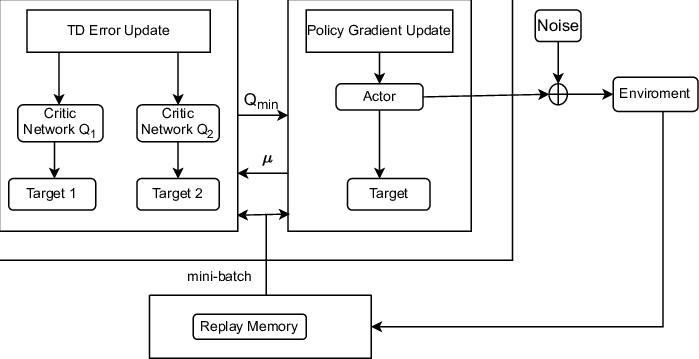
TD3 introduces a novel objective function to ensure stable policy updates. It clips the probability ratio between old and new policies to prevent large updates. The objective function is defined as a combination of the clipped surrogate objective and a regularization term, ensuring that the policy update does not deviate too much from the old policy.

#### Advantage Estimation

Advantage estimation in TD3 measures the relative value of taking an action compared to the average action. It is typically computed using Generalized Advantage Estimation (GAE), which considers the temporal difference error and a discount factor to balance immediate rewards and future expectations.

#### Algorithm

1. **Initialization:** Initialize the policy and value networks with random weights.
2. **Experience Collection:** Collect trajectories by running the policy in the environment.
3. **Policy Update:** Sample a minibatch of transitions and update the policy using the clipped surrogate objective.
4. **Value Update:** Sample a minibatch of transitions and update the value network to minimize the value loss.
5. **Update Old Policy:** After policy update, update the old policy to the current policy.
6. **Convergence:** Repeat the process until convergence, optimizing the policy and value networks to improve portfolio management decisions.



**3.4 TRAIN VALIDATE TEST**

**Train**

The primary goal of the **train** method is to iteratively improve the performance of the DDPG agent through learning and experience replay.

Training the Agent:

The agent interacts with the environment by choosing actions, receiving rewards, and learning from the outcomes.This process involves updating the agent's policy and value networks to better predict the optimal actions for maximizing rewards.

Tracking Performance:

The method keeps track of the cumulative returns (wealth) over multiple iterations.This helps in monitoring the agent's progress and understanding how well it is learning.

Validation During Training:

After each training iteration, the agent's performance is validated on a separate validation dataset.This helps to prevent overfitting and ensures that the agent is generalizing well to unseen data.

Model Checkpointing:

The agent's model is saved whenever it achieves the highest validation performance.This ensures that the best-performing model is preserved for future use.

Early Stopping:

The training process includes an early stopping criterion based on validation performance to prevent unnecessary computation and overfitting.

Visualization:

The cumulative returns for both training and validation are plotted and compared to a benchmark (Buy&Hold strategy).These plots help in visualizing the agent's learning progress and its effectiveness compared to a basic investment strategy.

**Validate**

The validate serves as a checkpoint to evaluate the agent's performance on unseen validation data during the training process.

Performance Assessment:

It provides a measure of how well the agent has learned to manage the portfolio based on its current policy.This helps in identifying if the agent is overfitting to the training data or if it is generalizing well.

Guiding Training Decisions:

The validation results are used to make decisions about saving the model or continuing training.If the agent performs well on the validation set, it indicates that the current model is effective.

Model Selection:

The validation process helps in selecting the best model by comparing the agent's performance across different iterations of training.

Preventing Overfitting:

Regular validation checks ensure that the model does not become too specialized to the training data, which can lead to poor performance on real-world data.

**Test**

The **test** evaluates the fully trained agent's performance on a completely separate test dataset, which has not been used during training or validation.

Final Performance Evaluation:It provides an unbiased assessment of the agent's ability to manage the portfolio in a realistic setting. This is the ultimate test to see how well the agent performs in practice.

Comparison with Benchmark:The agent's performance is compared to a Buy&Hold strategy to determine if it provides a significant improvement over a simple investment strategy.This comparison helps in understanding the value added by the DDPG agent.

Performance Metrics:The method calculates detailed performance statistics (e.g., cumulative return, daily returns) to provide a comprehensive evaluation of the agent.These metrics include standard financial performance measures which are crucial for assessing the quality of the investment strategy.

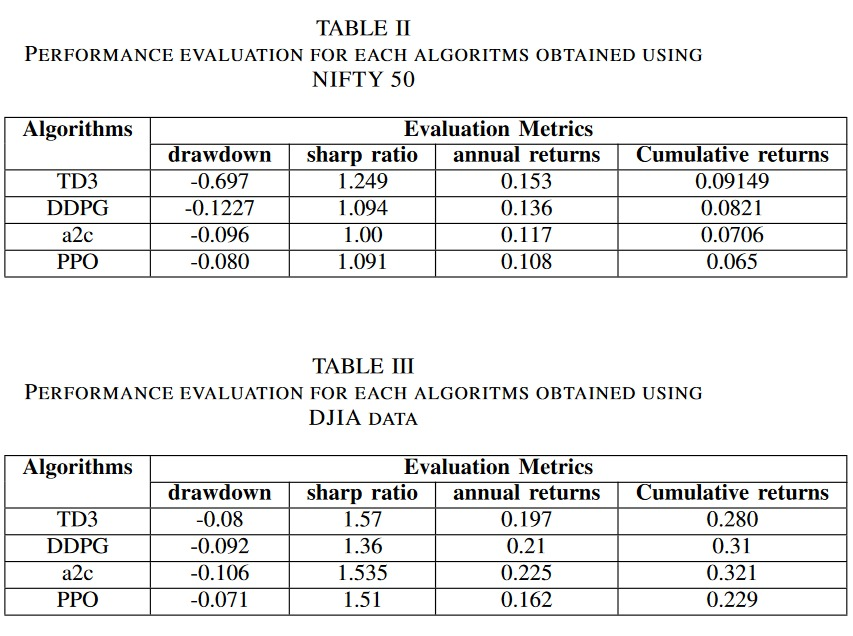
Visualization and Reporting:The results are visualized through plots, which makes it easier to understand the agent's performance over time.The performance statistics are saved in a CSV file for further analysis and reporting.

**3.5 Evaluate the agent’s performance**

The following metrics are evaluated for each agent to measure the performance

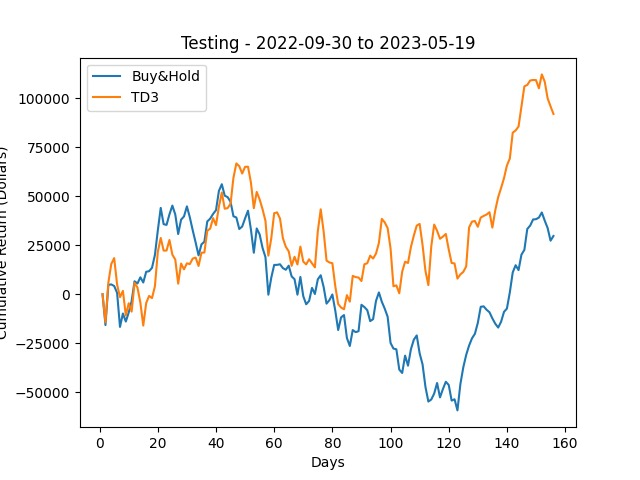
* Risk-Adjusted Returns: Sharpe ratio, Sortino ratio, Calmar ratio.
* Risk Measures: Annual volatility, Max drawdown, Daily value at risk.
* Return Measures: Cumulative returns, Annual returns.
* Distribution Characteristics: Skew, Kurtosis, Stability.
* Tail Risk: Tail ratio, Omega ratio.

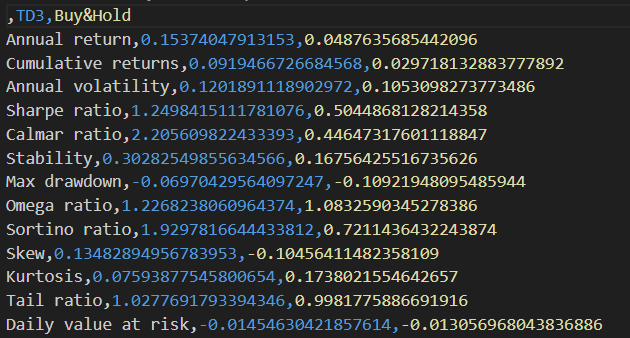
**4.Results**

****

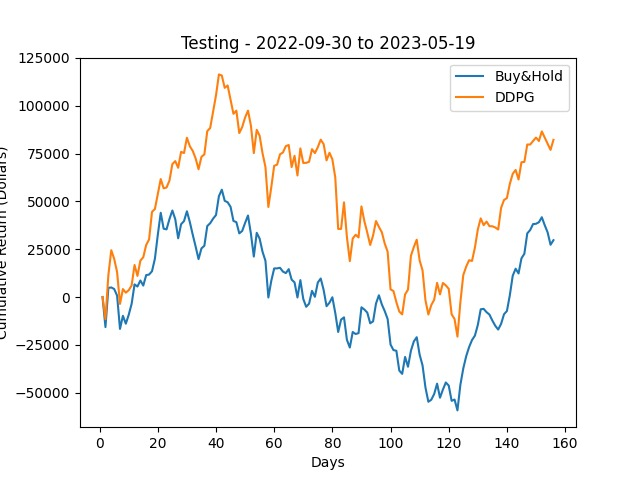
Results – NIFTY50

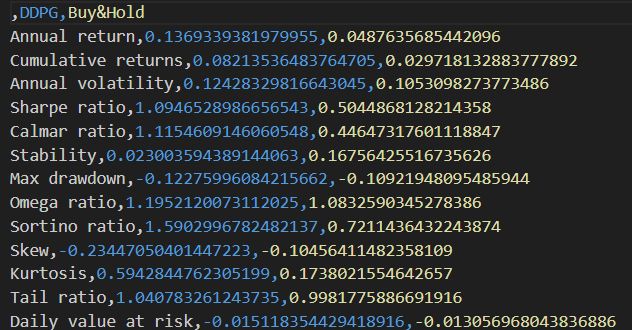
1)TD3

****

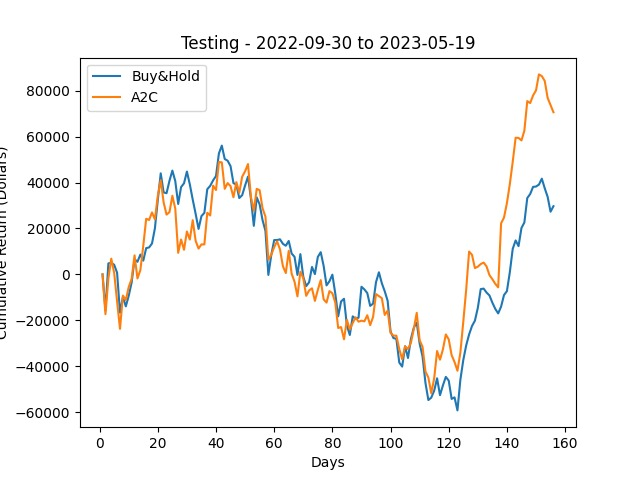
****

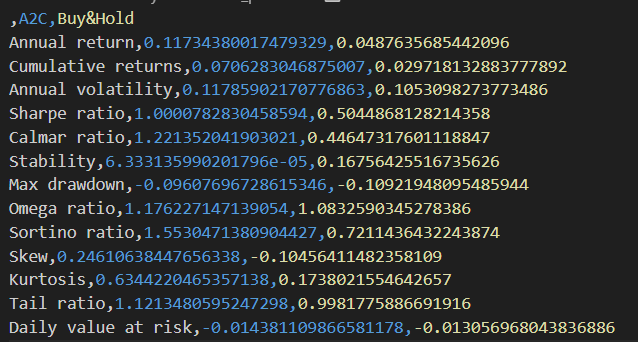
2)DDPG



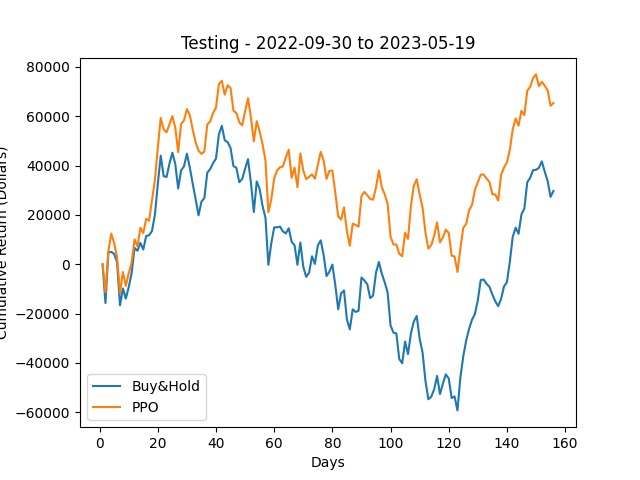


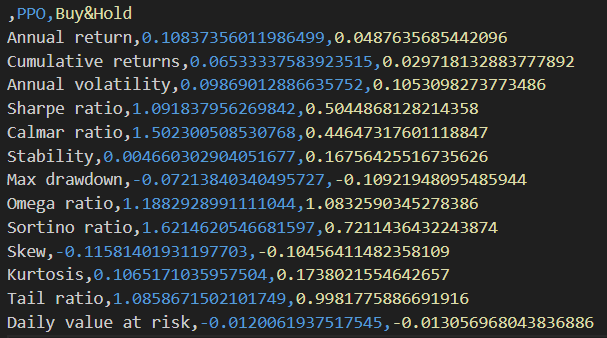
3)A2C





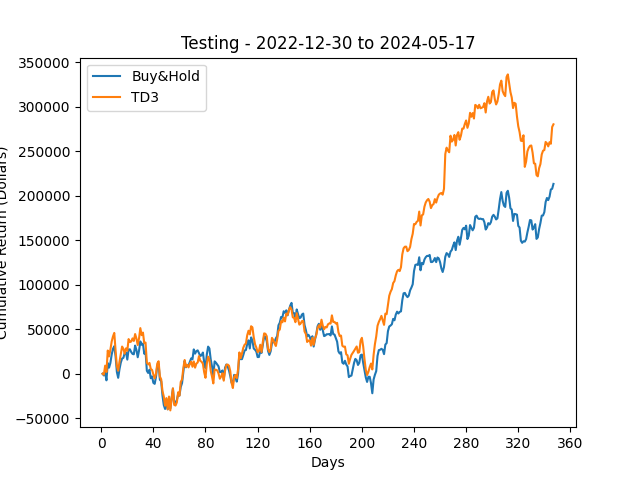
4)PP0

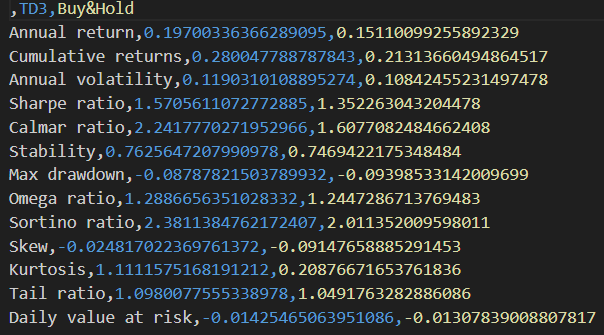
****

****

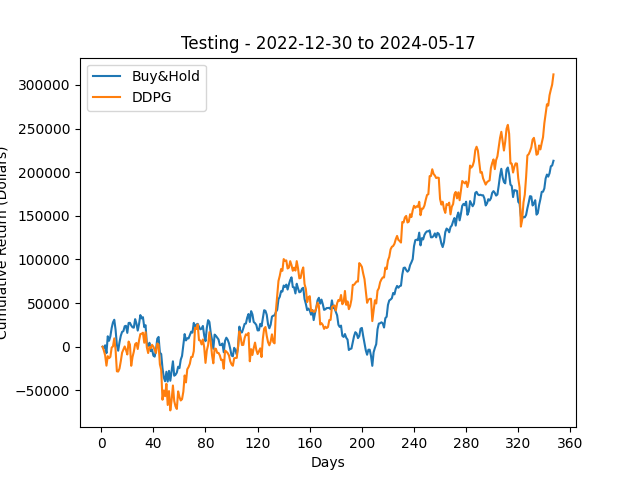
Results – DJIA

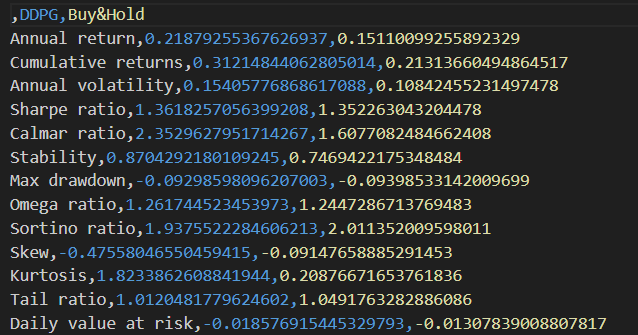
1)TD3



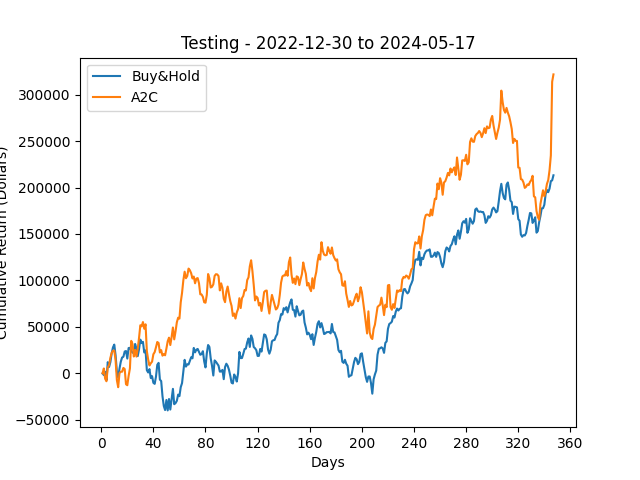


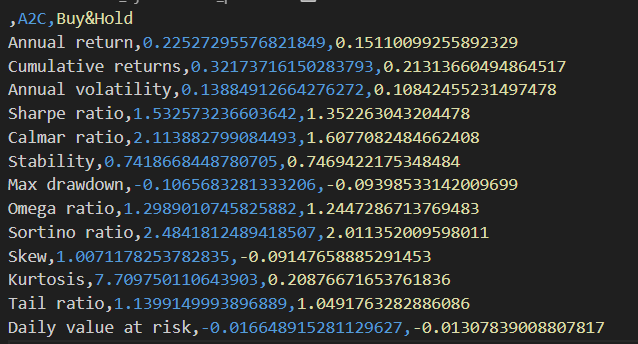
2)DDPG



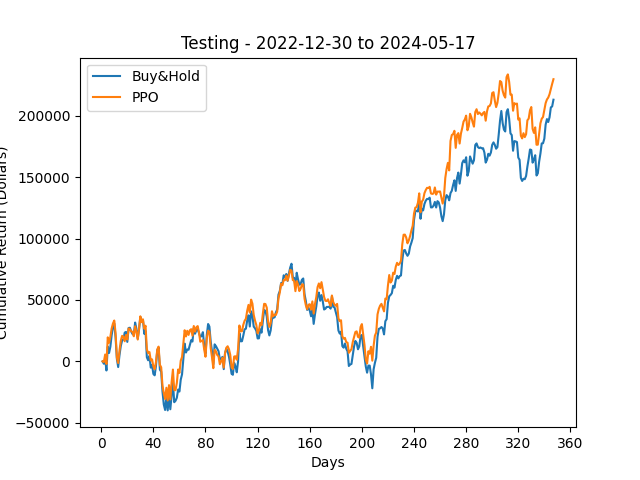


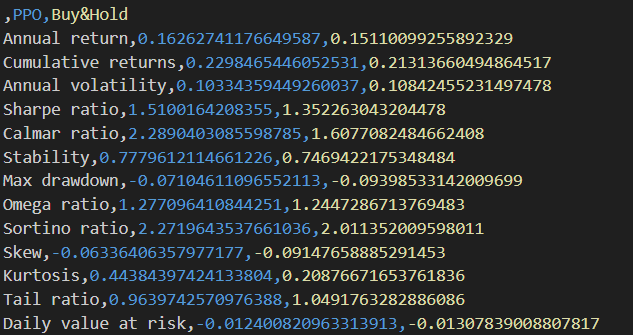
3)A2C





4)PP0





**5 REFERENCES**

1. Zhang, Huanming & Zhengyong, Jiang & Su, Jionglong. (2021). A Deep Deterministic Policy Gradient-based Strategy for Stocks Portfolio Management**.**
2. Liang, Z., Jiang, K., Chen, H., Zhu, J., & Li, Y. (2018). Deep Reinforcement Learning in Portfolio Management. *ArXiv, abs/1808.09940*.
3. <https://medium.com/analytics-vidhya/portfolio-optimization-using-reinforcement-learning-1b5eba5db072>