Reinforcement Learning Stacking Strategy for Stock Trading

Leiguang Ren
Department of Computer Science
Stanford University
Stanford, USA
lgren007@stanford.edu

Rui Xu
Department of Computer Science
Stanford University
Stanford, USA
ruxu@stanford.edu

Gabriel SantaCruz
Department of Computer Science
Stanford University
Stanford, USA
gsantac@stanford.edu

***Abstract*—This paper introduces a novel stacking-based RL strategy that integrates five advanced RL algorithms—Advantage Actor-Critic (A2C), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), Twin Delayed DDPG (TD3), and Soft Actor-Critic (SAC)—to enhance portfolio optimization. By training base models and incorporating their outputs into a final stacking agent, the proposed method improves portfolio return and reduces variances in trading performance. In our experiment, the proposed stacking agent achieved higher average portfolio returns and reduced variance compared to a rolling best-agent selection method that selects the best single agent periodically.**

Index Terms—Ensemble reinforcement learning, Markov Decision Process, automated stock trading, stacking strategy, actor-critic framework

I. Introduction

The primary objective of portfolio management is to maximize portfolio returns while effectively managing exposure to various market risks. Achieving this is inherently challenging due to the financial markets' complexity [1].

Over the years, a multitude of approaches have been developed to address the portfolio management problem. Traditional finance methods, such as Modern Portfolio Theory (MPT) introduced by Markowitz, focus on optimizing the expected return based on the mean and variance of portfolio returns [2]. These methods rely on well-defined statistical properties and assumptions about market behavior. More recently, computational techniques utilizing Markov Decision Processes (MDP) have been employed to model the sequential and dynamic nature of investment decisions, allowing for more adaptive strategies that consider temporal dependencies and evolving market [4].

In the past decade, reinforcement learning (RL) has emerged as a natural and powerful framework for portfolio management. RL algorithms are particularly well-suited for this task as they can learn optimal trading strategies through continuous interaction with the market environment. Despite its promising potential, individual RL agents often face limitations, including overfitting to specific market patterns and exhibiting high variance in performance across different and unseen market conditions [5]. These challenges undermine the reliability and effectiveness of single-agent RL approaches in real-world trading scenarios.

To address these issues, some approaches have attempted to enhance RL performance by periodically retraining multiple agents and selecting the best-performing agent for subsequent trading periods. While this periodic retraining and agent selection can improve return, this method relies on simple agent selection criteria without fully exploiting the diverse strengths of different RL algorithms [6], which is known to improve bias and variance [7].

To overcome these limitations and enhance the robustness and performance of RL-based portfolio management, this paper introduces an RL stacking strategy that integrates the strengths of multiple RL algorithms. Ensemble methods, particularly stacking, have been widely recognized in classical machine learning for their ability to reduce bias and variance by combining multiple models [6]. By integrating five advanced RL algorithms—Advantage Actor-Critic (A2C), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), Twin Delayed DDPG (TD3), and Soft Actor-Critic (SAC)—into a final agent trained using SAC, our approach leverages ensemble learning techniques commonly employed in classical ML [1]. This final agent effectively exploits the diverse strengths of each base agent while mitigating their individual weaknesses, resulting in a more robust and reliable trading strategy.

A. Related Works

We build upon FinRL, an open source library proposing DRL frameworks for automated stock trading in quantitative finance [5]. FinRL's library allows for seemless integration of our stock trading enviornment and provides baseline models to expand upon. In particular, we focus on their ensemble model that uses the five algorithms mentioned above. The ensemble model pretrains each of the agents so that when choosing an action, it selects the output of the optimal agent for a given time period when trading stocks. The combination of these different models makes it a strong benchmark to test the performance of our strategy. By definition, our stacking strategy is also considered an ensemble method, however we will refer to the method proposed by FinRL as ensemble and ours as stacking for the rest of the paper.

Stacking strategies have been used for generalization tasks that work to correct the biases for a learning set [7]. Wolpert describes stacking as scheme that aims to combine the models rather than simply picking one model and makes the comparison that with multiple agents, it's a more robust version of cross-validation [7]. Stacking proves to improve performance over choosing individual agents and we choose to integrate this scheme on existing deep reinforcement learning frameworks (FinRL) for stock trading [7].

II. Methodology

A. Market Environment Assumptions and Constraints

To accurately simulate real-world stock trading environments, we establish several key assumptions and impose specific constraints:

Transaction Costs: Each trade incurs a cost proportional to the amount traded, specifically a commission rate of 0.1%. This cost is incorporated into the reward function to penalize excessive trading and promote cost-effective strategies.

Market Impact: We assume that the agent's trades are sufficiently small to have no significant impact on market prices. This assumption allows us to simplify the training process by disregarding potential price movements caused by our trading activities.

Action Size: We cap the magnitude of actions that trading agents can execute, specifically limiting trades to a maximum of 1000 shares per stock.

Trading Resolution: Our approach focuses on end-of-day trading, assuming that all trading decisions are based on the closing prices of stocks. This reduces the size of the data we need to work with.

Number of Stocks: We restrict the number of tradable stocks within the portfolio, operating under the assumption that not all stocks are available for trading. This helps reduce the complexity of our problem.

B. Training Process

We employ five distinct reinforcement learning (RL) algorithms, each possessing unique strengths and weaknesses. We use these algorithms for both the base agents (sometimes referred to as Level-0 agents) and also stacking agent (often referred to as Meta-agent or Level-1 agent):

A2C: A synchronous policy gradient algorithm known for its stability during training [9].

DDPG: Suited for continuous action spaces and effective in handling high-dimensional problems [10].

PPO: Recognized for its robustness and efficiency in policy updates [11].

TD3: An enhancement over DDPG that addresses function approximation errors [12].

SAC: Optimizes a stochastic policy to enhance exploration capabilities [13].

There are two phases of training. During phase-1, each base agent is trained individually using the same training data. During phase-2, we incorporate pre-trained agents as part of the environment for stacking agent-2, who observes both the market environment as well as the output of each pre-trained agents as part of the State variables.

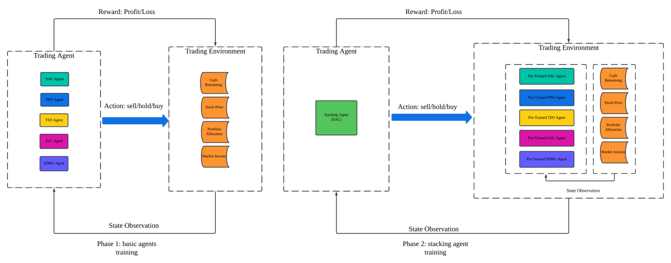

Fig. 1. Overview of stock trading stacking strategy

The final stacking model employs the Soft Actor-Critic (SAC) algorithm as the stacking agent. SAC was selected for its ability to optimize cumulative rewards while effectively integrating diverse inputs from base models, market indicators, and portfolio features. By dynamically leveraging the strengths of the base models, the stacking agent achieves superior adaptability and performance.

C. MDP Model

State space includes the following features:

- **Account-Level Features:** The available cash in the account (C_t) and the number of shares held for each stock ($shares_t^i$) at time t. These features are critical for calculating the total portfolio value:
- **Technical Indicators:** For each stock i, four features capturing relative price changes over the past four time steps:
$$\frac{P_t^i - P_{t-4}^i}{P_{t-4}^i}, \quad \frac{P_t^i - P_{t-3}^i}{P_{t-3}^i}, \quad \frac{P_t^i - P_{t-2}^i}{P_{t-2}^i}, \quad \frac{P_t^i - P_{t-1}^i}{P_{t-1}^i}.$$
These trends provide a snapshot of recent stock performance to inform trading decisions.
- **Base Model Outputs:** Actions from the five base agents are part of the environment in the stacking setting. For each stock i:
$$a_{\text{A2C}}^i, \quad a_{\text{DDPG}}^i, \quad a_{\text{PPO}}^i, \quad a_{\text{TD3}}^i, \quad a_{\text{SAC}}^i.$$

Actions are represented as a vector of continuous values between -1 and 1. A negative a_{stacking}^i indicate selling, and a positive a_{stacking}^i indicates buying.

$$shares_{t+1}^i = shares_t^i + h_{\text{max}} * a_{\text{stacking}}^i$$

Reward function aims to maximize portfolio returns and is defined as:

$$r_t = \text{Portfolio Value}_{t+1} - \text{Portfolio Value}_t$$

where the account value at time t is:

$$\text{Portfolio Value}_t = C_t + \sum_i P_t^i \cdot shares_t^i.$$

Transitions are probabilistic, as stock prices are unpredictable due to the complexity of environment, majority of which are unobserveable. While there is a rich history of modeling price dynamics, including seminal works like the

Black-Scholes model, this paper avoids explicitly modeling the transition function. Instead, we leverage model-free reinforcement learning, which does not depend on a transition model at all.

D. Formulation

The relationship between the base agents and the stacking agent can be represented as follows:

$$a_{\text{stacking}} = \pi_{\text{stacking}}(a|s, a_{\text{A2C}}, a_{\text{DDPG}}, a_{\text{PPO}}, a_{\text{TD3}}, a_{\text{SAC}})$$

π_{stacking} is the stacking agent's policy

s is the current state

$a_{A2C}, \ldots, a_{SAC}$ are actions proposed by base agents

The objective of the RL agents is to maximize the expected cumulative reward

$$J(\pi_{\text{stacking}}) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t \mid \pi_{\text{stacking}}\right]$$

where: r_t is the portfolio value change from day t to day $t+1$, $\gamma \in [0,1)$ is the discount factor, T is the terminal time step.

In RL, total expected reward is the primary metric for evaluating models, as it directly reflects the agent's ability to maximize returns and achieve its objectives in a given environment. However, since the benefits of stacking in classical machine learning literature often emphasize its ability to reduce bias and variance, we will also formalize **bias** and **variance** as additional metrics for calculation and analysis.

1) Bias: Bias quantifies the systematic deviation of the expected performance of a policy π from the optimal performance J^*. It measures how much the policy underperforms relative to the true optimal cumulative reward:

$$\text{Bias}(\pi) = \mathbb{E}[J^* - J(\pi)]$$

where J^* is the expected cumulative reward under the optimal policy π^*.

Empirical Calculation: Assuming that agent actions do not influence market prices, the maximum cumulative reward (J^*) can be computed (via. Dynamic Programming). However, for practical considerations, we will drop (J^*) from the calculation, looking at negative average return as proxy for Bias:

$$\text{Bias}(\pi) \approx \frac{1}{M}\sum_{m=1}^{M} -J_m(\pi)$$

where M is the number of trials, and $J_m(\pi)$ is the cumulative reward achieved in trial m.

2) Variance: Variance measures the variability in the performance of a policy π, reflecting the consistency of its outcomes across different training runs or scenarios:

$$\text{Variance}(\pi) = \mathbb{E}\left[(J(\pi) - \mathbb{E}[J(\pi)])^2\right]$$

Empirical Calculation: Variance is estimated by computing the deviations of cumulative rewards from their mean:

$$\text{Variance}(\pi) \approx \frac{1}{M}\sum_{m=1}^{M}\left(J_m(\pi) - \frac{1}{M}\sum_{k=1}^{M} J_k(\pi)\right)^2$$

E. Experimental Setup

1) Hyperparameter Tuning: Utilizing a grid search approach, we explored ten distinct hyperparameter configurations for each algorithm, encompassing parameters such as learning rates, batch sizes, discount factors, and entropy coefficients (see Table I for detailed settings). The optimal hyperparameters were selected based on their performance on a validation set.

Recognizing the challenge of assessing RL model performance in stock portfolio optimization—where the optimal policy is inherently unknown—we validated the selected hyperparameters within a simplified toy stock environment characterized by a predictable pyramidal price pattern (Figure 2). This validation confirmed that the RL models could effectively learn and adapt to the environment using the chosen hyperparameters.

TABLE I
HYPERPARAMETER VARIATIONS FOR EACH RL ALGORITHM

Algorithm	Hyperparameters
PPO	learning_rate: [1e-5, 1e-4, 1e-3], n_steps: [256, 512, 1024, 2048], batch_size: [64, 128], gamma: [0.98, 0.99], ent_coef: [1e-5, 1e-3], clip_range: [0.2, 0.3], gae_lambda: [0.95, 0.98]
A2C	learning_rate: [1e-5, 1e-4, 1e-3, 0.0007], n_steps: [5, 10, 20], gamma: [0.98, 0.99], ent_coef: [1e-5, 1e-3, 0.005], normalize_advantage: [True, False], use_rms_prop: [True, False]
DDPG	learning_rate: [1e-4, 1e-3], batch_size: [64, 128], buffer_size: [1e5, 1e6], tau: [0.005, 0.01], gamma: [0.98, 0.99], train_freq: [1, 10], action_noise: ['normal', 'ornstein_uhlenbeck']
SAC	learning_rate: [1e-4, 1e-3], batch_size: [64, 128], buffer_size: [1e5, 1e6], tau: [0.005, 0.01], gamma: [0.98, 0.99], train_freq: [1, 10], ent_coef: ['auto']
TD3	learning_rate: [1e-4, 1e-3], batch_size: [64, 128], buffer_size: [1e5, 1e6], tau: [0.005, 0.01], gamma: [0.98, 0.99], train_freq: [1, 10], action_noise: ['normal', 'ornstein_uhlenbeck']

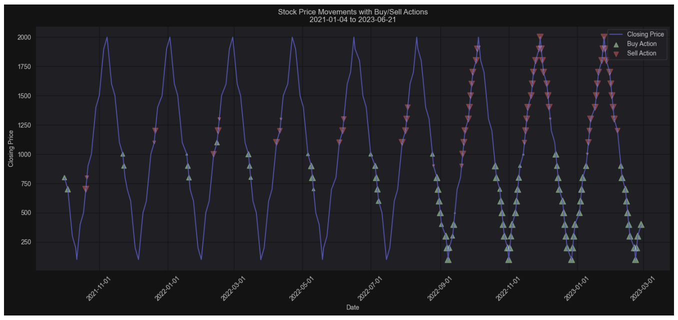

Fig. 2. Simple Stock World Environment illustrating a single stock following a pyramidal price pattern for hyperparameter tuning and the agent learning to buy low and sell high.

2) Training and Testing Framework: We conducted a total of M=10 experimental trials, each comprising the training of five individual agents—each employing one of the five advanced RL algorithms—and a *Stacking Agent* that integrates the outputs of the individual agents. To ensure the robustness and reproducibility of our comparisons, each trial was executed using a distinct random seed, mitigating the influence of stochastic variations inherent in RL training processes.

Training Period: January 1, 2010 – October 1, 2021

Testing Period: January 3, 2022 – January 3, 2023

F. *Performance Analysis*

Our evaluation consisted of two primary analyses:

1) **Individual Agents vs. Stacking Agent:** We compared the performance of individual agents against the Stacking Agent across multiple non-overlapping three-month trading windows.
2) **Stacking Agent vs. Ensemble Agent:** We compared the Stacking Agent's performance with that of the Ensemble Agent over a continuous one-year trading period. The Ensemble Agent dynamically replaced the production agent every three months with the best-performing individual agent based on its performance during the preceding three-month period.

TABLE II
PERFORMANCE OF MODELS ACROSS TIME PERIODS

Policy	Q1-2022	Q2-2022	Q3-2022	Q4-2022
BAH	-0.077 ± 0.0	-0.197 ± 0.0	-0.610 ± 0.0	-0.312 ± 0.0
a2c	-0.029 ± 0.017	0.005 ± 0.015	-0.102 ± 0.015	-0.113 ± 0.023
ddpg	-0.006 ± 0.021	-0.006 ± 0.023	-0.190 ± 0.025	-0.145 ± 0.034
ppo	-0.007 ± 0.012	-0.010 ± 0.010	-0.076 ± 0.017	-0.123 ± 0.022
sac	-0.044 ± 0.014	-0.021 ± 0.012	-0.130 ± 0.017	-0.106 ± 0.028
td3	-0.044 ± 0.024	-0.027 ± 0.020	-0.119 ± 0.026	-0.156 ± 0.032
stacking	-0.011 ± 0.012	-0.006 ± 0.017	-0.083 ± 0.016	-0.059 ± 0.021

1) Individual Agents vs. Stacking Agent: In the first analysis, we assess how each individual RL agent performs relative to the Stacking Agent over distinct three-month trading intervals within the testing period. Additionally, we include the *Buy and Hold* (BAH) strategy as a baseline for performance comparison. The BAH strategy involves purchasing a stock at the beginning of the trading period and retaining it without further transactions, serving as naive benchmark.

Our observations indicate that while the Stacking Agent does not consistently outperform all individual agents across every three-month window, it consistently ranks within the top performers across all evaluated periods.

2) Ensemble Agent vs. Stacking Agent: The second analysis evaluates the performance of an *Ensemble Agent* [5], which employs a rolling window strategy to select the best-performing individual agent based on historical performance metrics, against the *Stacking Agent*. Specifically, the Ensemble Agent utilizes the most effective single algorithm from the five evaluated agents to predict the next three months of trading activities. This selection process is iteratively repeated throughout the entire one-year evaluation period, ensuring that the Ensemble Agent adapts to evolving market conditions by continually updating its choice of the top-performing model.

During the testing period from January 3, 2022, to January 3, 2023, the Ensemble Agent achieved an end-of-year portfolio value of 76.1% (a **-23.9%** loss), the Stacking Agent reached 78.6% (a **-21.4%** loss), and the BAH strategy concluded the year at 70.6% (a **-29.4%** loss). Not only did Stacking Agent deliver higher average return, it also exhibited a significantly lower **maximum drawdown**.

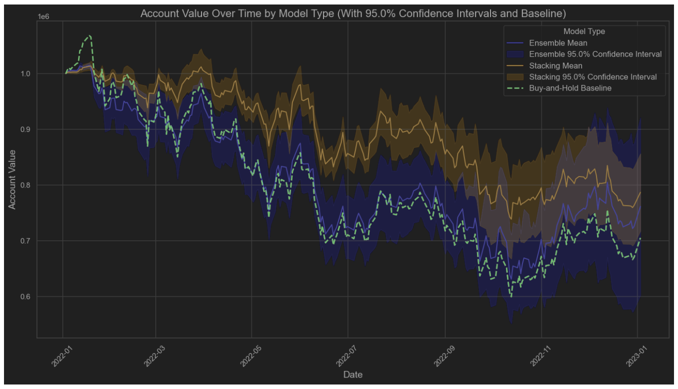

Fig. 3. Comparative performance of the Ensemble Agent and the Stacking Agent during the 2023 testing period.

G. *Robustness and Statistical Significance*

To ensure the reliability and generalizability of our findings, each experimental trial was conducted using different random seeds. This approach addresses the inherent stochasticity in reinforcement learning algorithms, as emphasized by [3], ensuring that our performance metrics are not biased by specific random initializations.

Our statistical analysis revealed that the Stacking Agent outperformed the Ensemble Agent, especially during bear market periods when stock prices were declining. Specifically, the 95% confidence interval for the Stacking Agent's performance did not overlap with that of the Ensemble Agent for majority of the year, indicating a statistically significant advantage.

Furthermore, an examination of both the mean returns and their corresponding confidence intervals demonstrates that the

Stacking Agent not only attained higher returns—indicating lower bias—but also exhibited reduced variance. Notably, the 95% confidence interval of the Stacking Agent is entirely contained within that of the Ensemble Agent. This finding confirms our hypothesis that applying stacking to Reinforcement Learning would reduce both bias and variance.

H. Further Considerations

1) Implementation Complexity and Performance: The dual-layer training process for base agents and the stacking agent significantly increases the complexity of hyperparameter tuning, training, and orchestration. Training the stacking agent is notably slow due to the reliance on base agents to generate predictions during each training step. This dependency results in training times that are approximately ten times longer than single-agent setups (e.g., one hour per episode on an Apple M3). To address performance constraints, we utilized parallelization via Ray, which allowed for evaluation across multiple random seeds. However, episodes still need to run serially, so capped number of episodes at just 10 for stacking agents, limiting their ability to achieve optimal performance. More extensive training might alter the comparative results.

2) Generalization: The evaluation of our stacking method is influenced by both the stochasticity inherent in training and the non-stationarity of financial markets. To account for stochasticity during training, we conducted multiple trials. However, the limited number of training episodes likely increased performance variability, reducing the robustness of our comparisons. Our evaluation relied on using one year of unseen data following a 10-year training period. To better assess the agent's ability to generalize across different market conditions, future studies could explore varying training and testing splits throughout the entire dataset. We would also adjust our measurement to normalize the return based on by theoretical achievable upperbound to make results comparable across different time periods.

3) Modeling Assumptions and Non-Stationarity: The state space relied heavily on short-term trends from the past five days of stock movements. More sophisticated technical indicators, fundamental business data, and a larger set of stocks—capturing inter-stock relationships—were excluded. Expanding the state space to incorporate these elements could significantly improve agents' performance. Financial markets are inherently complex, and the notion of training an RL agent over a decade to achieve consistent outperformance in the following year challenges assumptions about market efficiency. In theory, consistent arbitrage opportunities would quickly vanish due to competition. Future research could explore multi-agent frameworks to better simulate market dynamics (though this would likely suffer similar performance challenges).

III. Conclusion

This paper explored the application of stacking, a well-established technique in classical machine learning known for reducing bias and variance, within the reinforcement learning (RL) domain. Our experiments demonstrated initial evidence that stacking RL agents can improve both bias and variance in portfolio management tasks when compared to individual RL agents.

Studying stacking methodologies for portfolio management is an ambitious undertaking with numerous practical challenges. This work offers several directions for future exploration. For those interested in portfolio management, this project—built on the open-source FinRL framework [5]—serves as a useful reference and a foundation for developing novel trading strategies. For researchers focused on RL frameworks and methodologies, further investigation into stacking across a broader range of RL environments could provide valuable insights. In particular, testing stacking in environments that are explicitly modeled and stationary, such as those available in the RL Baselines3 Zoo framework, could help evaluate the robustness and versatility of this approach [15].

References

[1] Bao, Wenhang, and Xiao-yang Liu. "Multi-agent deep reinforcement learning for liquidation strategy analysis." arXiv preprint arXiv:1906.11046 (2019).

[2] Ndikum, Philip, and Serge Ndikum. "Advancing Investment Frontiers: Industry-grade Deep Reinforcement Learning for Portfolio Optimization." arXiv preprint arXiv:2403.07916 (2024).

[3] Lee, S. S., & Kim, J. Y. "On the Importance of Robustness and Statistical Significance in Deep Reinforcement Learning." *Proceedings of the 2023 International Conference on Learning Representations (ICLR)*, 2023.

[4] Yang, Hongyang and Liu, Xiao-Yang and Zhong, Shan and Walid, Anwar. "Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy." (September 11, 2020). Available at SSRN: https://ssrn.com/abstract=3690996 or http://dx.doi.org/10.2139/ssrn.3690996

[5] Liu, Xiao-Yang, et al. "FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance." arXiv preprint arXiv:2011.09607 (2020).

[6] Song, Yanjie, et al. "Ensemble reinforcement learning: A survey." Applied Soft Computing (2023): 110975.

[7] Wolpert, David H. "Stacked Generalization." Neural Networks, vol. 5, no. 2, 1992, pp. 241–259. ScienceDirect, https://doi.org/10.1016/S0893-6080(05)80023-1.

[8] McCandlish, Sam, et al. "An empirical model of large-batch training." arXiv preprint arXiv:1812.06162 (2018).

[9] Mnih, Volodymyr. "Asynchronous Methods for Deep Reinforcement Learning." arXiv preprint arXiv:1602.01783 (2016).

[10] Lillicrap, T. P. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[11] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

[12] Stephen Dankwa and Wenfeng Zheng. 2020. "Twin-Delayed DDPG: A Deep Reinforcement Learning Technique to Model a Continuous Movement of an Intelligent Robot Agent. In Proceedings of the 3rd International Conference on Vision, Image and Signal Processing (ICVISP 2019)." Association for Computing Machinery, New York, NY, USA, Article 66, 1–5. https://doi.org/10.1145/3387168.3387199

[13] Haarnoja, Tuomas, et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." International conference on machine learning. PMLR, 2018.

[14] De La Fuente, Neil, and Daniel A. Vidal Guerra. "A comparative study of deep reinforcement learning models: Dqn vs ppo vs a2c." arXiv preprint arXiv:2407.14151 (2024).

[15] Raffin, Antonin. RL Baselines3 Zoo. 2020. GitHub Repository, https://github.com/DLR-RM/rl-baselines3-zoo.