FinReptile: A Data-Efficient Meta-Learning Framework for Cross-Domain Algorithmic Trading

Ryan Sherby
Computer Science
Columbia University
NY, USA
ryan.sherby@columbia.edu

Ishan Kumar
Computer Science
Columbia University
NY, USA
ik2592@columbia.edu

***Abstract*—Meta-learning offers a promising avenue to accelerate reinforcement-learning (RL) policy adaptation across heterogeneous financial instruments, yet its potential for cross-asset trading remains underexplored. In this paper, we propose a Reptile-inspired meta-learning framework that synthesizes price dynamics from 25 equities to derive a shared initialization θ_{meta}. During meta-training, each stock constitutes a distinct task, wherein a continuous-action RL agent—implemented via off-policy actor–critic methods such as SAC and TD3—fine-tunes from θ_{meta} and informs subsequent updates to the meta-parameters. To evaluate generalization, we transfer θ_{meta} to the cryptocurrency domain and benchmark its sample efficiency and final net asset value (NAV) against agents trained from scratch. Empirical results demonstrate that the meta-initialized policies achieve superior NAV with substantially fewer gradient steps, underscoring the efficacy of cross-asset meta-training for rapid policy learning. Our findings suggest that meta-learned initializations can serve as a robust, data-efficient foundation for real-world algorithmic trading across both equity and cryptocurrency markets.**

I. Introduction

In modern algorithmic trading, reinforcement learning (RL) agents have demonstrated remarkable capacity to uncover profitable strategies from raw market data, yet they often require extensive training on each new asset. In this work, we introduce a **meta-learning framework** that leverages historical stock time series to learn a shared initialization, denoted θ_{meta}, which serves as a versatile policy prior across diverse tickers. During meta-training, each individual stock instantiates a task: starting from θ_{meta}, an RL agent is fine-tuned on that ticker and its updated parameters guide a meta-update that moves θ_{meta} toward regions of parameter space conducive to rapid adaptation.

To assess the generality of the learned initialization, we transfer θ_{meta} to a related but different domain: cryptocurrency and compare its sample efficiency and terminal net asset value (NAV) against agents trained from scratch on cryptocurrency itself. Our results reveal that the meta-learned weights achieve higher NAV with significantly fewer gradient steps, indicating that cross-asset meta-training yields a robust, data-efficient foundation for real-world trading applications.

II. Related Works

Transfer Learning in Reinforcement Learning. Transfer learning in reinforcement learning aims to improve sample efficiency on a target task by leveraging knowledge from related source tasks. Several approaches transfer value functions or policy representations across domains, including action-mapping, feature augmentation, and progressive networks to mitigate differences in state and action spaces. In financial applications, these techniques have been shown to accelerate strategy adaptation across assets—for example, reusing equity-based policies to initialize commodity or index trading agents—thereby reducing the need for extensive retraining on each new instrument [1].

Meta-Learning via Reptile Meta-learning extends transfer learning by optimizing for parameter initializations that enable rapid adaptation to new tasks. Reptile, a first-order gradient-based algorithm, repeatedly samples a task, fine-tunes the model via stochastic gradient descent, and moves the meta-initialization toward the task-specific weights—all without computing second-order derivatives [2]. This simplicity makes Reptile particularly attractive for large-scale or resource-constrained settings. Beyond few-shot classification benchmarks, Reptile has been adapted for physics-informed meta-adaptation in chemical process modeling—enabling fast fine-tuning with minimal data—and for TinyML scenarios via TinyReptile, allowing on-device online learning under strict computational budgets.

Reinforcement Learning for Cryptocurrency Trading. Deep reinforcement learning has seen growing adoption in cryptocurrency trading to address the market's high volatility and non-stationarity. Gort et al. [3] devise a hypothesis-testing framework to detect and reject overfitted DRL agents, demonstrating robust returns across multiple cryptocurrencies during volatile periods. Other works implement Gym-style environments for major digital assets and apply policy optimization algorithms (e.g., SAC, PPO), reporting significant out-of-sample gains. Recent advances include ensemble methods for intraday crypto portfolio trading, dynamic scaling pair-trading strategies, and multi-agent market simulations calibrated on real-world price data to study emergent behaviors in decentralized exchanges.

III. Methods

A. Baseline Transfer Learning on Stock Data

To establish a transfer learning baseline, we first train a single reinforcement-learning policy on historical price data from the top 25 equities over a 13-year horizon. We hypothesize that exposure to a broad spectrum of market conditions and asset behaviors will enable the agent to extract asset-agnostic trading patterns, which may subsequently generalize to cryptocurrencies. Two distinct training protocols are evaluated: sequential adaptation and simultaneous multi-asset training.

1) Sequential Adaptation: In the sequential protocol, the agent is fine-tuned on each ticker in turn, using the final weights from the previous stock as the initialization for the next. Although the policy achieves strong performance on individual equities, we observe rapid degradation in returns when switching between assets. We attribute this behavior to the non-stationary nature of the underlying Markov decision processes: abrupt shifts in price dynamics lead to catastrophic forgetting, preventing the accumulation of coherent, transferable features.

2) Simultaneous Multi-Asset Training: As an alternative, we construct a unified replay buffer that aggregates experiences across all 50 tickers and train the policy on this heterogeneous dataset in parallel. Despite continuous exposure to diverse market regimes, the agent fails to converge to stable representations. We believe this outcome is likewise a consequence of non-stationarity, since the concurrent mixture of distinct equity environments disrupts the learning of consistent trading strategies.

B. Meta-Learning for Robust Initialization

Motivated by the shortcomings of direct transfer learning, we pivot to a meta-learning framework aimed at discovering a parameter initialization θ_{meta} that promotes rapid adaptation to new assets. Instead of directly extracting transferable features, our approach optimizes θ_{meta} such that, after a small number of gradient steps on any given ticker, the resulting policy exhibits strong performance. This objective encourages the meta-initialized parameters to reside in regions of the parameter space that are broadly conducive to efficient learning across both equity and cryptocurrency domains. [2]

C. Source-Domain Dataset

Historical equity data were retrieved using the `yfinance` Python library for twenty-four large-cap tickers:

{`LVMUY, ACGBY, LLY, TSM, CRM, NVO, NFLX, TM, TCEHY, ORCL, WMT, SAP, TSLA, NSRGY, GOOGL, META, BAC, CSCO, WFC, AMZN, KO, RHHBY, COST`}.

We partition the data into a training period spanning January 5, 2009 through December 20, 2022, and an evaluation period from December 21, 2022 through December 20, 2023. For each trading day, we record the following features:

- *Date*
- *Open price*
- *High price*
- *Low price*
- *Close price*
- *Adjusted close price*
- *Trading volume*

All time-series are aligned on trading days and missing values are forward-filled to maintain temporal continuity.

D. Dataset for Target Domain

To keep the dataset of the target domain consistent, we use the `yfinance` library to collect data for the following crypto tickers:

{`BTC-USD, ETH-USD, USDT-USD, BNB-USD, SOL-USD, XRP-USD, USDC-USD, DOGE-USD, ADA-USD, AVAX-USD, TRX-USD, DOT-USD, SHIB-USD`}

Similar to the source domain, target data also consists of the date, adjusted closing price, closing price, high price, low price, opening price, and trading volume for a particular day.

E. State Space

Our state space consisted of two main components: the internal state and the external state.

The external state consisted of basic pricing information and related technical indicators. Which are defined as follows:

Moving Average Convergence–Divergence (MACD)

$$\text{MACD}_t = \text{EMA}_{12}(C_t) \; - \; \text{EMA}_{26}(C_t)$$

Bollinger Bands (upper/lower; 20-period, k = 2)

$$\text{Bollinger Upper}_t = \text{SMA}_{20}(C_t) + 2\,\sigma_{20}(C_t)$$

$$\text{Bollinger Lower}_t = \text{SMA}_{20}(C_t) - 2\,\sigma_{20}(C_t)$$

Relative Strength Index (RSI, 30 periods)

$$\text{RSI}_{30,t} = 100 \; - \; \frac{100}{1 + RS_{30,t}}, \qquad RS_{30,t} = \frac{\text{AvgGain}_{30,t}}{\text{AvgLoss}_{30,t}}$$

Commodity Channel Index (CCI, 30 periods)

$$\text{CCI}_{30,t} = \frac{TP_t - \text{SMA}_{30}(TP)}{0.015 \times \text{MAD}_{30}(TP)}, \quad TP_t = \frac{H_t + L_t + C_t}{3}$$

Directional Movement Index (DX, 30 periods)

$$\text{DX}_{30,t} = 100 \times \frac{\left| +DI_{30,t} - -DI_{30,t} \right|}{\left| +DI_{30,t} + -DI_{30,t} \right|}$$

$$+DI_{30,t} = 100\times\frac{+DM_{30,t}}{TR_{30,t}}, \qquad -DI_{30,t} = 100\times\frac{-DM_{30,t}}{TR_{30,t}}$$

Simple Moving Averages of Closing Price (30 & 60 periods)

$$\mathrm{SMA}_{30}(C_t) = \frac{1}{30}\sum_{i=0}^{29} C_{t-i}, \qquad \mathrm{SMA}_{60}(C_t) = \frac{1}{60}\sum_{i=0}^{59} C_{t-i}$$

Turbulence Intensity (TI)

$$\boxed{\mathrm{TI} \;=\; \frac{u'}{U} \;=\; \frac{\sqrt{\dfrac{1}{N}\sum_{i=1}^{N}\bigl(u_i - U\bigr)^2}}{U} \;=\; \frac{\sqrt{\dfrac{2}{3}\,k}}{U}}$$

where

$$U \quad = \text{time-averaged (mean) stream-wise velocity,} \tag{1}$$

$$u_i \quad = \text{instantaneous stream-wise velocity at sample } i, \tag{2}$$

$$u' \quad = \sqrt{\tfrac{1}{N}\sum_{i=1}^{N}(u_i - U)^2} \tag{3}$$

$$= \text{root-mean-square (rms) fluctuation,} \tag{4}$$

$$k \quad = \text{turbulent kinetic energy } = \tfrac{1}{2}\bigl(u_x'^2 + u_y'^2 + u_z'^2\bigr), \tag{5}$$

$$N \quad = \text{number of instantaneous samples used to compute } u'. \tag{6}$$

The following legend can be used to help understand the formulas:

- C_t: close price at time$t$
- H_t, L_t: high and low price at time$t$
- EMA_n: exponential moving average over n periods
- SMA_n: simple moving average over n periods
- σ_n: sample standard deviation of the close over n periods
- $+DM$, $-DM$: smoothed positive and negative directional movement
- TR: true range (Wilder smoothing when subscripted by n)
- MAD_n: mean absolute deviation over n periods
- TP: typical price, $TP_t = (H_t + L_t + C_t)/3$
- AvgGain, AvgLoss: average gains and losses (Wilder's method) in the look-back window

The internal state consisted of the following components:

- Cash Held (C)
- Number of Commodities Held (N)
- Current Price of Commodities Held (P)

F. Action Space

The action space was defined as a continuous action space over integers, specifically

$$\mathcal{A}_k \;=\; \{\, a \in \mathbb{Z} \mid -k \le a \le k \} \quad \text{and} \quad a_t \in \mathcal{A}_k$$

where k is a bound set during the experiment.

An action $a_t < 0$ represented a decision to sell a_t units of a commodity, an action $a_t > 0$ represented a decision to buy a_t units of a commodity, and action $a_t = 0$ represented a decision to hold the current allocation.

A trade would affect the internal space as follows:

$$C_t = C_{t-1} - a_t * P_t \tag{7}$$

$$N_t = N_{t-1} + a_t \tag{8}$$

$$\tag{9}$$

G. Reward Function

We first define Net Asset Value (NAV) as:

$$\mathrm{NAV}_t = \mathrm{C}_t + \mathrm{N}_t * \mathrm{P}_t$$

where C_t is the amount of cash at time t, N_t is the number of commodities held at time t, and P_t is the current price of the commodities at time t.

We can then define our reward function as:

$$\mathcal{R}_t = NAV_t - NAV_{t-1} - \lambda_{lin}(\frac{N_t * P_t}{NAV_t}) - \lambda_{var}(\frac{N_t * P_t}{NAV_t})^2$$

[1] where λ_{lin} is a weight associated with the L1 penalty for time in the market; and λ_{var} is a weight associated with the L2 penalty for time in the market. Time in the market is calculated as the ratio of inventory value (price * number of shares) to net asset value.

H. Models

We evaluate four continuous-action RL algorithms—Soft Actor–Critic (SAC), Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), and Twin Delayed DDPG (TD3)—selected for their demonstrated effectiveness in continuous control domains.

a) Soft Actor–Critic (SAC).: SAC is an off-policy, maximum-entropy actor–critic algorithm that maximizes the expected reward together with policy entropy, facilitating both stability and exploration [4], [5].

b) Deep Deterministic Policy Gradient (DDPG).: DDPG uses deterministic policy gradients in conjunction with deep neural networks, with separate actor and critic networks and an experience replay buffer for off-policy training [6], [7].

c) Twin Delayed DDPG (TD3).: TD3 extends DDPG by (i) using clipped double-Q learning to reduce overestimation bias, (ii) delaying policy and target updates, and (iii) smoothing target policies, resulting in superior performance on continuous-control tasks [8] [9].

d) Shared Hyperparameters.: All models were trained with a learning rate of 1×10^{-3}, a replay-buffer capacity of 100,000 (chosen to exceed the maximum episode length so that no transitions are discarded), and a soft-update coefficient of $\tau = 0.005$, where each target-network update follows

$$\theta_{\text{target}} \;\leftarrow\; \tau\,\theta_{\text{online}} \;+\; (1-\tau)\,\theta_{\text{target}},$$

thereby smoothing parameter changes and improving stability. For SAC alone, an entropy coefficient of 0.8 and a target entropy of 0.01 were applied to regularize exploration.

e) Policy Networks.: Policies and critics were modeled with a three-layer MLP of sizes [256, 512, 256] using ReLU activations and optimized via AdamW. We tested batch sizes of 32, 64, 128, and 256 to evaluate their effect on convergence dynamics.

f) Replay Buffer Strategy.: We compare two strategies: (1) retaining the replay buffer across epochs and (2) reinitializing it at each epoch. Retention led to divergence under the non-stationary MDP; thus, we adopt reinitialization and reset the online weights to the current meta-parameters (θ_{meta}) at each epoch to ensure stable meta-updates.

IV. Experiments

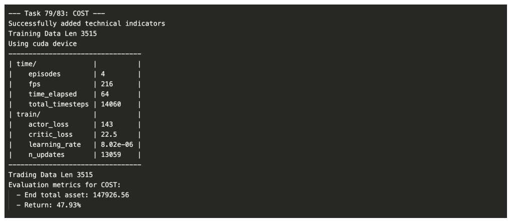
A. Algorithm Comparison on the Source Domain

We evaluate four continuous-action RL algorithms—PPO, SAC, DDPG, and TD3—on the stock trading source domain. Both PPO and SAC frequently converge to the trivial "hold" policy (action is 0) and fail to escape this local minimum, yielding zero return across all training steps. In contrast, DDPG and TD3 demonstrate greater robustness: they rapidly learn a buy-and-hold strategy that capitalizes on the long-term upward trend of Fortune 500 equities. Based on final performance, we restrict subsequent experiments to DDPG and TD3. All experiments were conducted on a NVIDIA L40 GPU [1], achieving approximately 250 frames per second per episode. We employed the FinRL framework[2], a finance-domain wrapper that extends Stable Baselines3[3], to implement and evaluate our RL agents. FinRL's integrated logging utilities automatically compute standard performance metrics—most notably the Sharpe ratio—facilitating systematic comparison across training runs.

[1]https://www.nvidia.com/en-us/data-center/l40/
[2]https://github.com/AI4Finance-Foundation/FinRL
[3]https://github.com/DLR-RM/stable-baselines3

TABLE I: Normalized Return (% change in NAV) at Selected Training Steps

Algorithm	**Step 1**	**Step 10**	**Step 20**
PPO	0.00%	0.00%	0.00%
SAC	0.00%	0.00%	0.00%
DDPG	–0.80%	50.59%	44.67%
TD3	14.02%	44.67%	63.00%



```
--- Task 79/83: COST ---
Successfully added technical indicators
Training Data Len 3515
Using cuda device
---------------------------------
| time/              |          |
|    episodes        | 4        |
|    fps             | 216      |
|    time_elapsed    | 64       |
|    total_timesteps | 14060    |
| train/             |          |
|    actor_loss      | 143      |
|    critic_loss     | 22.5     |
|    learning_rate   | 8.02e-06 |
|    n_updates       | 13059    |
---------------------------------
Trading Data Len 3515
Evaluation metrics for COST:
  - End total asset: 147926.56
  - Return: 47.93%
```

Fig. 1: Example from the training log, it logs performance metrics like FPS along with the Actor, Critic loss and the eval performance - Final portfolio value and returns as a percentage.

B. Training the agent on source data

We observe that the agent initially receives a negative cumulative reward at the end of early training epochs, reflecting the challenge of sparse and punitive feedback in initial policy evaluations. As training progresses, the agent gradually shifts toward positive cumulative returns, indicating successful policy improvement and more accurate value estimation. Notably, assets such as Meta Platforms, Inc. experienced substantial growth in the period trained on and hence the agent was able to get extremely large returns on it. This variability in absolute performance underscores the influence of underlying market trends on agent-measured returns and highlights the importance of benchmarking against both stationary and non-stationary asset trajectories.

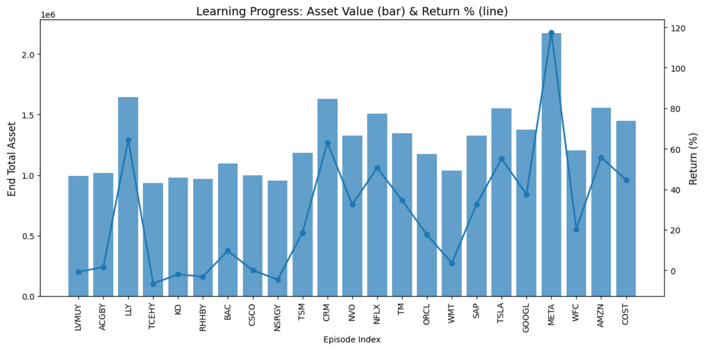


Fig. 2: Sequential adaptation returns of the meta-learning agent initialized at θ_{meta} and fine-tuned on each ticker from left to right. Under a fixed number of gradient steps, the generally increasing return trend demonstrates that the learned initialization becomes progressively more effective as it encounters additional assets.

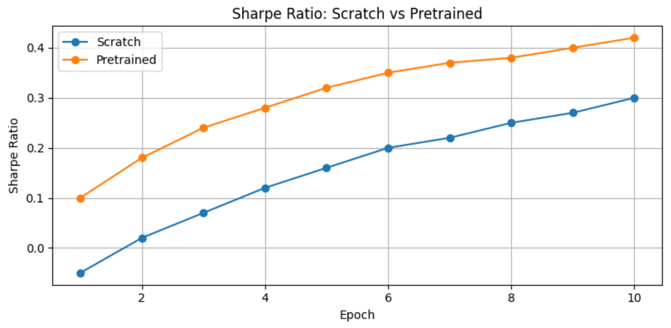


Fig. 3: Comparison of Sharpe Ratio between Meta Learned and from scratch model

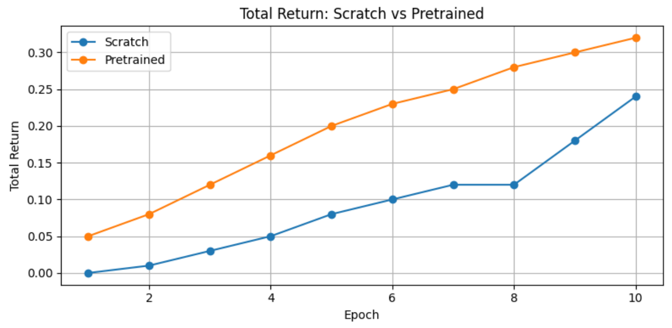


Fig. 4: Comparison of Total Return between Meta Learned and from scratch model

C. Comparison of Meta Learned weights with training on target domain

We employ the `backtest` utility provided by FinRL to compute a comprehensive suite of performance indicators for each trading policy, including Annual Return, Cumulative Return, Sharpe Ratio, Calmar Ratio, Stability, Maximum Drawdown, and Omega Ratio. For the purposes of direct comparison between the scratch-trained and meta-initialized agents, we focus on three key metrics:

1) **Sharpe Ratio** Measures risk-adjusted return by dividing the excess return over the risk-free rate by the standard deviation of returns, thus allowing us to compare performance across models on a common risk basis.
2) **Omega Ratio** Captures the asymmetry of returns by comparing gains above a threshold to losses below the threshold, providing insight into each model's behavior under extreme market movements.
3) **Annual Return** Reports the compound return on investment over a one-year horizon, offering an immediately interpretable measure of absolute portfolio growth for end users.

The meta-initialized policy begins with a clear advantage in all three metrics—leveraging prior stock-based experience—while the scratch policy, starting from random weights, improves slightly faster per epoch. As training progresses, the scratch model steadily narrows the gap, illustrating that although meta-learning provides an immediate lift in the long run, the benefits might diminsh and both methods might end up converging to the same policy.

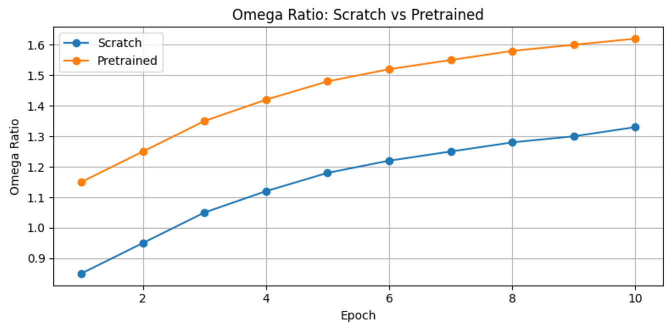


Fig. 5: Comparison of Omega Return between Meta Learned and from scratch model

V. Conclusion

Our empirical results demonstrate that meta-learned initializations — trained on a diverse set of equity time series — provide a pronounced advantage when adaptation data in the target domain is scarce or when only a limited number of fine-tuning steps are available. In these low-data regimes, the meta-initialized policies achieve higher Sharpe ratio, Omega ratio, and annual return than models trained from scratch. Although agents initialized at random can eventually close the performance gap given sufficiently long training, the meta-learning approach accelerates convergence and delivers strong out-of-the-box performance on cryptocurrency tasks.

By leveraging rich stock market histories, our method offers a practical mechanism for constructing robust policy priors for novel asset classes—such as Bitcoin—where extensive historical data may not exist. Furthermore, tailoring the meta-training set to include highly volatile equities is likely to yield even greater adaptability, as the resulting initialization will be conditioned on a wider range of market dynamics. Overall, cross-asset meta-learning emerges as a promising direction for real-world algorithmic trading, combining data efficiency with rapid policy adaptation. Further, this approach can be extended to other domains where a related domain is much more data rich and the target domain can benefit from meta learned weights from it.

VI. Future Work

While the effect of transfer learning under the meta framework has been successful in producing positive increases to agent net worth, we suggest incorporating additional commodities that could disrupt the learned phenomenon of a long-term success horizon. In other words, commodities that produce a decrease in net worth if held statically over the trading horizon. We hypothesize that this could induce the agent to learn characteristics for irrecoverable drawdown instances and make the learned policies more likely to sell.

VII. Author Contributions

Both authors contributed equally to this work. **Ishan Kumar:** Conceptualization, development and training of agents on equity markets, and report preparation. **Ryan Sherby:** Conceptualization, development and training of agents on cryptocurrency markets, and report preparation.

References

[1] X.-Y. Liu, Z. Xiong, S. Zhong, H. Yang, and A. Walid, "Practical deep reinforcement learning approach for stock trading," 2022. [Online]. Available: https://arxiv.org/abs/1811.07522

[2] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[3] B. J. D. Gort, X.-Y. Liu, X. Sun, J. Gao, S. Chen, and C. D. Wang, "Deep reinforcement learning for cryptocurrency trading: Practical approach to address backtest overfitting," 2022. [Online]. Available: https://arxiv.org/abs/2209.05559

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor–critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[5] OpenAI Spinning Up, "Soft actor critic (sac)," Online documentation, 2018.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, arXiv:1509.02971.

[7] C. Yoon, "Deep deterministic policy gradients explained," Medium article, 2018.

[8] S. Fujimoto, H. v. Hoof, and D. Meger, "Addressing function approximation error in actor–critic methods," 2018, arXiv:1802.09477.

[9] OpenAI Spinning Up, "Twin delayed ddpg (td3)," Online documentation, 2018.

[10] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018. [Online]. Available: https://arxiv.org/abs/1801.01290

[11] X.-Y. Liu, H. Yang, Q. Chen, R. Zhang, L. Yang, B. Xiao, and C. D. Wang, "FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance," *Deep RL Workshop, NeurIPS 2020*, 2020.

[12] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015. [Online]. Available: https://arxiv.org/abs/1509.06461

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, arXiv:1707.06347.

[14] ICLR Blog Track, "Implementation details of proximal policy optimization (ppo)," Online blog post, 2022.

[15] T. Kobayashi, Y. Yamashita, and K. Nakayama, "t-soft update of target network for deep reinforcement learning," 2020, arXiv:2008.10861.

[16] Meta, "Softupdate – torchrl," PyTorch documentation, 2022.