Data-Driven Deep Reinforcement Learning in Quantitative Finance

Xiao-Yang Liu[1], **Jingyang Rui**[2], **Jiechao Gao**[3], **Liuqing Yang**[1], **Hongyang Yang**[1], **Zhaoran Wang**[4], **Christina Dan Wang**[5], **Jian Guo**[6]
[1]Columbia University; [2]The University of Hong Kong; [3]University of Virginia;
[4]Northwestern University; [5]New York University (Shanghai); [6]IDEA Research.
XL2427@columbia.edu; zhaoranwang@gmail.com; guojian@idea.edu.cn

Abstract

Deep reinforcement learning (DRL) has shown huge potentials in quantitative finance recently. However, due to the high complexity of real-world markets, raw historical financial data often involve large noise and may not reflect the future of markets, degrading the performance of DRL agents in practice. By simulating the trading mechanism of real markets on processed datasets, market simulation environments play important roles in addressing this issue. However, the simulation accuracy heavily relies on the quality of processed datasets, while building and using datasets is often artisanal – painstaking and expensive. Moreover, training DRL agents on large datasets imposes a challenge on simulation speed. In this paper, we present a NeoFinRL framework that includes tens of Near real-market environments for data-driven Financial Reinforcement Learning. First, NeoFinRL separates financial data processing from the design pipeline of DRL-based strategy and provides open-source data engineering tools. Second, NeoFinRL provides tens of standard market environments for various trading tasks. Third, NeoFinRL enables massively parallel simulations by exploiting thousands of GPU cores.

1 Introduction

Compared to traditional supervised learning, deep reinforcement learning (DRL) has shown advantages in many complex decision-making problems [1, 2, 3]. Recent researches have also proved the feasibility of applying DRL in quantitative finance [4, 5, 6, 7]. However, due to the high cost of training a highly exploratory agent in real-world markets, researchers use historical financial datasets to train DRL agents [4, 5]. Nevertheless, due to the high complexity of real-world markets, raw historical financial data involve significant noise and may not reflect the future of markets. This issue usually degrades the performance of DRL agents in practice. It is challenging to apply DRL algorithms in real-time tradings.

Currently, researchers build their market simulation environments on processed datasets to address this issue. For example, FinRL library [5] provides a data processor connected to Yahoo! Finance and builds its stock trading environments based on historical price data. However, the data processor of FinRL is mainly designed for demonstration and not complete for different trading tasks. Users often need to clean data and extract features manually to achieve satisfactory performance. Moreover, there are no standard and unified evaluation protocols for different trading tasks. It is challenging and time-consuming for researchers to test and compare their trading strategies across different trading environments.

In this paper, we develop a NeoFinRL framework that includes tens of Near real-market environments for data-driven Financial Reinforcement Learning. First, we apply the DataOps paradigm [8] to the data engineering pipeline, providing agility to model deployment, as shown

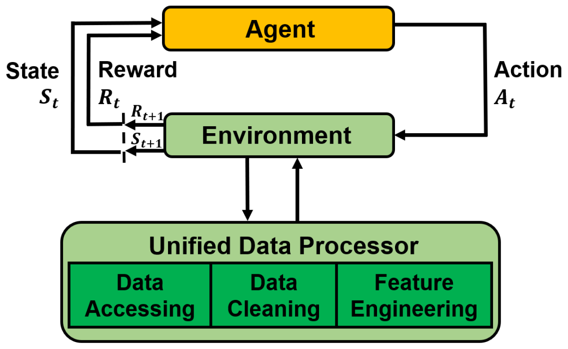


Figure 1: Overview of automated trading using deep reinforcement learning with DataOps paradigm.

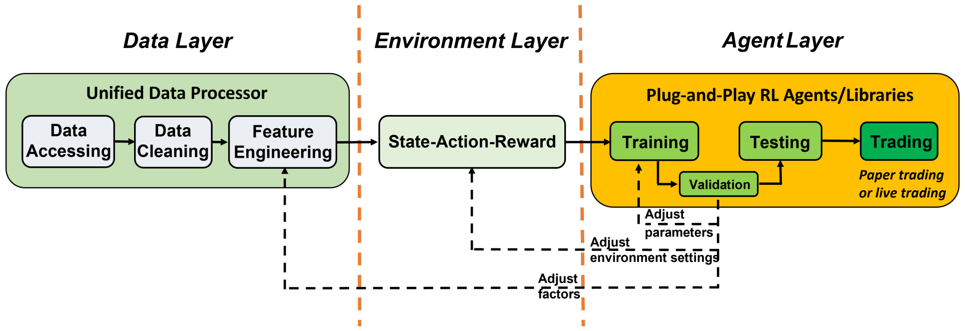


Figure 2: Overview of NeoFinRL.

in Fig. 1. We offer a unified and automated data processor enabling data accessing, data cleaning, and feature engineering. Second, we build tens of standard near real-market DRL environments for various trading tasks such as high-frequency trading, cryptocurrencies trading, portfolio allocation. The environments are directly connected to our data processors. High-quality large datasets can be generated efficiently and encapsulated into our environments. Third, to improve the performance of DRL agents in large datasets, we utilize thousands of GPU cores to perform massively parallel simulations in the training stage.

2 Proposed NeoFinRL Framework

MDP Model for Trading Tasks: We model a trading task as a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$ [9], where $\mathcal{S}$ and $\mathcal{A}$ denote the state space and action space, respectively, $P(s'|s, a)$ is the transition probability, $r(s, a)$ is a reward function, and $\gamma \in (0, 1)$ is a discount factor. Specifically, the state denotes the observations that a DRL agent receives from a market environment; the action space consists of actions that an agent is allowed to take at a state; the reward function $r(s, a, s')$ is the incentive mechanism for agents to learn a better policy. A trading agent aims to learn a policy $\pi(s_t|a_t)$ that maximizes the expected return defined as $R = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$.

Overview of NeoFinRL: We adopt a layered structure for DRL in finance, as shown in Fig. 2. NeoFinRL consists of three layers: data layer, environment layer, and agent layer. Each layer executes its functions and is relatively independent. Meanwhile, layers interact through end-to-end interfaces to implement the complete workflow of algorithm trading. Our codes are available online at `XXXX`.

DataOps for Data-Driven DRL in Finance: We follow the DataOps paradigm [8] in the data layer. First, we establish a standard pipeline for financial data engineering in RL, ensuring data of different

Platforms	Type	Supported Range and Frequency	Request Limits	Raw Data	Preprocessed Data	Account
Yahoo! Finance	US Securities	Depends on frequency, 1min	2,000/hour	OHLCV, df	Prices & Indicators, np.array	No
CCXT	Cryptocurrency	Depends on specific API, 1min	Depends on specific API	OHLCV, df	Prices & Indicators, np.array	No
WRDS.TAQ	US Securities	2003-now, 1ms	5 requests at the same time	Intraday Trades, df	Prices & Indicators, np.array	Yes
Alpaca	US Stocks, ETFs	2015-now, 1min	Depends on account	OHLCV, df	Prices & Indicators, np.array	Yes
JoinQuant	CN Securities	2005-now, 1min	3 requests at the same time	OHLCV, df	Prices & Indicators, np.array	Yes
QuantConnect	US Securities	1998-now, 1s	NA	OHLCV, df	Prices & Indicators, np.array	Yes

Table 1: Supported data platforms. OHLCV means open, high, low, close, volume data.

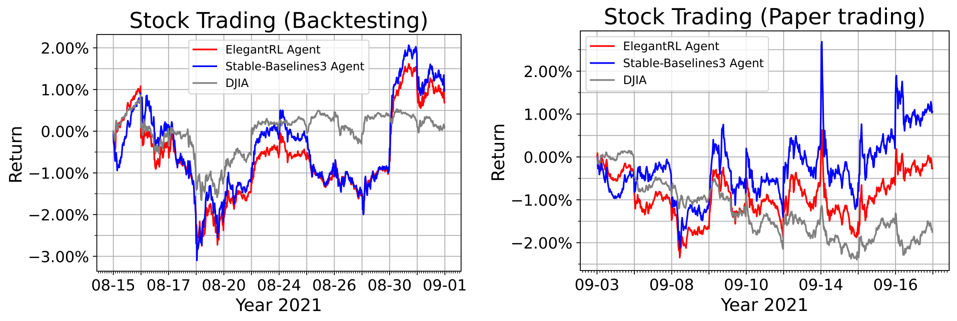


Figure 3: Cumulative returns (5-minute) of stock trading in backtesting and paper trading.

	ElegantRL [10]	**Stable-baselines3** [11]	**DJIA**
Cumul. return	0.968% / -0.652%	1.335% / 0.191%	0.099% / -1.56%
Annual return	22.425% / -16.746%	32.106% / 5.492%	2.108% / -35.522%
Annual volatility	15.951% / 14.113%	19.871% / 15.953%	9.196% / 9.989%
Sharpe ratio	1.457 / -1.399	1.621 / 0.447	0.289 / -4.894
Max drawdown	-2.657% / -1.871%	-2.932% / -1.404%	-1.438% / -2.220%

Table 2: Performance of backtesting (red) and paper trading (blue) for stock trading.

formats from different sources can be incorporated in a unified framework. Second, we automate this pipeline with a data processor, which can access data, clean data, and extract features from various data sources with high quality and efficiency. Our data layer provides agility to model deployment. The supported platforms and related information are shown in Table 1.

Massively Parallel Simulation: We utilize thousands of GPU cores to perform massively parallel simulation, which significantly accelerates the training process of DRL agents. The complete training process, including receiving states, taking actions, simulating markets, calculating rewards, and updating neural network models, is conducted on GPUs. In each CUDA core, a trading agent interacts with a market environment to produce state-action-reward sequences. Then all the sequences are transferred and stored in one replay buffer and used to update an aggregate learner and evaluator. By adopting this technique, we successfully achieve parallel simulation of hundreds of market environments to improve the performance of DRL trading agents in large datasets.

Plug-and-Play: In the development pipeline, we separate market environments from the data layer and the agent layer. Any DRL agent can be directly plugged into our environments, then trained and tested. Different agents/algorithms can be compared by running on the same benchmark environment to achieve fair evaluations.

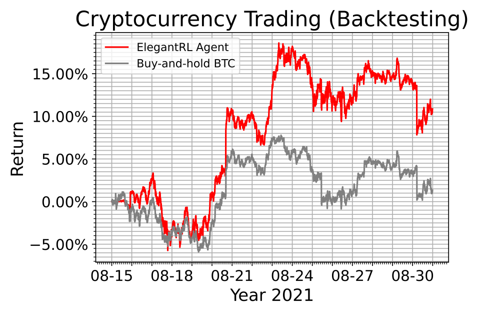
Training-Testing-Trading Pipeline: We employ a training-testing-trading pipeline. The DRL agent first learns from the training environment and is then validated in the validation environment for further adjustment. Then the validated agent is tested in historical datasets. Finally, the tested agent will be deployed in paper trading or live trading markets. First, this pipeline solves the information leakage problem because the trading data are never leaked when adjusting agents. Second, a unified pipeline allows fair comparisons among different algorithms and strategies.

3 Performance Evaluation

To provide benchmarks for researchers to test and compare with, we select several typical trading tasks and provide corresponding benchmark environments, including stock trading, portfolio allocation, cryptocurrency trading, and so on. We conduct experiments on stock trading and cryptocurrency trading.

3.1 Experiment Settings

Stock trading task: We select the 30 components in Dow Jones Industrial Average (DJIA) as our stock pool. We use the Proximal Policy Optimization (PPO) algorithm [12] of ElegantRL [10], Stable-baselines3 [11], and RLlib [13] to train agents and use DJIA as the baseline. We use 1-minute



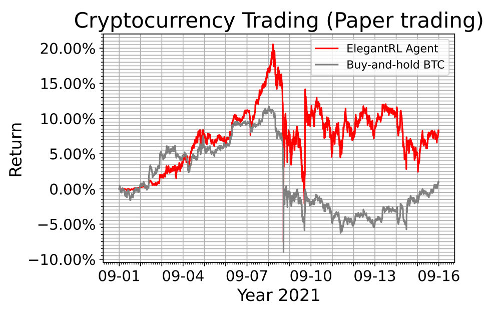


Figure 4: Cumulative returns (5-minute) of cryptocurrency trading in backtesting and paper trading.

	ElegantRL [10]	**BTC buy and hold**
Cumul. return	10.857% / 4.844%	1.332% / -1.255%
Annual return	360.823% / 121.380%	21.666% / 5.492%
Annual volatility	59.976% / 65.857%	47.410% / 57.611%
Sharpe ratio	2.992 / 1.608	0.657 / -0.113
Max drawdown	-6.396% / -10.474%	-7.079% / -14.849%

Table 3: Performance of backtesting (red) and paper trading (blue) for cryptocurrency trading.

data from 06/01/2021 to 08/15/2021 for training and data from 08/16/2021 to 08/31/2021 for validation (backtesting), and then we conduct paper trading from 09/03/2021 to 09/16/2021. The historical data and real-time data are accessed from the Alpaca database and paper trading API.

Cryptocurrency trading task: We select 10 representative cryptocurrencies as our pool. We use the PPO algorithm [12] of ElegantRL [10] to train an agent and use the Bitcoin (BTC) price as the baseline. We use 5-minute data from 06/01/2021 to 08/14/2021 for training and data from 08/15/2021 to 08/31/2021 for validation (backtesting). Then we retrain the agent using data from 06/01/2021 to 08/31/2021 and conduct paper trading from 09/01/2021 to 09/15/2021. The historical data and real-time data are accessed from Binance.

3.2 Trading Performance

Stock trading (backtesting): Both ElegantRL [10] agent and Stable-baselines3 [11] agent outperform DJIA in annual return and Sharpe ratio, as shown in Fig. 3 and Table 2. The ElegantRL agent achieves an annual return of 22.425% and a Sharpe ratio of 1.457. The Stable-baselines3 agent achieves an annual return of 32.106% and a Sharpe ratio of 1.621. To provide fair comparisons, we did not draw the curve of RLlib [13] since its performance is bad. Maybe we did not tune it well (hope to update it later).

Stock trading (paper trading): The performances of ElegantRL agent, Stable-baselines3 agent and the baseline DJIA are shown in Fig. 3 and Table 2. Both ElegantRL agent and Stable-baselines3 agent outperform the baseline. The paper trading results are consistent with the backtesting results.

Cryptocurrency trading (backtesting): The ElegantRL agent outperforms the benchmark (BTC price) in most performance metrics, as shown in Fig. 4 and Table 3. It achieves an annual return of 360.823% and a Sharpe ratio of 2.992.

Cryptocurrency trading (paper trading): The ElegantRL agent outperforms the benchmark (BTC price) in the paper trading stage, as shown in Fig. 4 and Table 3. The paper trading results are consistent with the backtesting results.

4 Conclusion

In this paper, we follow the DataOps paradigm to develop a NeoFinRL framework. NeoFinRL provides open data engineering tools, various market environments and enables massively parallel simulation. For future work, we will extend to incorporate more market environments and support more data sources and DRL libraries, making NeoFinRL more complete.

References

[1] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[2] Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311, 2013.

[3] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*, 2020.

[4] Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. Practical deep reinforcement learning approach for stock trading. *Workshop on Challenges and Opportunities for AI in Financial Services, NeurIPS*, 2018.

[5] Xiao-Yang Liu, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao, and Christina Dan Wang. FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance. *Deep RL Workshop, NeurIPS*, 2020.

[6] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deep reinforcement learning for trading. *The Journal of Financial Data Science*, 2(2):25–40, 2020.

[7] Tidor-Vlad Pricope. Deep reinforcement learning in quantitative algorithmic trading: A review. *arXiv preprint arXiv:2106.00123*, 2021.

[8] Crystal Valentine and William Merchan. Dataops: An agile methodology for data-driven organizations. `https://www.oracle.com/a/ocom/docs/oracle-ds-data-ops-map-r.pdf`, 2018.

[9] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[10] Xiao-Yang Liu, Zechu Li, Zhaoran Wang, and Jiahao Zheng. ElegantRL: A lightweight and stable deep reinforcement learning library. `https://github.com/AI4Finance-Foundation/ElegantRL`, 2021.

[11] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 07 2017.

[13] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 3053–3062. PMLR, 10–15 Jul 2018.