FinRL-Podracer: High Performance and Scalable Deep Reinforcement Learning for Quantitative Finance

Zechu Li
zl2993@columbia.edu
Columbia University

Xiao-Yang Liu*
xl2427@columbia.edu
Columbia University

Jiahao Zheng
jh.zheng@siat.ac.cn
Shenzhen Inst. of Advanced Tech.

Zhaoran Wang
zhaoranwang@gmail.com
Northwestern University

Anwar Walid†
anwar.i.walid@gmail.com
Amazon & Columbia University

Jian Guo‡
guojian@idea.edu.cn
IDEA Research

ABSTRACT

Machine learning techniques are playing more and more important roles in finance market investment. However, finance quantitative modeling with traditional supervised learning frameworks has a number of limitations, including the difficulty in defining appropriate labels, lack of consistency in modeling and trading execution, and lack of modeling the dynamic nature of the finance market. The development of reinforcement learning techniques is partially addressing these issues. Unfortunately, the steep learning curve and the difficulty in quick modeling and agile development are impeding finance researchers from using reinforcement learning in quantitative trading. In this paper, we propose an *RLOps in finance* paradigm and present a *FinRL-Podracer* framework to accelerate the development pipeline of deep reinforcement learning (DRL)-driven trading strategy and to improve both trading performance and training efficiency. FinRL-Podracer is a cloud-native microservices-based solution that features high performance and high scalability and promises *continuous training*, *continuous integration*, and *continuous delivery* of DRL-driven trading strategies, facilitating a rapid transformation from algorithmic innovations into a profitable trading strategy. First, we propose a generational evolution mechanism (namely, a cloud-native *orchestration* mechanism) to improve the trading performance of an DRL agent, and schedule the training of a DRL algorithm onto a GPU cloud via multi-level mapping. Then, we carry out the training of DRL components with high-performance optimizations on GPUs. Finally, we evaluate the FinRL-Podracer framework for a stock trend prediction task on an NVIDIA DGX SuperPOD cloud. FinRL-Podracer outperforms three popular DRL libraries *Ray RLlib*, *Stable Baseline 3* and *FinRL*, i.e., 12% $\sim$ 35% improvements in annual return, 0.1 $\sim$ 0.6 improvements in Sharpe ratio and 3× $\sim$ 7× speed-up in training time. We show the high scalability by training a trading agent in 10 minutes on an NVIDIA DGX SuperPOD cloud with 80 A100 GPUs, for a stock trend prediction task on NASDAQ-100 constituent stocks with minute-level data over 10 years.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Neural networks**; **Markov decision processes**; **Reinforcement learning**; **Explainability**; **Value iteration**.

KEYWORDS

RLOps in finance, deep reinforcement learning, stock trend prediction, scalability, GPU cloud

ACM Reference Format:
Zechu Li, Xiao-Yang Liu, Jiahao Zheng, Zhaoran Wang, Anwar Walid, and Jian Guo. 2021. FinRL-Podracer: High Performance and Scalable Deep Reinforcement Learning for Quantitative Finance. In *ACM International Conference on AI in Finance (ICAIF '21), October 15–16, 2020, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3490354.3494415

1 INTRODUCTION

Algorithmic trading is increasingly deployed in the financial investment process. A conventional supervised learning pipeline consists of five stages [30, 39], as shown in Fig. 1 (left), namely data pre-process, modeling and trading signal generation, portfolio optimization, trade execution, and post-trade analysis. Recently, deep reinforcement learning (DRL) [33, 35, 36] has been recognized as a promising alternative for quantitative finance [2, 6, 15, 16], since it has the potential to overcome some important limitations of supervised learning, such as the difficulty in label specification and the gap between modeling, positioning and order execution. We advocate extending the principle of *MLOps* [1] [1] to the *RLOps in finance* paradigm that implements and automates the continuous training (CT), continuous integration (CI), and continuous delivery (CD) for trading strategies. We argue that such a paradigm has vast profits potential from a broadened horizon and fast speed, which is critical for wider DRL adoption in real-world trading tasks.

The *RLOps in finance* paradigm, as shown in Fig. 1 (right), integrates middle stages (i.e., modeling and trading signal generation, portfolio optimization, and trade execution) into a DRL agent. Such a paradigm aims to help quantitative traders develop an end-to-end trading strategy with a high degree of automation, which removes

*Equal contribution.
†A. Walid finished this project at Bell labs, before joining Amazon.
‡Corresponding author.

New York '21, Nov. 3–5, 2021, New York, NY

ACM ISBN 978-1-4503-7584-9/20/10…$15.00
https://doi.org/10.1145/3490354.3494415

[1] MLOps is an ML engineering culture and practice that aims at unifying ML system development (Dev) and ML system operation (Ops).

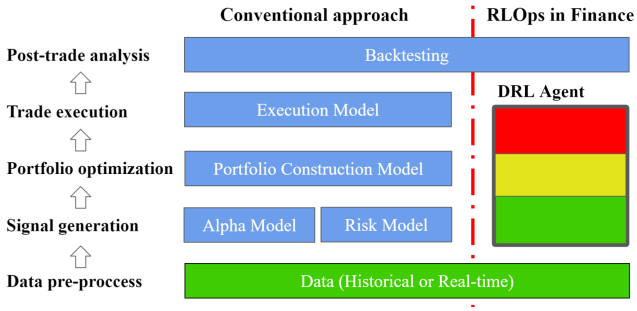

Figure 1: Software stack for an algorithmic trading process: conventional approach vs. RLOps in finance.

the latency between stages and results in a compact software stack. The major benefit is that it can explore the vast potential profits behind the large-scale financial data, exceeding the capacity of human traders; thus, the trading horizon is lifted up to a new level. Also, it allows traders to continuously update trading strategies, which equips traders with an edge in a highly volatile market. However, the large-scale financial data and fast iteration of trading strategy bring imperative challenges in terms of computing power.

Existing works are not satisfactory with respect to the usage of large-scale financial data and the efficiency of agent training. For DRL strategy design, existing works studied algorithmic trading on a daily time-frame [2, 24, 26, 41–44] or hourly time-frame [13], which is hard to fully explore the dynamics of a highly volatile market. For DRL library/package development, existing works may not be able to meet the intensive computing requirement of relatively high frequency trading tasks, large-scale financial data processing and tick-level trade execution. We evaluate the training time of three popular DRL libraries FinRL [24, 26], RLlib [19] and Stable Baseline3 [8] on NASDAQ-100 constituent stocks with minute-level data. Table 1 shows that it is difficult for them to effectively train a profitable trading agent in a short cycle time.

In recent years, scalable DRL frameworks and device-accelerated simulations have been recognized as the critical software development for the RLOps paradigm [11, 19, 20]. It is promising to utilize extensive computing resources, e.g., a GPU cloud, to accelerate the development pipeline of trading strategies. Therefore, we investigate DRL solutions on a GPU cloud, e.g., an NVIDIA DGX SuperPOD cloud [37] that is the most powerful AI infrastructure for enterprise deployments.

In this paper, we propose a *FinRL-Podracer* framework as a high-performance and scalable solution for *RLOps in finance*. At a high level, FinRL-Podracer schedules the training process through a multi-level mapping and employs a generational evolution mechanism. Such a design guarantees scalability on a cloud platform. At a low level, FinRL-Podracer realizes hardware-oriented optimizations, including parallelism encapsulation, GPU acceleration, and storage optimization, thus achieving high-performance. As a result, FinRL-Podracer can effectively exploit the super computing resources of a GPU cloud for relatively high frequency trading, which provides an opportunity to automatically design DRL trading strategies with fast and flexible development, deployment and production.

Our contributions can be summarized as follows

	Sharpe ratio	Max dropdown	Training time
RLlib [19]	1.67	**-23.248%**	**110 min**
SB3 [12]	**1.82**	-23.750%	150 min
FinRL [43]	1.35	-27.267%	345 min
QQQ	1.25	-28.559%	–

Table 1: Evaluations of existing DRL libraries on an NVIDIA DGX A100 server [4]. We evaluate on NASDAQ-100 constituent stocks with minute-level data by training from 01/01/2009 to 05/12/2019 and backtesting from 05/13/2019 to 05/26/2021. Invesco QQQ ETF is a market benchmark.

- We propose a *FinRL-Podracer* framework built on two previous projects, FinRL [24, 26] and ElegantRL [23] [2], to initiate a paradigm shift from conventional supervised approaches to *RLOps in finance*.
- FinRL-Podracer employs a generational evolution mechanism during the training of DRL agents and provides high-performance optimizations for financial tasks.
- We show the high scalability by training a trading agent in 10 minutes on an NVIDIA DGX SuperPOD cloud [37] with 80 A100 GPUs, for a stock trend prediction task on NASDAQ-100 constituent stocks with minute-level data over 10 years. We evaluate the trained agent for one year and show its trading performance outperforms three DRL libraries *FinRL* [24, 26], *Ray RLlib* [19] and *Stable Baseline3* [12], i.e., 12% $\sim$ 35% improvements in annual return, 0.1 $\sim$ 0.6 improvements in Sharpe ratio and 3× $\sim$ 7× speed-up in training time.

The remainder of this paper is organized as follows. Section 2 describes related works. Section 3 models a typical stock trend prediction task as a Markov Decision Process. In Section 4, we present the FinRL-Podracer framework and describe its evolution and training layers, respectively. In Section 5, we describe the experimental setup for the trading task and present experimental results. We conclude this paper in Section 6.

2 RELATED WORKS

This section summarizes related works from two aspects: DRL application in quantitative finance and the MLOps development.

2.1 DRL in Finance

With the the successes of DRL in playing games, e.g., Atari games [29] and GO games [34], more and more finance researchers show their interests in this area, and they have done some early attempts to applying DRL in quantitative finance investment. In this paper, we take the stock trend prediction (STP) problem as an example to introduce existing works and show great potentials of DRL in finance area.

Stock trend prediction task is often considered a challenging application of machine learning in finance due to its noisy and volatile nature. Traditionally, the STP problem is formulated as a supervised learning problem, where features of stocks are extracted from a past time window of technical indices, fundamental data and alternative data, and labels are usually extracted from a future

[2]ElegantRL is a scalable and elastic deep reinforcement learning library. It supports general robotic and game playing applications.

time window of concerned criteria such as rise/fall, returns, excess returns or Sharpe ratios. Recently, deep reinforcement learning has been applied to solving STP problems. Zhang *et al.* [44] constructed a trading agent using three DRL algorithms, DQN, PG, and A2C, for both discrete and continuous action spaces. Yang *et al.* [43] used an ensemble strategy to integrate different DRL algorithms, A2C, DDPG, and PPO based on the Sharpe ratio. They applied the idea of a rolling window, and the best algorithm is picked to trade in the following period. Recently, many researchers provide more DRL solutions for STP problems [3, 7, 18]. However, most existing works are based on several assumptions, which limits the practicality. For example, the backtesting has no impacts on the market; there are almost no other complex actions besides buying, holding, and selling; only one stock type is supported for each agent.

2.2 Principle of MLOps

Recently, Google trends put Machine Learning Operations (MLOps) as one of the most promisingly increasing trends [40]. MLOps is a practice in developing and operating large-scale machine learning systems, which facilitates the transformation of machine learning models from development to production [5, 27]. In essence, MLOps entails cloud computing power to integrate and automate a standard machine learning pipeline: (1) data transportation; (2) data transformation; (3) continuous ML training; (4) continuous ML deployment; (5) output production, thus building applications that enables developers with limited machine-learning expertise to train high-quality models specific to their domain or data [22, 38].

However, the training data of DRL is not prepared in advance compared with conventional supervised learning but collected through an agent-environment interaction inside the training process. Such a significant difference requires a re-integration of the automated pipeline and a re-scheduling of the cloud computing resources with respect to the conventional MLOps principle. Therefore, we advocate extending the principle of MLOps to the RLOps in the finance paradigm to seek an opportunity for the wider DRL adoption in production-class financial services.

3 STOCK TREND PREDICTION TASK

We describe the problem formulation of a typical financial task, stock trend prediction, which locates at the task layer of Fig. 2. Our setup follows a similar setting in [26][43].

A stock trend prediction task is modeled as a Markov Decision Process (MDP): given state $s_t \in \mathcal{S}$ at time t, an agent takes an action $a_t \in \mathcal{A}$ according to policy $\pi_\theta(s_t)$, transitions to the next state s_{t+1} and receives an immediate reward $r(s_t, a_t, s_{t+1}) \in \mathbb{R}$. The policy $\pi_\theta(s)$ with parameter θ is a function that maps a state to an action vector over n stocks. The objective is to find an optimal policy π_θ^* (parameterized by θ) that maximizes the expected return (the fitness score in Fig. 2) over T times slots

$$\pi_\theta^* = \underset{\theta}{\operatorname{argmax}}\, J(\pi_\theta), \text{ where } J(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t, s_{t+1})\right], \quad (1)$$

where $\gamma \in (0, 1]$ is a discount factor.

Then, for the stock trend predictiont task with n stocks, we specify the state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $r(s_t, a_t, s_{t+1})$, and the state transition, as in [26][43].

State space $\mathcal{S}$ describes an agent's perception of a market environment. We summarize various features that are used by human trader and use them to construct the state space:

- Balance $b_t \in \mathbb{R}_+$: the money left in the account at time t.
- Shares $\mathbf{h}_t \in \mathbb{Z}_+^n$: the number of shares for n stocks at t.
- Closing price $\mathbf{p}_t \in \mathbb{R}_+^n$: the closing prices of n stocks at t.
- Technical indicators help the agent make decisions. Users can select existing indicators or add new indicators. E.g., Moving Average Convergence Divergence (MACD) $\mathbf{M}_t \in \mathbb{R}^n$, Relative Strength Index (RSI) $\mathbf{R}_t \in \mathbb{R}_+^n$, Commodity Channel Index (CCI) $\mathbf{C}_t \in \mathbb{R}_+^n$, etc.

Action space $\mathcal{A}$ describes the allowed actions an agent can take at states s_t, $t = 1, ..., T$. For one stock, action is $a \in \{-k, ..., -1, 0, 1, ..., k\}$, where $k \in \mathbb{Z}$ or $-k \in \mathbb{Z}$ denotes the number of shares to buy or sell, respectively, and $a = 0$ means to hold. Users can set a maximum number of shares $h_{\max}$ for a transaction, i.e., $k \le h_{\max}$, or set a maximum ratio of capital to allocate on each stock.

Reward r_t for taking action a_t at state s_t and arriving at state s_{t+1}. Reward is the incentive for an agent to improve its policy for the sake of getting higher rewards. A relatively simple reward can be defined as the change of the account value, i.e.,

$$r_t = (b_{t+1} + \mathbf{p}_{t+1}^T \mathbf{h}_{t+1}) - (b_t + \mathbf{p}_t^T \mathbf{h}_t) - c_t, \quad (2)$$

where the first and second terms are the account values at s_{t+1} and s_t, and c_t denotes the transaction cost (market friction).

Transition (s_t, a_t, r_t, s_{t+1}). Taking action a_t at state s_t, the environment steps forward and arrives at state s_{t+1}. A transition involves the change of balance, number of shares, and the stock prices due to the market changes. We split the stocks into three sets: selling set S, buying set B and holding set H, respectively. The new balance is

$$b_{t+1} = b_t + (\mathbf{p}_t^S)^T \mathbf{k}_t^S - (\mathbf{p}_t^B)^T \mathbf{k}_t^B, \quad (3)$$

where $\mathbf{p}^S \in \mathbb{R}^n$ and $\mathbf{k}^S \in \mathbb{R}^n$ are the vectors of prices and number of selling shares for the selling stocks, and $\mathbf{p}^B$ and $\mathbf{k}^B$ for the buying stocks. The number of shares becomes

$$\mathbf{h}_{t+1} = \mathbf{h}_t - \mathbf{k}_t^S + \mathbf{k}_t^B \ge \mathbf{0}. \quad (4)$$

The super computing power is necessary to achieve the massively parallel simulations for an STP task. In the STP task, a DRL agent keeps observing and trading on the historical market data to sample trajectories (one trajectory is a series of transitions). However, the historical data has to be significantly large in order to provide a broad horizon. For example, the variety of the historical data is related with the **data volume** and **data type**:

- The data volume varies with respect to:
  - **the length of data period**: spans from one year up to more than ten years.
  - **the time granularity**: from daily-level to minute-level, second-level or even microsecond-level.
  - **the number of stocks**: from thirty (Dow 30) to hundreds (NASDAQ 100 or S&P 500), or even covers the whole market.
- The data type varies with respect to:
  - **the raw market data** includes data points of open-high-low-close-volume (OHLCV) for each stock, which provides a direct understanding of each stock performance.

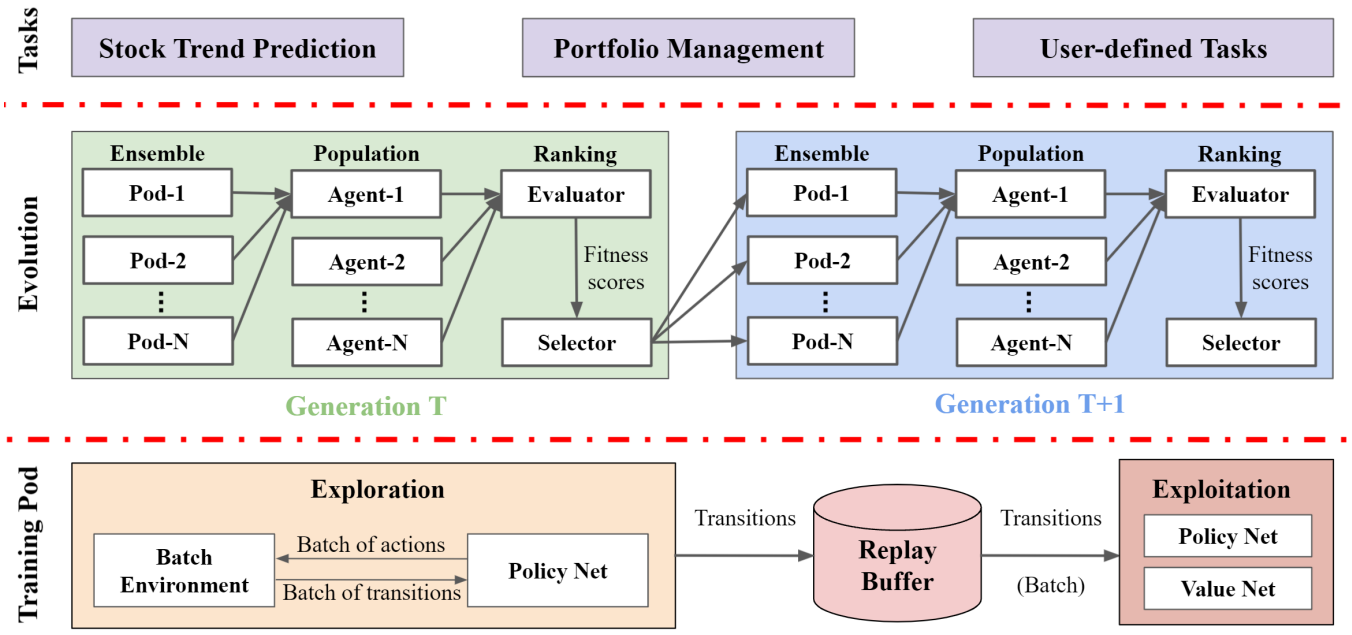

Figure 2: Overview of *FinRL-Podracer* that has three layers: trading task layer, evolution layer and training layer.

- **the alternative data** usually refers to the large-scale collection of both structured and unstructured data, e.g., market news, academic graph data, credit card transactions and GPS traffic. The agent could employ different encoders to analyze the insights of investment techniques provided by the alternative data.
- **the indexes** could be directly given as a kind of powerful technical indicator, which helps the agent make decisions.

In practice, the environment simulation, alternative data processing and index analyzing are computational expensive, therefore, a cloud-level solution is critical to a fast iteration of a trading strategy.

4 FINRL-PODRACER FRAMEWORK

We propose a *FinRL-Podracer* framework to utilize the super computing power of a GPU cloud for training DRL agents. We first present an overview of FinRL-Podracer and then describe its layered architecture.

4.1 Overview

Based on the experiments in Table 1, we found that existing DRL libraries/packages [12, 19, 24, 26] have three major issues that restrict the trading performance and training efficiency:

- There is no criteria to determine **overfitting or underfitting of models (agents)** during the training process. It is critical to overcome underfitting with more computing power and avoid overfitting that wastes computing power, while both cases would lead to suboptimal models.
- The training process of an agent is **sensitive to hyperparameters**, which may result in unstable trading performance in backtesting. However, it is tedious for human traders to search for a good combination of hyperparameters, and thus an automatic hyperparamter search is favored.
- **Computing power is critical** to effectively explore and exploit large-scale financial data. Sufficient exploration guarantees a good trading performance, and then smart exploitation results in good training efficiency. A strong computing power helps achieve a balance between exploration and exploitation.

Therefore, we provide a high-performance and scalable solution on a GPU cloud, *FinRL-Podracer*, to develop a profitable DRL-driven trading strategy within a small time window. To fully utilize a GPU cloud, say an NVIDIA DGX SuperPod cloud [37], we organize FinRL-Podracer into a three-layer architecture in Fig. 2, a trading task layer on the top, an evolution layer in the middle and a training layer at the bottom.

In the evolution layer, we employ a generational evolution mechanism and address the issues of overfitting and hyper-parameter sensitivity through a synergy of an *evaluator* and a *selector*. The evaluator computes the fitness scores $J(\pi_\theta)$ in (1) of a population of N agents and mitigates the performance collapse caused by overfitting. The hyper-parameter search is automatically performed via *agent evolution*, where the selector uses the fitness scores to guide the search direction. An effective cloud-level evolution requires a high-quality and scalable scheduling, therefore we schedule a population of parallel agents through a multi-level mapping and an efficient communication protocol.

In the training layer, we realize high-performance GPU-oriented optimizations of a decomposable DRL training pipeline with a lower-level parallelism. We locally optimize each component, namely environment, worker, replay buffer, and learner, through GPU acceleration, efficient parameter communication, and storage optimization.

Thus, we maximize the hardware usage and minimize the communication overhead, which allows each isolated component to be executed on a GPU cloud.

Such an evolution-and-training workflow on a GPU cloud pipelines the development of a strategy. It enjoys great scalability and high performance, which promotes fast and flexible development, deployment and production of profitable DRL trading strategies.

4.2 Scalable Evolution Layer

FinRL-Podracer exploits agent-parallelism to fully utilize the cloud computing resources. We propose a *generational evolution* mechanism to coordinate the parallel agents and to automatically search the best hyper-parameters. At present, we utilize an *evaluator* and a *selector* to schedule the agent evolution, where more modules can be incorporated, e.g., a monitor, an allocator, etc.

The evaluator evaluates each agent in the course of the training and keeps track of the best agent, which can effectively mitigate the overfitting issue. From our observation, it is difficult for users to use existing libraries [19][12][43] to train a profitable trading strategy because the overfitting agent may be treated as the best agent when the training process moves forward. When the dataset scales up, we need to increase the training time/steps to fully explore the large-scale data, making it harder to set an appropriate stop criteria, and the result agent may hardly be the best one. The evaluator effectively solves the problem: it evaluates the agent at each iteration, outputs a fitness score and records the best agent so far; when the fitness score in (1) drops, the evaluator would stop the training process and output the best agent as the final agent.

The selector acts as a central controller to perform a generational evolution in a genetic algorithm (GA) [28]. GA is an optimization algorithm inspired by natural evolution: at every *generation*, a population of *offsprings* is perturbed, and the evaluator calculates their fitness scores in (1) based on an objective function; then the selector recombines the offspring with the highest scores to form a new population for the next generation. Since the offspring is replicable, the concept of natural selection scales up well on a GPU cloud. As depicted in the evolution layer of Fig. 2, multiple agents with different hyper-parameter combinations are grouped into a population where each agent is an offspring. The synergy of the evaluator and selector enables FinRL-Podracer to naturally select the best agent for the future generation and eliminates the potential negative impact from poorly evolved agents, which effectively improves the stability and efficiency of the training. The experiment results in Section 5 will investigate the improvement brought by the generational evolution mechanism.

4.3 Packaging Worker-Learner into Pod

FinRL-Podracer achieves effective and efficient allocation of cloud resources through a multi-level mapping, which follows the principle of *decomposition*-and-*encapsulation*. We employ a worker-learner decomposition [9–11] that splits a DRL training process into four isolated components with encapsulated parallelism:

- **Environment**: simulates a financial market.
- **Rollout worker**: interacts with an environment.
- **Replay buffer**: stores transitions from a worker and feeds a batch of transitions to a learner.

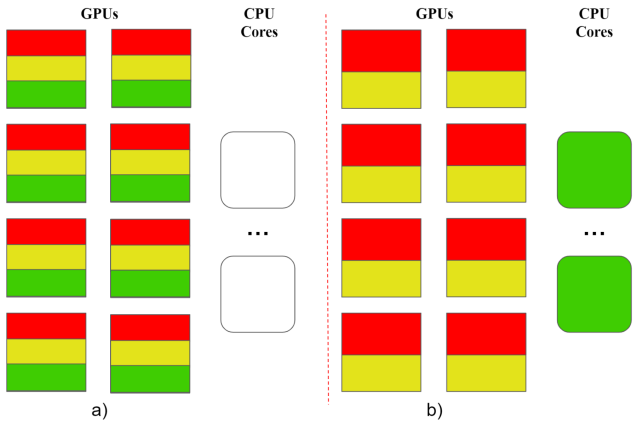

Figure 3: Two implementations of a training process: a) the environment simulation (green), action inference (yellow), and model update (red) are all located on GPUs. b) the environment simulation is executed on CPUs, and action inference and model update are on GPUs. An NVIDIA DGX-A100 server [37][4] contains 8 A100 GPUs.

- **Learner**: trains the neural networks.

Each agent contains the four types of components, which are packaged into a suite that is mapped into a GPU pod. We use multiprocessing to run those components as separate processes, and each process is mapped to a GPU container. Such a two-level mapping is natural since a GPU pod consists of multiple containers, while correspondingly a DRL training process of an agent consists of different components.

This pod-container structure enables scalable allocation of GPU computing resources. We take advantage of a GPU cloud software stack and use the Kubernetes (K8S) software to scalably coordinate pods among severs. Consider a cloud with 10 servers (i.e., 80 A100 GPUs), we encapsulate an agent (a package of components) into a pod, replicate it 10 times, and send them to a K8S server. Then, K8S distributes these 10 pod replicas to computing nodes that execute the training. The pod replication reflects the agent-parallelism, and it is highly scalable since a GPU cloud can support a large number of pods.

We propose an efficient protocol to guide the agent-to-agent communication to better support the agent evolution on a GPU cloud. Such a protocol aims to minimize the communication overhead as the number of agents scales up. The communication protocol in the evolution layer of Fig. 2 utilizes a parameter server (PS) architecture [17] and a mesh architecture as two steps. Take a population of 16 agents (due to the one-to-one mapping, 16 pods in total) as an example; in the first step, we redistribute 16 pods as 4 groups and select 1 pod from each group as the main pod that acts as a server to communicate with the remaining pods. In the second step, the 4 main pods communicate in pairs for 2 rounds. This communication protocol is scalable and efficient for a large number of agents: 1) such a protocol can flexibly adapt to an arbitrary number of agents,

with a change of configuration setting; 2) the protocol is decentralized, which effectively avoids the network bottleneck when the number of agents increases.

4.4 High Performance Training Layer

The optimization of each component is critical to the overall performance. We describe the hardware-oriented optimizations of each component, including parallelism encapsulation, GPU acceleration, and storage optimization.

Environment with GPU-acceleration. The environment simulation is both computing- and communication-intensive. We propose a batch mode to achieve massively parallel simulations, which maximizes the hardware utilization (either CPUs or GPUs). We instantiate multiple independent sub-environments in a batched environment, and a batched environment is exposed to a rollout worker that takes a batch of actions and returns a batch of transitions.

Fig. 3 a) presents a GPU-accelerated environment. Environments of financial tasks are highly suitable to GPUs because financial simulations involve "simple" arithmetics, where a GPU with thousands of cores has the natural advantages of matrix computations and parallelism. Then, financial environments written in CUDA can speed up the simulation. The GPU-accelerated environment also effectively reduces the communication overhead by bypassing CPUs, as supported by a GPU cloud [37]. The output transitions would be represented as a tensor already stored in GPU memory, which can be directly fetched by rollout workers, avoiding the data transfer between CPU and GPU back and forth.

Fig. 3 b) presents an environment on CPUs. There are some financial simulations with frequent CPU usage (addressing trading constraints), making it inefficient to compute on GPUs. In our experiments, some environments run much slower on GPUs than CPUs. Thus, we simulate those environments on CPUs.

Replay buffer on GPU. We allocate the replay buffer on the contiguous memory of GPUs, which increases the addressing speed and bypasses CPUs for faster data transfer. As the worker and learner are co-located on GPUs, we store all transitions as tensors on the contiguous memory of GPUs. Since the collected transitions are packed together, the addressing speed increases dramatically well when a learner randomly samples a batch of transitions to update network parameters.

Learner with optimizations. Inside a training process of an agent, we concurrently run multiple learners in parallel. All parallel learners are initialized with different random seeds and synchronized at the end of each training epoch. Such a design supports a vanilla ensemble strategy and stabilizes the learning process of the agent.

In addition, we present a novel and effective way for learners to communicate, i.e., sending the network parameters rather than the gradients. Most existing libraries [12, 19, 24, 26] send the gradients of learners for each sampled batch, following a traditional synchronization approach on supervised learning. Such an approach is inefficient for DRL algorithms since the learner will update the neural network hundreds of times within each training epoch, namely it needs to send gradients hundreds of times. By taking advantage of the soft update [21], we send the model parameters rather than the gradients. The parameter of the models is amenable to communication because model size in DRL is not comparable to that in other deep learning fields. Here, communication happens once at the end of each epoch, which is a significantly lower frequency of communication.

5 PERFORMANCE EVALUATION

We describe the GPU cloud platform, the performance metrics and compared methods, and then evaluate the performance of FinRL-Podracer for a stock trend prediction task.

5.1 GPU Cloud Platform

All experiments were executed on NVIDIA DGX-2 servers [4] in an NVIDIA DGX SuperPOD platform [37], a cloud-native supercomputer. We use 256 CPU cores of Dual AMD Rome 7742 running at 2.25GHz for each experiment. An NVIDIA DGX-2 server has 8 A100 GPUs and 320 GB GPU memory [4].

5.2 Performance Metrics

We evaluate the trading performance and training efficiency of the FinRL-Podracer for a stock trend prediction task.

Data pre-processing. We select the NASDAQ-100 constituent stocks as our stock pool, accessed at 05/13/2019 (the starting time of our testing period), and use the datasets with two time granularities: minute-level and daily. The daily dataset is directly downloaded from Yahoo!Finance, while the minute-level dataset is first downloaded as raw data from the Compustat database through the Wharton Research Data Services (WRDS) [32] and then pre-processed to an open-high-low-close-volume (OHLCV) format. We split the datasets into training period and backtesting period: the daily data from 01/01/2009 to 05/12/2019 for training; the minute-level data from 01/01/2016 to 05/12/2019 for training; For both datasets, we backtest on the same period from 05/13/2019 to 05/26/2021.

Evaluation metrics. Six common metrics are used to evaluate the experimental results:

- **Cumulative return**: the ratio between the final capital and the initial capital.
- **Annual return**: the geometric average amount of money earned by the agent each year.
- **Annual volatility**: the annual standard deviation of the return.
- **Sharpe ratio**: subtracting the annualized risk-free rate from the annualized return, and then dividing by the annualized volatility.
- **Max drawdown**: the maximum percentage loss during the trading period.
- **Training time**: the training time with respect to the cumulative return.

Compared methods. For trading performance evaluation, we compare FinRL-Podracer and vanilla FinRL-Podracer (without agent evolution) with FinRL [24, 26], RLlib [19], Stable Baseline3 [8], and NASDAQ Composite/Invesco QQQ ETF. We use Proximal Policy Optimization (PPO) [31] as the DRL algorithm in the reported results and fine-tune each library to maximize its performance. Each library is allowed to use up to 80 GPUs.

For training efficiency evaluation, the experiments are conducted on multiple GPUs. We compare with RLlib [19] since it has high performance on distributed infrastructure. However, both FinRL

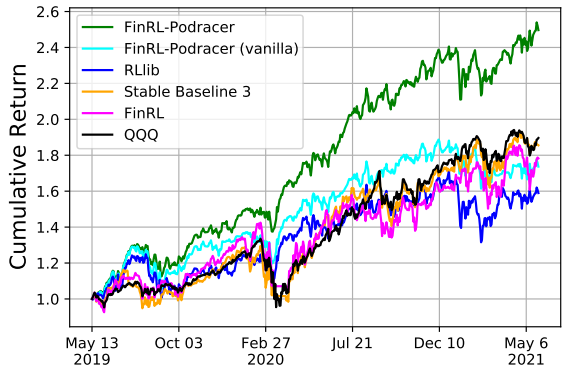

Figure 4: Cumulative returns on daily dataset during 05/13/2019 to 05/26/2021. Initial capital $1,000,000, transaction cost percentage 0.2%, and Invesco QQQ ETF is a market benchmark.

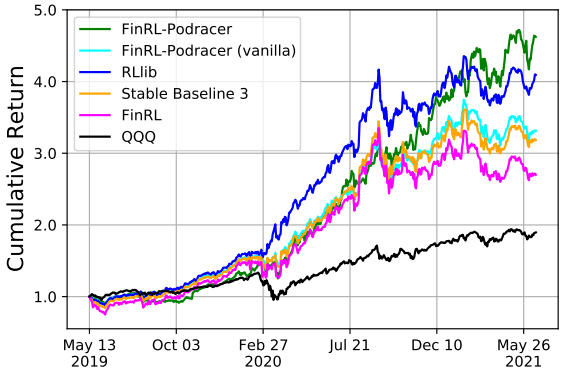

Figure 5: Cumulative returns on minute dataset during 05/13/2019 to 05/26/2021. Initial capital $1,000,000, transaction cost percentage 0.2%, Invesco QQQ ETF is a market benchmark.

	Cumul. return	Annual return	Annual volatility	Max dropdown	Sharpe ratio
FinRL-Podracer (**Ours**)	**149.553%/362.408%**	**56.431%/111.549%**	**22.331%**/33.427%	**-13.834%/-15.874%**	**2.12/2.42**
FinRL-Podracer (vanilla)	73.546%/231.747%	30.964%/79.821%	23.561%/31.024%	-18.428%/-21.002%	1.27/2.05
RLlib [19]	58.926%/309.54%	25.444%/99.347%	30.009%/31.893%	-23.248%/-22.292%	0.91/2.33
Stable Baseline3 [12]	85.539%/218.531%	35.316%/76.28%	31.592%/34.595%	-24.056%/-23.75%	1.12/1.82
FinRL [24]	78.255%/169.975%	32.691%/62.576%	37.641%/42.908%	-26.774%/-27.267%	0.94/1.35
Invesco QQQ ETF	89.614%	36.763%	28.256%	-28.559%	1.25

Table 2: Performance of stock trading on NASDAQ-100 constituent stocks with daily (red) and minute-level (blue) data.

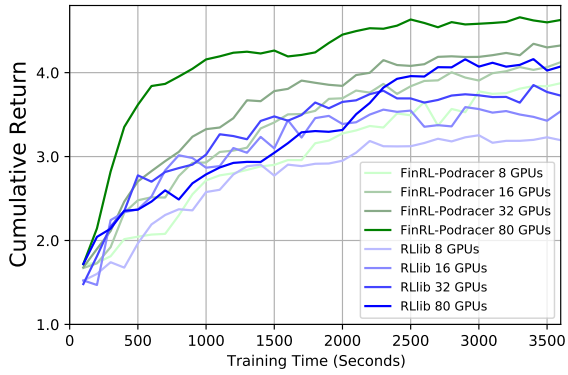

Figure 6: Generalization performance on backtesting dataset, using the model snapshots of FinRL-Podracer and RLlib [19] at different training time (wall clock time).

[24] and Stable Baseline 3 [8] do not support the training on multiple GPUs, thus we do not compare with them. To guarantee fairness, we keep all adjustable parameters and computing resources the same, such as the width of neural networks, total training steps, number of workers, and GPU and CPU resources.

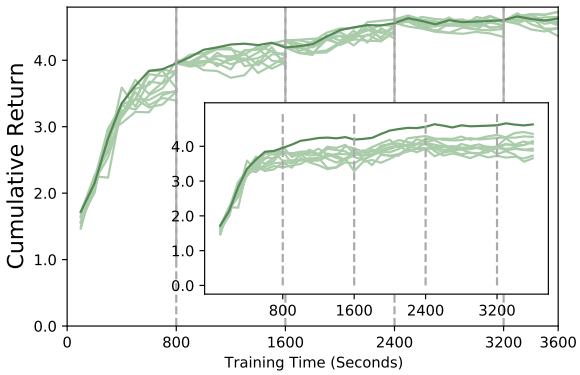

Figure 7: Generalization performance of a model along the agent evolution in the training process in FinRL-Podracer.

5.3 Trading Performance Analysis

We backtest the trading performance from 05/13/2019 to 05/26/2021 on both daily and minute-level dataset. From Fig. 4 and Fig. 5, all DRL agents are able to achieve a better or equal performance than the market in cumulative return, which demonstrates the profit potentials of DRL-driven trading strategies. Comparing Fig. 4 with Fig. 5, we observe that all methods have a much better performance on the minute-level dataset than that on the daily dataset. The trading

Hyper-parameters	Value
Total #GPUs	80
#GPUs per Agent	8
Optimizer	Adam
Learning rate	2^{-14}
Discount factor	$\gamma = 0.99$
Total steps	2^{20}
Batch size	2^{10}
Repeat times	2^{3}
Replay buffer Size	2^{12}
Ratio clip (PPO)	0.25
Lambda entropy (PPO)	0.02
Evaluation interval (second)	64

Table 3: Hyperparameter settings.

performance of most agents is almost the same as that of the market on daily dataset, however, all agents significantly outperform the market if they have a larger dataset to explore. With a higher granularity data, the Sharpe ratios are also lifted up to a new level. From Table. 2, agents achieve Sharpe ratios of 2.42, 2.05, 2.33, 1.82, 1.35 on the minute-level dataset, which are 0.3, 0.78, 1.42, 0.7, and 0.41 higher than those on the daily dataset. Therefore, we conclude that the capability to process large-scale financial data is critical for the development of a profitable DRL-driven trading strategy since the agent can better capture the volatility and dynamics of the market.

From Table 2, Fig. 4, and Fig. 5, we also observe that our FinRL-Podracer outperforms other baselines on both datasets, in terms of expected return, stability, and Sharpe ratio. As can be seen from Table 2, our FinRL-Podracer achieves the highest cumulative returns of 149.533% and 362.408%, annual returns of 56.431% and 111.549%, and Sharpe ratios of 2.12 and 2.42, which are much higher than the others. Furthermore, FinRL-Podracer also shows an outstanding stability during the backtesting: it achieves Max dropdown -13.834% and -15.874%, which is significantly lower than other methods. Consider the vanilla FinRL-Podracer as a direct comparison, we find that vanilla FinRL-Podracer has a similar trading performance with other baseline frameworks, and this is consistent with expectation as they all use the same DRL algorithm for all experiments. Such a performance gap between FinRL-Podracer and vanilla FinRL-Podracer demonstrates the significant impact of the generational evolution mechanism on the trading performance of a DRL-driven trading strategy. This proves the effectiveness of the hyper-parameter search and evaluation mechanism on the issues of hyper-parameter sensitivity and overfitting in financial problems.

5.4 Training Efficiency Analysis

We compare the training efficiency of FinRL-Podracer with RLlib [19] on a varying number of A100 GPUs, i.e., 8, 16, 32, and 80 A100 GPUs. We store the model snapshots at different training time, say every 100 seconds, then later we use each snapshot model to perform inference on the backtesting dataset and obtain the generalization performance, namely, the cumulative return.

In Fig. 6, as the number of GPUs increases, both FinRL-Podracer and RLlib achieve a higher cumulative return with the same training time (wall-clock time). FinRL-Podracer with 80 GPUs has a much steeper generalization curve than others, e.g., it can achieve a cumulative return of 4.0 at 800s, which means it learns in a much faster speed. However, FinRL-Podracer with 32 GPUs and 16 GPUs need 2, 200s and 3, 200s to achieve the same cumulative return, respectively. The generalization curves of RLlib with different numbers of GPUs are relatively similar, and we do not observe much speed-up. For example, RLlib needs approximately 2, 200s to achieve a cumulative return of 3.5, however, FinRL-Podracer needs 300s to achieve the same cumulative return, which is $3\times \sim 7\times$ faster than RLlib.

It is counter-intuitive that the increase of GPU resources not only makes FinRL-Podracer have a fast training, but also improves the trading performance over RLlib [19]. We know from Fig. 4 and Fig. 5 that the orchestrator mechanism promotes the trading performance of FinRL-Podracer, therefore, we empirically investigate the agent evolution process. Fig. 7 explicitly demonstrates an evolution of ten agents, where the selector chooses the best model to train in the next generation every 800s. The inner figure of Fig. 7 depicts the generalization curves of the ten agents in the first generation (without using the agent evolution mechanism). The curve with an orchestrator (the thick green curve) is much higher than the other ten curves.

6 DISCUSSION AND CONCLUSION

In this paper, we have proposed a high-performance and scalable deep reinforcement learning framework, *FinRL-Podracer*, to initiate a paradigm shift from conventional supervised learning approaches to *RLOps in finance*. FinRL-Podracer provides a highly automated development pipeline of DRL-driven trading strategies on a GPU cloud, which aims to help finance researchers and quantitative traders overcome the steep learning curve and take advantage of supercomputing power from the cloud platforms.

FinRL-Podracer achieved promising performance on a cloud platform, mainly by following the two principles, *the virtues of nested hierarchies* and *getting smart from dumb things* [14]. For low-level training, FinRL-Podracer realizes nested hierarchies by empolying hardware-oriented optimizations, including parallelism encapsulation, GPU acceleration, and storage optimizations. As a high level scheduling, FinRL-Podracer obtains a smart agent from hundreds of weak agents, which is the essence of ensemble methods, by employing a generational evolution mechanism. We further investigate the evolution and training layers in a followup work [25] for a cloud-native solution. We believe that ensemble multiple weak agents is preferable to aiming to train one strong agent. Thus we propose a new orchestration mechanism, a tournament-based ensemble training method [25] with asynchronous parallelism, which involves relatively low communication overhead. Also, we observe the great potential of massively parallel simulation, which lifts the exploration capability up into a potentially new dimension.

FinRL-Podracer is our first step from building a standard DRL pipeline of financial tasks to using DRL agents to understand the dynamics of the markets. We believe that the development of FinRL-Podracer is critical for the ecosystem of the FinRL community [24][26] because it offers opportunities for many future directions. First, FinRL-Podracer provides a way to take advantage of large-scale financial data. Therefore, it is possible to allow DRL agents to work in second or microsecond level and cover all stocks in the

future, which is meaningful for the exploration and understanding of the volatile and dynamic market. Moreover, training on the cloud makes DRL agents adapt to much more complex environment simulations and neural networks, thus can achieve wider DRL applications to various financial tasks, e.g., portfolio allocation, fraud detection, DRL-driven insights for yield improvement and optimization. Furthermore, the low-level optimizations in FinRL-Podracer could be also useful for the future development of financial simulators, such as using the GPU-accelerated techniques to reduce latency.

ACKNOWLEDGEMENT

This research used computational resources of the GPU cloud platform [37] provided by the IDEA Research institute.

REFERENCES

[1] Sridhar Alla and Suman Kalyan Adari. 2021. What Is MLOps? In *Beginning MLOps with MLFlow*. Springer, 79–124.
[2] H. Buehler, L. Gonon, J. Teichmann, and B. Wood. 2019. Deep hedging. *Quantitative Finance* 19 (2019), 1271 – 1291.
[3] L. Chen and Qiang Gao. 2019. Application of deep reinforcement learning on automated stock trading. *IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)* (2019), 29–33.
[4] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro* 41, 2 (2021), 29–35.
[5] Google Cloud. 2020. MLOps: Continuous delivery and automation pipelines in machine learning. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning#mlops_level_0_manual_process. *Google Cloud*, Jan. 07, 2020.
[6] Jacomo Corbo, Oliver Flemin, and Nicolas Hohn. 2021. It's time for businesses to chart a course for reinforcement learning. https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/its-time-for-businesses-to-chart-a-course-for-reinforcement-learning. *McKinsey Analytics*, Apirl. 01, 2021.
[7] Quang-Vinh Dang. 2019. Reinforcement learning in stock trading. *ICCSAMA* (2019).
[8] DLR-RM. 2021. Stable-baseline 3. https://github.com/DLR-RM/stable-baselines3.
[9] Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. 2020. SEED RL: scalable and efficient deep-RL with accelerated central inference. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
[10] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. 2018. IMPALA: scalable distributed deep-RL with importance weighted actor-learner architectures. In *Proceedings of the International Conference on Machine Learning (ICML)*.
[11] Matteo Hessel, Manuel Kroiss, Aidan Clark, Iurii Kemaev, John Quan, Thomas Keck, Fabio Viola, and Hado van Hasselt. 2021. Podracer architectures for scalable reinforcement learning. *ArXiv* abs/2104.06272 (2021).
[12] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2018. Stable baselines. https://github.com/hill-a/stable-baselines.
[13] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. 2017. A deep reinforcement learning framework for the financial portfolio management problem. *ArXiv* abs/1706.10059 (2017).
[14] Kevin Kelly. 1994. *Out of control: The rise of neo-biological civilization*. Addison-Wesley Longman Publishing Co., Inc.
[15] Marko Kolanovic and Rajesh T. Krishnamachari. 2017. Big data and AI strategies: machine learning and alternative data approach to investing. https://www.cognitivefinance.ai/single-post/big-data-and-ai-strategies. *J.P. Morgan Securities LLC*, May. 18, 2017.
[16] Petter N. Kolm and G. Ritter. 2019. Modern perspectives on reinforcement learning in finance. *Econometrics: Mathematical Methods & Programming eJournal* (2019).
[17] Mu Li, D. Andersen, J. Park, Alex Smola, Amr Ahmed, V. Josifovski, J. Long, E. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *BigDataScience '14*.
[18] Xinyi Li, Yinchuan Li, Yuancheng Zhan, and Xiao-Yang Liu. 2019. Optimistic bull or pessimistic bear: Adaptive deep reinforcement learning for stock portfolio allocation. *ICML Workshop on Applications and Infrastructure for Multi-Agent Learning* (2019).
[19] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, Ken Goldberg, and Ion Stoica. 2017. Ray RLLib: a composable and scalable reinforcement learning library. *ArXiv* abs/1712.09381 (2017).
[20] Jacky Liang, Viktor Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox. 2018. GPU-accelerated robotic simulation for distributed reinforcement learning. In *Conf. on Robot Learning (CoRL)*.
[21] T. Lillicrap, Jonathan J. Hunt, A. Pritzel, N. Heess, T. Erez, Yuval Tassa, D. Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971 (2016).
[22] Paul Lipton, Derek Palma, Matt Rutkowski, and Damian Andrew Tamburri. 2018. TOSCA solves big problems in the cloud and beyond! *IEEE Cloud Computing* (2018), 1–1. https://doi.org/10.1109/MCC.2018.111121612
[23] Xiao-Yang Liu, Zechu Li, Zhaoran Wang, and Jiahao Zheng. 2021. ElegantRL: A Scalable and Elastic Deep Reinforcement Learning Library. https://github.com/AI4Finance-Foundation/ElegantRL.
[24] Xiao-Yang Liu, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao, and Christina Dan Wang. 2020. FinRL: a deep reinforcement learning library for automated stock trading in quantitative finance. *Deep Reinforcement Learning Workshop at NeurIPS* (2020).
[25] Xiao-Yang Liu, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao, and Christina Dan Wang. 2021. ElegantRL-Podracer: Scalable and Elastic Library for Cloud-Native Deep Reinforcement Learning. *Deep Reinforcement Learning Workshop at NeurIPS* (2021).
[26] Xiao-Yang Liu, Hongyang Yang, Jiechao Gao, and Christina Dan Wang. 2021. FinRL: Deep reinforcement learning framework to automate trading in quantitative finance. *ACM International Conference on AI in Finance (ICAIF)* (2021).
[27] Rick Merritt. 2020. What Is MLOps? https://blogs.nvidia.com/blog/2020/09/03/what-is-mlops/. *NVIDIA*, Sep. 03, 2020.
[28] Melanie Mitchell. 1996. An introduction to genetic algorithms.
[29] V Mnih, K Kavukcuoglu, D Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540) (2015), 529–533.
[30] G. Nuti, Mahnoosh Mirghaemi, P. Treleaven, and Chaiyakorn Yingsaeree. 2011. Algorithmic trading. *Computer* 44 (2011), 61–69.
[31] John Schulman, F. Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *ArXiv* abs/1707.06347 (2017).
[32] Wharton Research Data Service. 2015. Standard & poor's compustat. Data retrieved from Wharton Research Data Service.
[33] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.
[34] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587) (2016), 484–489.
[35] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.
[36] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
[37] NVIDIA DGX A100 system reference architecture. 2020. *NVIDIA DGX SuperPOD: Scalable infrastructure for AI leadership*. NVIDIA Corporation.
[38] Damian A. Tamburri. 2020. Sustainable MLOps: Trends and challenges. In *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. 17–23. https://doi.org/10.1109/SYNASC51798.2020.00015
[39] P. Treleaven, M. Galas, and V. Lalchand. 2013. Algorithmic trading review. *Commun. ACM* 56 (2013), 76–85.
[40] Google Trends. [n.d.]. What Is MLOps? https://www.google.com/trends.
[41] Jia WU, Chen WANG, Lidong XIONG, and Hongyong SUN. 2019. Quantitative trading on stock market based on deep reinforcement learning. In *2019 International Joint Conference on Neural Networks (IJCNN)*. 1–8.
[42] Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. 2018. Practical deep reinforcement learning approach for stock trading. *NeurIPS Workshop* (2018).
[43] Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. 2020. Deep reinforcement learning for automated stock trading: An ensemble strategy. *ACM International Conference on AI in Finance (ICAIF)* (2020).
[44] Zihao Zhang, Stefan Zohren, and Stephen Roberts. 2020. Deep reinforcement learning for trading. *The Journal of Financial Data Science* 2(2) (2020), 25–40.