FinRL: Deep Reinforcement Learning Framework to Automate Trading in Quantitative Finance

Xiao-Yang Liu, Hongyang Yang
{xl2427,hy2500}@columbia.edu
Columbia University

Jiechao Gao
jg5ycn@virginia.edu
University of Virginia

Christina Dan Wang*
christina.wang@nyu.edu
New York University Shanghai

ABSTRACT

Deep reinforcement learning (DRL) has been envisioned to have a competitive edge in quantitative finance. However, there is a steep development curve for quantitative traders to obtain an agent that automatically positions to win in the market, namely *to decide where to trade, at what price* and *what quantity*, due to the error-prone programming and arduous debugging. In this paper, we present the first open-source framework *FinRL* as a full pipeline to help quantitative traders overcome the steep learning curve. FinRL is featured with simplicity, applicability and extensibility under the key principles, *full-stack framework, customization, reproducibility* and *hands-on tutoring*.

Embodied as a three-layer architecture with modular structures, FinRL implements fine-tuned state-of-the-art DRL algorithms and common reward functions, while alleviating the debugging workloads. Thus, we help users pipeline the strategy design at a high turnover rate. At multiple levels of time granularity, FinRL simulates various markets as training environments using historical data and live trading APIs. Being highly extensible, FinRL reserves a set of user-import interfaces and incorporates trading constraints such as market friction, market liquidity and investor's risk-aversion. Moreover, serving as practitioners' stepping stones, typical trading tasks are provided as step-by-step tutorials, e.g., stock trading, portfolio allocation, cryptocurrency trading, etc.

KEYWORDS

Deep reinforcement learning, automated trading, quantitative finance, Markov Decision Process, portfolio allocation.

1 INTRODUCTION

Deep reinforcement learning (DRL), that balances exploration (of uncharted territory) and exploitation (of current knowledge), is a promising approach to automate trading in quantitative finance [50][51][47][54][21][13]. DRL algorithms are powerful in solving dynamic decision making problems by learning through interactions with an unknown environment, and offer two major advantages of *portfolio scalability* and *market model independence* [6]. In quantitative finance, algorithmic trading is essentially making dynamic decisions, namely *to decide where to trade, at what price and what quantity*, in a highly stochastic and complex financial market. Incorporating many financial factors, as shown in Fig. 1, a DRL trading agent builds a multi-factor model to trade automatically, which is difficult for human traders to accomplish [4, 53]. Therefore, DRL has been envisioned to have a competitive edge in quantitative finance.

Many existing works have applied DRL in quantitative financial tasks. Both researchers and industry practitioners are actively designing trading strategies fueled by DRL, since deep neural networks are significantly powerful at estimating the expected return of taking a certain action at a state. Moody and Saffell [33] utilized a policy search for stock trading; Deng *et al.* [9] showed that DRL can obtain more profits than conventional methods. More applications include stock trading [35, 47, 51, 54], futures contracts [54], alternative data (news sentiments) [22, 35], high frequency trading [15], liquidation strategy analysis [3], and hedging [6]. DRL is also being actively explored in the cryptocurrency market, e.g., automated trading, portfolio allocation, and market making.

However, designing a DRL trading strategy is not easy. The programming is error-prone with tedious debugging. The development pipeline includes preprocessing market data, building a training environment, managing trading states, and backtesting trading performance. These steps are standard for implementation but yet time consuming especially for beginners. Therefore, there is an urgent demand for an open-source library to help researchers and quantitative traders to overcome the steep learning curve.

In this paper, we present a *FinRL* framework that automatically streamlines the development of trading strategies, so as to help researchers and quantitative traders to iterate their strategies at a high turnover rate. Users specify the configurations, such as picking data APIs and DRL algorithms, and analyze the performance of trading results. To achieve this, FinRL introduces a *three-layer* framework. At the bottom is an environment layer that simulates financial markets using actual historical data, such as closing price, shares, trading volume, and technical indicators. In the middle is the agent layer that implements fine-tuned DRL algorithms and common reward functions. The agent interacts with the environment through properly defined reward functions on the state space and action space. The top layer includes applications in automated trading, where we demonstrate several use cases, namely stock trading, portfolio allocation, cryptocurrency trading, etc. We provide baseline trading strategies to alleviate debugging workloads.

Under the three-layer framework, FinRL is developed with three primary principles:

- **Full-stack framework**. To provide a full-stack DRL framework with finance-oriented optimizations, including market data APIs, data preprocessing, DRL algorithms, and automated backtesting. Users can transparently make use of such a development pipeline.
- **Customization**. To maintain modularity and extensibility in development by including state-of-the-art DRL algorithms and supporting design of new algorithms. The DRL algorithms can be used to construct trading strategies by simple configurations.

*Corresponding author.

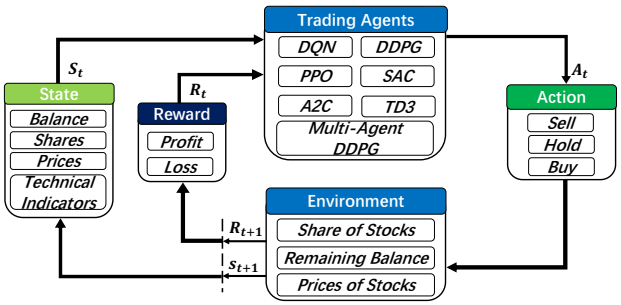

Figure 1: Overview of automated trading in FinRL, using deep reinforcement learning.

- **Reproducibility and hands-on tutoring**. To provide tutorials such as step-by-step Jupyter notebooks and user guide to help users walk through the pipeline and reproduce the use cases.

This leads to a unified framework where developers are able to efficiently explore ideas through high-level configurations and specifications, and to customize their own strategies at request.

Our contributions are summarized as follows:

- FinRL is the first open-source framework that demonstrates the great potential of applying DRL algorithms in quantitative finance. We build an ecosystem around the FinRL framework, which seeds the rapidly growing AI4Finance community.
- The application layer provides interfaces for users to customize FinRL to their own trading tasks. Automated backtesting module and performance metrics are provided to help quantitative traders iterate trading strategies at a high turnover rate. Profitable trading strategies are reproducible and hands-on tutorials are provided in a beginner-friendly fashion. Adjusting the trained models to the rapid changing markets is also possible.
- The agent layer provides state-of-the-art DRL algorithms that are adapted to finance with fine-tuned hyperparameters. Users can add new DRL algorithms.
- The environment layer includes not only a collection of historical data APIs, but also live trading APIs. They are reconfigured into standard OpenAI gym-style environments [5]. Moreover, it incorporates market frictions and allows users to customize the trading time granularity.

The remainder of this paper is organized as follows. Section 2 reviews related works. Section 3 presents the FinRL framework. Section 4 demonstrates benchmark trading tasks using FinRL. We conclude this paper in Section 5.

2 RELATED WORKS

We review the state-of-the-art DRL algorithms, relevant open-source libraries, and applications of DRL in quantitative finance.

2.1 Deep Reinforcement Learning Algorithms

Many DRL algorithms have been developed. They fall into three categories: value based, policy based, and actor-critic based.

A value based algorithm estimates a state-action value function that guides the optimal policy. Q-learning [49] approximates a Q-value (expected return) by iteratively updating a Q-table, which works for problems with small discrete state spaces and action spaces. Researchers proposed to utilize deep neural networks for approximating Q-value functions, e.g., deep Q-network (DQN) and its variants double DQN and dueling DQN [1].

A policy based algorithm directly updates the parameters of a policy through policy gradient [45]. Instead of value estimation, policy gradient uses a neural network to model the policy directly, whose input is a state and output is a probability distribution according to which the agent takes an action at the input state.

An actor-critic based algorithm combines the advantages of value based and policy based algorithms. It updates two neural networks, namely, an actor network updates the policy (probability distribution) while a critic network estimates the state-action value function. During the training process, the actor network takes actions and the critic network evaluates those actions. The state-of-art actor-critic based algorithms are deep deterministic policy gradient (DDPG), proximal policy optimization (PPO), asynchronous advantage actor critic (A3C), advantage actor critic (A2C), soft actor-critic (SAC), multi-agent DDPG, and twin-delayed DDPG (TD3) [1].

2.2 Deep Reinforcement Learning Libraries

We summarize relevant open-source DRL libraries as follows:
OpenAI Gym [5] provides standardized environments for various DRL tasks. OpenAI baselines [10] implements common DRL algorithms, while Stable Baselines 3 [37] improves [10] with code cleanup and user-friendly examples.
Google Dopamine [7] aims for fast prototyping of DRL algorithms. It features good plugability and reusability.
RLlib [25] provides highly scalable DRL algorithms. It has modular framework and is well maintained.
TensorLayer [11] is designed for researchers to customize neural networks for various applications. TensorLayer is a wrapper of TensorFlow and supports the OpenAI gym-style environments. However, it is not user-friendly.

2.3 Deep Reinforcement Learning in Finance

Many recent works have applied DRL to quantitative finance. Stock trading is considered as the most challenging task due to its noisy and volatile features, and various DRL based approaches [15, 35, 54] have been applied. Volatility scaling was incorporated in DRL algorithms to trade futures contracts, which considered market volatility in a reward function. News headline sentiments and knowledge graphs, as alternative data, can be combined with the price-volume data as time series to train a DRL trading agent. High frequency trading using DRL [38] is a hot topic.

Deep Hedging [6] designed hedging strategies with DRL algorithms that manages the risk of liquid derivatives. It has shown two advantages of DRL in mathematical finance, *scalable* and *model-free*. DRL driven strategy would become more efficient as the scale of the portfolio grows. It uses DRL to manage the risk of liquid derivatives, which indicates further extension of our FinRL library into other asset classes and topics in mathematical finance.

Cryptocurrencies are rising in the digital financial market, such as Bitcoin (BTC) [40], and are considered more volatile than stocks. DRL is also being actively explored in automated trading, portfolio allocation, and market making for cryptocurrencies [20, 39, 41].

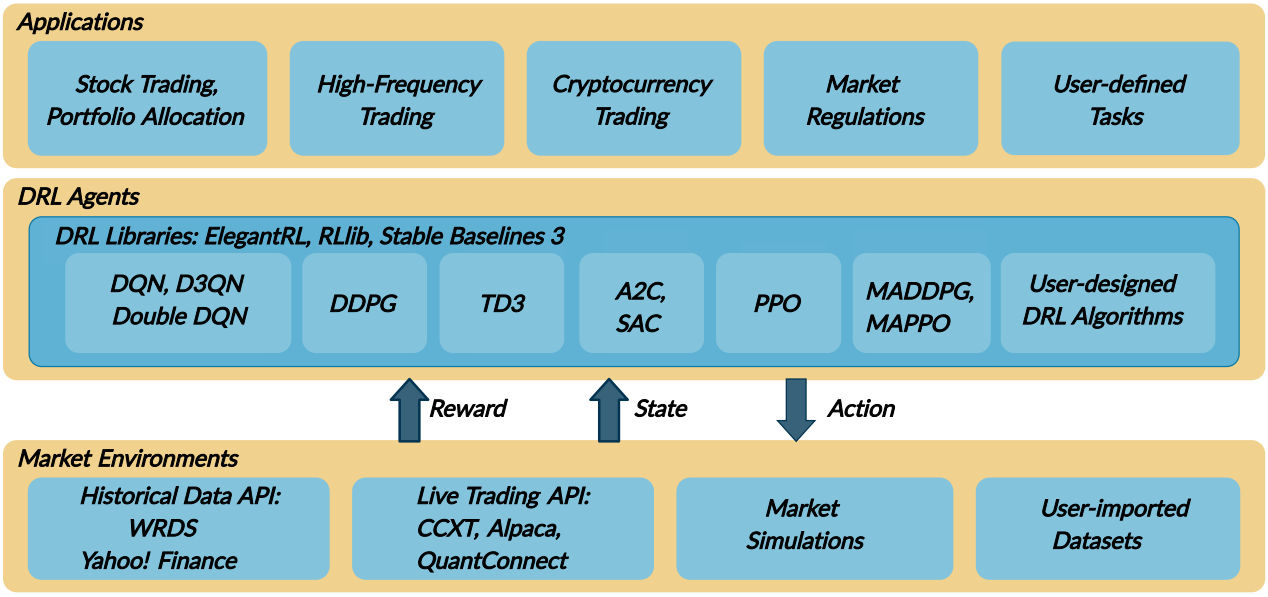

Figure 2: Overview of FinRL: application layer at the top, agent layer in the middle and environment layer at the bottom.

3 THE PROPOSED FINRL FRAMEWORK

In this section, we first present an overview of the FinRL framework and describe its layers. Then, we propose a training-testing-trading pipeline as a standard evaluation of the trading performance.

3.1 Overview of FinRL Framework

The FinRL framework has three layers, application layer, agent layer, and environment layer, as shown in Fig. 2.

- On the application layer, FinRL aims to provide hundreds of demonstrative trading tasks, serving as stepping stones for users to develop their strategies.
- On the agent layer, FinRL supports fine-tuned DRL algorithms from DRL libraries in a plug-and-play manner, following the unified workflow in Fig. 1.
- On the environment layer, FinRL aims to wrap historical data and live trading APIs of hundreds of markets into training environments, following the defacto standard Gym [5].

Upper-layer trading tasks can directly call DRL algorithms in the agent layer and market environments in the environment layer.

The FinRL framework has the following features:

- **Layered architecture**: The lower layer provides APIs for the upper layer, ensuring *transparency*. The agent layer interacts with the environment layer in an exploration-exploitation manner. Updates in each layer is independent, as long as keeping the APIs in Table 2 unchanged.
- **Modularity and extensibility**: Each layer has modules that define self-contained functions. A user can select certain modules to implement her trading task. We reserve interfaces for users to develop new modules, e.g., adding new DRL algorithms.
- **Simplicity and applicability**: FinRL provides benchmark trading tasks that are reproducible for users, and also enables users to customize trading tasks via simple configurations. In addition, hands-on tutorials are provided in a beginner-friendly fashion.

Key components	Attributes
State	Balance $b_t \in \mathbb{R}_+$; Shares $\boldsymbol{k}_t \in \mathbb{Z}_+^n$
	OHLCV data $\boldsymbol{o}_t, \boldsymbol{h}_t, \boldsymbol{l}_t, \boldsymbol{p}_t, \boldsymbol{v}_t \in \mathbb{R}_+^n$
	Technical indicators; Fundamental indicators
	Smart beta
	NLP market sentiment features
Action	Buy/Sell/Hold; Short/Long
	Portfolio weights
Rewards	Change of portfolio value
	Portfolio log-return
	Shape ratio
Environment	Dow-30, NASDAQ-100, S&P-500
	Cryptocurrencies
	Foreign currency and exchange
	Futures and options
	Living trading

Table 1: Key components and attributes. OHLCV stands for Open, High, Low, Close and Volume.

3.2 Application Layer

On the application layer, users map an algorithmic trading strategy into the DRL language by specifying the state space, action space and reward function. For example, the state, action and reward for several use cases are given in Table 1. Users can customize according to their own trading strategies.

State space $\mathcal{S}$. The state space describes how the agent perceives the environment. A trading agent observes many features to make sequential decisions in an interactive market environment. We allow the time step t to have *multiple levels of granularity*, e.g., daily, hourly or a minute basis. We provide various features for users to select and update, in each time step t:

- Balance $b_t \in \mathbb{R}_+$: the account balance at the current time step t.

- Shares $\boldsymbol{k}_t \in \mathbb{Z}_+^n$: current shares for each asset, where n represents the number of stocks in the portfolio.
- Open-high-low-close (OHLC) prices $\boldsymbol{o}_t, \boldsymbol{h}_t, \boldsymbol{l}_t, \boldsymbol{p}_t \in \mathbb{R}_+^n$ and trading volume $\boldsymbol{v}_t \in \mathbb{R}_+^n$.
- Technical indicators, including Moving Average Convergence Divergence (MACD), Relative Strength Index (RSI), etc.
- Fundamental indicators, including return on assets (ROA), return on equity (ROE), net profit margin (NPM), price-to-earnings (PE) ratio, price-to-book (PB) ratio, etc.

Action space $\mathcal{A}$. The action space describes the allowed actions that an agent can take at a state. An action of one share is $a \in \{-1, 0, 1\}$ where $-1, 0, 1$ represent selling, holding, and buying, respectively; an action of multiple shares is $a \in \{-k, ..., -1, 0, 1, ..., k\}$ where k denotes the maximum number of shares to buy or sell, e.g., "Buy/Sell 10 shares of AAPL" is 10 or -10, respectively.

Reward function. The reward function $r(s, a, s')$ is the incentive for an agent to learn a better policy. FinRL supports user-defined reward functions to reflect risk-aversion or market friction [6, 54] and provides commonly used ones [13] as follows:

- The change of the portfolio value when taking action a at state s and arriving at new state s' [35, 50, 51], $r(s, a, s') = v' - v$, where v' and v are portfolio values at s' and s, respectively.
- The portfolio log return [18], $r(s, a, s') = \log(v'/v)$.
- The Sharpe ratio for trading periods $t = 1, ..., T$ [34],

$$\text{Sharpe ratio} = (\mathbb{E}(R_t) - R_f)/\text{std}(R_t), \tag{1}$$

where $R_t = v_t - v_{t-1}$, and R_f is the risk-free rate.

3.3 Agent Layer

FinRL allows users to plug in and play with standard DRL algorithms, following the unified workflow in Fig. 1. As a backbone, we fine-tune three representative open-source DRL libraries, namely Stable Baselines 3 [37], RLlib [25] and ElegantRL [28]. User can also design new DRL algorithms by adapting existing ones.

3.3.1 Agent APIs. FinRL uses unified Python APIs for training a trading agent. The Python APIs are flexible so that a DRL algorithm can be easily plugged in. To train a DRL trading agent, as in Fig. 2, a user chooses an environment (i,e., StockTradingEnv, StockPortfolioEnv) built on historical data or live trading APIs with default parameters (env_kwargs), and picks a DRL algorithm (e.g., PPO [42]). Then, FinRL initializes the agent class with the environment, sets a DRL algorithm with its default hyperparameters (model_kwargs), then launches a training process and returns a trained model.

The main APIs are given in Table 2, while the details of building an environments, importing an algorithm, and constructing an agents are hidden in the API calls.

3.3.2 Plug-and-Play DRL Libraries. Fig. 3 compares the three DRL libraries. The details of each library are summarised as follows.

Stable Baselines 3 [37] is a set of improved implementations of DRL algorithms over the OpenAI Baselines [10]. FinRL chooses to support Stable Baselines 3 due to its **advantages**: 1). User-friendly, 2). Easy to replicate, refine, and identify new ideas, and 3). Good documentation. Stable Baselines 3 is used as a base around which new ideas can be added, and as a tool for comparing a new approach against existing ones. The purpose is that the simplicity of these tools will allow beginners to experiment with a more advanced tool set, without being buried in implementation details.

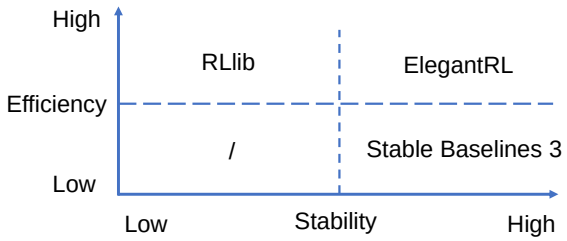

Figure 3: Comparison of DRL libraries.

RLlib [25] is an open-source high performance library for a variety of general applications. FinRL chooses to support RLlib due to its **advantages**: 1). High performance and parallel DRL training framework; 2). Scale training onto large-scale distributed servers; and 3). Allowing the multi-processing technique to efficiently train on laptops. RLlib natively supports TensorFlow, TensorFlow Eager, and PyTorch, but most of its internals are framework agnostic.

ElegantRL [28] is designed for researchers and practitioners with finance-oriented optimizations. FinRL chooses to support ElegantRL due to its **advantages**: 1). Lightweight: core codes have less than 1,000 lines, less dependable packages, only using PyTorch (train), OpenAI Gym [5] (env), NumPy, Matplotlib (plot); 2). Customization: Due to the completeness and simplicity of the code structure, users can easily customize their own agents; 3). Efficient: Performance is comparable with RLlib [25]; and 4). Stable: As stable as Stable Baselines 3 [37].

ElegantRL supports state-of-the-art DRL algorithms, including both discrete and continuous ones, and provides user-friendly tutorials in Jupyter Notebooks. ElegantRL implements DRL algorithms under the Actor-Critic framework, where an agent consists of an actor network and a critic network. The ElegantRL library enables researchers and practitioners to pipeline the disruptive "design, development and deployment" of DRL technology.

Customizing trading strategies. Due to the uniqueness of different financial markets, customization becomes a vital character to design trading strategies. Users are able to select a DRL algorithm and easily customize it for their trading tasks by specifying the state-action-reward tuple in Table 1. We believe that among the three state-of-the-art DRL libraries, **ElegentRL** is a practically useful option for financial tasks because of its completeness and simplicity along with its comparable performance with RLlib [25] and stability with Stable Baselines 3 [37].

3.4 Environment Layer

Environment design is crucial in DRL, because the agent learns by interacting with the environment in a trial and error manner. A good environment that simulates real-world market will help the agent learn a better strategy. Considering the stochastic and interactive nature, a financial task is modeled as a Markov Decision Process (MDP), whose state transition is shown in Fig. 1.

The environment layer in FinRL is responsible for observing current market information and translating those information into states of the MDP problem. The state variables can be categorized

Function	Description
env = StockTradingEnv(df, **env_kwargs)	Return an environment instance of the Env class with data and default parameters.
agent = DRLAgent(env)	Instantiate a DRL agent with a given environment env.
model = agent.get_model(model_name, **model_kwargs)	Return a model with name and default hyperparameters.
trained_model = agent.train_model(model)	Launch the training process for the agent and return a trained model.

Table 2: Main APIs of FinRL.

into the state of an agent and the state of the market. For example, in the use case stock trading, the state of the market includes the open-high-low-close prices and volume (OHLCV) and technical indicators; the state of an agent includes the account balance and the shares for each stock.

The RL training process involves observing price change, taking an action and calculating a reward. By interacting with the environment, the agent updates iteratively and eventually obtains a trading strategy to maximize the expected return. We reconfigure real market data into gym-style training environments according to the principle of *time-driven simulation*. Inspired by OpenAI Gym [5], FinRL provides strategy builders with a collection of universal training environments for various trading tasks.

3.4.1 Standard Datasets and Live Trading APIs. DRL in finance is different from chess, card games and robotics [44, 52], which may have physical engines or simulators. Different financial tasks may require different market simulators. Building such training environments is time-consuming, so FinRL provides a set of representative ones and also supports user-import data, aiming to free users from such tedious and time-consuming work.

NASDAQ-100 index constituents are 100 stocks that are characterized by high technology and high growth.

Dow Jones Industrial Average (DJIA) index is made up of 30 representative constituent stocks. DJIA is the most cited market indicator to examine market overall performance.

Standard & Poor's 500 (S&P 500) index constituents consist of 500 largest U.S. publicly traded companies.

Hang Seng Index Index (HSI) constituents are grouped into Finance, Utilities, Properties and Commerce & Industry [19]. HSI is the most widely quoted indicator of the Hong Kong stock market.

SSE 50 Index constituents [12] include the best representative companies (in 10 industries) of A shares listed at Shanghai Stock Exchange (SSE) with considerable size and liquidity.

CSI 300 Index constituents [8] consist of the 300 largest and most liquid A-share stocks listed on Shenzhen Stock Exchange and SSE. This index reflects the performance of the China A-share market.

Bitcoin (BTC) Price Index consists of the quote and trade data on Bitcoin market, available at https://public.bitmex.com/.

3.4.2 User-Imported Data. Users may want to train agents on their own data sets. FinRL provides convenient support for users to import data, adjust the time granularity, and perform the training-testing-trading data split. We specify the format for different trading tasks, and users preprocess and format the data according to our instructions. Stock statistics and indicators can be calculated using our support, which provides more features for the state space. Furthermore, episodic total return and Sharpe ratio can also assist performance evaluation.

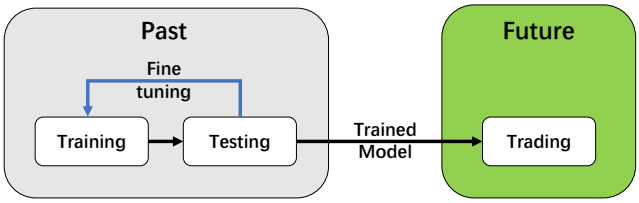

Figure 4: The training-testing-trading pipeline.

3.5 Training-Testing-Trading Pipeline

The "training-testing" workflow used by conventional machine learning methods falls short for financial tasks. It splits the data into training set and testing set. On the training data, users select features and tune parameters; then evaluate on the testing data. However, financial tasks will experience a *simulation-to-reality gap* between the testing performance and real-live market performance. Because the testing here is offline backtesting, while the users' goal is to place orders in a real-world market.

FinRL employs a "training-testing-trading" pipeline to reduce the simulation-to-reality gap. We use historical data (time series) for the "training-testing" part, which is the same as conventional machine learning tasks, and this testing period is for backtesting purpose. For the "trading" part, we use live trading APIs, such as CCXT, Alpaca, or Interactive Broker, allowing users carry out trades directly in a trading system. Therefore, FinRL directly connects with live trading APIs: 1). downloads live data, 2). feeds data to the trained DRL model and obtains the trading positions, and 3). allows users to place trades.

Fig. 4 illustrates the "training-testing-trading" pipeline:

Step 1). A training window to retrain an agent.

Step 2). A testing window to evaluate the trained agent, while hyperparameters can be tuned iteratively.

Step 3). Use the trained agent to trade in a trading window.

Rolling window is used in the training-testing-trading pipeline, because the investors and portfolio managers need to retrain the model periodically as time goes ahead. FinRL provides flexible selections of rolling windows, such as monthly, quarterly, yearly windows, or by users' specifications.

4 HANDS-ON TUTORIALS AND BENCHMARK PERFORMANCE

We provide hands-on tutorials and reproduce existing works as use cases. Their configurations and commands are available on Github.

4.1 Backtesting Module

Backtesting plays a key role in evaluating a trading strategy. FinRL library provides an automated backtesting module based on Quantopian pyfolio package [36]. It is easy to use and consists of various

individual plots that provide a comprehensive image of the performance. In order to facilitate users, FinRL also incorporates market frictions, market liquidity and the investor's degree of risk-aversion.

4.1.1 Incorporating Trading Constraints. Transaction costs incur when executing a trade, such as broker commissions and the SEC fee. We allow users to treat transaction costs as parameters in the environments: 1). **Flat fee** is a fixed amount per trade; and 2). **Per share percentage** is a percentage rate for every share, e.g., 0.1% or 0.2% are most commonly used.

Moreover, we need to consider market liquidity for stock trading, e.g., the bid-ask spread that is the difference between the best bid and ask prices. In our environment, users can add the bid-ask spread as a parameter. For different levels of risk-aversion, users can add the standard deviation of the portfolio returns into the reward function or use a risk-adjusted Sharpe ratio as the reward function.

4.1.2 Risk-aversion. An investor may prefer conservative trading in highly volatile markets. For a worst case scenario as the 2008 global financial crisis, FinRL employs the turbulence index turbulence_t to measure extreme fluctuation [23]:

$$\text{turbulence}_t = (\boldsymbol{y_t} - \boldsymbol{\mu})^T \boldsymbol{\Sigma^{-1}} (\boldsymbol{y_t} - \boldsymbol{\mu}) \in \mathbb{R}, \tag{2}$$

where $\boldsymbol{y_t} \in \mathbb{R}^n$ is the return at t, $\boldsymbol{\mu} \in \mathbb{R}^n$ is the average of historical returns, and $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ is the covariance matrix of historical returns. $turbulence_t$ can be used to control buying/selling actions. If $turbulence_t$ is higher than a preset threshold, the agent halts and will resume when $turbulence_t$ becomes lower than the threshold.

4.2 Baseline Strategies and Trading Metrics

Baseline trading strategies are provided to compare with DRL strategies. Investors usually have two mutually conflicting objectives: the highest possible profits and the lowest possible risks [43]. We include three conventional strategies as baselines.

Passive trading strategy [31] is an easy and popular strategy that has the minimal trading activities. Investors simply buy and hold index ETFs [46] to replicate a broad market index or indices such as Dow Jones Industrial Average (DJIA) index and Standard & Poor's 500 (S&P 500) index.

Mean-variance and min-variance strategy [2] both aim to achieve an optimal balance between the risks and profits. It selects a diversified portfolio with risky assets, and the risk is diversified when traded together.

Equally weighted strategy is a type of portfolio allocation method. It gives the same importance to each asset in a portfolio.

FinRL includes common metrics to evaluate trading performance:
Final portfolio value: the amount of money at the end of the trading period.
Cumulative return: subtracting the initial value from the final portfolio value, then dividing by the initial value.
Annualized return and standard deviation: geometric average return in a yearly sense, and the corresponding deviation.
Maximum drawdown ratio: the maximum observed loss from a historical peak to a trough of a portfolio, before a new peak is achieved. Maximum drawdown is an indicator of downside risk over a time period.
Sharpe ratio in (1) is the average return earned in excess of the risk-free rate per unit of volatility.

4.3 Hands-on Tutorials

We provide tutorials to help users walk through the strategy design pipeline, i.e., get familiar with the stat-action-reward specifications in Table 1 and the agent-environment interactions in Fig. 1.

Tutorial 1: Stock trading

First, users specify the *state* at the application layer, i.e., the number of stocks, technical indicators, the initial capital, etc. Second, users provide start/end dates for training/testing periods, set the time granularity. FinRL instantiates an environment for the task, while the operations are transparent to users. FinRL uses standard APIs to download data and obtains a Pandas DataFrame containing the open-high-low-close prices and volume (OHLCV) data. FinRL preprocesses the OHLCV data by filling missing data and calculates technical indicators that are passed into the state. Third, users select a DRL library and a DRL algorithm. FinRL has default hyperparameters for daily stock trading task. During the testing period, users can tune these parameters to improve the trading performance. Finally, FinRL feeds time series data of the portfolio value into a backtesting module to plot charts. Please see examples in Section 4.4 and Section 4.6.

Tutorial 2: Analyzing Trading Performance

Before deploying a trading strategy, users need to fully evaluate its trading performance via backtesting. The trading performance can be easily evaluated using the automatic backtesting module in Section 4.1. The commonly used trading metrics and baseline strategies are given in Section 4.2.

Cumulative return and Sharpe ratio are widely used metrics to evaluate overall performance of trading strategies. To gain more details about the strategy, the distribution of returns over the testing period and annualized return are provided to examine if the return is stable and consistent. Annualized volatility and maximum drawdown measure the robustness.

4.4 Use Case I: Stock Trading

We use FinRL to reproduce both [50] and [51] for stock trading. The ensemble strategy [51] combines three DRL algorithms (PPO [42], A2C [32] and DDPG [26]) to improve the robustness.

The implementation is easy with FinRL. We choose three algorithms (PPO, A2C, DDPG) in the agent layer, and an environment with start and end dates in the environment layer. The implementations of DRL algorithms and data preprocessing are transparent to users, alleviating the programming and debugging workloads. Thus, FinRL greatly facilitates the strategy design, allowing users to focus on improving the trading performance.

Fig. 5 and Table 3 show the backtesting performance on Dow 30 constituent stocks, accessed at 2020/07/01. The training period is from 2009/01/01 to 2020/06/30 on a daily basis, and the testing period is from 2020/07/01 to 2021/06/30. The performance in terms of multiple metrics is consistent with the results reported in [51] and [50], and here we show results in a recent trading period. We can see from the DJIA index that the trading period is a bullish market with an annual return of 32.84%. The ensemble strategy achieves a Sharpe ratio of 2.81 and an annual return of 52.61%. It beats A2C with a Sharpe ratio of 2.24, PPO with a Sharpe ratio of 2.23, DDPG with a Sharpe ratio of 2.05, DJIA with a Sharpe ratio of 2.02, and min-variance portfolio allocation with a Sharpe ratio of 1.98,

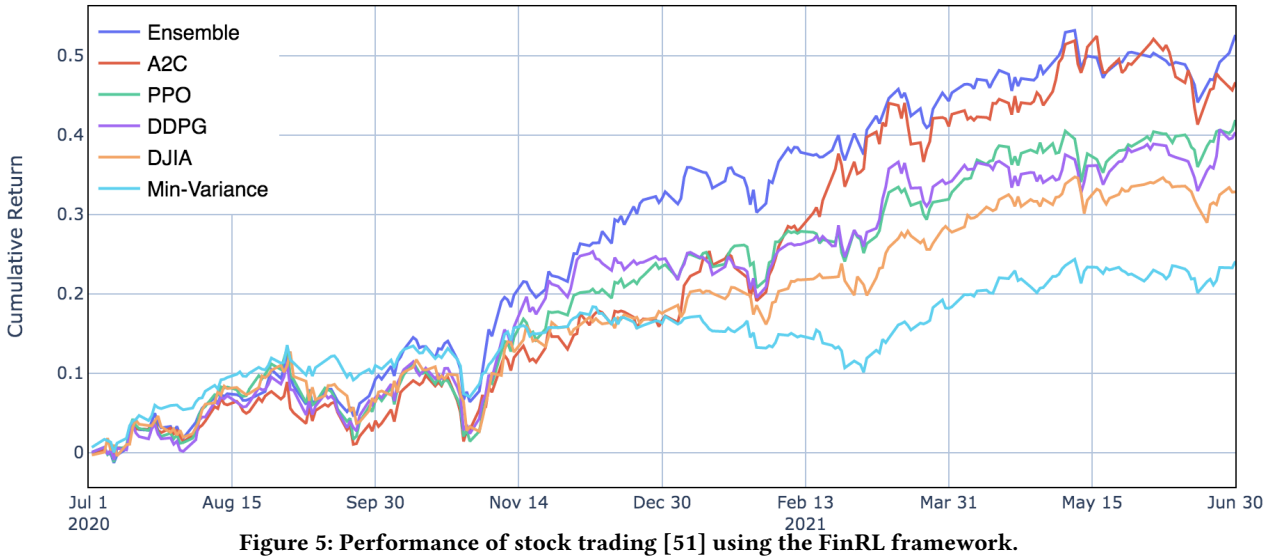

Figure 5: Performance of stock trading [51] using the FinRL framework.

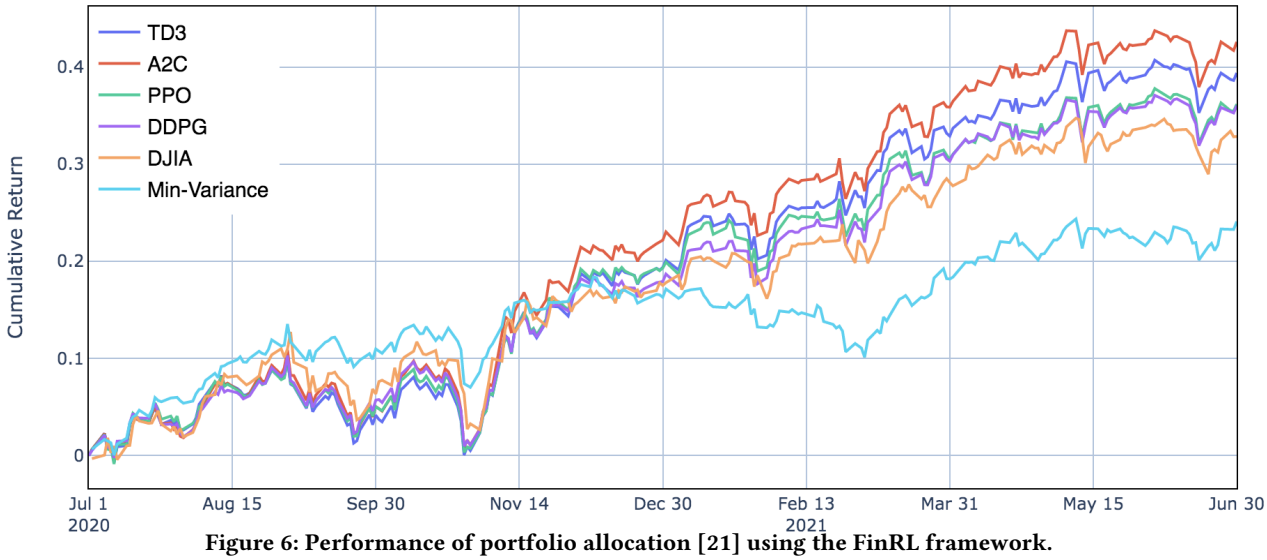

Figure 6: Performance of portfolio allocation [21] using the FinRL framework.

07/01/2020-06/30/2021	Ensemble [51]	A2C	PPO	DDPG	TD3	Min-Var.	DJIA
Initial value	1M	1M	1M	1M	1M	1M	1M
Final value	1.52M	1.46M; 1.43M	1.42M; 1.36M	1.40M; 1.36M	1.39M	1.24M	1.33M
Annualized return	52.61%	46.65%; 42.57%	41.90%; 36.17%	40.34%; 36.01%	39.38%	24.10%	32.84%
Annualized Std	15.53%	17.86%; 15.51%	16.33%; 15.20%	17.28%; 14.38%	15.08%	11.2%	14.5%
Sharpe ratio	2.81	2.24; 2.36	2.23; 2.11	2.05; 2.21	2.28	1.98	2.02
Max drawdown	-7.09%	-7.59%; -9.04%	-9.41%; -8.68%	-8.10%; -8.46%	-8.92%	-6.97%	-8.93%

Table 3: Performance of stock trading and portfolio allocation over the DJIA constituents stocks using FinRL. The Sharpe ratios of the ensemble strategy and the individual DRL agents exceed those of the DJIA index, and of the min-variance strategy.

respectively. Therefore, the backtesting performance demonstrates that FinRL successfully reproduces the ensemble strategy [51].

4.5 Use Case II: Portfolio Allocation

We reproduce a portfolio allocation strategy [21] that uses a DRL agent to allocate capital to a set of stocks and reallocate periodically.

FinRL improves the reproducibility by allowing users to easily compare the results of different settings, such as the pool of stocks to trade, the initial capital, and the model hyperparameters. It utilizes the agent layer to specify the state-of-the-art DRL libraries. Users do not need to redevelop the neural networks and instead they can just plug-and-play with any DRL algorithm.

Fig. 6 and Table 3 depict the backtesting performance on Dow 30 constituent stocks. The training and testing period is the same with Case I. It shows that each DRL agent, namely A2C [32], TD3

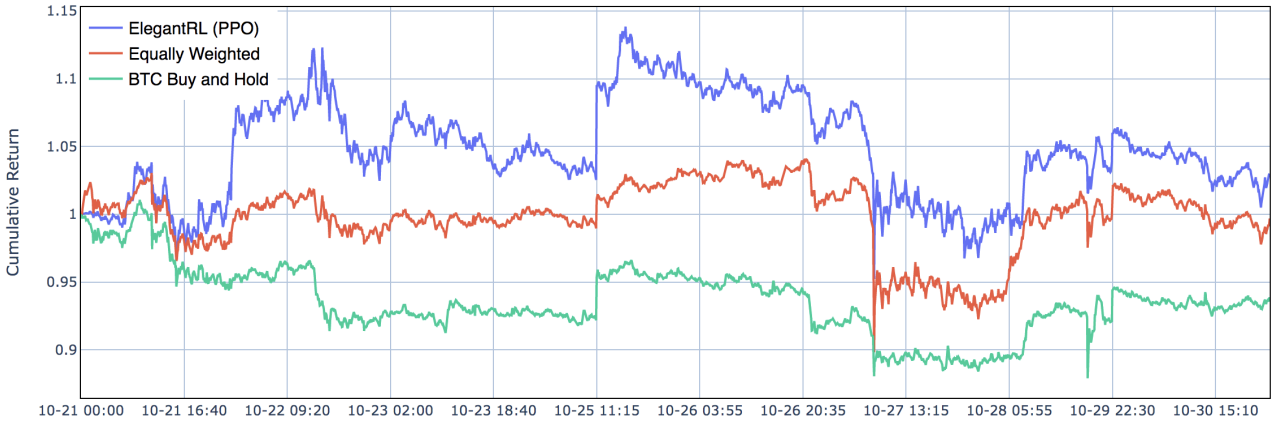

Figure 7: Cumulative returns (5-minute interval) of trading top 10 market cap cryptocurrencies using FinRL.

[14], PPO [42], and DDPG [26], outperforms the DJIA index and the min-variance strategy. A2C has the best performance with a Sharpe ratio of 2.36 and an annual return of 42.57%; TD3 is the second best agent with a Sharpe ratio of 2.28 and an annual return of 39.38%; PPO with a Sharpe ratio of 2.11 and an annual return of 36.17% and DDPG with a Sharpe ratio of 2.21 and an annual return of 36.01%. Therefore, using FinRL, users can easily compare the agents' performance with each other and with the baselines.

4.6 Use Case III: Cryptocurrencies Trading

We use FinRL to reproduce [20] for top 10 market cap cryptocurrencies [1]. FinRL provides a full-stack development pipeline, allowing users to have an end-to-end walk-through of how to download market data using APIs, perform data preprocessing, fine-tune DRL algorithms, and get automated backtesting performance.

Fig. 7 shows the backtesting performance on the ten cryptocurrencies with transaction cost. The training period is from 2021/10/01 to 2021/10/20 with a 5-minute interval, and the testing period is from 2021/10/21 to 2021/10/30. The portfolio with the PPO algorithm from the ElegantRL library [28] has the highest cumulative return of 103%, equally weighted portfolio strategy has the second highest cumulative return of 99%, while BTC buy and hold strategy has a cumulative return of 93%. Therefore, the backtesting performance shows that FinRL successfully reproduce [20] with completeness and simplicity.

5 ECOSYSTEM OF FINRL AND CONCLUSIONS

In this paper, we have developed an open-source framework, FinRL, to help quantitative traders overcome the steep learning curve. Customization is accessible on all layers, from market environments, trading agents up towards trading tasks. FinRL follows a training-testing-trading pipeline to reduce the simulation-to-reality gap. Within FinRL, historical market data and live trading APIs are reconfigured into standardized environments in OpenAI gym-style; state-of-the-art DRL algorithms are implemented for users to train trading agents in a pipeline; and an automated backtesting module is provided to evaluate trading performance. Moreover, benchmark schemes on typical trading tasks are provided as practitioners' stepping stones.

Ecosystem of FinRL Framework. We believe that the open-source community will greatly promote AI for finance in both academia and industry. As the AI4Finance community is growing rapidly, FinRL provides an open-source ecosystem that features Deep Reinforcement Learning in finance for all-level users.

FinRL offers a full-stack framework that consists of various markets, SOTA DRL algorithms, finance tasks (portfolio allocation, cryptocurrency trading, high-frequency trading), live trading support, etc. For entry-level users, FinRL aims to provide a demonstrative and educational culture with hands-on documents to help beginners get familiar with DRL in finance applications. For intermediate-level users, such as full-stack developers and professionals, FinRL provides ElegantRL [28], a lightweight and scalable DRL library with finance-oriented optimizations. For advanced-level users, such as investment banks and hedge funds. FinRL delivers FinRL-Podracer [24, 29], a cloud-native solution with high performance and high scalability training.

FinRL also develops other useful tools to support the ecosystem. FinRL-Meta [30] adds financial data engineering with a unified data processor and hundreds of market environments. Explainable DRL for portfolio management [17] and DRL ensemble strategy for stock trading [50, 51] are also implemented using FinRL.

Future work. Future research directions would be investiaging DRL's potential on limit order book [48], hedging [6], market making [16], liquidation [3], and trade execution [27].

[1]The top 10 market cap cryptocurrencies as of Oct 2021 are: Bitcoin (BTC), Ethereum (ETH), Cardano (ADA), Binance Coin (BNB), Ripple (XRP), Solana (SOL), Polkadot (DOT), Dogecoin (DOGE), Avalanche (AVAX), Uniswap (UNI).

REFERENCES

[1] Joshua Achiam. 2018. Spinning up in deep reinforcement learning. https://spinningup.openai.com
[2] Andrew Ang. August 10, 2012. Mean-variance investing. *Columbia Business School Research Paper No. 12/49.* (August 10, 2012).
[3] Wenhang Bao and Xiao-Yang Liu. 2019. Multi-agent deep reinforcement learning for liquidation strategy analysis. *ICML Workshop on Applications and Infrastructure for Multi-Agent Learning* (2019).
[4] Stelios D Bekiros. 2010. Fuzzy adaptive decision-making for boundedly rational traders in speculative stock markets. *European Journal of Operational Research* 202, 1 (2010), 285–293.
[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI gym. *arXiv preprint arXiv:1606.01540* (2016).
[6] Hans Buehler, Lukas Gonon, Josef Teichmann, Ben Wood, Baranidharan Mohan, and Jonathan Kochems. 2019. Deep hedging: Hedging derivatives under generic market frictions using reinforcement learning. *Swiss Finance Institute Research Paper* 19-80 (2019).

[7] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. 2018. Dopamine: A research framework for deep reinforcement learning. *http://arxiv.org/abs/1812.06110* (2018).

[8] Ltd China Securities Index Co. 2017. CSI 300. http://www.csindex.com.cn/uploads/indices/detail/files/en/145_000300_Fact_Sheet_en.pdf

[9] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. 2016. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems* 28, 3 (2016), 653–664.

[10] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. 2017. OpenAI baselines. https://github.com/openai/baselines.

[11] Hao Dong, Akara Supratak, Luo Mai, Fangde Liu, Axel Oehmichen, Simiao Yu, and Yike Guo. 2017. TensorLayer: A versatile library for efficient deep learning development. In *Proceedings of the 25th ACM International Conference on Multimedia*. 1201–1204.

[12] Shanghai Stock Exchange. 2018. SSE 180 Index Methodology. http://www.sse.com.cn/market/sseindex/indexlist/indexdetails/indexmethods/c/IndexHandbook_EN_SSE180.pdf

[13] Thomas G. Fischer. 2018. *Reinforcement learning in financial markets - a survey.* FAU Discussion Papers in Economics. Friedrich-Alexander University Erlangen-Nuremberg, Institute for Economics.

[14] Scott Fujimoto, Herke Van Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning* (2018).

[15] Prakhar Ganesh and Puneet Rakheja. 2018. Deep reinforcement learning in high frequency trading. *ArXiv* abs/1809.01506 (2018).

[16] Sumitra Ganesh, Nelson Vadori, Mengda Xu, Hua Zheng, Prashant Reddy, and Manuela Veloso. 2019. Reinforcement learning for market making in a multi-agent dealer market. *NeurIPS'19 Workshop on Robust AI in Financial Services.*

[17] Mao Guan and Xiao-Yang Liu. 2021. Explainable deep reinforcement learning for portfolio management: an empirical approach. *ACM International Conference on AI in Finance (ICAIF)* (2021).

[18] Chien Yi Huang. 2018. Financial trading as a game: A deep reinforcement learning approach. *arXiv preprint arXiv:1807.02787* (2018).

[19] Hang Seng Index. 2020. Hang Seng index and sub-indexes. https://www.hsi.com.hk/eng/indexes/all-indexes/hsi

[20] Zhengyao Jiang and J. Liang. 2017. Cryptocurrency portfolio management with deep reinforcement learning. *Intelligent Systems Conference (IntelliSys)* (2017), 905–913.

[21] Zhengyao Jiang, Dixing Xu, and J. Liang. 2017. A deep reinforcement learning framework for the financial portfolio management problem. *ArXiv* abs/1706.10059.

[22] Prahlad Koratamaddi, Karan Wadhwani, Mridul Gupta, and Sriram G. Sanjeevi. 2021. Market sentiment-aware deep reinforcement learning approach for stock portfolio allocation. *Engineering Science and Technology, an International Journal.*

[23] Mark Kritzman and Yuanzhen Li. 2010. Skulls, financial turbulence, and risk management. *Financial Analysts Journal* 66 (10 2010).

[24] 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. *ACM International Conference on AI in Finance (ICAIF)* (2021).

[25] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. 2018. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning (ICML).*

[26] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. *ICLR* (2016).

[27] Siyu Lin and P. Beling. 2020. A deep reinforcement learning framework for optimal trade execution. In *ECML/PKDD.*

[28] 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.

[29] Xiao-Yang Liu, Zechu Li, Zhuoran Yang, Jiahao Zheng, Zhaoran Wang, Anwar Walid, Jian Guo, and Michael Jordan. 2021. ElegantRL-Podracer: Scalable and elastic library for cloud-native deep reinforcement learning. *Deep RL Workshop, NeurIPS 2021* (2021).

[30] Xiao-Yang Liu, Jingyang Rui, Jiechao Gao, Liuqing Yang, Hongyang Yang, Zhaoran Wang, Christina Dan Wang, and Guo Jian. 2021. Data-driven deep reinforcement learning in quantitative finance. *Data-Centric AI Workshop, NeurIPS.*

[31] B. G. Malkiel. 2003. Passive investment strategies and efficient markets. *European Financial Management* 9 (2003), 1–10.

[32] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.

[33] John Moody and Matthew Saffell. 2001. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks* 12, 4 (2001), 875–889.

[34] J. Moody, L. Wu, Y. Liao, and M. Saffell. 1998. Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting* 17 (1998), 441–470.

[35] Abhishek Nan, Anandh Perumal, and Osmar R Zaiane. 2020. Sentiment and knowledge based algorithmic trading with deep reinforcement learning. *ArXiv* abs/2001.09403 (2020).

[36] Quantopian. 2019. Pyfolio: A toolkit for portfolio and risk analytics in Python. https://github.com/quantopian/pyfolio.

[37] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. 2019. Stable Baselines3. https://github.com/DLR-RM/stable-baselines3.

[38] Francesco Rundo. 2019. Deep LSTM with reinforcement learning layer for financial trend prediction in FX high frequency trading systems. *Applied Sciences* 9 (10 2019), 1–18.

[39] Jonathan Sadighian. 2019. Deep reinforcement learning in cryptocurrency market making. *arXiv: Trading and Market Microstructure* (2019).

[40] Svetlana Sapuric and A. Kokkinaki. 2014. Bitcoin is volatile! Isn't that right?. In *BIS.*

[41] Otabek Sattarov, Azamjon Muminov, Cheol Lee, Hyun Kang, Ryumduck Oh, Junho Ahn, Hyung Oh, and Heung Jeon. 2020. Recommending cryptocurrency trading points with deep reinforcement learning approach. *Applied Sciences* 10.

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

[43] William F Sharpe. 1970. *Portfolio theory and capital markets.* McGraw-Hill College.

[44] 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.

[45] Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12.* MIT Press, 1057–1063.

[46] Evgeni B. Tarassov. 2016. Exchange traded funds (ETF): History, mechanism, academic literature review and research perspectives. *Microeconomics: General Equilibrium & Disequilibrium Models of Financial Markets eJournal* (2016).

[47] Nelson Vadori, Sumitra Ganesh, Prashant Reddy, and Manuela Veloso. 2020. Risk-sensitive reinforcement learning: a martingale approach to reward uncertainty. *International Conference on AI in Finance (ICAIF)* (2020).

[48] Svitlana Vyetrenko, David Byrd, Nick Petosa, Mahmoud Mahfouz, Danial Dervovic, Manuela Veloso, and Tucker Hybinette Balch. 2020. Get real: Realism metrics for robust limit order book market simulations. *International Conference on AI in Finance (ICAIF)* (2020).

[49] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3-4 (1992), 279–292.

[50] Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. 2018. Practical deep reinforcement learning approach for stock trading. *NeurIPS Workshop* (2018).

[51] 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).

[52] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and X. X. Hu. 2019. Experience replay optimization. *International Joint Conference on Artificial Intelligence (IJCAI).*

[53] Yong Zhang and Xingyu Yang. 2017. Online portfolio selection strategy based on combining experts' advice. *Computational Economics* 50, 1 (2017), 141–159.

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