Deep Reinforcement Learning and Transfer Learning Methods Used in Autonomous Financial Trading Agents

Ciprian Paduraru, Catalina Camelia Patilea and Stefan Iordache
Department of Computer Science, University of Bucharest, Romania

Keywords: Deep Reinforcement Learning, Transfer Learning, Algorithmic Trading, Deep Q-Networks, Double Deep Q-Networks.

Abstract: It is reported that some of the largest companies from the banking and business sectors are investing massively in the field of trading with automated methods. The methods used vary from classical time series based methods to Deep Learning and more recently Reinforcement Learning (RL). The main goal of this work is first to improve the state of the art in RL-based trading agents. Then, we focus on evaluating the robustness of the trained agents when they are transferred to different trading markets than the ones they were trained on. The framework we developed, *RL4FIN*, is open source and can be tested by both academia and industry. The evaluation section shows the improvements over state-of-the-art using some public datasets.

1 INTRODUCTION

Today, computing and economics are more closely linked than ever, from critical transaction systems to algorithms capable of predicting the onset of major economic crises. On a smaller scale, such as managing personal savings or bills, the economy can be easily managed by individuals and groups without the use of machines or advanced computer algorithms. However, it becomes more interesting when we add large data sets and larger financial portfolios that need to be managed. Large companies in the trading space are investing in dedicated research teams focused on developing algorithms, machine learning engines, and agents capable of making second-by-second decisions about investments without the need for human intervention.

The work we present in this research paper focuses first on discovering current approaches to automated trading based on machine learning algorithms and, in particular, reinforcement learning (RL) agents. The framework proposed in this paper is a complete data-driven pipeline, from data collection and processing to training offline and online agents, to production-ready evaluation. In the second part, we assume that we have a powerful pre-trained agent capable of strong generalization. Going into details, our contribution can be divided into three main segments:

- Building a reusable environment capable of adapting multiple reinforcement learning agents, on-policy and off-policy strategies. The environment was developed based on OpenAI Gym (Brockman et al., 2016) and can be extended to adapt to multiple markets, trading types (day trading, scalping, swing trading, position trading, etc.) or contract types (futures, options, contract for difference-CFD). Based on the common interface, users are then able to leverage existing open source training agent libraries such as Stable-Baselines3 (Raffin et al., 2021) or RLlib (Liang et al., 2018) with minimal prior knowledge of RL.
- Review and improve the state of the art of reinforcement learning implementations for algorithmic trading. In the Evaluation section, we show that the agent proposed in this paper performs better than traditional investment strategies on a number of commonly used metrics.
- The addition of transfer learning methods from one agent to another in the field of economics is also a novelty in the field that, according to our research, has not been explored in this niche literature.

The proposed framework, named *RL4FIN*, is open source (due to the limitation of double-blind testing, we will not list it here) and can be used by both the academic community and industry.

The paper is organized as follows. The next section surveys the methods used in the literature for RL methods in trading and transfer learning of such

Paduraru, C., Patilea, C. and Iordache, S.
Deep Reinforcement Learning and Transfer Learning Methods Used in Autonomous Financial Trading Agents.
DOI: 10.5220/0012194000003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 2*, pages 15-26
ISBN: 978-989-758-680-4; ISSN: 2184-433X

agents. Section 3 provides a general introduction to the RL domain. The proposed framework is outlined in Section 4. The evaluation results are presented in Section 5. Finally, the last section presents the conclusions and plans for future work.

2 RELATED WORK

Automatic trading using artificial intelligence algorithms has been explored recently in the literature. The overall purpose is to replace manual human labor with faster and improved decision-making, correlating previous states and actions, market parameters, and influence factors.

Reinforcement Learning in Trading. The work in DeepTrader(Wang et al., 2021) covers initial portfolio management, asset scoring, and market scoring. To create complete portfolios, the method first uses Long-Short Term Networks (LSTMs) (Hochreiter and Schmidhuber, 1997) to analyze specific assets over time, encapsulate knowledge, and score them. Their decision-making process is based on Graph Convolutional Networks (GCNs) (Kipf and Welling, 2017). Our current methods could also be improved by using LSTM in the network used to encapsulate the agent knowledge, either directly after processing the input layer or during the action sections, as proposed in another study in (Bakker, 2001). This remains a topic for future work and study. Our research has a different focus than their work in DeepTrade, as it focuses more on automated trading. The evaluation of our proposed methods shows improvement in this area, especially in the basic cases of trading single asset types.

FinRL (Liu et al., 2022a) (Li et al., 2021) (Liu et al., 2022b) pursues a similar goal to ours by acting in the field of trade, using reinforcement algorithm methods and creating a complete framework that enables trade at the industrial level. Their work covers multiple processes such as data sources, post-processing, feature engineering, and RL development agents. Their method uses DQN to trade entire portfolios and achieves superior performance. However, while we focus on individual asset trading types, our methods achieved better results on some public datasets. We intuitively attribute this improvement to two facts: a) the features of individual assets are extracted using a deep neural network rather than manual feature engineering, and b) the careful implementation of a DQN variant (DDQN) after our empirical evaluation of algorithms and hyperparameters on the public datasets.

Augmenting Reinforcement Learning with Knowledge Transfer. Transfer learning is widely used in the fields of computer vision and natural language processing, and recently the idea has been applied to the field of reinforcement learning. There are several surveys and research projects (Zhu et al., 2022) (Beck et al., 2022) (Taylor and Stone, 2009) that cover this mix of methods. Their work focuses on showing ways in which information and knowledge can be transferred between agents. The authors first emphasize the difficulty and that a pre-trained agent cannot be easily transferred to another environment and produce good results without major adjustments. In our research, we found only the work of the QuantNet (Koshiyama et al., 2021) project, which deals with the financial sector. They are developing a method for evaluating a machine learning algorithm constructed with LSTMs against several stock markets. In our proposed framework, the knowledge transfer method has been shown to improve end-episode returns by more than 15% when the models are applied to different markets and datasets. This result empirically indicates the potential that the proposed methods have in transferring strategies and experience from one asset to another in the field of automated trading. The idea of introducing knowledge transfer in our work also improved the average learning speed by more than 4-fold. A compilation of RL algorithms is proposed in (Yang et al., 2021), while an empirical evaluation of each RL method is reported in (Kong and So, 2023).

3 REINFORCEMENT LEARNING OVERVIEW

The first part of this section discusses the general concepts and terminology of the RL field, linking the theory and methods generally used in the economic context. For deeper details, the reader is invited to read (Sutton and Barto, 2018a). The second part focuses on explaining the mechanisms of transfer learning and the specifics used when applied to the RL domain.

3.1 Methods

Reinforcement Learning (RL) (Sutton and Barto, 2018b) is an artificial intelligence strategy that focuses mainly on developing agents that can interact with different environments. The agents are designed to explore the world and acquire expertise by performing actions, receiving rewards for doing so, and transitioning to other states from one time step to the next. In most environments, states are unlabeled, so

the main goal of reinforcement learning is to maximize the gain or reward function over multiple time steps. To describe reinforcement learning as a semi-supervised strategy for learning agents, we outline the basic components and terminology of this class of algorithms below:

The trading problem is modeled as a Partially Observable Markov Decision Process (POMDP) (Puterman, 2014). Its components and the RL specifics used by the proposed framework are described below.

- **Environment States ($s \in S$)** - a current representation of the states the agent can occupy at each time t: $s_t(NAV, M-NAV, R, M-R, POS, C, T]$, where: NAV represents the net asset value and its derivatives, R represents the return (reward), POS represents the position array (long, hold, short), C represents the trading cost, and T represents the position change or executed trade. The prefix M describes the same concepts but for the market actor (the underlying asset), while the non-M components represent the components corresponding to the trading agent trained in the environment.
- **Agent Actions ($a \in A$)** - Agents need to learn a decision process and take the most profitable actions. At each timestep, t, the action taken by the agent is in the set $a_t \in \{long, short, hold\}$. Long means that the shares are bought and held until profitable, then sold later when it becomes most convenient, i.e., long-term investment strategy. Short represents the moment shores are bought from a broker with the hope to return that investment with a higher profit and if the market declines, on the other hand, the buyer will be at a loss. Finally, when choosing hold, there is no action.
- **Agent Policy (π)** - The decision process that provides for actions given a state. In most RL implementations, this can take the form of an array, matrix, or function that is used to approximate the next action when the representations of the states/actions are continuous or inefficient from a computational point of view to be represented using either of the previous two methods. It is usually expressed as either a: (a) deterministic policy, i.e. $\pi(s)a$, which means that in state s the chosen action is a, (b) a stochastic policy, i.e. $\pi(a|s)$, which represents a distribution over the action space A at a state s. In the trading agent case, this probability distribution refers to the three actions described above and indicates how likely each is to occur at a given time step t.
- **Reward Function** - A signal value indicating how good the action performed by the agent was. The goal is to maximize the cumulative reward over a sequence of steps, i.e., an episode. Specifically for our work, the reward is the cumulative return at the end of the day after deducting all trading costs. In the case of a holding action, the agent is penalized 0.1% per trading period.
- **Episodes** - An episode (or trajectory) is considered a sequence of steps, actions, and rewards. Starting from an initial state s_0, the agent follows the policy to sample actions at each time step t, i.e., $a_t \sim \pi(s_t)$, then obtain a reward r_t from the environment. Thus the trajectory taken by the agent can be formalized as $\tau = (s_0, a_0, r_0, \ldots, s_t, a_t, r_t, \ldots, s_T, a_T, r_T)$. It can be either a finite or infinite horizon, but in this paper, we focus on the former case. In this case, T represents the length of the episode/trajectory.
- **Value Functions** - This function is conceived as a qualitative measure of states ($V(s)$ function), or combinations of states of action ($Q(s,a)$). If $V(s)$ is considered to be the average estimated value of any action that follows in state s, and $Q(s,a)$ is the estimated value for performing action a in state s, then $A(s,a) = Q(s,a) - V(s)$, the advantage function, denotes the estimated advantage value of being in state s and performing action a first.

Environments can be either deterministic or non-deterministic (e.g., an agent acting on a slippery surface might try to keep moving but end up in a different direction). At each timestamp t, a transition is decided via a probability function $P(s_{t1}|s_t, a_t)$ given by the policy sampling process. As mentioned earlier, after each action, outcomes are collected in the form of rewards, which are the main measure of the agent's performance. A discount factor $\gamma \in [0,1]$ is used to balance immediate rewards with longtime horizon rewards, i.e., a value of $\gamma 1$ means that the agent considers each reward along a trajectory with equal importance, while a value of γ The goal of the trained agent is then to maximize the cumulative reward of a trajectory, i.e., $R(\tau)\sum_{t't}^{T}\gamma^{t'}r_{t'}$.

The challenges we encountered during this research were related to the nature of the economic field and how the concepts can be mapped to the reinforcement learning field. For example, the absence of final states conducted us in the direction of model-free reinforcement learning algorithms, mainly using agents derived from Q-Learning: Deep Q-Networks (DQN) (Mnih et al., 2013), Duelling Deep Q-Networks (Duelling-DQN) (Wang et al., 2016) and Double Deep Q-Network (DDQN) (van Hasselt et al., 2015).

3.2 Transfer Learning

When applying knowledge transfer from one agent to another (using reinforcement learning), the design of the solution must answer some important questions:

- Which knowledge should be marked as transferable?

  Due to the higher computational complexity of reinforcement learning algorithms compared to supervised and unsupervised learning methods, one can choose between several sources of information: the experience replay buffer, the probability distribution of actions, and the value or action-value functions.

- Are the source and target agents compatible for transfer learning, and if not, what is the adapter interface and how can this affect performance with respect to the algorithm used for training?

  This is mostly a matter of trial and error, as reinforcement learning algorithms are divided into several branches, e.g. model-free vs. model-based or on-policy vs. off-policy classes. Depending on the configurations of the state and action space, this results in different performance and computational overhead. If you also use transfer learning, the complexity of the decision becomes even greater.

- Are there differences between the environments? This is mainly related to the application domain. In the area of stock trading, for example, there are several ways to model an environment, starting with classic buy and sell actions and ending with options trading, where a contract expiration can be chosen for each action.

All of these combinations of factors and algorithmic decisions are typically resolved through a trial-and-error process, where the evaluation process focuses on simulations and obtaining feedback on various metrics and objectives. There are several techniques that can be used to add transfer learning layers to RL agents after answering the above questions:

- Reward Shaping (RS). Used for adjusting the rewards distribution or function with the objective of aiding the agent towards learning its policy.
- Learning from an Expert. More known as Imitation Learning, but in the case of RL this is focused on extracting segments of the experience replay buffer from the expert agent (teacher) and transferring it to the target (student).
- Policy Transfer. There are two main methods in this segment: *policy distillation* (Czarnecki et al., 2019), and *policy reuse* (Zhou et al., 2019), the latter performing better for the purpose of this paper.
- Actions and Tasks Mapping. A method of transfer learning that groups actions at a hierarchical level (Qi et al., 2021).

4 RL4FIN FRAMEWORK

We build the proposed framework and research objectives of this paper around four subthemes, which are addressed in the following subsections.

4.1 Data Acquisition and Processing

Collecting data for financial projects proved to be a challenge. We had to consider three important issues and tradeoffs in making our decisions:

- **Quantity** - AI generally performs better when it can learn from larger data sets. Even in the case of RL agents, where the agent can learn on its own, it is important to have a real and large dataset available to start learning. In our case, generating synthetic data was not an option.
- **Quality** - Several data sources proved to be inaccurate, making them unsuitable for a research project focused on trading real assets. Without rich and qualitative data and mechanisms to detect outliers, extraction of technical indicators would have been impossible.
- **Cost** - The cost of obtaining data in the financial industry is determined by the following factors:
  - real-time data is difficult to acquire without high costs.
  - large amounts of data at small time intervals can only be sourced over short periods of time without a significant increase in cost.
  - access to trusted data sources, such as Nasdaq Data Link [1], can be obtained at a high financial cost, but with large gains as one is later able to develop high-end algorithms.

Considering and weighing these factors, our main data provider in this work is AlphaVantage[2], a well-known source of financial data in the algorithmic trading industry. After obtaining an academic license for advanced indicators and advanced historical data, we developed a data pipeline shown in Figure1.

[1] https://data.nasdaq.com/tools/api
[2] https://www.alphavantage.co/

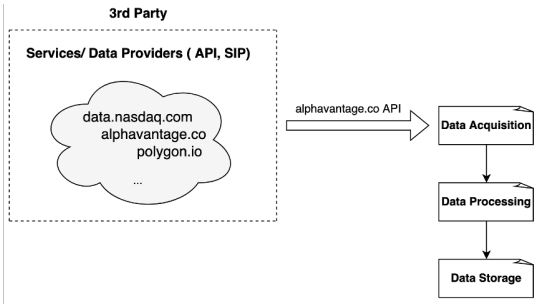

Figure 1: The design of data collection and processing is simple. The first step consists of multiple API calls to the AlphaVantage provider, followed by aggregation and processing. In the second step, multiple indicators are computed to enrich the information and prepare it for loading into the environment. Data storage is done using HDF5 (Folk et al., 2011) files to increase performance and reduce I/O processing time.

4.2 The Environment

An important goal of this project was to create a versatile environment that could be reused in both the academic and industrial communities and that could be easily extended to different types of trading, portfolios, and risk management. To provide separation between the implementation of the environment and the algorithms used to train the agents, the project uses the OpenAI Gym (Brockman et al., 2016) interface for environments. The advantage of this common interface is that open-source libraries of RL algorithms can be used with minimal effort to test different classes of algorithms and methods, even by users unfamiliar with low-level RL methods.

The customized environment implemented in the *RL4FIN* framework consists of several technical indicators extracted from stock market data:

- Base indicators: Indicators for open, close, high, low and volume data.
- Advanced Technical Indicators: Simple Moving Average (SMA), Exponential Moving Average (EMA), Weighted Moving Average (WMA), Relative Strength Index (RSI), Stochastic Oscillator (STOCH), Average Directional Movement Index (AVX), Bollinger Bands (BBANDS) and On Balance Volume (OBV) (Dongrey, 2022). These indicators were selected to provide diversity and enrich the dataset in several directions. In the field of machine learning methods, these composite indicators have strong similarities to the concepts of feature engineering or data enrichment.

Two environmental parameters specific to financial trading proved to be important for the optimization of RL agents in the context of this work.

- Trading Interval - Short-term trading strategies such as scalping and day trading are very different from the medium- or long-term variants of swing trading and position trading (Tino et al., 2001). This is an important consideration in the training and evaluation of the trading agent.
- Maximum number of steps per trading episode - In order to evaluate agents and compare them with other strategies and algorithms, the framework sets different length limits for episodes. The goal is to obtain agents that are able to make profits within a given time window.

Most work in the literature focuses on a simpler action scheme: *buy, hold*, and *sell*. The current version of the open-source framework follows this scheme and supports the same operations. However, these can be extended by the user, e.g. by adding more functions for specific areas such as trading futures, options or CFDs.

The states, or observations at any timestep t, $s_t \in S$ are composed of the following components:

- $balance_t \in \mathbb{R}_+$: Available balance.
- $base_t \in \mathbb{R}_+$: Base indicators (open, high, low, adjusted close, volume) for the asset traded.
- $MA_t \in \mathbb{R}_+$: Advanced technical indicators (SMA, EMA, WMA) for the asset traded.
- $RSIt \in \mathbb{R}_+$: Relative strength index (RSI) for the asset traded.
- $STOCHt \in \mathbb{R}_+$: Stochastic Oscillator (STOCH) for the asset traded.
- $AVXt \in \mathbb{R}_+$: Average Directional Movement Index (AVX) for the asset traded.
- $BBANDSt \in \mathbb{R}_+$: Bollinger Bands (BBANDS) for the asset traded.
- $OBVt \in \mathbb{R}_+$: On Balance Volume (OBV) for the asset traded.

The three actions are represented with numbers, i.e., 1 for long (buy), 0 for hold (no action), and -1 for short (sell). The entire asset is traded in a single step, which means that the goal is to increase the potential balance at the end of the simulation.

4.3 Training Trading Agents, Particularities and Observations

The principle is to train an agent capable of making decisions in such a way that, when placed on the market data, it will make the highest possible profit.

After testing several reinforcement learning algorithms based on action value functions $Q - values$

($Q(s,a)$ representing the estimated value of applying action a in state s), the algorithm based on Double Deep Q-Networks (DDQN) (van Hasselt et al., 2015) proved to be the best. The problem observed with its simpler version, i.e., regular DQN (Mnih et al., 2013), is the problem of overestimation, which has also been noted in the literature (Sutton and Barto, 2018a) in some use cases. In the financial trading domain, the overestimation led the agent to choose mainly the *long* action, which is actually a buy-and-hold strategy suitable for long-term portfolios. In the experiments, as explained in more detail in the Evaluation section, DQN achieved lower profits with trading intervals of 1 day and 200 trading steps (equivalent to one year in the market). Given these facts, it was decided to keep the current default implementation of DDQN as the basis for training the agent.

There are two main classes of hyperparameters used in the training process, as listed in Tables 2 and 3. The balance between exploiting the current agent policy and exploring the environment to discover new optimal states and actions is achieved in this work by applying an adaptive epsilon-greedy method (dos Santos Mignon and de Azevedo da Rocha, 2017). The trial-and-error process to determine the optimal values for these hyperparameters is performed by *Grid Search* (Liashchynskyi and Liashchynskyi, 2019) as a standard approach, i.e., dividing the allowed range of values into equal parts and performing experiments with each combination of subranges. Some of the parameters in Table 3 are presented in the form of ranges because the values in this range have similar effects on performance.

Figure 2 shows the interaction between the customized environment used by the *RL4FIN* framework and the agents customized for financial trading.

Table 1: ε-Greedy Parameters to balance exploitation vs exploration.

Parameter	Value
Epsilon start	1
Epsilon end	0.01
Maximum decay steps	250
Decay factor	0.99

4.4 Transfer Learning

The knowledge transfer of a trading agent trained on one financial market to another is evaluated in two ways: (a) without fine-tuning, by transferring the strategy directly, and (b) with fine-tuning, by adjusting the weights of the agent network in a training process of shorter duration using the new environment and data set. Fine-tuning is usually done by freezing the first part of the network and then retraining the last few layers (or even just the last layer) with a small sample from the new market dataset (Figure3). The intuition behind this is that the first part of the network typically learns generalities, while the end of the network generates details and specifics of the environments and datasets used for training. While the detailed results are presented in the next section, we outline the key observations below:

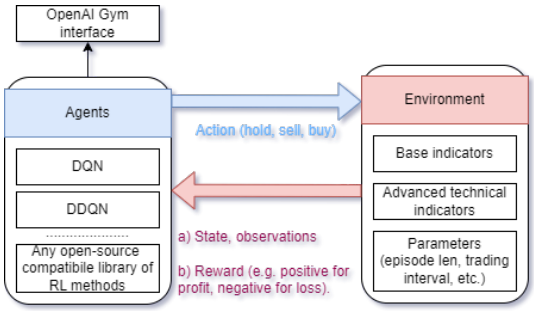

Figure 2: Overview of the interaction between the trading agent and the environment. The agents used by the framework use an interface compatible with OpenAI Gym, so most of the RL open-source libraries with algorithms can be reused. For demonstration purposes, we re-implemented customized DQN and DDQN algorithms as explained in 4.3. The environment contains a set of financial indicators that provide information about how the agent behaves in the financial market, as well as other parameters needed for the simulation that can be fully customized by the user.

Table 2: The main parameters used for training the trading agent with the DDQN method.

Parameter	Value
Discount factor	0.99
Target network update frequency	100 steps
Optimizer	Adam
L2 regularization factor	10^{-6}
Learning Rate	$\in [10^{-4}, 5 \times 10^{-4}]$
Batch Size	4096
Number of Episodes	$\in [750, 1000]$
Dropout Rate	$\in [0.3, 0.5]$

- Transferring between different markets with similar environments and rules: the agent that was directly transferred in this case made a profit that was even higher than the baseline of human traders.
- adjusting trading intervals, i.e., changing rules: as expected, changing the baseline interval from 1 day to other intervals (e.g., 60 minutes, 5 minutes, 1 second) did not produce profits or stable results.

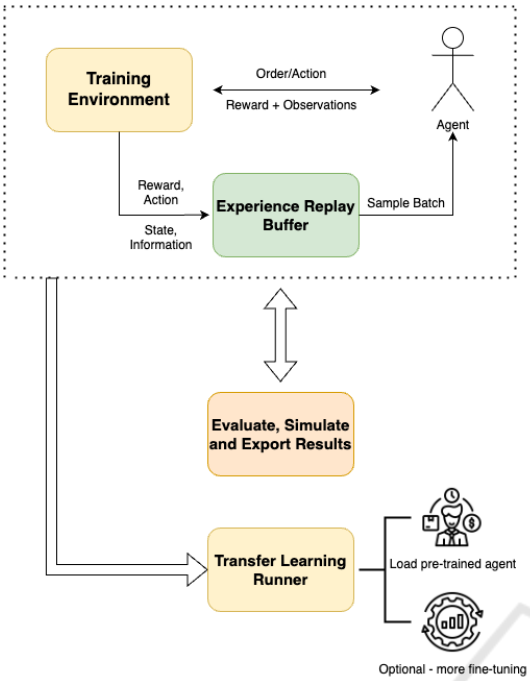

Figure 3: The flow from training the teacher agent in the marketplace and then transferring its knowledge representation to the (student) agent, with and without fine-tuning. The yellow blocks contain the fine-tuning part. The top layer contains the modules used for training the agents, such as an environment component, a base agent (based on the OpenAI Gym interface), data structures for storing and querying experience, evaluation, control, and common simulation functionalities. The lowest layer contains the functionalities to perform transfer learning.

4.5 Framework Development Methodology

Below, we briefly mention some important aspects of the methodology used to develop the framework.

- **Covering Software Engineering Principles** - Our research was driven by software development principles and best practices such as SOLID, Don't repeat yourself (DRY), testing and evaluation practices in machine learning (e.g., the 80/20% rule where the model does not see test data during training time), and test-driven development (TDD) (Martin, 2008).
- **Re-usability, Elasticity, and Modularity** - As described in Figure3, the developed framework was built with reusability in mind, a key aspect for future development and research. Components within the framework were developed with the separation of domains in mind, are highly modular, and are fully customizable by the user.
- **Proof of Concept** - Even though the framework was tested on datasets that were publicly available through an academic license, the project is designed as a backbone solution for applying reinforcement learning methods to trading systems in the general case, with the possibility of extending it to different types of environments and dataset specificities.

The production and experimental architecture is shown in Figure4. All agents are encapsulated in one component (center), as are the API of the clients that interact with the framework, and the implementations of data aggregation and live updating.

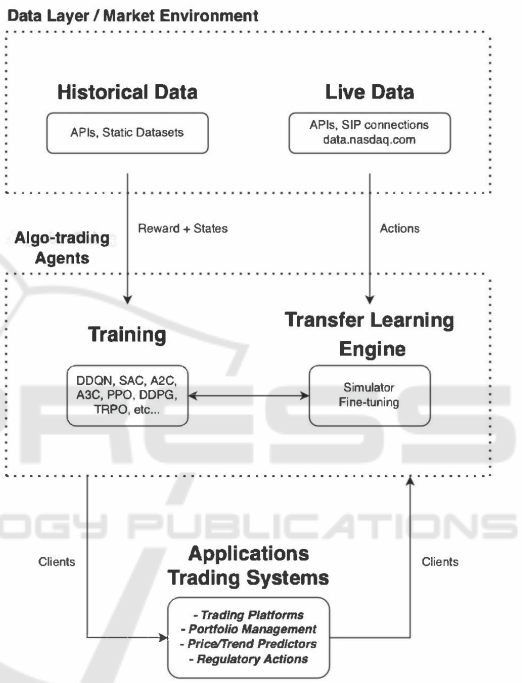

Figure 4: Production and experimentation ready architecture of the components of the framework.

5 EVALUATION

In this section, we first present the evaluation results of training RL trading agents on different datasets. Then we analyze how the trained agents perform on market datasets other than the trained ones, with or without changing the trading rules, with and without fine-tuning. All these experiments were conducted in our open-source framework. The results below show the agents that perform best with the DDQN method, i.e., with the best hyperparameters found with the Grid Search method.

The datasets used in this research are NASDAQ

100[3] (Web API-based access, 30 years of historical data) and AlphaVantage[4] (Web API-based access). The latter provides more than 60 economic indicators. Data and benchmarks for various industries were selected from these datasets. In particular, data for AAPL (Apple Inc.), QQQ (Invesco QQQ Trust - Exchange-Traded Fund), Nvidia (NVDA) and AMD were extracted to prove that algorithmic trading can be applied regardless of the industry segment. The length of an episode in our evaluation is usually 200 steps, where each step corresponds to one day, which means a whole year of trading in total.

Evaluations were performed on an Apple Mac M1 Max processor with 32 GB RAM and 10 cores CPU, demonstrating the feasibility of the framework to perform inference on consumer hardware.

5.1 Reinforcement Learning Agent Evaluation and Results

The *market* in our work case represents the real value recorded in the dataset of the asset over a year, at the end of each trading day (step).

We use two metrics as mentioned below.

- *Annual Returns* - expresses the increase in value of an asset over a period of time, in our case one year. The formula behind this ratio has three main parameters: the value at the end of the measured period (*EndVal*), the starting value of the asset (*StVal*), and the dividend yield (*Div*): $Return = \frac{(EndVal - StVal)Div}{StVal}$.

  The goal of typical trading agents is to generate positive returns and to outperform the market benchmark, meaning that the agent's decision would have improved the value of the asset held over the course of the year.

- *Agent Out-performance* - The purpose of this metric is to ensure consistent decisions in the benchmark. Specifically, it measures how many times the agent took the correct action (long, hold, short) during the year. For example, a bad decision would be if our agent makes a short decision, but in the market (the ground truth in the benchmark scenario) the value of the asset has increased, so the trading agent should have made a hold decision. The goal of our work is to outperform the benchmark in more than 50% of the time steps over an entire episode to achieve outperformance.

[3]https://data.nasdaq.com/tools/api
[4]https://www.alphavantage.co/

Figures 5 and 6 show the results of training sessions on the daily Apple (AAPL) dataset, an extensive dataset with over 20 years of continuous and qualitative information. Another successfully conducted experiment showed that agents with reinforcement learning running in a longer episodic time frame (1000 episodes) surpassed the market by more than tripling the profit achieved (from 10% to over 30% of annual revenue). The results are shown in Figure 7.

Figure 5: After various adjustments to the hyperparameters, and using the DDQN method, the best agent after empirical evaluation was trained with 500 episodes on the daily AAPL dataset and 200 episode steps.

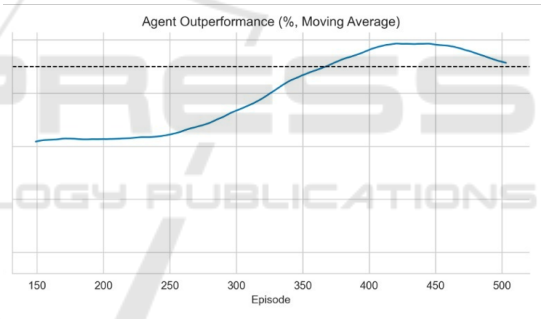

Figure 6: In the same experiment with the AAPL dataset, the agent performs better than the market benchmark.

Figure 7: Experiment was done on the Invesco QQQ ETF (ticker - QQQ).

Last but not least, we checked whether the agent correctly selected the actions long, hold, and short, adhering to the learned strategy. We tracked *diversity* of policies but also considered the risks associ-

ated with these actions. We would expect to see many hold/long actions and fewer short actions, as the latter are generally considered risky. Figure8 is representative of the distribution of actions across policies.

Annual return results are compared using two modern methods shown in Tables 3 and 5.

Table 3: Comparison against (Yang et al., 2021), evaluation with their A2C version.

	RL4FIN (our)	**The work in (Yang et al., 2021) (A2C algorithm comparison)**
Return averaged on all datasets	67.3%	60.0%

Table 4: Performance compared to FinRL on the same interval of time (2019).

Data used: 2019	**RL4FIN (our)**	**FinRL** (Liu et al., 2022a)
Return - AAPL	45.8%	36.88%
Return - QQQ	31.5%	27.51%

5.2 Knowledge Transfer Evaluation and Results

We first evaluate how a pre-trained agent performs in different markets without fine-tuning. This process involved simulations and evaluations of multiple datasets (other financial market tickers). During this step, only the environment was changed (trading rules and parameters) to allow an in-depth comparison of different trading strategies. These results and observations are shown in Table5.

Figure9 is representative of the applicability of transfer learning toward our research and shows that positive gains can only be made when the agent is transferred to another market or stock.

Fine-tuning was added to easily adapt the agent's policy to strategies that might apply in other scenarios. For example, more volatile areas of the stock market, such as the electric car area, require an agent that is able to allocate multiple actions toward short rather than long, marking gains that would be impossible to achieve with a simple buy-and-hold strategy. To improve the results, we applied the following scheme. First, load a pre-trained agent and freeze 50% of the shifts. Then fine-tune the agent on the new dataset for 200 episodes (one-fifth of the normal training scenario). Several hyperparameters were considered. The main ones found in our experiments were: the number of episodes, $\varepsilon - decay$, and $\varepsilon - greedy$.

Table 5: Transfer Learning without Fine-Tuning.

Environment Variations	**Observations**
No Changes	50-70% return over a time frame of 200 trading steps.
Decreased Trading Interval	After several experiments the conclusion extracted was the following: scalping and day-trading strategies require specially designed, direct knowledge transfer being faulty. Between -30% to 10% return over a time frame of 200 trading steps. Trading intervals reduced to 1 minute.
Increased Trading Interval	Position trading style isn't suitable without a specially trained agent. Between -10% to 20% return over a time frame of 200 trading steps. Trading intervals increased to 1 week.
Decreased Steps Num	Due to the agent's nature results were lower than baseline, but with a slight profit. The algorithm isn't designed to process sudden changes in the environment. Between 10% to 40% return over a time frame of 20 trading steps. The trading interval is fixed to 1 day.
Increased Steps Num	Due to the agent's nature results were lower than baseline, but with a slight profit. The algorithm isn't designed to process sudden changes in the environment. Between 20% to 30% return over a time frame of 500 trading steps. The trading interval is fixed to 1 day.

Table6 lists several variations and experiments to extend the hyperparameters to give an idea of how the results can vary with fine-tuning. The ideal ratio between the number of maximal episodes and the number of episodes devoted to random exploration is obtained from empirical evaluation (25%).

The profit achieved during the fine-tuning experiments increased and outperformed all benchmark scenarios on the subset selected for evaluation. Comparing the figures9 and10, one can observe the previously mentioned increase in annual return. The values were obtained after each agent was evaluated for 200 episodes after the fine-tuning step for each dataset.

The usefulness of transferring knowledge from one agent to another is also reflected in the average speedup achieved in training. For example, the time required to train an agent from scratch was reduced from nearly 10 hours to about 2 hours. Agents are able to train and adapt to a new market faster (even with customized trading rules) when starting from a pre-trained set of policies or networks. Memory requirements are also lower, as there is no need to store the entire collection of experience data from the train-

Figure 8: During the first 200 episodes, the agent is designed to explore the environment by choosing random actions, so the distribution for each action is close to 33%. As the training progresses, it slowly shifts to a more stable strategy, choosing mainly long actions as a safety net, while short actions are rarely chosen. This diagram can be used as evidence that our agent is learning and not sticking to a usual trading strategy like buy and hold.

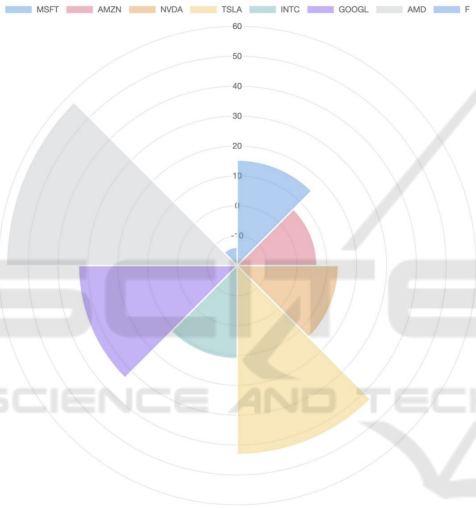

Figure 9: Transfer learning without fine-tuning. Positive gains were obtained for most of the benchmarks in the data sets. To understand the visualization, for example, in the case of AMD (shaded in gray), the agent generated a return of almost 60%, which outperforms the market baseline of 50% return. This is a clear case of the agent outperforming the market, even without fine-tuning. In contrast, for the Nvidia (NVDA) data (colored orange), the agent does not outperform the market base.

ing session, but a small portion of it is sufficient during the fine-tuning process.

All fine-tuned agents targeted higher annual returns directly and faster, proving that this solution is suitable for practice in the financial industry, Figure 11.

With the aforementioned setup, agent training took an average of 2-4 hours. The fine-tuning process took about 1 hour, and the inference time, i.e., the time required for the agent to make a single prediction for a moving window of previously observed data, averaged between 1.2 and 1.5 seconds.

Table 6: Transfer learning with fine-tuning experiments, comparison of the two main parameters involved: the number of episodes and ε-decay.

Num Episodes	Epsilon decay	Observations
100	25	Insufficient number of episodes.
100	50	Insufficient number of episodes. Too many exploratory episodes.
100	75	Insufficient number of episodes. Too many exploratory episodes.
200	**50**	**Best results over all experiments.**
200	100	Too many exploratory episodes, but with good results overall.
200	150	Too many exploratory episodes. Slow results.
400	**100**	**Best results over all experiments.**
400	200	Too many exploratory episodes, but with good results overall.
400	300	Too many exploratory episodes. Slow results.

6 CONCLUSIONS AND FUTURE WORK

This paper presents an open-source framework, *RL4FIN*, developed for trading systems that can be used for experiments in both academia and indus-

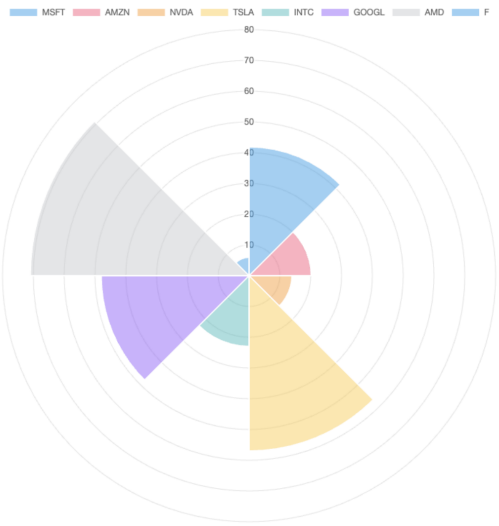

Figure 10: Transfer Learning with Fine-Tuning, with the same setup as Figure 9.

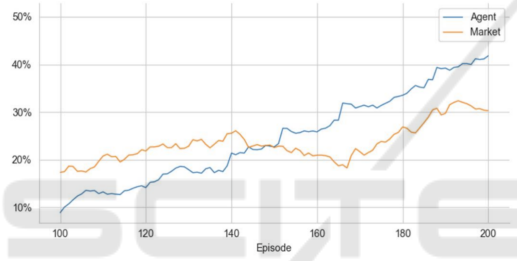

Figure 11: Transfer Learning with Fine-Tuning - Training behavior.

try. The proposed RL-based agents and methods outperform some of the state-of-the-art methods in public benchmarks, demonstrating their usefulness. Moreover, the framework provides transfer learning adaptation and an experimentable solution to transfer knowledge from one market to another. The work has shown that fine-tuning to the new datasets is necessary, especially when trading rules change.

Although financial applications can be formulated in multiple ways, the usability of our methods can be further developed to adapt to multiple scenarios in the future. Therefore, we plan to report in future work on the difficulties of developing a solution that can accommodate all trading or portfolio management strategies. Regarding the technical aspects, we plan to extend our methods by carefully investigating and implementing other RL methods, such as actor-critical mechanisms. We are also considering extending the framework to several types of trading in finance (e.g., futures, options, CFDs, buy-sell). As suggested in another study in (Bakker, 2001), LSTM networks can also enhance the capabilities of RL agents as an alternative to current knowledge embedding.

REFERENCES

Bakker, B. (2001). Reinforcement learning with long short-term memory. In Dietterich, T., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press.

Beck, N., Rajasekharan, A., and Tran, H. (2022). Transfer reinforcement learning for differing action spaces via q-network representations.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Czarnecki, W. M., Pascanu, R., Osindero, S., Jayakumar, S., Swirszcz, G., and Jaderberg, M. (2019). Distilling policy distillation. In Chaudhuri, K. and Sugiyama, M., editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 1331–1340. PMLR.

Dongrey, S. (2022). Study of market indicators used for technical analysis. *International Journal of Engineering and Management Research*, 12:64–83.

dos Santos Mignon, A. and de Azevedo da Rocha, R. L. (2017). An adaptive implementation of e-greedy in reinforcement learning. *Procedia Computer Science*, 109:1146–1151. 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal.

Folk, M., Heber, G., Koziol, Q., Pourmal, E., and Robinson, D. (2011). An overview of the hdf5 technology suite and its applications. pages 36–47.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks.

Kong, M. and So, J. (2023). Empirical analysis of automated stock trading using deep reinforcement learning. *Applied Sciences*, 13(1).

Koshiyama, A., Blumberg, S., Firoozye, N., Treleaven, P., and Flennerhag, S. (2021). Quantnet: transferring learning across trading strategies. *Quantitative Finance*, 22:1–20.

Li, Z., Liu, X.-Y., Zheng, J., Wang, Z., Walid, A., and Guo, J. (2021). FinRL-podracer. In *Proceedings of the Second ACM International Conference on AI in Finance*. ACM.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., and Stoica, I. (2018). RLlib: Abstractions for distributed reinforcement learning. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062. PMLR.

Liashchynskyi, P. and Liashchynskyi, P. (2019). Grid search, random search, genetic algorithm: A big comparison for nas.

Liu, X.-Y., Xia, Z., Rui, J., Gao, J., Yang, H., Zhu, M., Wang, C. D., Wang, Z., and Guo, J. (2022a). Finrl-

meta: Market environments and benchmarks for data-driven financial reinforcement learning.

Liu, X.-Y., Yang, H., Chen, Q., Zhang, R., Yang, L., Xiao, B., and Wang, C. D. (2022b). Finrl: A deep reinforcement learning library for automated stock trading in quantitative finance.

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, USA, 1 edition.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Qi, J., Zhou, Q., Lei, L., and Zheng, K. (2021). Federated reinforcement learning: techniques, applications, and open challenges. *Intelligence and Robotics*.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Sutton, R. S. and Barto, A. G. (2018a). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, 2nd edition edition.

Sutton, R. S. and Barto, A. G. (2018b). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685.

Tino, P., Schittenkopf, C., and Dorffner, G. (2001). Financial volatility trading using recurrent neural networks. *IEEE transactions on neural networks*, 12(4):865–874.

van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning.

Wang, Z., Huang, B., Tu, S., Zhang, K., and Xu, L. (2021). Deeptrader: A deep reinforcement learning approach for risk-return balanced portfolio management with market conditions embedding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(1):643–650.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning.

Yang, H., Liu, X.-Y., Zhong, S., and Walid, A. (2021). Deep reinforcement learning for automated stock trading: An ensemble strategy. In *Proceedings of the First ACM International Conference on AI in Finance*, ICAIF '20, New York, NY, USA. Association for Computing Machinery.

Zhou, W., Yu, Y., Chen, Y., Guan, K., Lv, T., Fan, C., and Zhou, Z.-H. (2019). Reinforcement learning experience reuse with policy residual representation. pages 4447–4453.

Zhu, Z., Lin, K., Jain, A. K., and Zhou, J. (2022). Transfer learning in deep reinforcement learning: A survey.