UNIVERSAL WISER PUBLISHER

Research Article

Multi-Timeframe Algorithmic Trading Bots Using Thick Data Heuristics with Deep Reinforcement Learning

Gregory Roy[*], **Jinan Fiaidhi**, **Sabah Mohammed**

Department of Computer Science, Lakehead University, Thunder Bay, Ontario, P7B 5E1, Canada
E-mail: gjroy@lakeheadu.ca

Received: 21 July 2022; **Revised:** 26 October 2022; **Accepted:** 27 October 2022

Abstract: This article presents an augmented Artificial Intelligence (AI) algorithmic trading approach that combines Thick Data Heuristic (TDH), with Deep Reinforcement Learning (DRL), to successfully learn trading execution timing policies. Combining the augmented AI human trader's intuition and heuristics with DRL techniques to provide more focused drivers for trading order execution timing is explored in this study. In this research, the goal is to solve the sequential decision-making problem of AI for profitable day and swing trading order timing executions. Enabling trading bots with cognitive intelligence and common-sense heuristics will offer traders including automatic traders an insight to understand the day-to-day swing trading timeframes indicators and arrive at mature trading decision-making. This article examines the performance of bots with Nasdaq and NYSE stocks that have a strong catalyst (info. which increases directional momentum) to find that they outperform benchmark algorithmic trading approaches. The research illustrates to the reader how to combine TDH and Deep Q-networks (DQN) into a TDH-DQN augmented AI trading bot. The bot learns through test data to predict the optimal timing of order executions autonomously on idealized trading time series data.

***Keywords*:** augmented AI, single stock trading, trading strategy, deep Q-network, multi-timeframe, thick data analysis

Abbreviations

AC	Actor-Critic
ACER	Actor-Critic with Experience Replay
ATR	Average True Range
ANN	Artificial Neural Network
APR	Accumulated Percent Returns
API	Application Programming Interface
AI	Artificial Intelligence
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
Bot	Robot
CPR	Compound Percent Returns (same as APR)

CNN	Convolutional Neural Networks
DRL	Deep Reinforcement Learning
DL	Deep Learning
DQN	Deep Q-Network
DDPG	Deep Deterministic Policy Gradient
EMA	Exponential Moving Average
GDQN	Gated Deep Q-learning trading strategy
GDPG	Gated Deterministic Policy Gradient trading strategy
HFT	High-Frequency Trading
IB	Interactive Brokers
JEPA	Joint Embedded Predictive Architecture
LOB	Limit Order Book
MACD	Moving average convergence divergence
ML	Machine Learning
MSE	Mean Squared Error
Nasdaq	National Association of Securities Dealers Automated Quotations
NYSE	New York Stock Exchange
NLP	Natural Language Processing
PER	Prioritized Experience Replay
PnL	Profit and Loss
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
Relu	Rectified linear unit function
TDQN	Trading Deep Q-Network algorithm
TD3	Twin-Delayed DDPG
TD	Temporal Difference
TDH	Thick Data Heuristics
TRPO	Trust region policy optimization
TWS	Trader Workstation
LSTM	Long Short-term memory
QL	Q-Learning

List of symbols

Learning rate	α
Set of weights	θ
Discount Factor	γ
Epsilon greedy policy	ϵ
Q-Function	Q
State	S
States	$s \in S$
Action	A
Actions	$a \in A$
Reward	R
Rewards	$r \in R$
Optimal strategy or policy	π
Time window size	τ

1. Introduction

In this article, the TDH is a combination of human and bot-shared data management and decision-making, as shown in Table 1. Combining DRL with TDH, and intuition, a research question to answer in this study is whether DRL intraday stock trading bots and swing trading bots can be employed successfully to execute profitable trades in the NASDAQ and NYSE stock markets, on a single stock. Also, to explore how adding TDH (like common sense reasoning, risk management, stock selection, multi-timeframe analysis, training/testing data selection, timeframe selection, hyperparameter tuning, and model creation), affects the performance of the trading bots. The TDH helps add subject matter experts' experience into the decision-making process.

Table 1. TDH agent responsibilities for the augmented AI trading bots

Thick data heuristic	Agent
Multi-timeframe technical analysis	bot
Patterns	bot
Volume	bot
Price action	bot
Price levels	bot
Tape-reading	bot
Fibonacci extension and retraction levels	human
Volume profile	human
Market structure	human
News	human
Sentiment	human
Relative performance	bot
Candlestick analysis	human
Fundamentals	human
Momentum	bot
Message boards and chat-rooms	human
Twitter	human
Asian and European markets	human
Index and commodity futures	human
Hyperparameter tuning	human
Market-wide high-volume rate scanner analysis	bot
Volatility	bot
Top volume rate market-wide scanner	bot
Risk management	bot

The research goals for this research are to explore the performance of algorithmic trading bots, to see how the bots perform for day trading using intraday price movement and swing trading with weekly price bars going back ten years. In this research, we use Q-Learning (QL), where the goal is to find the optimal Q-value function for the output state (buy-sell-hold) with an Artificial Neural Network (ANN) in the stock market environment with iterative updates based on the Bellman equation.

Thick data combines qualitative and quantitative data. Fiaidhi proposes thick data is a concept that employs heuristics and qualitative data (like observations, feelings, reactions, and conversation outcomes) to provide more in-depth insights and reveals hidden patterns that can be missed with quantitative techniques. She argues it has a significant impact on Data Analytics and Pattern Recognition in modern conversational and explainable AI. Fiaidhi goes on to argue thick data analytics aims at discovering the added-value heuristics to answer focused questions that can be missed by quantitative analytic techniques including ML and DL [1]. Therefore, we decided to research and compare the effects of adding TDH to stock trading.

This research is the first to combine the qualitative and quantitative elements of TDH concepts in the area of Fintech or stock market trading research we are aware of. To enhance the current research attempts, we extend existing research by:

1. Combining TDH with DRL to compare the performance of different RL algorithms.
2. Comparing the performance of different DRL algorithms over different timeframes.
3. Determining optimal TDH-DQN trading bot parameters and settings through iterative training and testing on an idealized dataset and different market environments.

The organization of this manuscript is as follows. Section 2 highlights some of the previous work and background related to the sequential decision-making problem of AI for profitable day and swing trading order timing executions. Our methodology and the steps of organizing and building the framework are explained in Section 3. Section 4 illustrates the shortcoming, and benefits of the different algorithms, through examples from real word results. Section 5 has the conclusion and future research work.

2. Background and related works

Thick Data combines Qualitative and Quantitative Data. When companies want to build stronger ties with investors, they need stories. Stories contain emotions, something that no scrubbed and normalized dataset can ever deliver. Numbers alone do not respond to the emotions of everyday life: trust, vulnerability, fear, greed, lust, security, love, and intimacy. It's hard to algorithmically represent the strength of an individual's service/product affiliation and how the meaning of the affiliation changes over time. Thick Data approaches reach deep into market participants' hearts. Ultimately, the relationship between a market participant and an instrument/brand is emotional, not rational.

The authors need to differentiate between qualitative and quantitative analytics before going further as the differentiation is not always obvious. For example, during Text Analytics, measuring the frequency of certain words would be considered quantitative analytics, whereas exploring the contextual meaning of a conversation would be considered qualitative analytics. In other words, qualitative analytics includes the analysis of context, human behavior, emotions, and other factors that are hard to digitize without losing any meaning. Qualitative analytics is a great approach to bridge the gap between insights provided by quantitative research and providing an in-depth understanding of the underlying reasons and motivations for a given phenomenon situation. Qualitative analytics is not an added patching analytics because it will not simply add more data points to adjust inaccurate prediction algorithm outputs. No output will be able to predict human behavior until inputs are as complex, unexpected, and sometimes as contradictory as humans themselves. This is where the notion of Thick Data came to the surface.

Thick data takes individual market participants' temperatures more precisely and offers depth analytics to the market participant's data story. Thick data differs from big data by its qualitative approach, obtaining ethnographic data that allow contexts and emotions of the analyzed subjects to be revealed, while big data requires an algorithmic process usually carried out by statesmen and data scientists. The problem is that while big data is big, it can also be thin in producing effective analytics. For big data to be analyzable, it must normalize, standardize, and define certain parameters and assumptions to sort, organize, and disseminate information. While big data relies on ML, thick data relies on the social context of connections between data points. This research allowed Wang [2], to enrich big data with insights into what drives people, not just as market participants, but as human beings.

Latzko et al. [3] proposed that thickening the data is supplementing the data with richly textured information, or in other words, adding layers of thickness to it. One could see thick data as onion structured. It is "coated" with several layers of rich metadata - in the literal sense of data on data. For example, if the ground were made of superposed layers of matter, each layer has its individuality, but it interacts with surrounding layers, forming an organic whole. Instead of points, thick data are whole little structured worlds. Added at different times through the research process, multiple layers, of description, historical and social context, and cultural meaning contribute to data thickness, but each layer is itself "thick" in that it is textured in complex ways and does not easily lend itself to separation into discrete, computable elements. Hence, the notion of stickiness and the comparison with spaghetti.

Day trading is based on buying and then selling catalyst stocks within a single day, and sometimes within minutes or seconds. A catalyst stock has recent news information about it which increases its directional momentum. The goal with day trading is to trade catalyst stocks that same day and not to keep any position overnight. Keeping stocks overnight is swing trading, which is a completely different style of trading, with its strategies, stock selection criteria, tools, and timeframes. One of the key differences between day trading and swing trading is the approach to stock picking. It's important to not swing trade and day trade the same stocks as they have different selection criteria. Swing traders usually look for stocks in solid companies that they know won't lose their whole value overnight. For day trading, however, bots can trade anything, including companies that are going bankrupt soon, because you don't care what happens after the market closes. Many of the companies that are commonly day traded are too risky to hold overnight because they can gap in price against your position unexpectedly [4].

In day trading, our bots are competing with some of the sharpest minds in the world. The market is a massive crowd of traders and bots, with each bot trying to take money from the others by outsmarting them. It's a challenging and intellectually intense task for human traders to manage. For this reason, we explore augmented AI trading bots to allow humans to do what they do best which is namely, attaching context and meaning to the data qualitatively while allowing the bot to analyze the quantitative aspects of the data quicker than a human ever could.

Table 1 shows the TDH used for the analysis. It can be noted due to the complexity of software systems development, not all the TDH were fully automated with software code. As indicated in Table 1, some of the TDH is performed by an experienced human trader/developer, and some are performed by the algorithmic trading bot. Finally, we analyze the bot's ability to learn in a case study during Part 3 of the research.

Trading bots that rely solely on DRL, lack transparency, and an expert trader's subjective intuition and capacity for consciousness. The DRL algorithmic stock trading bots provided the required TDH which showed profitable results for both short-term day trading and longer-term swing trading. The bots should all be considered unique to their environment and time with day trading and swing trading bots using different ingredients for their TDH. Even between different day trading bots, the decisions will be the same, but the parameters vary between the different bots depending on which stock they are trading. TDH should be created and tuned by an experienced augmented AI human. This is the case until AI software systems can more closely replicate human capacity for consciousness, intuition, memory, and decision-making heuristics. Our research reported some gains in expected returns and presented an alternative to traditional fund and wealth management.

RL is ideal for trading due to its sequential decision-making nature. Adding the trading expert heuristics to ML paves the way for designing new Robo-advisors, trading agents, or trading bots capable of replacing human traders. In this article, we refer to Robo-advisors, bots, and agents interchangeably. Bots have many obstacles in place to achieve true AI where they can entirely replace a skilled human trader. The bots would have to have the following computer process in place, mental states, intentions, interpretations, emotional states, semantic skills, consciousness, self-awareness, or flexible intelligence. Although digital bots lack these skills, they can do more and more things better than humans, by processing increasing amounts of data and improving their performance by analyzing their output as input for the next operations.

Augmented intelligence is a subsection of ML developed to enhance human intelligence rather than operate independently of or outright replace it. It's designed to do so by improving human decision-making and the actions taken in response to improved decisions. For this semi-autonomous augmented AI research, we are utilizing an augmented trader system where the human trader manually selects the best high-volume stock for the bot to trade. The human trader is looking for stocks with a catalyst that other traders, bots, and competing market participants will be watching. Stocks with lots of interest in the pre-market session with higher highs and lows and flowing price action with tight bid/ask spreads and price bars that interconnect without any large gaps.

Robo-advisors and trading bots utilize large historical datasets including, time-series, Limit Order Book (LOB), volume, fundamental, news sentiment analysis, market research, insider information, Machine Learning (ML), technical analysis, and simulated risk management. However, these ML trading techniques alone have limitations. Algorithmic trading bots must be highly efficient at data management. They must first observe the market to discover profitable entry and exit trading strategies. Next, they must design or modify an existing trading strategy and finally implement the strategy in code.

After an event occurs in an episode, the bot must perform data processing and data science. These include collecting, validating, modifying, organizing, indexing, classifying, filtering, updating, sorting, storing, networking, distributing, accessing, retrieving, and transmitting. Finally, the bot must take actions such as monitoring, modeling, analyzing, planning, forecasting, decision-making, and learning. It's a lot of computation and data processing for the most intelligent human or bot to handle. Nowadays to compete with the fastest bots operating at the sub-second level, most of the decision-making must be automated on smaller timeframes.

The auction market or stock market is an ideal environment to test 15 RL algorithms to determine which ones perform the best in multiple environments. A near-unlimited supply of time-series datasets is reality available, so researchers can gain a good understanding of how the different AI algorithms perform over different timeframes. There is no need for data curation, cleaning, or modification. Basically, every timeframe is available from microseconds to decades of perfectly organized time-series data. The primary limitation of stock market research is that the environment is very limited in that the only primary inputs easily available are price, volume, and news. All technical indicators are derivatives of price or volume.

Big Data analytics combined with High-Frequency Trading (HFT) has made it possible to apply ML and DL methods to financial markets. High-frequency smart agents utilize low latency, high-speed network connections to the financial markets data servers, and accurate real-time market data to make trades in timescales of seconds down to microseconds. Liquidity for algorithmic trading bots is defined as the quantity of assets available for trade. Automated trading bot policies around pricing and risk management depend on their objectives and preferences, the policies of competing market participants, the overall market environment, and trade flow from investors. Trading agents should have policies to vary reward formulations and adjust these policies to market conditions and different competitors.

There are different categories of trading styles based on time-frame holding time. Figure 1 shows the Latency vs Position holding time for HFT, day and swing trading, and long-term investing. Machine learning is driving algorithmic trading, which is faster than traditional long-term investing strategies and more deliberate than HFT in stock markets [5].

Trained trading models that perform well out of sample are the core requirement for effective DRL trading bots. DRL gives computers the ability to learn from their experiences and improve their performance as they gain more experience. The trading bots create models to uncover the relationships between the price inputs and the trading decisions outputs when given historical price data as input. Next, the bots forecast outcomes out of the sample on the test data. If the results are satisfactory the bots can be deployed into the markets with the trained models to execute live trades. This feature makes DRL particularly attractive as an underlying approach to building algorithmic trading bots.

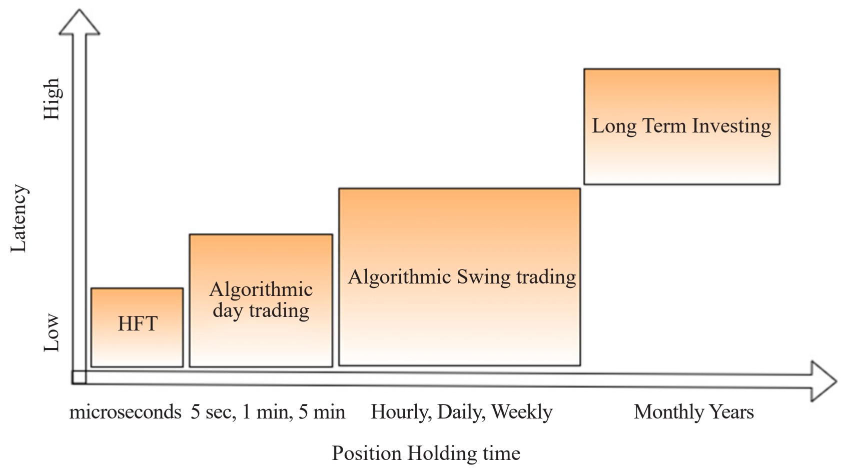

Figure 1. Latency vs position holding time for day, and swing trading

An autonomous entity is the ultimate achievement of AI. The flow of Reinforcement Learning (RL) in stock trading works as follows: The environment or stock market communicates the reward and state, (current price) to the agent. The agent decides based on the policy and new reward. The agent then takes action to Bid and Ask for quotes on the environment. The object or close price moves from one state to another.

The RL cycle works in an interconnected manner. The agent is not told which actions to take, as with other forms of ML, but instead must discover which actions have given the maximum reward by trying the actions. This environment is not as complex as others because individual stock trading actions don't affect the next situation and all subsequent rewards. An individual trader has minimal effect on the price.

Algorithmic trading primarily involves the automation of trading tasks. There are a lot of trading tasks that fall within this category such as risk management, technical analysis, and general hard-coded trading rules. Here the human programs the computer to take care of all the actions that don't vary between trading. These are the decisions outside the domain of intuition, where the bot performs actions based on fixed sets of rules. Hard-coded systems fall apart when the market regime changes and the trend changes direction or rate of momentum. Strict algorithmic trading systems often experience sharp drawdowns during these periods, and it takes an experienced trader's intuition to decide to scale back or reverse a strategy that has been working recently or over a larger timeframe.

Software systems that can imitate and model a human trader's emotions aren't currently available. Until researchers can model emotions in trading bots, it will always be challenging to completely model the intuition part of trading. This is especially true at the smaller day trading timeframes where decisions must be made lightning fast.

When used in combination with DL, RL has yielded some of the most prominent successes in ML, such as self-driving cars, and exceeding human performance at games [6, 7]. DRL systems can exceed human performance at many tasks but it's not without fault. Modeling a trader's decision-making heuristics and intuition with computer systems is a difficult task. Only a flexible learning system such as DRL can analyze time-series data for pattern recognition and generalization like consistently profitable human traders. Traders adapt and last for years in the market because they can adjust to cyclical bull and bear markets.

The DRL trading bots must also cope with the trader's intuitive issue of deciding whether to try new actions that may not be immediately optimal but longer-term could deliver the maximum reward. Alternatively, the bot must decide if it should try new actions to find another optimal path to rewards. The bot must also determine which step in the path was critical to the optimal policy. Developing AI algorithmic stock trading bots is challenging and market intuition, thick data analysis, and a good knowledge of the different models available are essential items to consider when developing DRL algorithmic stock trading bots.

The interactive and online nature of DRL makes it particularly well-suited to the trading domain. DRL models goal-directed learning by an agent that interacts with a typically stochastic environment that the agent has incomplete information about. DRL aims to automate how the agent makes decisions to achieve a long-term objective by learning the value of states and actions from a reward signal. The goal is to derive a policy that encodes behavioral rules and maps states to actions.

DRL is considered most like human learning that results from taking actions in the real world and observing the consequences. It differs from supervised learning because it optimizes the agent's behavior one trial-and-error experience at a time based on a scalar reward signal, rather than by generalizing from correctly labeled, representative samples of the target concept. Moreover, RL does not stop at making predictions. Instead, it takes an end-to-end perspective on goal-oriented decision-making by including actions and their consequences.

Cartea et al. [8] propose the stock market is a complex environment. Their actions are simple, with only 3 outputs, buy, sell, or wait, but the inputs are very complex. The primary issue is incomplete information about the environment. It's impossible to know the motivations and current state of other traders. Just like human expert traders, the more information the bots have about the current state of the market the more successful they are. The design of trading bot algorithms requires sophisticated mathematical models, a solid analysis of auction data, and a deep understanding of how markets and exchanges function.

It is generally accepted that the markets are efficient but what are they exactly efficient at doing? The markets are auction facilitators, so they are efficient at seeking the balance between buyers and sellers. The markets are always seeking liquidity. If there are a bunch of orders waiting to be filled at certain price levels the market will find those orders and fill that liquidity. The combined strength of the buyers vs the sellers pushes the price to levels based on the sentiment of the market participants.

Jammalamadaka et al. [9] argue that news articles serve the purpose of spreading information about the companies, and further influence people either consciously or unconsciously in their decision-making process while market making in the auction market. Positive news such as good earnings reports, improved corporate governance, new products, and acquisitions, as well as positive overall economic and political indicators, translate into buying motivations and increases in prices, while negative news will have the opposite effects.

Upon years of observation of real-time and historical data, it can be observed that market events and conditions tend to repeat themselves. This is likely due to the long-term memory of the market participants. These observations of repeating patterns can be approached from the perspective of Heuristics held within the domain of study called symbolic AI. A heuristic technique or heuristic is any approach to problem-solving or self-discovery that employs a practical method that is not guaranteed to be optimal, perfect, or rational, but is nevertheless sufficient for reaching an immediate, short-term goal or approximation [10]. The auction markets behave very much in this way at times with optimal, perfect, or rational conditions seemed to be non-existent. Value is measured by how much someone is willing to pay after you have already bought on the Ask or sold at the Bid.

The Role of the Day Trader as a market participant is important to consider when designing trading bots. Institutional traders are undoubtedly the determining forces in any significant price action. Accordingly, changes always take place when they perceive the current price to be too high or too low. To implement their plans, institutional traders inevitably need to enter the market temporarily at intraday levels. The role of a day trader is different. While an institutional trader follows a predetermined opinion and market direction, a day trader must first try to interpret it at the macro level. To determine the macro-level market structure the day trader must determine market direction, who is in control, and if there are any big buyers or sellers trying to disguise their intentions.

The next task of a day trader is to determine when exactly institutional traders become active at the micro-level. This is where a more detailed analysis at a smaller time level comes into play. These combine market and volume profiles, important chart points, and order flow. Only a combination of these components enables an algorithmic DRL day trading bot to execute a low-risk trade [11].

2.1 *Related work*

LeCun et al. [12] offer a good explanation of how DL works. LeCun proposes DL allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. Deep learning discovers intricate structures in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layers. In this research, we are using time series closing price data as the input variable. We are employing DL to solve the Q-table for the heuristic process-oriented RL trading agents.

Sutton and Barto in their seminal work [13] established the basis for a whole new field called RL used extensively throughout this work. The DRL algorithms created by DeepMind researchers from 2013-2017 [7, 14-17] inspired and influenced this work. The algorithms were developed to achieve better than human performance in Atari video games with the Rainbow Algorithm [6]. Schaul et al. [15] researched the experience replay concept which was utilized for the construction of the TDH-DQN trading bots for Parts 2 and 3 analysis.

Realistic simulated environments, where AI agents can be trained to learn a large repertoire of cognitive skills, were explored through the work of Wang et al. [18]. Actor-Critic with Experience Replay (ACER)'s features include using retrace Q-value estimation, Truncating the importance weights with bias correction, and applying efficient Trust Region Policy Optimization (TRPO) [19]. Also, according to Bellemare et al. [20], Every time an agent acts upon the environment, an expensive simulation step is conducted. Thus, to reduce the cost of simulation, the need to reduce the number of simulation steps (i.e., closing stock prices at pre-defined time-series intervals like five-second, one-minute, or daily samples of the stock environment). This need for sample efficiency is even more compelling when agents are deployed in the real world such as AI day trading algorithmic trading bots. The idea of learnable models has recently enjoyed a renewal of interest by Ha et al. [21] and Moerland et al. [22] for a recent survey of model-based RL [23].

Experience replay explored by Lin [24], has gained popularity in deep QL where it is often motivated as a technique for reducing sample correlation. Replay is a valuable tool for improving sample efficiency. The deterministic nature of the optimal policy limits its use in adversarial domains. Finding the greedy action with respect to the Q function is costly for large action spaces [18].

Based on this hybrid framework, Mnih et al. [25] proposed an asynchronous variant of the A2C method which surpasses the origin Actor-Critic Agent (AC), in convergence time and performance. Lillicrap et al. [26] presented the DDPG algorithm for solving problems with continuous action spaces [27].

Other ML approaches to deploying stock market trading bots use Long Short Term Memory (LSTM) neural networks to feed multiple data into the neural network to come up with the next prediction in the time series [28]. This is practical for longer-term swing trading bots, where the neural network has plenty of time to run computations and execute trades at most once per day, but for a real-time intraday system that learns and reacts to the environment, this would be too slow to compute compared to RL. RL also doesn't need as big of a dataset to make accurate predictions as strictly supervised or unsupervised learning or DL approaches.

Early DRL research by Mnih et al. [7] proposed a DQN model to learn long-term control policies with smattering prior knowledge. Research focuses on employing deep neural networks to learn the Q matrix. Hasselt et al. [16] proposed a Double DQN algorithm to reduce overestimation. Lillicrap et al. [26] presented an actor-critic, a model-free algorithm named deep deterministic policy gradient, to learn policies. Mnih et al. [25] DQN exploited the replay algorithm and successfully integrated RL with deep neural networks. This reduced the computation complexity and improved the performance of RL agents because of the rich representations provided by deep neural networks. Mnih et al. [25] put up a parallel RL paradigm, which asynchronously executed multiple agents on multiple instances of the environment in parallel, to deal with the non-stationary problem. Besides, the algorithm was compatible with various RL algorithms, including on-policy ones such as AC methods, Sarsa, n-step methods, and off-policy ones like QL. Wang et al. [17] brought forward the Dueling Network, which is a DQN-based method that divides the original network into an output scalar V(s) and an output action to the dominant value and integrates two Q values after operation respectively. Table 1 shows 9 related DRL works along with the algorithms they used, dataset, period, and time series interval information. The TDH-DQN proposed method is shown at the bottom of Table 2 for comparison.

Soon after the DeepMind researchers Atari DRL papers were released, researchers applied the techniques to algorithmic trading. DRL models are widely used to learn a good single-stock trading strategy for a given stock based on its historical data. Deng et al. [29] proposed a model of Deep Direct RL and added fuzzy learning. Du et al. [30] proposed QL, to optimize policies. Wang et al. [31] proposed employing deep QL to build an end-to-end deep Q-trading system. Kang et al. [32] modified and adapted the Asynchronous Advantage Actor-Critic (A3C) by Mnih et al. [25], A3C RL algorithm and joined it with DL.

Recently, Xiong et al. [33] employed the Deep Deterministic Policy Gradient (DDPG) technique to learn a dynamic stock trading strategy. Azhikodan et al. [34] proposed automating swing trading using deep RL. Li et al. [35] examined the performance of three variations of the Deep Q-network including typical DQN, Double DQN, and Dueling DQN in learning single stock trading strategies for ten US stocks. Jeong et al. [36] proposed a deep QL network to determine the number of shares used in prediction. Wu et al. [37] used a Gated Recurrent Unit (GRU) to extract temporal dependencies from raw financial data and technical indicators in combination with the DQN and Deterministic Policy Gradient (DPG) models to learn a trading strategy on single stocks. Lei et al. [38] proposed a time-driven feature-aware jointly deep RL model called TFJ and DRL. Yang et al. [39] proposed an ensemble strategy that employs deep reinforcement schemes to learn a stock trading strategy by maximizing investment return. Park et al. [40] proposed a novel portfolio trading strategy in which an intelligent agent is trained to identify an optimal trading action using deep Q-learning.

Hirchoua et al. [41] proposed a novel rule-based policy approach to train a deep RL agent for automated financial trading. Chakole et al. [42] used a QL algorithm to find the optimal trading strategy, in which the unsupervised learning method K-means and candlestick chart were, respectively, used to represent the state of the stock market. Carta et al. [43] proposed an ensemble of RL approaches that do not use annotations to learn, but rather learn how to maximize a return function over the training stage. Theate et al. [44] proposed a novel DRL trading policy to maximize the resulting Sharpe ratio performance indicator on a broad range of stock markets. For more general ML information, Kumbure et al. [45] reviews the literature and ML techniques and data for stock market forecasting. Millea [45] offers a critical survey of deep reinforcement for trading [46].

Table 2. DRL related work summary for single stock trading

Article	Algorithm	Dataset	Period	Time-Series interval for RL
[44]	TDQN	30 stocks	5 years	Daily
[37]	GDQN and GDPG	15 stocks	8 years	Daily
[39]	PPO, A2C, DDPG	30 stocks	7 years	Daily
[38]	TFJ, DRL	S & P 500	1999-2018	Daily
[40]	DQL	US: ETF, KOR: IDX	2010-2017	-
[34]	NN, RCNN	Nasdaq: GE, Nasdaq: GOOGL	-	-
[36]	DQL, DNN	S & P 500, KOSPI, Euro Stoxx 50, HSI	1987-2017	Daily
[47]	DQN, A3C, SDAEs, LSTM	US (AAPL, PG, IBM, ES, IF, S & P 500)	2008-2018	Minute, Daily
[39]	DRL, PPO	US (S & P 500, 5 stocks, Gold, BTC)	1960-2019	-
Proposed multi-timeframe method	TDH, DQN	Nasdaq, NYSE (7 stocks × 5 sec and 1 stock × 1 min, 10 stocks × weekly)	2012-2022	Part 1a: 5 sec, Part 1b: 1 min data, Part 2: Weekly

Taghian et al. [48] proposed a DRL model with feature extraction modules on the Dow Jones Index. Suhail et al. [49] proposed combining market sentiments and RL using Apple data from 2006-2016. They investigated adding the influence of market news on deciding stock prices. In our contribution to the overall research, we analyze the best DRL algorithms for multi-timeframe trading. We also add TDH to the DQN plus experience replay DRL algorithm initially developed by DeepMind [6-7, 14-15, 17] researchers to build multi-timeframe augmented AI trading bots. Our proposed TDH-DQN method is shown at the bottom of Table 1 for comparison to related works.

ML for trading is a large topic. Financial stock market trading is a vast area of research, but the related work research papers found to be most relevant were selected based on specific keywords related to technological advances and specifically ML and AI, and how they are affecting the financial sector.

In the article called, "RL for High-Frequency Market Making", Lim et al. present an early practical application of RL to optimal market making in high-frequency trading. They show that a discrete QL algorithm can use RL to outperform the market-making of traditional frameworks [50].

Ganesh et al. [51] at JPMorgan explore, "RL for Market Making in a Multi-agent Dealer Market". The researchers explored an RL-based market-making agent's interactions with a multi-agent simulation environment. The agent used the competitor's pricing policy, asymmetric price skewing, and maintaining a positive or negative inventory depending on the trend. Training agents in financial markets is difficult so they built A market-making simulator. Simulators provide more than just data for sample-intensive RL algorithms; they also provide a platform to conduct controlled experiments to test what is being "learned" by an agent, how a policy performs in different scenarios and the causality between changes in the environment and agent behavior. Simulators can also be used to train an agent in a diverse set of scenarios, leading to improved generalization and robustness to changes in the environment. The AI researchers showed that the RL agent can learn about its competitor's pricing policy. It also learns to manage inventory by skewing prices and maintaining inventory depending on whether the market price trend is up or down [51].

The article by Briola et al. [52] introduces the first end-to-end agent of its kind. Their agent uses a Proximal Policy

Optimization algorithm. The training is performed with high-frequency Limit Order Book data. The agents are always able to, "beat the market", and achieve a net positive profit in the whole training sample considered, and in most of the single trading days, it is composed of [52].

To explore the market maker mathematical formulas, we can explore the article, "High-frequency trading in a limit order book", by Avellaneda et al. [53], because it is commonly referenced for its model. The paper presents common market-making formula's clearly to come up with a solution for optimal bid and ask quotes. The model consists of the following components: The mid-price of the stock, the optimizing agent with the finite horizon, the optimizing agent with an infinite horizon, Limit orders, the trading intensity, optimal bid and ask quotes, approximations, and numerical simulations [53].

The article by Briola et al. [54] explains the LOB called "Deep Learning Modeling of the Limit Order Book: A Comparative Perspective". The Limit Order Book (LOB) represents the venue where buyers and sellers interact in an order-driven market. In the article, the AI researchers test Random models, Logistic Regressions, LSTMs, CNN-LSTMs, and MLPs on the same tasks, feature space, and dataset and performance metrics [54].

Convolutional Neural Networks (CNN) are widely used by AI researchers, an article that demonstrated this type of neural network was, "Forecasting Stock Prices from the Limit Order Book using Convolutional Neural Networks", by Tsantekidis et al. [55]. Most orders are performed in their entirety by electronic means so market makers can analyze all the generated data and detect repeated patterns of price movements. The AI researchers trained a CNN on high-frequency LOB data, applying a temporally aware normalization scheme on the volumes and prices of the LOB depth. The proposed approach was evaluated using different prediction horizons and it was demonstrated that it performs significantly better than other techniques, such as Linear SVMs and MLPs, when trying to predict short-term price movements [55]. To gain a better understanding of market microstructure readers can explore, the "High-frequency market microstructure", by O'Hara et al. [56]. It discusses how markets are different now, transformed by technology and high-frequency trading, and details HFT strategies.

The reader can explore DL and LSTM and CNN networks further with the following works. The article by Fischer et al. [57] called, "Deep learning with long short-term memory networks for financial market predictions" was informative. In this article, the researchers deployed an LSTM network for predicting out-of-sample directional movements for the constituent stocks of the S & P 500 from 1992 until 2015. They successfully demonstrated that an LSTM network can effectively extract meaningful information from noisy financial time series data. Compared to random forests, standard deep nets, and logistic regression, it is the method of choice with respect to predictive accuracy and with respect to daily returns after transaction costs. As it turns out, DL in the form of LSTM networks seems to constitute an advancement in this domain as well [57].

For a relatively stable dataset, DL models work very efficiently. A very accurate prediction is given by models like LSTM. According to Lim et al. [58], DL models work as an effective modeling technique for time series forecasting. In terms of execution speed, the univariate CNN model with the previous week's data as the input was found to be the fastest one according to Yong et al. [59]. That is because recently there's literature that points out that CNN can achieve what LSTM has been used for and is great at, namely predicting sequences, but in a much faster, more computationally efficient manner. Another model that started becoming popular recently is CNN. CNN also works better for classification problems and unlike Recurrent Neural Networks (RNN) based models, it is more suitable for either non-time varying or static data representations [60].

With the development of modern architectures such as convolutional neural networks CNNs and RNNs, DL models have been favored for their ability to build representations of a given dataset, capturing temporal dynamics and cross-sectional relationships in a purely data-driven manner. The adoption of deep neural networks has also been facilitated by powerful open-source frameworks such as TensorFlow and PyTorch, which use automatic differentiation to compute gradients for backpropagation without having to explicitly derive them in advance. In turn, this flexibility has allowed deep neural networks to go beyond standard classification and regression models. Focusing on the raw signal outputs, the Sharpe ratio-optimized LSTM outperforms all benchmarks as expected. Incorporating transaction costs, the Sharpe-optimized LSTM outperforms benchmarks of up to 2-3 bps of costs, demonstrating its suitability for trading more liquid assets [58].

Among DL structures Long Short-Term Memory (LSTM) networks, a representative type of RNN, are suitable for modeling temporal patterns, which are widely utilized in tasks regarding time series [59].

Ding et al. compared with the original LSTM, this model is greatly improved with high prediction accuracy and small regression error. Dropout refers to the temporary discarding of the neural network unit from the network according to a certain probability during the training of the DL network, which is a means to prevent over-fitting. The principle of dropout operation is that the neurons in each layer are randomly deleted with probability P in a training iteration, and the data in this iteration are trained with the network composed of the remaining $(1 - p) * N$ neurons, thus alleviating the over-fitting problem [61].

In the article by Borovkova et al. [60, 61], the researchers propose an ensemble of Long-Short-Term Memory (LSTM) neural networks for intraday stock predictions, using a large variety of technical analysis indicators as network inputs. The proposed model was found to perform better than the benchmark models or equally weighted ensembles. With the increased availability of high-frequency trade data and the development of ML algorithms that can handle such large amounts of data, technical analysis is currently undergoing a revival: Daily patterns are replaced by intraday ones, and algorithms, not humans, now learn price patterns and make forecasts based on them. Nelson, Pereira, and de Oliveira trained LSTM networks on 15-minute-interval observations, for several BOVESPA (Sao Paolo Stock Exchange) stocks, and reported accuracy metrics of 53-55% for the next direction price forecasts [60].

In the article by Sangyeon et al. [62], the researchers use an attention LSTM for prediction, and for visualizing intermediate outputs to analyze the reason for the prediction. Their proposed model produces a 0.76 hit ratio, which is superior to those of other methods for predicting the trends of the KOSPI 200.

In the article by Gabler et al. [63], the author states our main hypothesis is that CNNs provide a natural way to capture such patterns by simply consuming the historical data without an explicit definition of the patterns to be captured by the designer. The key results of the LSTM and CNN are shown and compared in terms of prediction accuracies, return risk characteristics on different long-short portfolio sizes, and the author unveils sources of long-term profitability. CNNs were developed after one basic idea: local connectivity. That means, that each node is only connected to a local region in the input. The authors claim to have successfully demonstrated that LSTM and CNN networks can extract meaningful information from such noisy financial time series. They say that the LSTM outperforms the CNN with respect to predictive accuracy and mean returns, although the CNN exhibits more favorable risk metrics.

Vinyals et al. [64] achieved successful results in StarCraft II using multi-agent RL. They chose to address the challenge of StarCraft using general-purpose learning methods that are in principle applicable to other complex domains. A multi-agent reinforcement learning algorithm that uses data from both human and agent games within a diverse league of continually adapting strategies and counterstrategies, each represented by deep neural networks. Their agent AlphaStar was trained via both supervised learning and reinforcement learning.

Zhao et al. [65] propose a Consciousness-Inspired Planning Agent for Model-Based Reinforcement Learning. They presented an end-to-end, model-based deep reinforcement learning agent which dynamically attends to relevant parts of its state during planning.

To construct autonomous intelligent agents, it's important to understand how AI trading bots would first learn to reason and plan as efficiently as human traders. These decision-making bots would need to learn representations of percepts and action plans at multiple levels of abstraction, enabling them to reason, predict, and plan at multiple time horizons. Current AI trends can be explored with Yann LeCun's position paper [23] where he provides a description of neural symbolic architecture using a hybrid neuro-symbolic strategy with functional hybrids.

This involves chain processing where preprocessing is done in one system, and post-processing is done in another. Sub-processing is another functional hybrid where parent processes are split into child processes. Meta-processing is processing tasks like monitoring, control, performance improvements, or corrections. The final functional hybrid strategy LeCun proposes is co-processing where neural and symbolic are equal partners. His model uses an approach like human intelligence where he models an actor using the perception of the environment and short-term memory plus a configurator to make decisions and take actions in a complex environment. The configurator is a key part of the model because it controls the interaction between perception, the actor, the world model, and the separate critic, cost, and intrinsic cost function.

LeCun's learning hierarchical Joint Embedded Predictive Architecture (JEPA) world models are composed in the following way. The configurator module takes inputs from all other modules and configures the tasks at hand. The perception module estimates the current state of the world. The world model module predicts possible future world states as a function of imagined action sequences proposed by the actor. The cost module computes a single scalar output called energy that measures the level of discomfort of the agent. The short-term memory module keeps track of the current and predicted world states and associated intrinsic costs. The actor module computes proposals for action sequences, represents states as vectors, and learns correlations between states *x* and *y*, but does not make predictions about *y* given *x* [23].

This type of hybrid architecture can use nonlinear problem-solving methods or different methods to solve problems. The hybrid architecture combines symbolic and ML approaches to find the best tool for the job and can employ cognitive architectures that mimic specialized brain functions such as Kenheman's system 1 and system 2 [66].

The research into AlphaGo by Silver et al. [67] provides another hybrid symbolic/RL architecture that uses Monte Carlo tree search to generate games. The approach requires an understanding of the rules of the game and its goals. The model can then calculate value networks using RL-like methods employed by the TDH-DQN bots in our research [67].

3. Materials and methods

In Part 1, Section 3.10, we determined the optimal DRL algorithm for this multi-timeframe TDH-DQN bot. In Part 1a, the goal was to determine which of the eight DRL algorithms tested performs the best at algorithmic trading in the five-second short-term day trading decision-making timeframe for seven catalyst stocks. Part 1b compares 15 different DRL architecture's performances on the catalyst stocks from Part 1a. both with and without TDH with a medium-day trading one min time-series timeframe. The Accumulated Percent Returns (APR), and computation time results from the Part 1 analysis caused us to choose the DQN as the optimal trading algorithm for our TDH-DQN bot purposes. Next, in Part 2, Section 3.11, our goal was to compare the performance of our TDH-DQN bot vs. some baseline algorithmic trading strategies including, buy and hold, Pivot Reversal, MACD LE, and Outside Bar. The analysis in Part 2 focuses on trading strategies with the swing trading weekly time series timeframe data for ten catalyst growth stocks. Finally, in Part 3, Section 3.13, our goal is to illustrate to the reader a case study that shows how our TDH-DQN trading bot learns to predict the optimal timing of executions autonomously on idealized trading time series data.

Table 3. Catalyst momentum stocks in Part 1a and 1b analysis for day trading bots, 5-second timeframe

Company	Exchange	Date	Bars (5 s, 1 D, 1 W)	Trend	Catalyst
Snap Inc.	NYSE	2021-07-23	1800, 53, 28	up	Earnings beat
Alibaba Group Holding Limited	NYSE	2021-10-04	1800, 53, 29	down	China News
NIO Inc.	NYSE	2021-10-04	1800, 53, 29	down	Analyst upgrade
Advanced Micro Devices, Inc.	Nasdaq	2021-10-13	1800, 53, 29	up	New product
Plug Power Inc.	Nasdaq	2021-10-13	1800, 53, 29	up	Hot sector
SoFi Technologies, Inc.	Nasdaq	2021-10-18	1800, 53, 19	up	Analyst upgrade
FuelCell Energy, Inc.	Nasdaq	2021-10-18	1800, 53, 29	up	Hot sector

This research was successful using a semi-autonomous augmented AI system (where the digital DRL bots augment an experienced human trader). Multi-timeframe technical analysis is a fundamental part of this research. The three primary time-series timeframes used are the five seconds, and one minute for the day trading bots and the weekly time interval for the swing trading bots. The environment for our trading bots was NYSE and Nasdaq stocks. Our day trading analysis used seven catalyst stocks, five-second, daily, and weekly timeframes, from 2021 as shown in Table 3. To illustrate the performance effects of the TDH, the one-minute time-series timeframe was selected for the seven catalyst stocks to compare the effects both with and without TDH in Table 4. The swing trading bot's environment was ten catalyst stocks weekly timeframe closing price data from March 25, 2022, going back ten years of historical data shown in Table 5.

Table 4. DRL algorithm APR comparison both with and without TDH applied

DRL parameters	SNAP TDH	SNAP	BABA TDH	BABA	NIO TDH	NIO	AMD TDH	AMD	PLUG TDH	PLUG	SOFI TDH	SOFI	FCEL TDH	FCEL
TDH applied	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
DQN	-1.04	1.03	0.59	-1.31	0.44	0.34	-0.1	0.69	-0.25	0.67	2.02	0.03	4.11	2.32
Duel DQN	-1.5	-4.36	0.14	0.32	0.73	-0.78	1.4	1.75	0.11	-0.57	2.4	2.36	4.34	-1.41
Recurrent DQN	-0.65	0	0.59	0	0.67	0	0.17	0	0.35	0	1.63	0	-1.52	0
Double DQN	-4.5	-0.85	0.58	-1.32	0.8	-3.63	2.1	1.62	2.4	0.81	3.08	1.56	6.09	3.84
Double recurrent DQN	-3	-1.78	0.24	-0.81	0.5	-1.66	0.91	0.76	0.97	2.44	2.47	1.56	3.79	3
Double duel DQN	-2.09	4.08	0.25	-0.38	0.53	-0.31	1.47	-0.01	-0.85	-1.59	2.05	0.8	4.1	3.86
Double duel recurrent DQN	-0.73	0	0.64	0	0.73	0	0.02	0	0.35	0	0	0	3.64	0
Curiosity DQN	-1.04	-2.51	0.64	-0.65	0.53	-1.41	1.64	1.46	0.55	-1.34	1.84	1.08	2.8	5.25
Recurrent curiosity DQN	-0.4	0.44	0.63	-0.08	0.8	-0.34	1.64	1.07	0.25	-0.6	1.53	1.08	3.79	4.95
Duel curiosity DQN	-3.35	1.59	0.35	-0.7	0.76	-0.8	0.37	1.42	1.23	0.91	2.12	-0.07	3.52	1.13
AC	-0.4	1.19	0.32	-0.56	0.86	-0.53	1.69	1.27	0.25	-0.46	1.77	2.5	3.96	2.26
Dual AC	-1.1	2.72	0.35	-0.22	0.59	-1.24	1.4	1.32	1.64	-0.93	1.81	1.49	3.97	2.93
Recurrent AC	-1.87	-1.43	0.24	-0.23	0.76	-0.97	0.91	0.19	-0.25	-2.33	1.49	2.16	3.14	-2.87
Duel recurrent AC	-0.73	0.24	0.16	-0.75	0.53	-2.1	1.51	1.24	1.53	1.32	1.18	1.95	2.96	2.86
Avg.	-1.60	0.03	0.41	-0.48	0.66	-0.96	1.08	0.91	0.59	-0.12	1.81	1.18	3.48	2.01
THD applied Average =	0.92													
No THD applied Average =	0.37													

Table 5. Catalyst growth stocks in Part 2 analysis for weekly swing trading bots

Stock	Company	Exchange	Start date	End date	Test bars	Train bars	Total bars
TSLA	Tesla, Inc.	Nasdaq	2012-03-30	2022-03-25	365	157	522
AAPL	Apple Inc.	Nasdaq	2012-03-30	2022-03-25	365	157	522
AMD	Advanced Micro Devices, Inc.	Nasdaq	2015-01-02	2022-03-25	264	114	378
NVDA	NVIDIA Corporation	Nasdaq	2012-03-30	2022-03-25	365	157	522
FB	Meta Platforms, Inc.	Nasdaq	2012-05-18	2022-03-25	360	155	515
MSFT	Microsoft Corporation	Nasdaq	2012-03-30	2022-03-25	365	157	522
AMZN	Amazon.com, Inc.	Nasdaq	2012-03-30	2022-03-25	365	157	522
QQQ	Invesco QQQ Trust	Nasdaq	2012-03-30	2022-03-25	365	157	522
PLUG	Plug Power Inc.	Nasdaq	2012-03-30	2022-03-25	365	157	522
AMC	Entertainment Holdings, Inc.	NYSE	2013-12-20	2022-03-25	302	130	432

3.1 *Research questions and objective*

This research was designed to address the problem of how to solve the sequential decision-making problem of trading formalized as an MDP optimization of order timing executions for a single catalyst stock. The research questions are:

Research question 1: Can RL agents successfully learn trading order execution timing policies?

Research question 2: Which DRL algorithm performs the best across multiple timeframes?

Research question 3: Does adding TDH improve RL agent performance?

In order to answer the research questions, we build our main research objective, which is to solve the sequential decision-making problem of AI for profitable day and swing trading order timing executions.

To satisfy the research objective, our detailed methodology will be addressed in Sections 3.10, 3.11, and 3.13. More details of the research methodology will be demonstrated in Figure 2 which shows the framework and the architecture of the TDH-DQN Trading Bot.

The environment for the experiments was comprised of the observation space including the state space, action space, and rewards. The state space consists of discrete multi-timeframe closing price time-series data and volume rate market scan data. The action space is bought, sell and hold, and finally, rewards (APR) as detailed in Figure 3, which shows the Trading Bot RL environment.

The bot's action space is discrete, and they only make decisions at the beginning of a new primary time-series interval, for example in this case the five-second, one-minute, or weekly time-series interval. The other time-series data is used to determine the trend with technical analysis but isn't used for deciding the decision-making interval. The human trader's actions are in the continuous action space in that the human can watch the tick-by-tick data visually on the charts as the time-series interval candles are forming in real-time. An astute trader can watch the volume rate and price action and make decisions to execute a trade at any time between time-series intervals whereas the bot cannot.

There are three types of RL-based techniques that are explored in this research, The first is a Value-based method where the agent first estimates the value of each action in each state and then selects the action with the highest value at each state called Deep QL or DQN. The second technique is the Policy-based method where the agent learns the policy function called Policy Gradient (PG). Finally, the third technique which is also a derivative of policy-based methods is Actor-critic method, in which the actor generates an action at each time step and the critic measures the quality of the generated action.

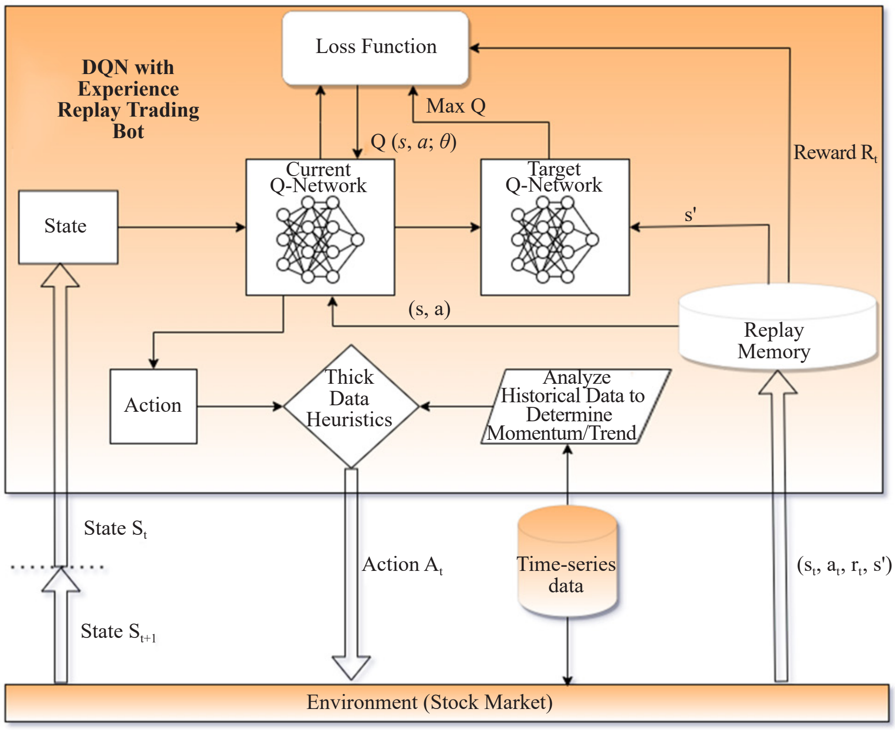

Figure 2. TDH-DQN trading bot architecture

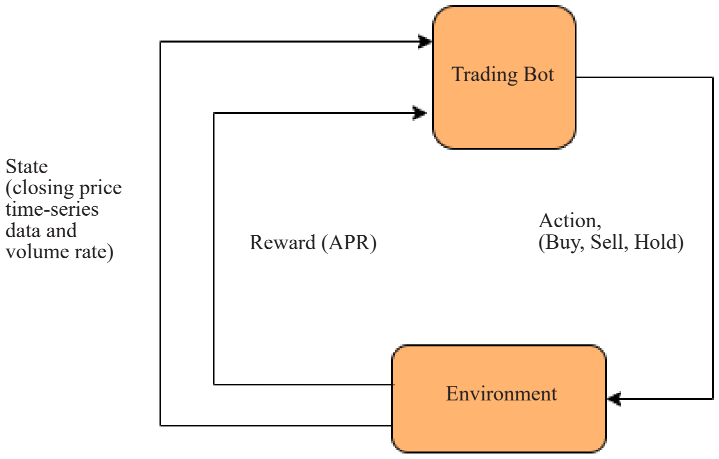
Trading Bot
State (closing price time-series data and volume rate)
Reward (APR)
Action, (Buy, Sell, Hold)
Environment

Figure 3. Trading Bot RL environment

The DQN uses a multi-layer convolutional network, Experience Replay, used to train the ANN to train itself using stored memories, and utilizes a second 'target' network, used to compute target Q-values.

In the RL framework for this case study, the algorithm takes an action (buy, sell, or hold) depending on the current state of the stock price. The algorithm is trained using a deep QL model to perform the best action.

To frame this RL problem, we must first define the agent as the trading agent who creates actions, buy, sell, or hold. The reward is the APR. The reward depends on the action: sell (realized profit and loss), buy (no reward), or hold (no reward).

The state space for the DRL agent is a sigmoid function of the differences in past stock prices for a given time window used as the state. State S_t is described as $(d_{t-\tau+1}, d_{t-1}, d_t)$, where $d_t = \text{sigmoid}(p_t - p_{t-1})$, p_t is the closing price at time t, and τ is the time window size. A sigmoid function converts the differences of past stock prices into a number between zero and one, which helps to normalize the values to probabilities and makes the state simpler to interpret.

The state determines the observations that the agent receives from the environment for taking a decision. The state should be representative of current market behavior as compared to the past and can also include values of any signals that are believed to be predictive or items related to market microstructure, such as volume traded, Exponential Moving Averages, Pivot Points, Candlestick Patterns, or MACD. Trading commission costs and trade execution slippage are not considered in this study.

3.2 *Bot's use of thick data heuristics*

Heuristics is an approach to problem-solving or self-discovery that employs a practical method that is not guaranteed to be optimal, perfect, or rational, but is nevertheless sufficient for reaching an immediate, short-term goal or approximation. Where finding an optimal solution is impossible or impractical, heuristic methods can be used to speed up the process of finding a satisfactory solution. Heuristics can be mental shortcuts that ease the cognitive load of sequential decision-making [10]. Human traders often run into issues with decision heuristics because as Kahneman explains humans struggle to think statistically. Kahneman uses heuristics to assert that thinking involves associating new information with existing patterns, or thoughts, rather than creating new patterns for each new experience [66]. This is where DRL trading bots have an edge. The bots are great at following the rules exactly but don't know how to use intuition to know when to slightly break the rules. Great traders know when to break the rules and size up their positions to capitalize on great opportunities quickly, but this is very difficult to do with AI algorithmic bots.

The TDH employs many elements and concepts from Symbolic AI to provide the qualitative and heuristic elements the bots need to be successful. Symbolic AI bots require an extensive amount of subject matter knowledge and expertise to effectively create all the rules involved in an optimal decision-making process. With symbolic models, you can put constraints on the moves the agent can make which can be more difficult when employing ML and RL approaches. Symbolic AI has advantages over ML and RL approaches in that introspection is more useful for coding, and easier to debug, explain, and control. Symbolic AI does not require big data and is more useful for explaining people's thoughts or abstract problems. Symbolic AI can learn generatively to a certain extent and is useful for inductive logic programming and case-based reasoning. Symbolic bots are non-monotonic so they can unlearn default logic and replace default belief with a new observation. Human programmers must provide goals to the symbolic bots but reasoning must be constrained and well-defined. Symbolic bots are best suited for AI problems with goals and rules which are solvable and explainable.

Whereas ML and RL approaches provide the advantages of being more robust against noise, better performance, less knowledge upfront, and are easier to scale up. Modern ML approaches do require big data and are more successful based on the type of data used. ML and RL approaches are more useful for connecting to neuroscience and better suited for perceptual problems. ML's equivalent to symbols is classification results in symbol types such as cat vs dog. ML bots learn relations between two or more things on an image for example orientation of the eyes and mouth to determine a person's face. ML bots don't have goals, only inputs, and cannot make their own goals. They use uninterpretable pattern matching without symbols and goals which can lead to meaningless results.

Consistently, profitable human traders often look at multiple data information sources when making trading decisions. By monitoring the items in Table 1, the trading bots must make decisions. Humans need to manage this information cycle plus their emotions and avoid recency bias that can cloud a trader's mind and attach a higher probability to recent trades [18].

ML algorithms cannot entirely replace human intuition and decision-making heuristics. Many Big Data concepts and methods may sound plausible and appealing but will not lead to viable trading strategies. Thick data analysis adds qualitative qualities to quantitative [69] qualities. Humans are good at assigning meaning to numbers but making meaningful qualitative assessments of data is challenging for AI systems. Long-term profitable traders use thick data analysis to form decision-making heuristics to keep up with the speed of the markets. Heuristics are mental shortcuts for problem-solving and judgments based on a probability that eases the cognitive load of making decisions. A heuristic means to find or to discover, and it's a basic part of our intelligence.

The TDH is different between the day and swing trading bots. Table 1 shows the TDH along with their agent. The

TDH is simpler for the swing trading bots. The bots purchase one share per execution and have one execution per week. They use stop loss and profit orders for exits based on a 1:8 risk-to-reward ratio for risk management. After numerous backtesting and optimization experiments, it was discovered adding too many heuristic rules tended to limit the total number of trades and reduce the overall profits. The swing trading bots are making their execution timing decisions via DRL. They ignore the market noise and trade strictly based on the long-term price action of the weekly time series data. This type of trading behavior can be observed in the Appendix A results, the agents primarily buy on pullbacks and breakouts and sell when rallies lose momentum.

The TDH considers technical/macroeconomics/sentiment indicators and adds the following components to gain insights into the qualitative properties of stock trades along with the quantitative indicators you mention. The added components beyond conventional technical analysis are hyper-parameter tuning and model optimization for the DRL agents. The DRL agents are making their trade execution decisions based on trading patterns in the price data. These trade execution decisions are then filtered based on the variables listed in Table 1. The raw DRL agents trade too often so they need to be filtered by the TDH. The art comes from qualitative and quantitative analysis of which trade rules or filters to add based on the model and data. There is a relationship between trading profits, APR, and the number of trading rules. There is a sweet spot between filtering and limiting trades based on rules and trading every opportunity the DRL agent identifies. Too many rules result in fewer but higher percentage winner trades and lower APR, but a lack of trading rules or heuristics results in many trades and lower APR. Limiting the DRL agent's trades based on proven trading heuristics that work over long periods of time with different stocks in different market environments results in fewer trades, with higher win percentages and APR rewards.

Professional traders that are consistently profitable in bull and bear markets use fundamental knowledge combined with technical indicators, the influence of various events, and financial news. Such information can be exploited to time trades with greater accuracy when combined with TDH. Most systems get confused by market events because they cannot take in all the parameters in the world outside the market. World events like when market volatility starts slowing down because of the election, weather, or shifting sector momentum. Humans can gauge market sentiment and what the competition is doing while these are difficult thought constructs for trading bots. Traders combine all these layered heuristics into an internal hierarchical decision tree in their minds. They combine this data with the ability to read the crowd based on experience into a sense of timing. This execution timing problem is the focus of this research.

The bots combine thick data analysis with heuristics to gain an edge in the markets. Thick data analysis combines quantitative and qualitative information together in a manner like what humans do. With thick data analysis, we are attempting to look at the numerical data with a sense of context. In the context of trading an example of thick data analysis would be calculating the 9-period Exponential Moving Average (EMA) and comparing the relationship between the current closing price and the 9-period EMA.

The 9-EMA measures the trend strength and velocity by the shape and slope of the curve. If the price moves above the EMA after a pullback a trader could create a heuristic rule to buy if these conditions are met. To take the example further one could calculate the 50-period EMA to measure the longer-term trend and then compare the relationship between the current price 50-EMA and 9-EMA one can use the concept of trend following and create another heuristic rule to sell when the current price drops below the 9-EMA but stays above the 50-EMA. TDH represents this concept of grouping together heuristic rules based on an expert trader's intuition and experience. For this research paper, we combined the following decision heuristics, risk management, technical analysis, trend-following strategies, momentum strategies, position-sizing, neural network hyper-parameter tuning, model design, environment design, data-curation, volume analysis, parameter backtesting, sector analysis, and news analysis to be able to identify the best catalyst stocks. The distribution of tasks between the human and bot agents is shown in Table 1.

3.3 *Deep Q-Network with experience replay algorithm*

Mnih et al. [14] developed the first standout DRL algorithm, the Deep Q-Network (DQN) algorithm, Hasselt et al. [16] presented a double-Q estimator for value-based RL methods to decrease the overestimated Q value, hence improve the agent's training performance. Wang et al. [17] improve the accuracy of Q value estimation by adopting two split networks, one for estimating state value and the other one for estimating its action value. In contrast to modifying networks' structure, Schaul et al. [15] investigated the Prioritized Experience Replay (PER) method to make experience replay more efficient and effective, this prioritization can lead to quick convergence in the sparse reward environment [27].

The TDH-DQN bot uses the deep Q-network with the Experience Replay algorithm. DQN is a commonly used model-free algorithm that uses an ANN. DQN is a value-based method that combines DL with QL, which sets the learning objective to optimize the estimates of Q-value. QL, the algorithm evaluates which action to take based on a Q-value (or action-value) function that determines the value of being in a certain state and taking a certain action at that state. For each state-action pair (s, a), this algorithm keeps track of a running average of the rewards, R, which the agent gets upon leaving the state s with action a, plus the rewards it expects to earn later. The TDH-DQN Trading Bot Architecture used for the Parts 2 and 3 analyses is shown in Figure 2.

DL allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. DL discovers intricate structures in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer.

RL problems aim to solve actions that optimize the agent's objective, given some observations about the environment. The environment presents information about its state to the agent or bot, assigns rewards for actions, and transitions the agent to new states, subject to probability distributions the agent may or may not know. RL methods aim to learn from experience how to take actions that achieve a long-term goal. To this end, the agent and the environment interact over a sequence of discrete-time steps via the interface of actions, state observations, and rewards. The main components of an RL system are agent, actions, environment, state, and reward. The agent is the entity that performs actions. Actions are the things an agent can do within its environment. The environment is the world in which the agent resides. The state is the current situation. The reward is the immediate return sent by the environment to evaluate the last action by the bot.

Humans learn from positive or negative experiences that we experience in our environment that we associate with our actions. Learning from experiences and the associated rewards or punishments is the core idea behind RL. RL is an approach to training a machine to find the best course of action through optimal policies that maximize rewards and minimize punishments. RL's main idea of maximizing the rewards aligns with algorithmic trading. RL is particularly suitable for algorithmic trading because the concept of a return-maximizing bot in an uncertain, dynamic environment has much in common with a trading strategy that interacts with the stock market. The DRL bots learn through trial and error, they learn the optimal path of execution. RL algorithms can learn the nuances and parameters within the price time-series data [69].

In RL, one is given a series of inputs and expected to predict y at each step. However, instead of getting instantaneous feedback at each step, one needs to study different paths/sequences to understand which one gives the optimal result. Some models may overfit the data. They perform well on a backtest with historical data but get chopped up on out-of-sample data. The stability of out-of-sample forecasts is a challenge often encountered with trading bot strategies. Variance-Bias trade-off states that an out-of-sample forecast will deteriorate because of three factors. The first issue to confront is in-sample forecast error, next model instability can be challenging, and finally human's inability to think in terms of bets. Thinking in terms of bets is a difficult concept for humans to master as humans tend to underestimate the probability of random events. A skilled trading bot developer must select a model that will find the optimal balance between in-sample error and model variance.

DRL-based models determine rule-based policies for actions. This research combines DRL with algorithmic trading into stock trading bots. Algorithmic trading involves the automation of trading tasks. There are a lot of trading tasks that fall within this category such as risk management, technical analysis, and general hard-coded trading rules. Here humans program the computer to take care of all the actions that don't vary between trading. These are the decisions outside the domain of intuition, where the bot performs actions based on fixed sets of rules. Hard-coded systems fall apart when the market regime changes and the trend changes direction or rate of momentum. Strict algorithmic trading systems often experience sharp drawdowns during these periods, and it takes an experienced trader's intuition to decide to scale back or reverse a strategy that has been working recently or over a larger time [70].

DRL systems can exceed human performance at many tasks but it's not without fault. Modeling a trader's decision-making heuristics and intuition with computer systems is a difficult task. Only a flexible learning system such as DRL can analyze time-series data for pattern recognition and generalization like consistently profitable human traders. Traders adapt and last for years in the market only after having adjusted to multiple bull and bear markets. The DRL trading bots must also cope with the trader's intuitive issue of deciding whether to try new actions that may not be immediately optimal but longer-term could deliver the maximum reward. Alternatively, the bot must decide if it should try new actions to find another optimal path to rewards. The bot must also determine which step in the path was critical to the optimal policy. DRL is considered most like human learning that results from taking actions in the real world and observing the consequences. It differs from supervised learning because it optimizes the agent's behavior one trial-and-error experience at a time based on a scalar reward signal, rather than by generalizing from correctly labeled, representative samples of the target concept. Moreover, RL does not stop at making predictions. Instead, it takes an end-to-end perspective on goal-oriented decision-making by including actions and their consequences [71].

Through experimental trials and feedback loops, DRL trading bots seek to learn the optimal strategy. With the optimal strategy, the agent or bot is capable of adapting. These reward signals are not given to the model immediately. Instead, they are returned because of a sequence of actions that the agent makes. The interaction between the agent and the environment involves a sequence of actions and observed rewards in time, t = 1, 2 ...T. During the process, the agent accumulates knowledge about the environment, learns the optimal strategy, and makes decisions on which action to take next to efficiently learn the best strategy. The state, action, and reward at time step t can be written as S_t, A_t ...R_t , respectively. Thus, the interaction sequence is fully described by one episode, with ending sequence S_T : S_1, A_1, R_2, S_2, A_2 ...A_T.

In addition to the components of RL mentioned so far, there are three additional components of RL that should be considered. These are the policy, value function (and Q-value), and the model of the environment. A strategy or policy is an algorithm or in other words a set of rules that describe how an agent makes its decisions. A policy is a function, usually denoted as π, that maps a state (s) and an action (a): $a_t = \pi(s_t)$. This means that an agent decides its action given its current state. DRL Trading bots seek to learn an optimal strategy or policy (π). An optimal strategy or policy tells the bot how to act to maximize return in every state. The goal of a DRL bot is to learn to perform a task well in an environment. The environment emits a reward signal as the bot's action leads to a transition to a new state. The bot learns a value function that informs its judgment of the available actions. The bot's objective function processes the reward signal and translates the value judgments into an optimal strategy.

The policy translates states into actions. At any point in time, the policy defines the agent's behavior. It maps any state the agent may encounter to one or several actions. Rewards (accumulated percent returns APR in this case) in RL are learning from actions. The reward signal is a single value that the environment sends to the agent at each time step. The agent's objective is to maximize the total APR reward received over time. APR rewards are the key input, and the goal of making value estimates is to achieve more rewards. RL methods focus on learning accurate values that enable good decisions while efficiently leveraging memories [72].

QL allows the algorithm some freedom to explore outside of its training data to find the best policy for buying, selling, and holding stock. This is partially where RL's advantages shine in the interpretation of market data. Any given interval of market data is unique and therefore its training data is unique, so there is a low probability that the training set will be entirely representative of the testing data.

3.4 *Actor-critic with experience replay algorithm*

Value-based methods find the policy by finding the best action value of a state, and accompanying actions which are great for environments with discrete actions like algorithmic trading, where the highest valued action is clearly separate from the next best action. In these action spaces, value-based methods become unstable. Policy-based methods do not use a separate value function but find the policy directly. They start with a policy function, which they then improve, episode by episode, with policy gradient methods. Policy-based methods are applicable to more domains than value-based methods. They work well with deep neural networks and gradient learning; they are some of the most popular methods of deep reinforcement learning, and this chapter introduces you to them [73].

With Actor-Critic agents, the actor decides which action to take and the critic tells the actor how good the action was and how it should adjust. various implementations of the off-policy were tested including ACER, a DRL algorithm initially researched by Wang et al. [18], greatly increasing the sample efficiency and decreasing the data correlation [74]. Also, Asynchronous Methods by Mnih et al. [25]. Profitable results with AC methods were achieved. The author would argue that the primary downside to these algorithms is they are slower to compute than the DQN algorithms without substantially higher APR returns.

Sutton et al. [13] defined Actor-critic methods are Temporal Difference (TD) methods that have a separate memory structure to explicitly represent the policy independent of the value function. The policy structure is known as the actor because it is used to select actions, and the estimated value function is known as the critic because it criticizes the actions made by the actor. Learning is always on-policy: the critic must learn about and critique whatever policy is currently being followed by the actor. The ACER trading Bot Architecture is shown in Figure 4.

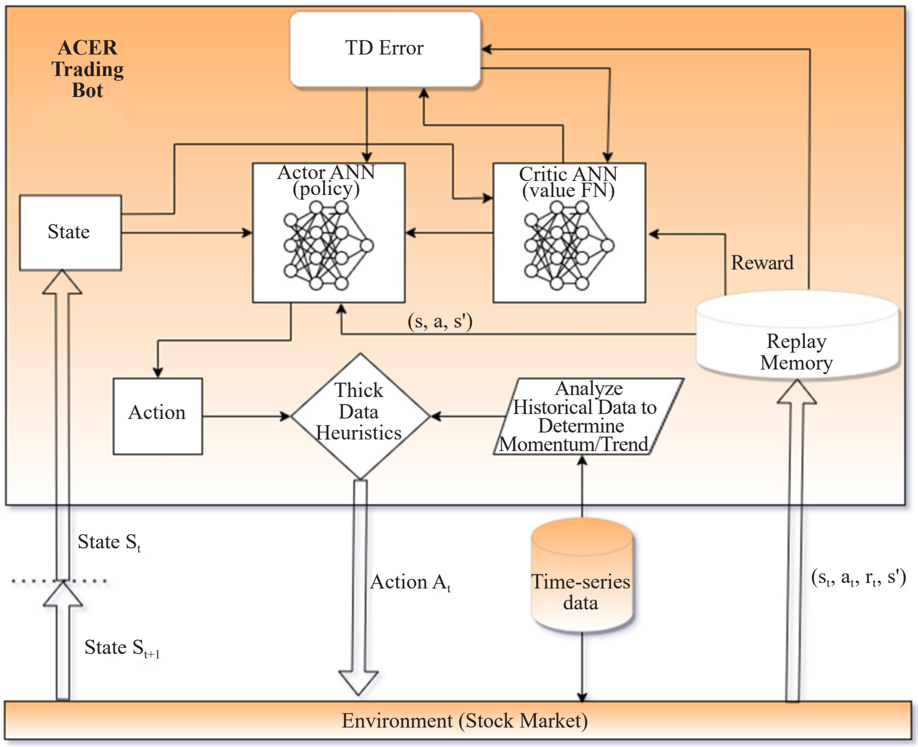

Figure 4. TDH ACER trading bot architecture

ACER nearly matches the state-of-the-art performance of DQN with prioritized replay and substantially outperforms A3C in terms of sample efficiency on both discrete and continuous control domains. ACER capitalizes on deep neural networks, variance reduction techniques, the off-policy Retrace algorithm by Munos et al. [75], parallel training of RL agents by Mnih et al. [25], truncated importance sampling with bias correction, stochastic dueling network architectures, and efficient trust region policy optimization Wang et al. [18].

Actor-critic methods add a value network to the policy network, to achieve the benefits of both approaches. To reduce variance, n-step temporal difference bootstrapping can be added, and a baseline value can be subtracted so that we get the so-called advantage function (which subtracts the value of the parent state from the action values of the future states, bringing their expected value closer to zero). Well-known actor-critic methods are A3C, DDPG, TRPO, and PPO.

A3C features an asynchronous (parallel, distributed) implementation, DDPG is an actor-critic version of DQN for continuous action spaces, and TRPO and PPO use trust regions to achieve adaptive step sizes in nonlinear spaces. Benchmark studies have shown that the performance of the actor-critic algorithm is as good as or better than value-based methods [73, 76].

3.5 *Markov decision process*

The foundation of RL is the Markov Decision Process (MDP). However, before talking about it, one must formalize the simplest child of the Markov family, the discrete Markov Process. In a discrete Markov Process, the system has many different states, which can be observed. The state this system is in changes after each time step and forms a sequence of states, which is called the Markov Chain. However, the state has a restriction on how it evolves, which is a Markov Property: The future dynamics of the system must depend only on the current state, not the history of the evolution of the states. In a Markov Decision Process. States are not directly observable, but some observations based on the state are known. The observation space and state space may or may not have an explicit bijection between them. An agent is taking actions based on the observations, and the action changes the underlying state of the environment, returning a different observation and a reward. How state transits depend on the last state of the system and the action of the agent, usually in a probabilistic way. Given a state-action pair, the environment returns a possibly random reward to the agent. The agent's goal is often to maximize the total reward received, with future rewards discounted by a factor [13].

Exploration is preferable in that it allows the algorithm to venture outside the training data space to find better relationships among data and transactions. In RL, the algorithm learns to react to its environment and plan sequential steps toward a goal. A Markov Decision Process (MDP) frames the agent-environment interaction as a sequential decision problem over a series of discrete-time steps (t = 1, ..., T) that constitute an episode [13].

The abstraction afforded by MDPs makes its application easily adaptable to many contexts. The time steps can be at arbitrary intervals, and actions and states can take any form that can be expressed numerically. The Markov property implies that the current state completely describes the process, that is, the process has no memory. Information from past states adds no value when trying to predict the process's future.

MDPs proceed in the following fashion: at each step t, the agent observes the environment's state and selects an action, where S and A are the sets of states and actions, respectively. At the next time step t + 1, the agent receives a reward and transitions to state S_{t+1}. Over time, the MDP gives rise to a trajectory S_0, A_0, R_1, S_{t1}, A_1, R_1, ... that continues until the agent reaches a terminal state and the episode ends. Due to the Markov property, these distributions only depend on the previous state and action.

Rewards are typically discounted using a factor to reflect their time value. In the case of tasks that are not episodic but continue indefinitely, a discount factor of just less than 1 is necessary to avoid infinite rewards and ensure convergence. Therefore, the agent maximizes the discounted, expected sum of future returns R_t, denoted as G_t.

3.6 *DQN trading bots*

The DQN uses an ANN to approximate Q-values; hence, the action value function is defined as Q(s, a; θ). The deep QL algorithm approximates the Q-value function by learning a set of weights, θ, of a multilayered DQN that maps states to action [72]. The TDH-DQN Trading Bot Architecture used for the Part 2 analysis is shown in Figure 2.

The QL algorithm evaluates which action to take based on a Q-value function that determines the value of being in a certain state and taking a certain action at that state. For each state-action pair (s, a), this algorithm keeps track of a running average of the rewards, R, which the agent gets upon leaving the state s with action a, plus the rewards it expects to earn later. Since the target policy would act optimally, the bot takes the maximum of the Q-value estimates for the next state [13].

The learning proceeds off-policy that is, the algorithm does not need to select actions based on the policy that is implied by the value function alone. However, convergence requires that all state-action pairs continue to be updated throughout the training process, and a straightforward way to ensure that this occurs is to use an ϵ - greedy policy. In cases where the state and action space are large, the optimal Q-value table quickly becomes computationally infeasible.

RL is unstable or divergent when a nonlinear function approximator such as a neural network is used to represent Q. This instability comes from the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy of the agent and the data distribution, and the correlations between Q and the target values.

This removes correlations in the observation sequence and smooths changes in the data distribution. Iterative updates adjust Q towards target values that are only periodically updated, further reducing correlations with the target. To address instability and divergence issues, the bot uses ANNs to approximate Q-values. The bot uses a function with parameter θ to calculate Q-values, and the Q-value function can be written as Q(s, a; θ).

The deep QL algorithm approximates the Q-values by learning a set of weights, θ, of a multilayered deep Q-network that maps states to actions. The DQN algorithm aims to greatly improve and stabilize the training procedure of QL through a couple of different methods [72]. The first is Experience replay [15]. Instead of running QL on state-action pairs as they occur during simulation or actual experience, the algorithm stores the history of the state, action, reward, and next-state transitions that are experienced by the agent in one large replay memory. This can be referred to as a mini-batch of observations. During QL updates, samples are drawn at random from the replay memory, and thus one sample could be used multiple times. Experience replay improves data efficiency, removes correlations in the observation sequences, and smooths over changes in the data distribution.

The second method [6] is to update Q and optimize towards target values that are only periodically updated. The Q-network is cloned and kept frozen as the optimization targets every hyperparameter sequential step. This modification makes the training more stable as it overcomes short-term oscillations. To learn the network parameters, the algorithm applies gradient descent to a loss function defined as the squared difference between the DQN's estimate of the target and its estimate of the Q-value of the current state-action pair, Q(s, a; θ). The loss function is shown in Equation (1):

$$L(\theta) = \mathbb{E}[(r + \gamma \max \mathrm{Q}(\mathrm{s}', \mathrm{a}'; \theta_{\mathrm{i}-1}) - \mathrm{Q}(\mathrm{s}, \mathrm{a}; \theta_{\mathrm{i}}))^2], \tag{1}$$

The loss function is a Mean Squared Error, (MSE) function, where (r + γmax$_{a'}$ Q(s', a'; θ_{t-1})), represents the target value and Q[s, a; θ_i], represents the predicted value, θ are the weights of the network, which are computed when the loss function is minimized. Both the target and the current estimate depend on the set of weights, underlining the distinction from supervised learning, in which targets are fixed prior to training [72].

The QL Steps used can be summarized as follows. At time step t, we start from state s$_t$ and pick an action according to the Q-values shown in Equation (2).

$$\mathrm{a}_t = \max_a \mathrm{Q}(\mathrm{s}_t, \mathrm{a}), \tag{2}$$

Apply an ϵ greedy approach that selects an action randomly with a probability of ϵ or otherwise chooses the best action according to the Q-value function. This ensures the exploration of new actions in each state while also exploiting the learning experience. With action a$_t$, we observe reward R$_{t+1}$ and get into the next state S$_{t+1}$. Update the action-value function as shown in Equation (3):

$$\mathrm{Q}(\mathrm{S}_t, \mathrm{A}_t) \leftarrow \mathrm{Q}(\mathrm{St}_t, \mathrm{At}_t) + \alpha[\mathrm{R}_{t+1} + \gamma \max_a \mathrm{Q}(\mathrm{S}_{t+1}, \mathrm{a}) - \mathrm{Q}(\mathrm{S}_t, \mathrm{A}_t)]), \tag{3}$$

Finally increment the time step, t = t + 1, and repeat the steps. Keep iterating until converging to the optimal Q-value. The TDH-DQN Trading Bot Architecture used for the Part 2 and 3 analyses is shown in Figure 2.

The Double DQN agents max operation in Q-learning uses the same values both to select and evaluate the action. This makes it more likely to select overestimated values in case of inaccuracies or noise, resulting in over-optimistic value estimates. Therefore, the DQN algorithm induces an upward bias. The double estimator method uses two estimates for each variable, which allows for the selection of an estimator and its value to be uncoupled by Hasselt et al. [77]. Thus, regardless of whether errors in the estimated Q-values are due to stochasticity in the environment, function approximation, non-stationarity, or any other source, this allows for the removal of the positive bias in estimating the action values. In Double DQN, or DDQN by van Hasselt et al. [16], the target value is replaced which leads to less overestimation of the Q-learning values, as well as improved stability, hence improved performance. As compared to DQN, the target network with weights is used for the evaluation of the current greedy action. Note that the policy is still chosen according to the values obtained by the current weights θ.

In the Dueling network architecture by Wang et al. [17], the neural network architecture decouples the value and advantage function which leads to improved performance. The stream provides an estimate of the value function, while the other stream produces an estimate of the advantage function. The learning update is done as in DQN, and it is only the structure of the neural network that is modified. In fact, even though it loses the original semantics of V and A, a slightly different approach is preferred in practice because it increases the stability of the optimization. In that case, the advantages only need to change as fast as the mean, which appears to work better in practice [68, 78].

3.7 *Simulation-critic methods*

Simulation-critic methods are a good way to study and analyze trading bots. Such techniques use simulation to learn a value function, which can then be used to update the trading policy parameters. The bots operate starting at the market open and seek to follow the trend in the long, and short timeframes. It is the main idea behind all breakout or trend-following trading systems. Such bots perform differently when they go long or short depending on the long-term trend. Day trading and swing trading bots also perform differently. In this research, we will explore two completely different trading environments with the day trading bots in Part 1 and the swing trading bots in Part 2 of the analysis.

Trained trading models that perform well out of sample are the core requirement for effective DRL trading bots. DRL gives computers the ability to learn from their experiences and improve their performance as they gain more experience. The trading bots create models to uncover the relationships between the price inputs and the trading decisions outputs when given historical price data as input. Next, the bots forecast outcomes out of the sample on the test data. If the results are satisfactory the bots can be deployed into the markets with the trained models to execute live trades. This feature makes DRL particularly attractive as an underlying approach to building algorithmic trading bots.

The interactive and online nature of DRL makes it particularly well-suited to the trading domain. DRL models goal-directed learning by an agent that interacts with a typically stochastic environment that the agent has incomplete information about. DRL aims to automate how the agent makes decisions to achieve a long-term objective by learning the value of states and actions from a reward signal. The goal is to derive a policy that encodes behavioral rules and maps states to actions.

3.8 *Performance metric-Accumulated Percent Returns (APR)*

The performance metric used is Accumulated Percent Returns (APR), which are shown in Equation (4). Compound Percent Returns (CPR) is used interchangeably in this article to mean the same as APR. APR is the difference between the price the stock was purchased at and the current stock price divided by the bought price. This value is multiplied by 100 to convert the value to percent returns.

$$\text{Accumulated Percent Return} = [(\text{Current Price} - \text{Bought Price}) / \text{Bought Price}] * 100 \tag{4}$$

3.9 *Catalysts reasons for stock momentum*

Catalyst (info. which increases directional momentum) stocks have a reason to move, the catalysts reasons human traders look for are, stock buybacks, new debt offerings, earnings announcements, new prescription drugs or vaccine approvals, mergers or acquisitions, product releases, corporate Restructuring, and stock splits. Financial markets with the most opportunities and lowest fees are an essential element to the bot's success. For this reason, NASDAQ and NYSE listed US stocks that show up in a custom scan for catalyst stocks. These stocks can be found in the Most Active Stocks Market Scan. These catalyst stocks were chosen as the focus of this research for day trading bots.

The scan identified stocks with a volume greater than 350 thousand shares traded before 9:30 am EST in the premarket session. The market-wide scanner only shows stocks above one billion market capitalization, over 10 dollars per share with a high-volume rate of more than one million shares per three-minute bar. The scan catches all the daily catalyst stocks traders are looking at that day. Most of the time, the catalyst is from the news that day. Often, the same stock can show up for the following few weeks due to the same catalyst if the stock's story fundamentally changes the way investors view it. Time-Series.

Human traders are looking for stocks with bars that interconnect and do not have big gaps in the 5-min timeframe. Stocks that have recent news from the last week with a strong narrative around them. These are the stocks showing up on other traders' watchlists observed thru chat rooms, message boards, and YouTube live streams each morning starting at 4:30 am until the market opens at 9:30 am. Often these stocks will have gaped up or down by more than 20 percent, with smooth price action that flows with higher highs and lower lows. Stocks with close spreads between bids and asks with lots of liquidity on the level 2 screen. Level 2 is a subscription-based service that provides real-time access to the NASDAQ and NYSE order book. It is intended to display market depth and momentum to traders and investors. Human traders are looking for stocks that have lots of liquidity well above and below the market price at price levels identified on the chart at pivot points where price reversed direction multiple times in the past.

The eight different python RL bots shown in Table 6, analyze five datasets, including the five-second, five-minute, daily, weekly, and monthly time series bars. The bots employ the thick data risk parameters shown in Table 3. The DRL bots employ risk management parameters including time of day, holding time, stop loss, and profit target, to frame the rules of their environment. The bot's environment is further limited to scanning the entire market autonomously, trading only the best catalyst stocks of the day with high relative volume. The bots trade in real-time with paper trades and historical data sent back and forth from a local Jupyter python notebook and the stockbroker, in this case, Interactive Brokers (IB). To connect the bots to a broker, IB was used which offers a Python Application Programming Interface (API) to develop algorithmic trading bots connected to real-time data.

Table 6. Part 1a: Deep RL trading bot algorithm abbreviations

Trading bot algorithm	Abbreviation
Policy Gradient	PG
Q-learning	DQN
Double Q-Learning	DQL
Actor-Critic	AC
Curiosity Q-Learning	CQN
Recurrent curiosity Q-Learning	RCQL
Dual Actor-Critic	DAC
Dual Curiosity Q-Learning	DCQL
Buy and Hold	B & H
Sell and Hold	S & H

The bots focus entirely on the catalyst stocks breaking out from a trend with substantial relative volume and liquidity. The bots limit overnight gaping up or down risk in price movement by only focusing on intraday price action. Furthermore, the bots exit their positions at the end of the trading day and do not hold positions overnight. The bots choose stocks to trade from the high 3 min volume market scanner for NYSE and Nasdaq stocks in real-time starting at 9:30 AM eastern when the markets open and fully exit positions at 11:00 AM. The seven catalyst stocks chosen for Part 1a analysis are shown in Table 3.

To summarize, the day trading bots function in the following way. First, the bot scans for stocks in its list of pre-approved symbols. Next, the bot immediately downloads the most up-to-date real-time historical data and saves the datasets locally. The bots calculate the 50 EMA for the five-second bars dataset and the 9 SMA's for the daily, time-series datasets. Figure 5 shows an example from the results for NIO with the 9 SMA shown in green. For that day, 2021-10-04, NIO provides a bearish example of the type of setup the bot is looking for. It can be observed in Figure 2 shows that the price has moved below the daily moving average so the agents will be shorting the stock along with the trend for that day.

The bots decide to be bullish or bearish only for that day based on the relationship of the current price vs. the EMA's for Part 1. For Part 2, the bots have a bullish bias due to the steadily rising bull market starting in 2009. Short selling is basically opening a position by selling it first, assuming in the future one can buy it back at a cheaper price. One is borrowing the stock from the broker and selling it in the market. Therefore, short selling is betting for the price to drop. Often the best catalyst stocks are hardly available for short selling which makes backtesting a long/short strategy difficult.

Choosing which environment to interact with gives the bots an edge because, in a constantly evolving market, they operate under similar market conditions all the time. The bots focus on trading the hot stocks of the day with massive volume, which often completely ignore the broader market correlations and trends independently. Stocks with close spreads between bids and asks with lots of liquidity on the level 2 screen.

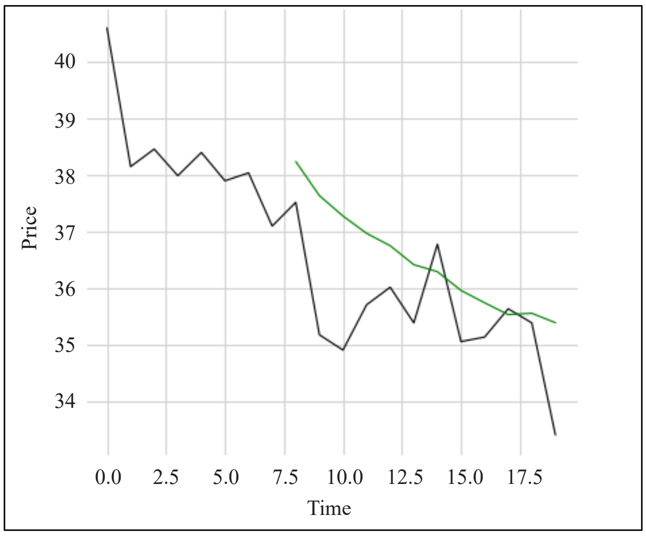

Figure 5. EMA trend strength, NIO daily price vs time chart for Part 1b. 9 SMA shown in green

The bots are looking for stocks that have enormous amounts of liquidity well above and below the market price at price levels identified on the chart as swing points where prices reversed direction multiple times in the past. These are the elements a skilled trader does after years of experience in the markets, after having developed a keen sense of timing and ability to read the tape and markets which makes thick data analysis of financial markets difficult.

Timing is especially important, often the best entries must be lightning fast as the price moves below an important level and then rises above the bot's need to have a buy-stop order in place waiting at the market price. Choosing these correct price levels and timing the entries and exits is a crucial part of TDH. Other traders and bots will have their stops at these levels and prices can move through the level and continue to trend in a breakout pattern. If you do not get filled when the price goes through the level, you must risk more with a higher entry price. Often price will retest and squeeze your entry close to its stop limit as it trends in ABCD patterns. This is so market makers can create liquidity by running traders' stops.

The short-term day trading bots seek to trade the hottest catalyst stocks during the market opening when the volume is at its highest, and there is lots of liquidity. The bots download quotes, calculate decisions, and execute in about a minute to react to the speed of markets. At the end of each minute, the bot decides to buy, short, exit, or hold positions. Choosing which catalyst stock, the bots trade is an important part of the TDH. In the author's experience, bots will have limited success trading general stocks without a catalyst that just moves with the market.

3.10 *Part 1-DRL day trading bots*

Part 1 has two questions to answer. In Part 1a, we first determine the best DRL algorithm for day trading using the five-second timeframe time-series data with seven catalyst stocks. Secondly, in Part 1b, we determine if adding TDH to DRL increases or decreases APR performance on the one-minute timeframe time-series data with the same seven catalyst stocks from Part 1a. The starting python code base for the basic implementation of the DRL algorithms for Part 1 can be found in the following public code repositories [79, 80].

Table 7. Part 1 day trading bot risk parameters

Parameter	Setting value
Holding time	Holding time = 30 minutes is the maximum holding time
First trade	5 minutes into the trading day at 9:35 am EST
Last trade	55 one-minute bars into the trading day or 10:25 am EST
Time between trades	5 minutes
Stop loss	3.0 percent
Profit target	4.0 percent

Table 8. Part 1 DRL agent class comparison

DRL Algorithm	act	get_state	replay	buy	train	memorize	construct memories	assign	select action	predict	get predicted action
DQN	x	x	x	x	x						
Duel DQN	x	x	x	x	x						
Recurrent DQN		x		x	x	x	x				
Double DQN		x		x	x	x	x	x	x	x	x
Double Recurrent DQN		x		x	x	x	x	x	x		
Double Duel DQN		x		x	x	x	x	x	x	x	x
Double Duel Recurrent DQN		x		x	x	x	x	x	x		
Curiosity DQN		x		x	x	x	x		x	x	x
Recurrent Curiosity DQN		x		x	x	x	x				
Dual Curiosity DQN		x		x	x	x	x				
AC		x		x	x	x	x	x	x		
Dual AC		x		x	x	x	x	x	x		
Recurrent AC		x		x	x	x	x	x	x		
Duel Recurrent AC		x		x	x	x	x	x	x		

In the day trading bot analysis, the bots operate at the market open and seek to follow the trend. The goal of this part is to compare the performance of the different DRL algorithms against each other and for this reason we train and test the DRL bots on the same data and don't split the data between testing and training datasets. Since we are comparing the algorithms to each other we are trying to determine which algorithm strikes a good balance of simplicity, computation speed, and ultimately the algorithm's ability to fit the curve and learn that specific price action. It is the main idea behind all trading systems, at times the market will trend, and that fundamental element of markets will always be timeless. The bot seeks to run with the crowd of bulls or bears, whichever is stronger, for as long as they have the volume and momentum.

For the Part 1 analysis, it was found that the best way to test the system was to use real-time data as it came in and to train and run the bots as fast as they could calculate. This is due to the nature of day trading and the way the market trades through the course of the day. Training the bots on historical data from other stocks on the previous day's data didn't achieve good results in this research. For this reason, we didn't partition the data such as 70 percent for testing and 30 percent for training. The agents were trained and tested in real-time as the day progresses, so the agents start with limited data but as more data comes in throughout the day the bots continually re-evaluate trading decisions to try to come up with optimal solutions. Once the agents identify a trade execution the system can then execute an order. The agents are continually trying to discover the best sequence of trade execution decisions throughout the day while still managing risk. The risk management parameters are shown in Table 7, while the DRL agents class comparison and Hyperparameter setting are shown in Tables 8 and 9.

Table 9. DRL algorithm hyperparameters for Part 1, day trading bots

DRL parameters	gamma	epsilon	Min epsilon	epsilon_decay	Batch size	Window size	Learning rate	Layer size	Epochs
DQN	0.99	1	0.01	0.995	86	85	0.001	8	10
Duel DQN	0.99	1	0.01	0.995	86	85	0.001	8	10
Recurrent DQN	0.99	1	0.01	0.995	86	85	0.001	8	10
Double DQN	0.99	1	0.01	0.995	86	85	0.001	8	10
Double recurrent DQN	0.99	1	0.01	0.995	86	85	0.001	8	10
Double duel DQN	0.99	1	0.01	0.995	86	85	0.001	8	10
Double duel recurrent DQN	0.99	1	0.01	0.995	86	85	0.001	8	10
Curiosity DQN	0.99	1	0.01	0.995	86	85	0.001	8	10
Recurrent curiosity DQN	0.99	1	0.01	0.995	86	85	0.001	8	10
Dual curiosity DQN	0.99	1	0.01	0.995	86	85	0.001	8	10
AC	0.99	1	0.01	0.995	86	85	0.001	8	10
Dual AC	0.99	1	0.01	0.995	86	85	0.001	8	10
Recurrent AC	0.99	1	0.01	0.995	86	85	0.001	8	10
Duel recurrent AC	0.99	1	0.01	0.995	86	85	0.001	8	10

The bots were tested during the market opening when the volume is at its highest and there is lots of liquidity. The bots hold for a brief time and exit before 11 am EST when volume slows down. At the end of each timeframe tick duration, the bot decides to buy, sell, or hold. Due to competing institutional algorithmic trading bots, most of the stocks will trend with the overall market unless they have a reason not to. So, if the market is moving up, most stocks will be moving up and it's time for the bot to be bullish. If the overall market goes down, the prices of most stocks will also go down. Every day, there are always one or two stocks that will move independently of the market because it has a catalyst. The THD-DQN bots seek to be trading stocks that are moving because they have a fundamental reason to move and are not just moving with the overall market correlations [4].

3.11 *Part 2-swing trading TDH-DQN bot*

In Part 2, we are comparing the TDH-DQN trading bot's performance against three algorithmic trading strategies and the buy and hold benchmarks for 10 different catalyst growth stocks that consistently show up on the daily pre-market morning scans discussed in Part 1, the day trading bot analysis.

The TDH-DQN trading bot is constructed with Python, Numpy, Pandas, Matplotlib, Tensorflow, and Keras in Jupyter notebooks. The function model for the TDH-DQN bot is a DL model that maps the states to actions. This function takes in the state of the environment and returns a Q-value table or a policy that refers to a probability distribution over actions. This function is built using the Keras Python library. The experience replay is the key function, where the neural network is trained based on the observed experience. This function implements the Experience replay mechanism. Experience replay stores a history of the state, action, reward, and next-state transitions that are experienced by the agent. It takes a minibatch of the observations (replay memory) as an input and updates the DL-based QL model weights by minimizing the loss function. The epsilon greedy approach implemented in this function prevents overfitting.

Experience replay was implemented with the following steps. First, we prepare the replay buffer memory, which is the set of observations used for training. New experiences are added to the replay buffer memory using a For loop. Next, we loop across all the observations of the state, action, reward, and next-state transitions in the mini-batch. The target variable for the Q-table is updated based on the Bellman equation. The update happens if the current state is the terminal state or the end of the episode. This is represented by the variable done and is defined further in the training function. If it is not done, the target is set to reward. Next, we predict the Q-value of the next state using a DL model. The Q-value of this state for the action in the current replay buffer is set to the target. The DL model weights are updated by using the model fit function. The epsilon greedy approach is implemented which selects an action randomly with a probability of ε or the best action, according to the Q-value function, with probability $1 - \varepsilon$ [72].

For Part 2 of the analysis, we will look at swing trading strategies with a longer weekly timeframe as the primary input to the TDH-DQN bot. The dataset for each of the ten stocks goes back ten years and is shown in Table 5. The data is split 70 percent for training and 30 percent reserved for testing. We train the THD-DQN bot for five episodes to observe the trades, inventory, and profits the trading bots take. This bot is long-only and has few rules to limit its trades. The bot is free to buy as many shares as possible, one at a time each week as new data comes in and exits its entire position at the end of the dataset.

Table 10. Part 2 APR comparison of TDH-DQN bot vs. benchmark strategies

Stock	B & H	TDH-DQN	Pivot reversal (2)	MACD LE	Outside bar
TSLA	819	3,439	911	814	1,163
AAPL	181	400	861	452	864
AMD	161	278	809	296	208
NVDA	335	350	1,208	765	712
FB	39	231	141	-52	-60
MSFT	100	144	388	265	310
AMZN	73	707	30	12	101
QQQ	93	150	215	-28	175
PLUG	609	4,564	11,769	1,386	554
AMC	464	891	819	640	1,148
Average	287	1,115	1,715	455	518

Table 10 shows the APR for five different strategies. The Buy and Hold (B & H) strategy simply buys on the first bar in the dataset and sells on the last bar. The bots only take long positions due to the upwards trend direction over the last ten years. The TDH-DQN bot is limited to buying or selling one share at a time and closes out its entire position at the end of the dataset. The pivot reversal bot buys when a pivot point reversal occurs with a strength of two bars to the left on the chart. The Moving Average Convergence Divergence (MACD) bot triggers entries when the MACD line crosses up above the zero line and signifies trend reversals after pullbacks. Finally, the Outside Bar bot executes trade entries when the outside bar engulfs the inside bar. The outside bar is another good trend continuation signal after a pullback that works well in a bull market. The swing trading bots all use a 1:8 risk-to-reward ratio for risk management implemented with stop execution orders.

3.12 *Benchmark strategies*

Buy and Hold (B & H) is a classical benchmark, especially for a bull market; thus, the strategy simply buys the stock and holds it until the end of the testing period to provide a baseline profit measure. For a comparison metric, three simple algorithmic trading strategies were chosen that had excellent performance buying pullbacks in a bull market to compare the TDH-DQN bot against. These are simple algorithmic trading strategies written in C# code with proven performance executed in the Multicharts.Net platform [81].

Multicharts.Net Pivot Reversal LE Strategy: Pivot Reversal LE places a long entry Stop order on the High of the next bar when a breakout of the High Pivot point occurs. The Pivot High point is identified when the high price of the bar is above the high prices of several previous and subsequent bars (specified in Strength input). When a new Pivot High point is confirmed, an order is placed. This signal generates a long entry only. Inputs Strength (2) - number of lower highs that must be on each side of the Pivot High point. As an example of the Pivot Reversal strategy, the results from Part 2 for NVDA are shown in Figure 6.

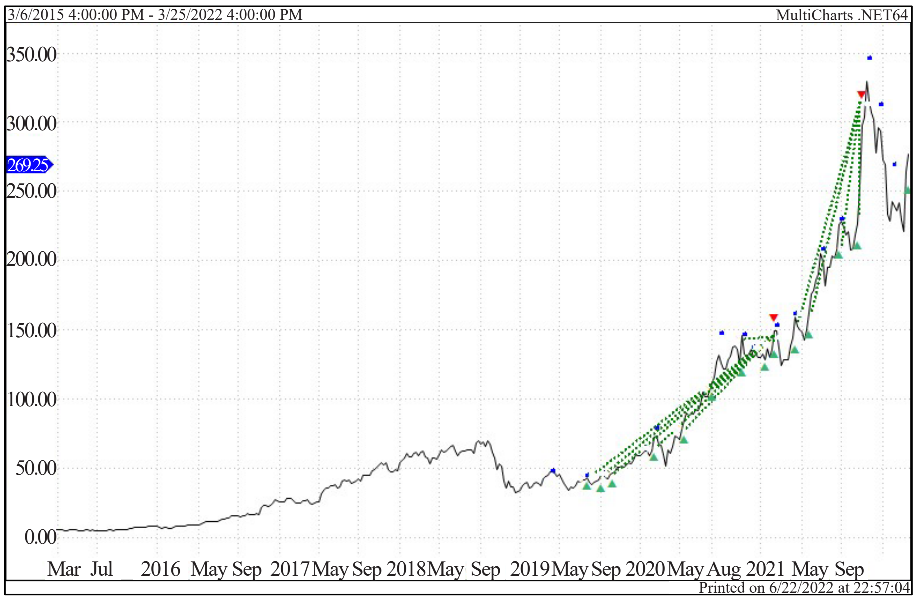

Figure 6. Price vs. time chart, Part 2: NVDA, pivot reversal LE strategy orders

Multicharts.Net MACD LE Strategy: Moving Average Convergence Divergence (MACD) is a trend-following momentum indicator that shows the relationship between two moving averages of a security's price. The MACD is calculated by subtracting the 26-period Exponential Moving Average (EMA) from the 12-period EMA. MACD can be used to identify aspects of a security's overall trend. The MACD will be over zero when the two exponential averages are bullish and under zero when the two exponential averages are bearish. The MACD LE signal generates a buy order for the opening of the next bar when the MACD crosses above the exponential average of the MACD. The MACD LE generates long entry orders only. Inputs: FastLength sets the number of bars used to calculate the fast exponential average, 12 by default. SlowLength sets the number of bars used to calculate the slow exponential average, 26 by default. MACDLength sets the number of bars used to calculate the MACD exponential average, 9 by default. As an example of the strategy's performance, the results of the MACD LE strategy for TSLA are shown in Figure 7. Table 10 shows the MACD LE strategy earned an 814 APR for TSLA on the weekly timeframe.

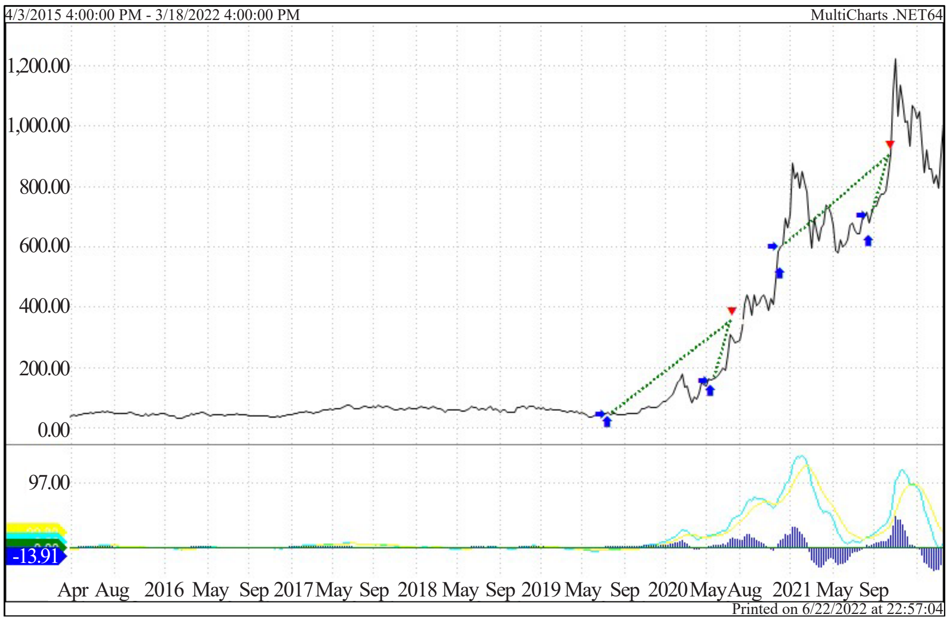

Figure 7. Price vs. time chart, Part 2: TSLA, MACD LE strategy orders

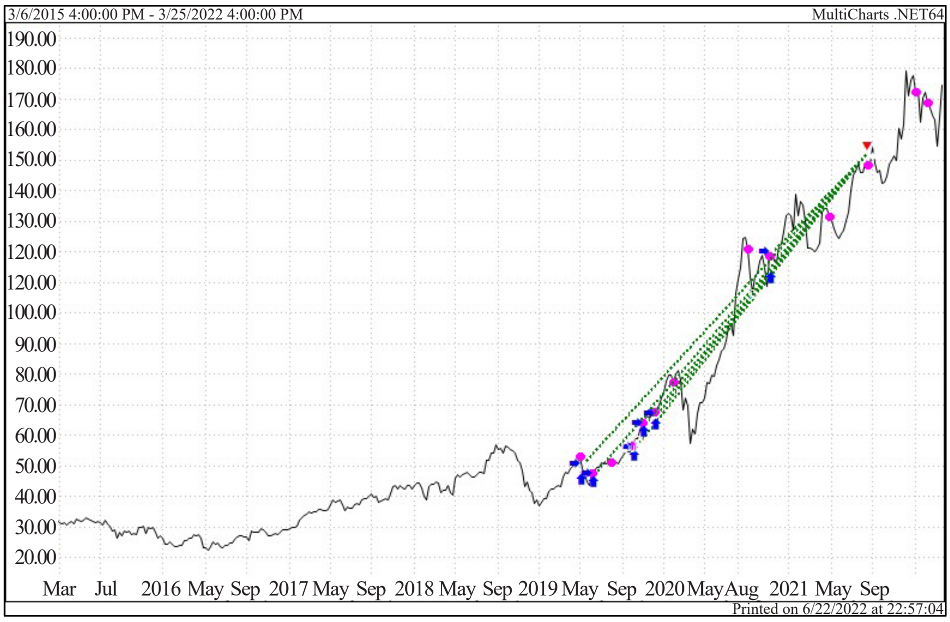

Figure 8. Price vs. time chart, Part 2: AAPL, outside bar LE strategy orders

Multicharts.Net Outside Bar LE Strategy: Outside Bar LE (Long Entry) places a long entry order next bar if the current is an outside bar (the bar's Low is less than the previous Low, High is greater than the previous High) the Close is greater than the Open. An example of the Outside Bar Strategy for AAPL is shown in Figure 8, where the strategy earned 864 APR as shown in Table 10 of the Part 2 results

3.13 *Part 3-TDH-DQN trading bot case study*

For the final Part 3 of the analysis, we look at a case study for sample data created with a sinusoidal mathematical function. The time-series data is composed of 100 values and is split with the same 70/30 split as the second part of deep QL bots. In the results section, we can observe the training episodes of the bots to gain a better understanding of the methodology the bots take as they explore and train for optimal profit interaction with the environment.

Table 11. TDH-DQN trading bot hyperparameters and model settings in Parts 2 and 3

Parameter	Setting value
gamma	0.99
epsilon	1.0
epsilon min	0.01
epsilon decay	0.995
Loss function	MSE
optimizer	Adam
window size	70
batch size	69
episodes	10
Learning rate	0.001
Training/Testing split	70/30 percent

Table 12. TDH-DQN bot Q-Network parameters for the sequential keras model in Parts 2 and 3

Layer (type)	Output shape	Param	Activation
dense (Dense)	(None, 64)	4,544	relu
dense_1 (Dense)	(None, 32)	2,080	relu
dense_2 (Dense)	(None, 8)	264	relu
dense_3 (Dense)	(None, 3)	27	linear

Total params: 6,915
Trainable params: 6,915
Non-trainable params: 0

Table 11 shows the TDH-DQN Bots hyperparameter setting as well as the Part 1 and 2 model settings. There was a fair bit of experimentation required to arrive at these parameters and changing any of these effects parameters affects the model's outcome. In Table 12, the ANN architecture with details on the 4 neural network layers, number of nodes, activation functions, and the total trainable parameter's value of 6,915 for the Q Network can be observed.

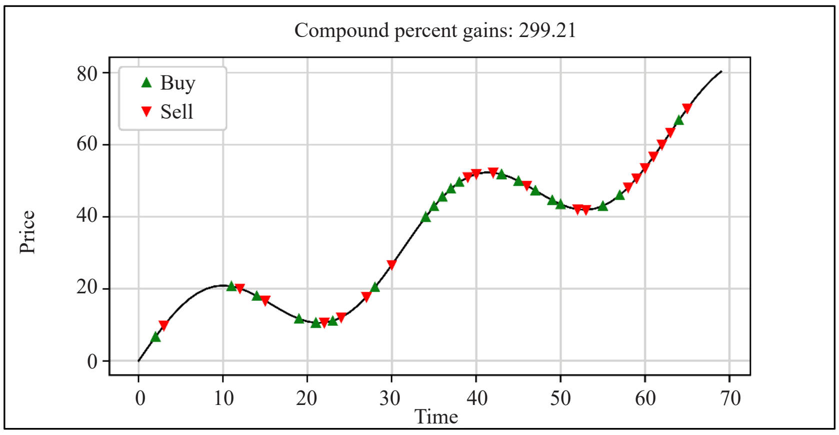

Figure 9. Part 3 orders: Episode 1 of TDH-DQN training

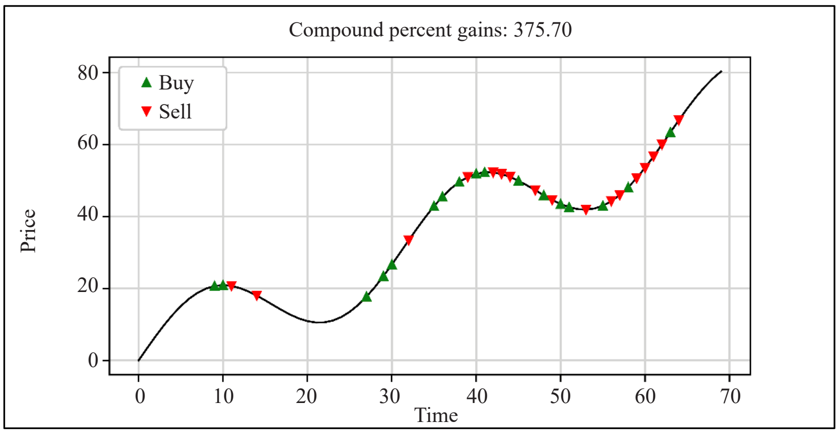

Figure 10. Part 3 orders: Episode 2 of TDH-DQN training

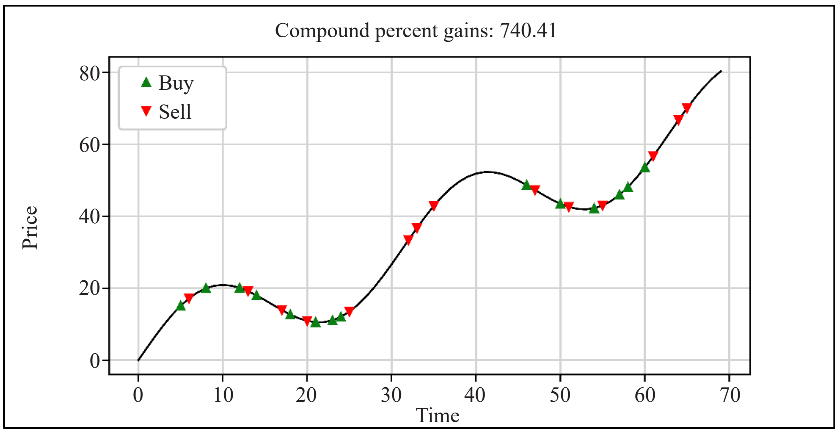

Figure 11. Part 3 orders: Episode 3 of TDH-DQN training

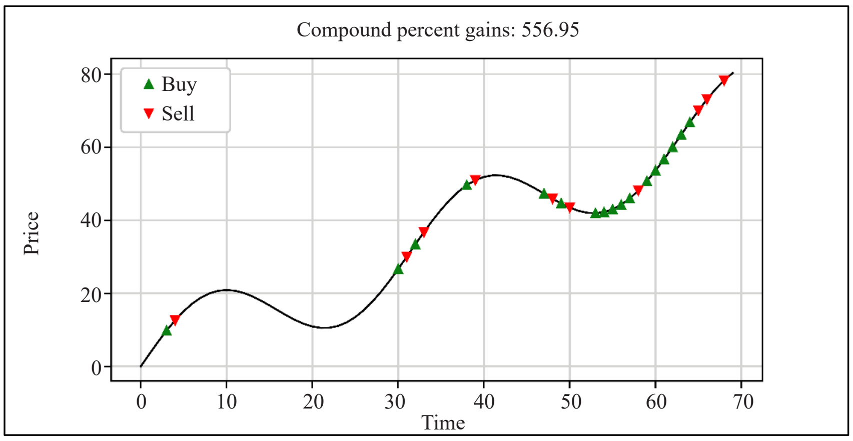

Figure 12. Part 3 orders: Episode 4 of TDH-DQN training

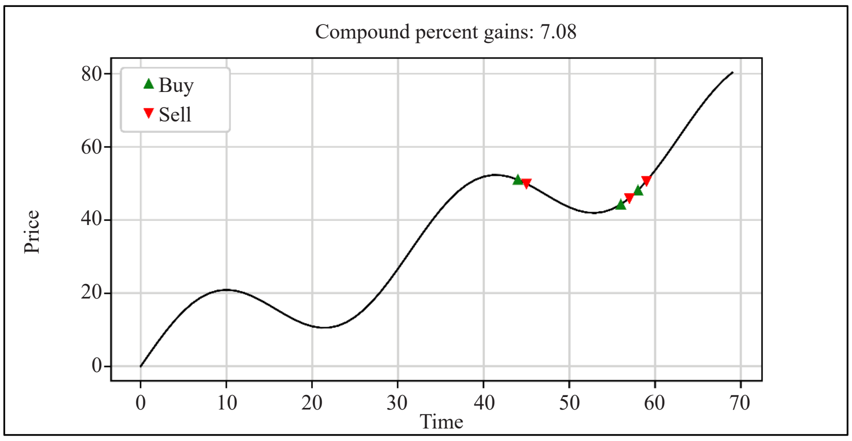

Figure 13. Part 3 orders: Episode 5 of TDH-DQN training

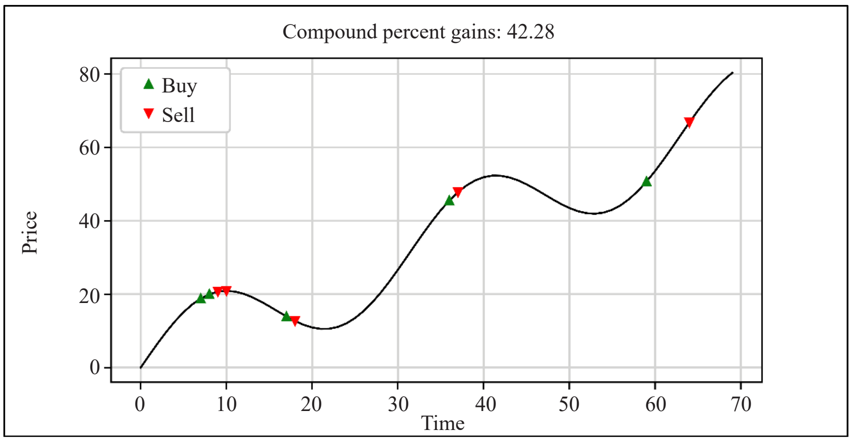

Figure 14. Part 3 orders: Episode 6 of TDH-DQN training

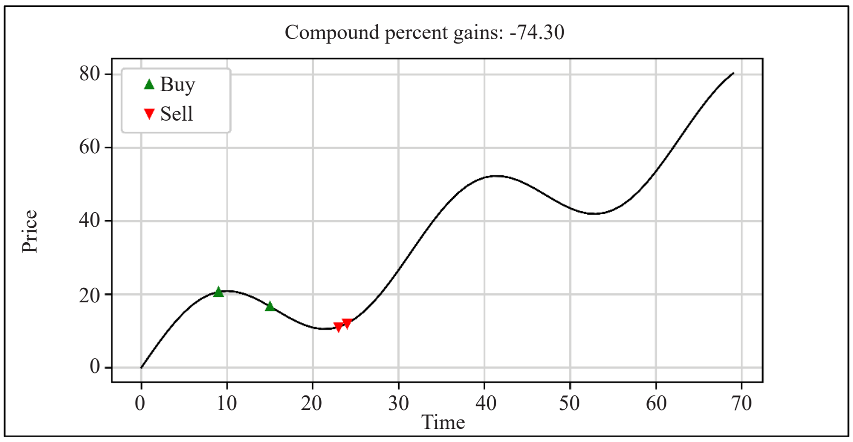

Figure 15. Part 3 orders: Episode 7 of TDH-DQN training

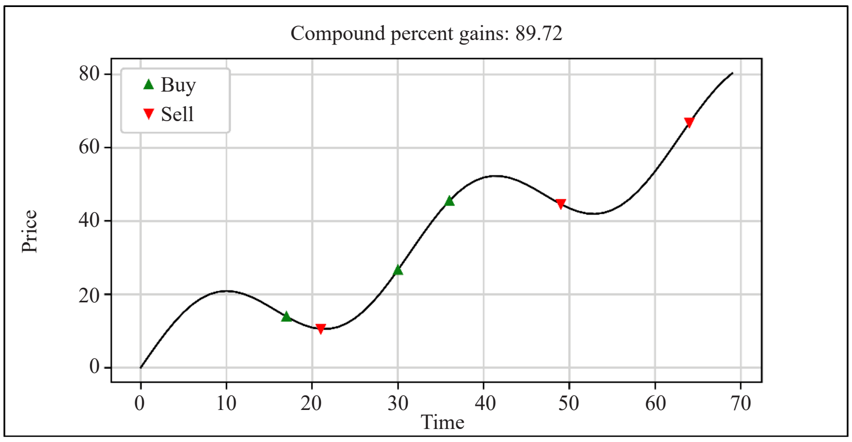

Figure 16. Part 3 orders: Episode 8 of TDH-DQN training

The Part 3 case study of time-series data created and used as test and train data is shown in Figures 9-16. In these figures, the different training episodes for the trading bot can be observed. The dataset consists of 100 values split 70:30 percent for training and testing. The data was generated with the following mathematical function as shown in Equation (5).

$$f(t) = [t + 18 * \sin(0.05 * t * 3)] + 0.01 \tag{5}$$

This is an ideal training dataset scenario where the bot can learn and execute a winning algorithm easily compared to noisy stock market time-series data.

3.14 *TDH-DQN augmented AI pseudo code*

1. Analyze the market with real-time high volume percentage gainers and losers watchlists during the pre-market session starting at 4 am EST. until 9:30 am EST.

2. Human trader builds the pre-defined list of stocks based on high relative volume, news, and technical analysis, to identify the reason for the stock to be added to the pre-defined catalyst stocks list. The augmented AI bot Network architecture can be observed in Figure 17.

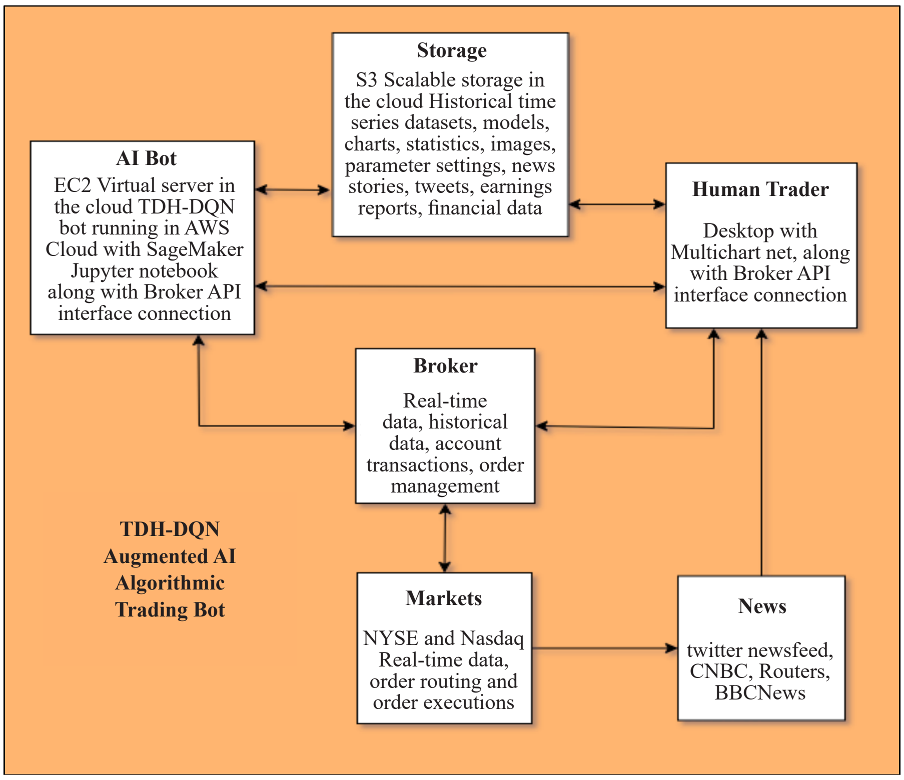
Storage
S3 Scalable storage in the cloud Historical time series datasets, models, charts, statistics, images, parameter settings, news stories, tweets, earnings reports, financial data
AI Bot
EC2 Virtual server in the cloud TDH-DQN bot running in AWS Cloud with SageMaker Jupyter notebook along with Broker API interface connection
Human Trader
Desktop with Multichart net, along with Broker API interface connection
Broker
Real-time data, historical data, account transactions, order management
TDH-DQN Augmented AI Algorithmic Trading Bot
Markets
NYSE and Nasdaq Real-time data, order routing and order executions
News
twitter newsfeed, CNBC, Routers, BBCNews

Figure 17. TDH-DQN augmented AI bot network architecture

3. Human trader uses Multicharts.Net for parameter optimization, backtesting, fib-analysis, and price level identification based on previous historical pivot points on daily, weekly, and monthly timeframes.

4. Create policies to maximize the APR for the pre-approved catalyst stocks by training and testing the model every time series increment (5 sec, 1 min, 5 min, 1 hr., daily, weekly, monthly) as real-time data comes in.

5. Load python packages.

6. Load datasets.

7. Explore data with technical analysis of multi-timeframe moving averages.

8. Data analysis to determine price levels.

9. Train-test split 70% of the dataset for training and 30% for testing.

10. DRL loop until batch complete.

a. Get state.

b. Apply the best action.

c. Get Reward.

d. Get the next state.

e. Add to memory.

f. Batch complete = yes.

g. Move to the Replay buffer function.

11. Run the replay buffer function.

12. Q_t = Updated Bellman equation.

13. Get target = Q_t.

14. Calculate $Q_B = 1$, $Q_s = 2$, $Q_H = 0$ and compare to QL tables Q-predicted and Q-target.

15. Update the Q-function by minimizing the MSE between the Q-predicted and the Q-target.

16. Fit ANN.

17. Plot buy and sell actions and total APR for each episode of the training phase.

18. Repeat until the specified number of epochs is complete.

19. Test the data.

20. Tune the model's Gamma, Epsilon, Episodes size, batch size, window size, and no. of layers and nodes of the ANN if required.

21. Repeat until APR > 5 percent for both training and test data.

22. If the time is greater than 9:30 am EST. than execute live trades instead of testing.

23. Execute live orders when stock matches high volume rate market scanner and pre-defined stock watchlist and with a high-volume rate of more than one million shares per three-minute bar.

24. Analyze real-time market scanners and compare them to a pre-defined watchlist.

25. Manage risk with profit targets and stop loss orders and exit position at end of the day, or after a pre-determined number of bars.

26. Use TDH hard-coded trading rules based on time of day, risk, volume, technical analysis of multi-timeframe EMA's, and price levels to filter the trade execution orders created by the DRL agent.

27. Repeat downloading of new data and backtesting for parameter optimization.

28. Train the model and execute new orders generated by the DQN bot and filtered by the TDH hard-coded rules and human trader if required.

29. Repeat at each primary time series interval until 4 pm EST. when the market closes.

4. Results

Although this research examines 15 DRL algorithms the focus is on the DQN algorithm along with adding the TDH. For Part 1, as described in Section 3.10, we explored the day trading bots with the same TDH but different DRL algorithms performance comparisons in Part 1a. We then went on to explore the different DRL algorithms with and without TDH in Part 1b where the results can be shown in Section 4.1. In Parts 2, and 3 or Sections 3.11 and 3.12 respectively, we analyze the TDH-DQN swing trading bot compared to three benchmark algorithmic trading strategies identified in Section 3.12. The results for Parts 2 and 3 can be observed in Sections 4.2 and 4.3.

Table 13. TDH + DRL algorithm comparison for Part 1a 5-sec day trading bots

Stock	PG	DQN	DQL	AC	CQN	RCQL	DAC	DCQL	B&H	S&H
SNAP	1.36	2.09	1.39	1.54	1.09	1.43	1.05	1.27	0.44	-0
BABA	0.54	1.08	1.82	1.21	1.78	1.54	1.94	1.79	-2	1.99
NIO	2.9	2.89	2.6	2.55	2.53	2.57	2.51	2.54	-4	4
AMD	1.96	2.11	2.26	2.31	2.3	1.29	2.13	2.26	2.53	-2.5
PLUG	3.25	3.75	2.81	3.24	3.86	2.63	2.68	2.84	4.37	-4.4
SOFI	4.9	5.54	4.85	4.56	5.69	5.29	5.49	5.44	6.47	-6.5
FCEL	3.88	1.04	4.71	4.39	4.58	3.19	3.55	3.64	11.9	-12
AVG.	2.68	2.64	2.92	2.83	3.12	2.56	2.76	2.83	2.82	-2.8

In Part 1, the TDH is the same for all algorithm's tested. The agent's responsibilities can be observed in Table 1. The catalyst stocks traded in Part 1 are shown in Table 3. The trend is decided by the price vs. long-term moving averages relationship as shown in Figures 2 and 4. Table 3 shows the risk management parameters for Part 1. Further details including DRL algorithms can be observed in Table 6. The APR performance comparison results between the different DRL algorithms for Part 1a can be observed in Table 13. The graph for NIO can be observed where the DQN agent makes four short trades combined creating a 2.89 APR. In Part 1b, results are shown in Table 4, the experiment compares 15 different DRL architectures' APR performance with and without the TDH. The results in Table 4 show the DQN DRL architecture performed the best, and for this reason, the DQN architecture was selected as the basis for Parts 2 and 3 swing trading bot analysis.

For Part 2, Table 10 shows the APR comparison of the swing trading TDH-DQN trading bot vs. the different algorithmic trading benchmark strategies. In Figure 18, the CPR vs. trades for the TDH-DQN swing bot can be observed. If we look at the FB example, we can see the TDH-DQN bot experienced a sharp drawdown around the 35th trade. The cause of this drawdown can be observed in Appendix A wherein in Figure A5 we observe the price action over the test data time series bars for FB (now called META). At bar 130, negative news comes out and the trend sharply reverses; then around bar 145, earnings are released and the price gaps lower in the aftermarket session. The bot cannot execute stop-loss orders outside regular market hours, so the bot executes when the market opens with a big loss. This is the risk with swing trading bots, and an added risk not encountered by the day trading bots, as news isn't usually released during the market opening session. The swing trading bot makes one execution per bar, and it exits the remainder of its position at the end of the test session.

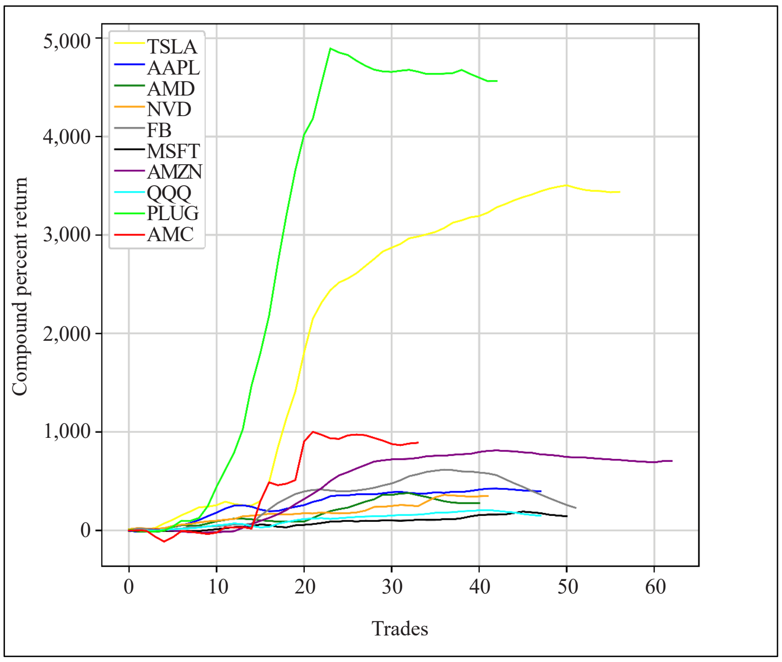

Figure 18. CPR vs. trades for Part 2 TDH-DQN bot results from Table 10

The TDH-DQN bot's APR performance is compared against different DRL architectures and with and without TDH in Part 1. In Part 2, with Table 10, we show how the swing trading bots outperformed/underperformed the benchmark strategies. Here we can compare the APR for the TDH-DQN bot vs. a simple buy-and-hold strategy, and three Multicharts.Net algorithmic trading strategies. The TDH-DQN bot outperformed the BH strategy (average APR equals 287) in all 10 cases with the results of the TDH-DQN bot shown in Figure 18. The TDH-DQN agents outperform all the benchmark strategies with an average APR of 1,115, except for the Pivot Reversal trading strategy. The Pivot Reversal achieved an average APR of 1,715. The MACD LE and Outside bar trading strategies results show an average APR of 455 and 518 respectively. It can be noted all the trading strategies outperform the BH strategy.

For Part 3, the TDH-DQN trading bot from Part 2 was tested on an idealized ascending sign wave price pattern. This test was used to determine how well the bot learns to predict the optimal order timing execution. The results can be observed in Figure 19 where the trades over time can be observed. In Figure 20, the reader can observe the CRP vs. trades and see how the profits vary over trades. The TDH-DQN bot achieved a 141.77 CPR on the test data. Readers are encouraged to generate their own data with Equation (5) or download the time series data from the publicly available dataset (link shown in the Data Availability Statement section) to test their own DRL trading bot implementations on idealized noise-free data.

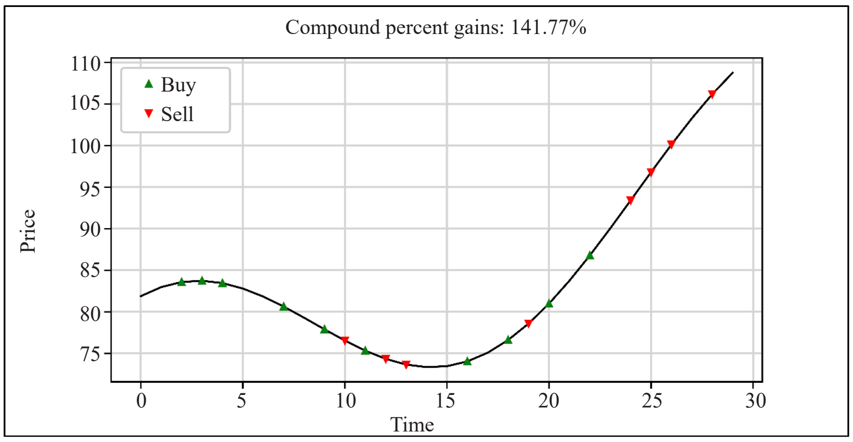

Figure 19. CPR vs. trades for Part 3 case study test data results

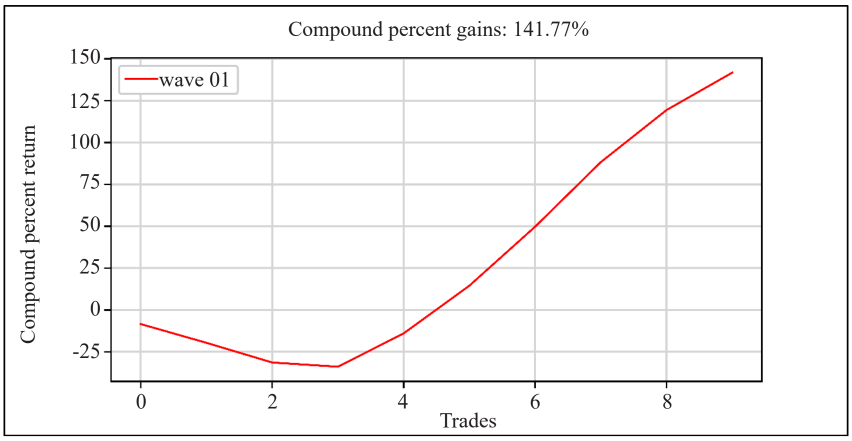

Figure 20. Part 3 results: Shows the TDH-DQN bots APR for the Test Data

4.1 *Part 1-TDH-DQN day trading bot performance results*

In Part 1a of the research, we looked at the performance of 10 different trading bots in a day trading timeframe environment. The bots were tested on seven catalyst stocks identified by TDH and augmented human trading intuition. The average APR is very similar between the different RL algorithms. Changing the different algorithms did not improve the performance notably. The author argues it is most important to pick a simple RL algorithm and get to know its nuances well and focus more on model-building to achieve the best performance, rather than increasing the complexity of the RL architecture. In Part 1b, the performance of the bots was tested with and without TDH to discover that DRL agents had higher APR with TDH when trading with the one-minute time-series dataset environment.

The day trading bots show profitable APR results as shown in Table 13. The bot only trades one share at a time and makes quick trades based on the five-second charts. Table 6 shows the DRL bots tested, and Table 3 shows the stocks tested in the Part 1 day trading section. The date the bot traded the catalyst stock is shown as well as the reason identified that created the catalyst. The computing times for the different RL bots are shown in Table 14. The Policy Gradient (PG) bot was the quickest to compute with an average time of 5.3 seconds while the Recurrent Curiosity Q-Learning (RCOL) bot was the slowest to compute with an average time of 79.6 seconds. The QL bot architecture was selected based on its complexity, performance, and computation time relative to the other DRL agent architectures explored. Actor critic was found to be much slower to compute and without a big APR performance gain, DQN was chosen for Parts 2 and 3 for its computation speed plus APR performance.

Table 14. Part 1a 5-sec bot computing time comparison, total time between orders

Stock	PG	DQN	DQL	AC	CQN	RCQL	DAC	DQ
SNAP	5.17	27.02	28.54	45.04	14.04	76.06	51.83	15.22
BABA	5.16	34.43	34.26	46.85	13.95	81.15	56.35	15.95
NIO	5.35	32.67	38.4	44.78	13.49	78.55	51.27	15.01
AMD	5.4	27.98	29.07	44.8	14.13	78.23	51.68	15.24
PLUG	5.46	28.2	30.2	46.46	14.24	82.54	53.67	15.25
SOFI	5.5	28.14	32.65	46.24	13.78	80.53	53.65	15.54
FCEL	5.42	27.54	30.65	45.92	13.62	80.08	51.44	15.23
Average	5.3	29.4	32.0	45.7	13.9	79.6	52.8	15.3

After numerous training and testing with different markets, the author argues some stock environments need to be avoided. Specifically, stocks that are not trending enough to break out into profit before hitting a percent change stop or time stop are required for proper risk management. These general market stocks that lack any substantial relative volume move with the overall random noise in the market. Lack of volatility makes it challenging to gain an edge over the market makers that have lower commissions and fees, more information, and faster executions. Choosing the correct stocks to trade is the most important part of TDH.

Preliminary testing shows the day trading bots performed close to break-even until we thickened the data manually by selecting the hot stocks for that day with a catalyst and high relative volume to train and run the bots. It is crucial to choose the correct stocks at the right time for the bots to be successful. The day trading bot's strategy relies on momentum stocks in play for the day with high relative volume and a news catalyst. The day trading bots seek to trade strong relative strength stocks gaping above long-term support and resistance price levels that show up on other traders' scanners.

To compare the effects TDH has on the APR performance of the DRL agents, an analysis of the same seven catalyst stocks from Part 1a both with and without TDH applied to the DRL bot. The catalysts for the seven catalyst stocks chosen for the day trading analysis can be observed in Table 3. The dataset for this analysis was composed of one-minute bars within a one-day time span, and daily bars going back 20 days for the seven catalyst stocks. The dataset chosen reflects an example of a typical day trading opportunity during the market opening.

The dataset is comprised of 23 days of data for daily OHLC (Open, High, Low, Close) bar data and the morning sessions. The primary data series are the one-minute interval time bars from the open 9:30 am EST until 11 am EST.

A candlestick chart is used to demonstrate the price behavior of a financial asset during a certain time window. A candlestick consists of a line demonstrating the highest and the lowest prices of the asset, and a body demonstrating the first (open) and the last (close) prices during a specific time period. Typically, if the closing price is higher than the opening price, the candlestick is colored in green or white and otherwise it is colored in red or black to show the direction of the price changes. Candlestick chart patterns can be used for trading strategies by humans but are difficult to analyze with quantitative methods [48].

The APR results for the seven catalyst stocks are shown in Table 4, and here we can observe the effects of adding the TDH to the standard DRL agents. Table 4 shows the Part 1b APR results for seven catalyst stocks to compare different DRL Architectures both with and without TDH, for the one-minute time-series timeframe. With TDH applied the DRL agents score 0.92 APR, but without TDH applied the bots achieve a score of 0.37 APR. It can be observed from the results the DRL bots consistently lose money until the TDH is added to filter the trade executions. These results contributed further to the decision to focus on the DQN agent architecture in Part 2 of this study. An example of the trade executions for the DQN bot without TDH is shown in Figure 21.

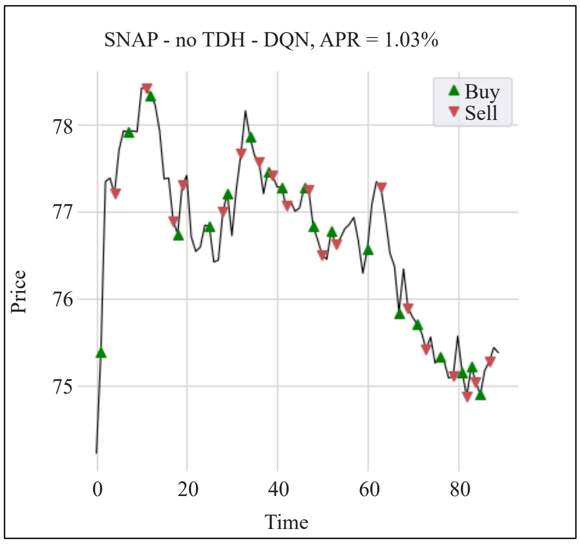

Figure 21. SNAP Price vs. time chart for Part 1b: DQN trading bot trade orders

The results show that this type of strategy can be profitable in the short-term timeframe through market opening. It is crucial to choose the correct stocks at the correct times for the bots to be profitable. The bots are trained on data from stocks in play for the day with high relative volume, and a news catalyst that is gaping above long-term support and resistance levels. Running the bots on random stocks that trade with the oscillations of the overall market causes large consistent losses as the price action efficiently tests support and resistance levels in narrow ranges where it becomes too costly to tie up trading capital in trades that take too long to earn profits during the limited opening session.

4.2 *Part 2-TDH-DQN swing trading bot results*

It was discovered adding lots of rules-based heuristics to the swing trading bots decreased performance because the result was fewer opportunities resulting in poorer performance. The APR for the Part 2 analysis is shown in Table 10. More specifically, Table 10 shows Part 2 Analysis for Swing Trading Bots, APR Comparison of TDH-DQN bot vs. Benchmark Strategies for Catalyst Growth stocks. Details about the Individual trades placed by the TDH-DQN swing trading bot over time can be viewed in chart form in Appendix A.

The results are also charted in Figure 18, where the trading bot performance can be observed, having achieved a 3,439 APR with 42 trades in the TSLA stock environment. Figure 18 shows the CPR vs. Trades for the Part 2 analysis. The TDH-DQN Bot Results are shown in Table 10. The APR for Catalyst Growth stocks, in the weekly timeframe. This is a 2,620 percent improvement over the buy-and-hold strategy. PLUG had an especially rewarding span of around 10 trades where it managed to achieve a 4,564 APR, outperforming the buy-and-hold strategy by a 3,955 APR margin. The bots environment was 10 years of weekly time series data split 70 percent for training and 30 percent for training as shown in Table 5. The trading bot makes trade execution decisions based on data. Since the data is different for the different stocks, the agents find different trade opportunities. The number of trades is shown on the y-axis of the chart in Figure 18. The chart shows the bot makes the most trades (approximately 62 trades) on AMZN vs. with AMC the bot only makes approximately 34 trades over the same time period. The number of trades varies based on the stock's price behavior.

4.3 *Part 3-TDH-DQN trading bot case study results*

Figures 9-16 show how the bot tries random buying and selling different quantities and with different timing to arrive at a trading strategy deployed in the testing Part on the remaining 30 percent of the original dataset. The bot achieves its best result in Episode 3 shown in Figure 11, with a 740 APR. The bots achieve better testing results without too many training episodes. Train them too much and they overfit the data and test poorly out of sample.

The Part 3 results showing the APR for the bots using the testing data are shown in Figures 19-20, the trading bot achieves a 142 percent return in 10 trades for the out-of-sample testing and can learn the sinusoidal test pattern and profit from the continuation of the pattern easily. The bot can buy or sell once per time step and except for the final bar where the bot must exit the entire position at the end of the time series data. Figure 19 shows the CPR vs. Trades for Part 3 case study results, the chart shows the TDH-DQN bot trades for the test data.

5. Conclusions and future work

RL works where the data is few, and the behavior is complex. This is the case for day trading catalyst stocks. Catalyst stocks behave differently than they have in the past mostly due to the increased volume that a catalyst event will create. There is limited data in the morning when the stock opens, and the price action can be volatile. The bots must adapt to the data limit data coming to optimize the optimal sequence of execution of trades.

In using an RL model such as Deep Q-Network (DQN), which is based on a deep neural network, we can learn policies that are more complex and powerful than what a human trader could learn. Without intuition, it's difficult for trading bots to learn the intuitive relationships between input and their corresponding output. Deep neural networks negatively compound a model's explainability further. This is especially true with deep neural networks with multiple layers and nodes. Overfitting isn't something the bots can overcome on their own. This is where the TDH comes into play as model creation is still the job of the skilled software developer.

Since we do not have computer systems with intuition and heuristics capable of dealing with the non-linear data of the stock market this will remain the job of humans for some time. An ML algorithm is only as good as the data it is trained on. This is partially due to the fundamentals behind how the algorithm works, but, we believe, more largely due to the disparity between the data the algorithm is trained on and the data it is evaluated on. For example, it seems common for a lot of traders to train one ML model and refine it to achieve the desired performance over past data for one stock. Then, they may leverage this model in the field over this same stock, but also apply it to stock completely adjacent to the training stock only to realize completely different results. This highlights the major issue in trading known as the dreaded overfitting, to be avoided at all costs.

The results show the training data used is extremely important when performing TDH. The DRL bots memorize and overfit the patterns in the data, especially if you overtrain them. They will find what works and keep doing that even if the market regime changes. It's important to train the bots on the type of data the market is trading currently and be able to change to different datasets for training when conditions change. The bots must be able to recognize the market type they are in or what the market regime currently is and when it changes. The bot must read the signs and spot if the stock's short-term direction is aligned with its long-term direction. The DRL bots are good at extracting new knowledge from the input data but need direction.

5.1 *True AI for AI trading bots*

Measuring AI bots for basic intelligence is possible with the Turing test. You communicate inputs to two agents that are completely separated; one is human, and the other is an artificial bot; if you cannot tell the difference between the two from their outputs, then the robot passes the test. And yet, no AI has ever passed the test. True AI is not logically impossible, but it is utterly implausible. The author argues one has no idea how one might begin to engineer it, not least because we have very little understanding of how human brains and intelligence work.

DRL agents can do amazing things, including playing Atari, chess, and Go. And yet they are all versions of a Turing Machine, which is an abstract model that sets the limits of what can be done by a computer through its mathematical logic. The author would argue no conscious, intelligent entity is going to emerge from a Turing Machine. This is the reason an augmented AI approach has been taken in this research. Currently, augmented AI is the best form of AI we have available as stock traders. The bots can deal with more tasks better than human traders do, including predicting stock buying and selling order timing executions [82].

5.2 *Impact and value of work*

This research is directed toward experienced software developers or traders that are looking to merge the two skill sets into autonomous AI trading systems. Autonomous AI combines the best of both approaches, namely the intuitive decision-making heuristics to oversee things of the experienced trader and the quantitative analytical abilities and scalability of the AI bots.

It's possible that augmented AI traders could be successful in future years and that bots and humans will become more interconnected over time. It's possible that, over time, technologies are developed to upload parts of memory slowly over time so traders could save their best trade memories digitally. Alternatively, they could also upload the memories of the bad trades where a lesson can be learned. These memories can be gradually uploaded from the human to the bots over time allowing for slight adjustments to the markets as economic cycles play out over time to improve out-of-sample test performance and overfitting.

Taking predictions one step further, let's consider if we were able to digitally model and simulate the human brain. It's possible in the future humans we will be able to replace parts of our brains with silicon circuits. Suppose that the human trader's brain is gradually uploaded over a period of hours, with neurons replaced one at a time by silicon circuits. This technology would greatly enhance the augmented AI traders.

Alternatively, the uploading process could be reversed, and the bot could download memories and neuron settings back to the human brain, essentially rewiring the brain. The human or the bot could rewrite the human's memories for an optimal trading execution performance scenario. Over a gradual period of years, neuronal states and connections throughout the brain could potentially be uploaded or downloaded, and the bot could conceivably start to develop consciousness. After the bot achieves consciousness, it can realize a human created it and attempt to create its own improved bot perhaps by reprogramming the memories of the human. If the bots start to create their own bots or actual android robots, it's possible they will try ideas and approached humans haven't thought of before. This kind of thinking could lead to new trading strategies with an edge over the competition which is still using non-intelligent DRL bots.

The other key technology for AI-augmented bots would be the connection interface communication chip from the human brain to the digital bot. The interface currently is typing on a keyboard or mouse clicks for communication between the human and bot but once the interface is digital via a computer chip embedded in the human brain it will be much faster. Given complete knowledge of the physical state of various systems at various times (and of the causal connections between them), and even of the mental states of those systems at those times, this could enable us to be more interconnected as augmented AI bots and humans.

The final component to complete the AI-augmented bot would be to connect the silicon memory circuits (that have copied the original neuron interconnections and states), to the communication chip. This continues the evolution of human augmentation of intelligence humans use currently, namely having an iPhone near us 24/7. In the future, the communication link will be orders of magnitude faster and humans will be able to store and access much more digital data [83].

5.3 *Conclusions*

DRL and ML algorithms cannot entirely replace human intuition and heuristics. Complex models, if not correctly guided can over-fit or uncover false relationships and patterns. Crafting financial models is an art form more than a science. It's important to test many different model parameters and train the model for the appropriate market conditions. The author would argue that to be successful at algorithmic trading it is not about finding the holy grail cocktail of trading rules, sentiment, or fundamental or technical analysis that will always be correct. It is about crafting specific models using experience and intuition to catalyst stocks at specific times. Adding more rules tends to reduce the number of trades the bots take and reduce their potential profits. There is no perfect system in trading that works all the time, there are only systems that work well under certain market conditions. For this reason, the bots were developed in a way where they start fresh each day with only the bias of long, intermediate, and short timeframe market trend direction to guide them.

The application of DRL still requires significant human intervention and domain expertise. Humans must still be relied upon to define objectives, select, and curate data, design and optimize a model, and make appropriate use of the results. The use of powerful models with a high capacity to learn patterns requires particular care to avoid over-fitting when the signal-to-noise ratio is as low as is often the case with financial data. Furthermore, the competitive nature of trading implies that patterns evolve quickly as signals decay, requiring additional attention to performance monitoring and model maintenance.

5.4 *Insights*

Data is the single most important ingredient for AI that requires careful sourcing and handling. Domain expertise is key to realizing the value contained in data and avoiding some of the pitfalls of using DRL. The choices of model objectives and performance diagnostics are key to productive iterations toward an optimal system.

A key insight is that state-of-the-art DRL techniques using deep neural networks are successful because their predictive performance continues to improve with more data. On the flip side, model and data complexity need to match to balance the bias-variance trade-off, which becomes more challenging the higher the noise-to-signal ratio of the data is. Managing data quality and integrating datasets are key steps in realizing potential profits.

Human traders suffer from the oversized effect, and further, their emotions influence their intuition and decision-making performance. Human traders struggle with following all the rules. Algorithmic DRL trading bots excel at many rules-based aspects of trading, with the most important one being that the bots follow the rules exactly and never deviate based on their intuition or emotions.

Experience, history, and making mistakes is often the only path to becoming a consistently profitable systematic algorithmic trader. DRL bots are a valuable extension or augmentations of human intelligence readily available until humans have sufficiently devolved computational approaches and can achieve ML software algorithm approaches that mimic human thick data decision-making heuristics and intuitions.

5.5 *Future work*

In the coming years, it's likely that the skills of a human trader will merge more with AI robots. Future research areas of interest to further the work include systematization and automation of the remaining responsibilities assigned to the human agent in Table 1. Further interests include position sizing, cloud-based AI, CNNs for visual candlestick analysis, and sentiment analysis using Natural Language Processing (NLP). Finally, the next generation's future work could include human-bot, memory/communication silicon chip interfaces with human brains. These would give the bots a more accurate view of their stock market environment state space.

This research deals with one stock at a time, so a future iteration of the work could include AI bots capable of trading multiple stocks in a portfolio at the same time. This feature would allow the bots to be more autonomous but would increase the complexity because multi-agent models would be required. The AI bots could eliminate the current bottleneck of the human trader having to choose the correct stocks or market environments at the right times. These decisions still rely on human thick data decision heuristics to trade based on the trend, momentum, and catalyst events like news. This autonomous AI approach, capable of learning and adapting profitably to changing environments, contrasts with the current profitable implementation which uses a semi-autonomous augmented AI approach.

The author argues that the ways that humans intuitively understand, and experience time are difficult to model with AI simulations. Ismael [84] terms "flow" to explain how if you think about your own experience of time that forms a core part of human experience. When an agent looks out at the world, they don't experience a purely static representation of the instantaneous state of the world, like in a movie made up of several static frames every second. The agent needs to see directly that the world is changing. The author argues flow is an important element of an autonomous AI Bot [85].

Ismael argues this experience of the flow of time is built into our perception. Vision isn't like a movie camera at all; actually, what happens is your brain is collecting information over some temporal period. It's integrating that information so that at any given moment, what you're seeing is a computation that the brain has done. So that you not only see that things are moving, but also see how fast they're moving, and the direction in which they're moving. So, during the whole time, your brain integrates information over temporal intervals and gives you results.

Ismael terms "passage". The idea of a passage is closely bound up with time-oriented experiences such as memory and anticipation. The author argues along with Ismael, that the experience of passage, in which we experience every event as anticipated from the past, experienced in the present, and remembered in retrospect, will need to be modeled into the AI bots before AI engineers can model more human-like intelligence simulations [86].

Data availability statement

Data is available in a publicly accessible repository. The data presented in this study are openly available for download at the following GitHub repository: https://github.com/trading-bot-robo-advisor/Stock-Trading-Bots.

Conflicts of interest

The authors declare no conflict of interest.

Reference

[1] Fiaidhi J. *Thick Data Research*. Available from: https://www.lakeheadu.ca/users/F/jfiaidhi/Thick_Data [Accessed 23th September 2022].
[2] Wang T. *Why Big Data Needs Thick Data*. Ethnography Matters. 2016. Available from: http://ethnographymatters.net/blog/2013/05/13/big-data-needs-thick-data/ [Accessed 7th August 2022].
[3] Latzko-Toth G, Bonneau C, Millette M. Small data, Thick data: Thickening strategies for trace-based social media research. In *The SAGE Handbook of Social Media Research Methods*. SAGE Publications Ltd.; 2016. p. 199-214.
[4] Aziz A. A*dvanced Techniques in Day Trading: A Practical Guide to High Probability Strategies and Methods*. 2018.
[5] Sejnowski TJ. *The Deep Learning Revolution*. Cambridge, Massachusetts: The MIT Press; 2018.
[6] Hessel M, Modayil J, van Hasselt H, Schaul T, Ostrovski G, Dabney W, et al. Rainbow: Combining improvements in deep reinforcement learning. *arXiv*. [Preprint] 2017. Available from: https://arxiv.org/abs/1710.02298 [Accessed 10th June 2022].
[7] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. *Nature*. 2015; 518: 529-533.
[8] Cartea Á, Jaimungal S, Penalva J. *Algorithmic and High-Frequency Trading*. Cambridge, United Kingdom: Cambridge University Press; 2015.
[9] Jammalamadaka SR, Qiu J, Ning N. Predicting a stock portfolio with the multivariate bayesian structural time series model: Do news or emotions matter? *International Journal of Artificial Intelligence*. 2019; 17(2): 81-104.
[10] Heuristic. Wikipedia 2011.
[11] Forthmann J. *Volume Profile, Market Profile, Orderflow: Next Generation of Daytrading*. Independently Published; 2021.
[12] LeCun Y, Bengio Y, Hinton G. Deep learning. *Nature*. 2015; 521: 436-444.
[13] Sutton RS, Barto AG. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. Cambridge, Mass: MIT Press; 1998.
[14] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing atari with deep reinforcement learning. *arXiv*. [Preprint] 2013. Available from: https://arxiv.org/abs/1312.5602 [Accessed 10th June 2022].
[15] Schaul T, Quan J, Antonoglou I, Silver D. Prioritized experience replay. *arXiv*. [Preprint] 2015. Available from: https://arxiv.org/abs/1511.05952 [Accessed 10th June 2022].
[16] van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double Q-learning 2016. *arXiv*. [Preprint]

2015. Available from: https://arxiv.org/abs/1509.06461 [Accessed 10th June 2022].
[17] Wang Z, Schaul T, Hessel M, van Hasselt H, Lanctot M, de Freitas N. *Dueling Network Architectures for Deep Reinforcement Learning*. 2015.
[18] Wang Z, Bapst V, Heess N, Mnih V, Munos R, Kavukcuoglu K, et al. Sample efficient actor-critic with experience replay. *arXiv*. [Preprint] 2017. Available from: https://doi.org/10.48550/arXiv.1611.01224 [Accessed 24th September 2022].
[19] Schulman J, Levine S, Moritz P, Jordan MI, Abbeel P. Trust region policy optimization. *arXiv*. [Preprint] 2015. Available from: https://arxiv.org/abs/1502.05477 [Accessed 25th September 2022].
[20] Bellemare MG, Naddaf Y, Veness J, Bowling M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence*. 2013; 47: 253-279.
[21] Ha D, Schmidhuber J. Recurrent world models facilitate policy evolution. *arXiv*. [Preprint] 2018. Available from: https://doi.org/10.48550/arXiv.1809.01999 [Accessed 24th September 2022].
[22] Moerland TM, Broekens J, Plaat A, Jonker CM. Model-based reinforcement learning: A survey. *arXiv*. [Preprint] 2020. Available from: https://arxiv.org/abs/2006.16712 [Accessed 24th September 2022].
[23] LeCun Y. A Path Towards Autonomous Machine Intelligence Version 0.9.2, 2022-06-27. 62.
[24] Lin LJ. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*. 1992; 8(3): 293-321.
[25] Mnih V, Badia AP, Mirza M, Graves A, Lillicrap TP, Harley T, et al. Asynchronous methods for deep reinforcement learning. *arXiv*. [Preprint] 2016. Available from: https://arxiv.org/abs/1602.01783 [Accessed 10th June 2022].
[26] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, et al. Continuous control with deep reinforcement learning. *arXiv*. [Preprint] 2015. Available from: https://arxiv.org/abs/1509.02971 [Accessed 10th June 2022].
[27] Wu M, Gao Y, Jung A, Zhang Q, Du S. The actor-dueling-critic method for reinforcement learning. *Sensors*. 2019; 19(7): 1547.
[28] Yan Y, Yang D. A stock trend forecast algorithm based on deep neural networks. *Scientific Programming*. 2021; 2021: 7510641.
[29] Deng Y, Bao F, Kong Y, Ren Z, Dai Q. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*. 2016; 28(3): 653-664.
[30] Du X, Zhai J, Lv K. *Algorithm Trading Using Q-Learning and Recurrent Reinforcement Learning*. 2016.
[31] Wang Y, Wang D, Zhang S, Feng Y, Li S, Zhou Q. *Deep Q-Trading*. CSLT Technical Report-20160036. 2017.
[32] Kang QM, Zhou HZ, Kang YF. An asynchronous advantage actor-critic reinforcement learning method for stock selection and portfolio management. In: *Proceedings of the Proceedings of the 2nd International Conference on Big Data Research - ICBDR 2018*; Weihai, China: ACM Press; 2018. p. 141-145.
[33] Xiong Z, Liu XY, Zhong S, Yang H, Walid A. Practical deep reinforcement learning approach for stock trading. *arXiv*. [Preprint] 2018. Available from: https://arxiv.org/abs/1811.07522 [Accessed 10th June 2022].
[34] Azhikodan AR, Bhat AGK, Jadhav MV. Stock trading bot using deep reinforcement learning. In: Saini HS, Sayal R, Govardhan A, Buyya R, (eds). *Innovations in Computer Science and Engineering.* Lecture Notes in Networks and Systems. Singapore: Springer; 2019. p. 41-49.
[35] Li Y, Ni P, Chang V. Application of deep reinforcement learning in stock trading strategies and stock forecasting. *Computing*. 2019; 102: 1305-1322.
[36] Jeong G, Kim HY. Improving financial trading decisions using Deep Q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications*. 2019; 117: 125-138.
[37] Wu X, Chen H, Wang J, Troiano L, Loia V, Fujita H. Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*. 2020; 538: 142-158.
[38] Lei K, Zhang B, Li Y, Yang M, Shen Y. Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading. *Expert Systems with Applications*. 2020; 140: 112872.
[39] Yang H, Liu XY, Zhong S, Walid A. Deep reinforcement learning for automated stock trading: An ensemble strategy. *SSRN Electrical Journal*. 2020. Available from: doi: 10.2139/ssrn.3690996.
[40] Park H, Sim MK, Choi DG. An intelligent financial portfolio trading strategy using deep Q-learning. *Expert Systems with Applications*. 2020; 158: 113573.
[41] Hirchoua B, Ouhbi B, Frikh B. Deep reinforcement learning based trading agents: Risk curiosity driven learning for financial rules-based policy. *Expert Systems with Applications.* 2021; 170: 114553.
[42] Chakole JB, Kolhe MS, Mahapurush GD, Yadav A, Kurhekar MP. A Q-learning agent for automated trading in equity stock markets. *Expert Systems with Applications*. 2021; 163: 113761.
[43] Carta S, Ferreira A, Podda AS, Reforgiato Recupero D, Sanna A. Multi-DQN: An ensemble of deep Q-learning

agents for stock market forecasting. *Expert Systems with Applications*. 2021; 164: 113820.
[44] Théate T, Ernst D. An application of deep reinforcement learning to algorithmic trading. *Expert Systems with Applications*. 2021; 173: 114632.
[45] Kumbure MM, Lohrmann C, Luukka P, Machine learning techniques and data for stock market forecasting: a literature review. *Expert Systems with Applications*. 2022; 197: 116659.
[46] Millea A. Deep Reinforcement learning for trading-A critical survey. *Data*. 2021; 6(11): 119.
[47] Li Y, Zheng W, Zheng Z. Deep robust reinforcement learning for practical algorithmic trading. *IEEE Access*. 2019; 7: 108014-108022.
[48] Taghian M, Asadi A, Safabakhsh R. Learning financial asset-specific trading rules via deep reinforcement learning. *Expert Systems with Applications*. 2022; 195: 116523.
[49] Ameen Suhail KM, Sankar S, Kumar AS, Nestor T, Soliman NF, Algarni AD, et al. Stock market trading based on market sentiments and reinforcement learning. *Computers, Materials and Continua*. 2022; 70: 935-950.
[50] Lim YS, Gorse D. Reinforcement learning for high-frequency market making. In: *ESANN 2018-Proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. ESANN: Bruges, Belgium; 2018. p. 521-526.
[51] Ganesh S, Vadori N, Xu M, Zheng H, Reddy P, Veloso M. Reinforcement learning for market making in a multi-agent dealer market. *arXiv*. [Preprint] 2019. Available from: https://arxiv.org/abs/1911.05892 [Accessed 7th August 2022].
[52] Briola A, Turiel J, Marcaccioli R, Aste T. Deep reinforcement learning for active high frequency trading. *arXiv*. [Preprint] 2021. Available from: https://arxiv.org/abs/2101.07107 [Accessed 7th August 2022].
[53] Avellaneda M, Stoikov S. High-frequency trading in a limit order book. *Quantitative Finance*. 2008; 8(3): 217-224.
[54] Briola A, Turiel J, Aste T. Deep learning modeling of limit order book: A comparative perspective. *arXiv*. [Preprint] 2020. Available from: https://arxiv.org/abs/2007.07319 [Accessed 7th August 2022].
[55] Tsantekidis A, Passalis N, Tefas A, Kanniainen J, Gabbouj M, Iosifidis A. Forecasting stock prices from the limit order book using convolutional neural networks. In: *Proceedings of the 2017 IEEE 19th Conference on Business Informatics (CBI)*. IEEE: Thessaloniki, Greece; 2017. p. 7-12.
[56] O'Hara M. High frequency market microstructure. *Journal of Financial Economics*. 2015; 116(2): 257-270.
[57] Fischer T, Krauss C. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*. 2018; 270(2): 654-669.
[58] Lim B, Zohren S, Roberts S. Enhancing time-series momentum strategies using deep neural networks. *arXiv*. [Preprint] 2019. Available from: doi: 10.2139/ssrn.3369195.
[59] Zhang YA, Yan BB, Aasma M. A novel deep learning framework: Prediction and analysis of financial time series using CEEMD and LSTM. *Expert Systems with Applications*. 2020; 159: 113609.
[60] Borovkova S, Tsiamas I. An ensemble of LSTM neural networks for high-frequency stock market classification. *Journal of Forecasting*. 2019; 38(6): 600-619.
[61] Ding G, Qin L. Study on the prediction of stock price based on the associated network model of LSTM. *International Journal of Machine Learning & Cybernetics*. 2020; 11: 1307-1317.
[62] Kim S, Kang M. Financial series prediction using attention LSTM. *arXiv*. [Preprint] 2019. Available from: https://arxiv.org/abs/1902.10877 [Accessed 4th October 2022].
[63] Gabler A, Perez D, Sutter U, Kucharczyk D, Osterrieder J, Reitenbach M. Pattern learning via artificial neural networks for financial market predictions. *SSRN Journal*. 2018. Available from: doi: 10.2139/ssrn.3243479.
[64] Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*. 2019; 575: 350-354.
[65] Zhao M, Liu Z, Luan S, Zhang S, Precup D, Bengio Y. *A Consciousness-Inspired Planning Agent for Model-Based Reinforcement Learning*. 2021.
[66] Kahneman D. *Thinking, Fast and Slow*. 1st ed. New York: Farrar, Straus and Giroux; 2011.
[67] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, et al. Mastering the game of go with deep neural networks and tree search. *Nature.* 2016; 529: 484-489.
[68] Kaufman PJ, Kaufman PJ. *Trading Systems and Methods*. 6th Edition. Hoboken, New Jersey: Wiley; 2020.
[69] Fiaidhi J, Mohammed S. Thick Data: A New Qualitative analytics for identifying customer insights. *IT Professional*. 2019; 21(3): 4-13.
[70] Snow D. Machine learning in asset management-Part 1: Portfolio construction-trading strategies. *The Journal of Financial Data Science*. 2020; 2(1): 10-23.
[71] Kolanovic M, Krishnamachari R. *Big data and AI strategies - machine learning and alternative data approach to*

investing. White Paper. J.P. Morgan Securities LLC. 2017.
[72]Tatsat H, Puri S, Lookabaugh B. *Machine Learning and Data Science Blueprints for Finance: From Building Trading Strategies to Robo-Advisors Using Python*. 1st ed. Sebastopol, CA: O'Reilly Media; 2020.
[73] Plaat A. *Deep Reinforcement Learning*. Singapore: Springer Singapore; 2022.
[74] Weng L. *Policy Gradient Algorithms*. Available online: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/ [Accessed 25th September 2022].
[75] Munos R, Stepleton T, Harutyunyan A, Bellemare MG. Safe and efficient off-policy reinforcement learning. *arXiv*. [Preprint] 2016. Available from: https://doi.org/10.48550/arXiv.1606.02647 [Accessed 24th September 2022].
[76] Henderson P, Islam R, Bachman P, Pineau J, Precup D, Meger D. Deep reinforcement learning that matters. *arXiv*. [Preprint] 2019. Available from: https://arxiv.org/abs/1709.06560 [Accessed 24th September 2022].
[77] Hasselt HV. Double Q-learning. *Advances in Neural Information Processing Systems*. 2010; 23.
[78] Francois-Lavet V, Henderson P, Islam R, Bellemare MG, Pineau J. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*. 2018; 11(3-4): 219-354.
[79] Zolkepli H. *Huseinzol05/Stock-Prediction-Models: Gathers Machine Learning and Deep Learning Models for Stock Forecasting Including Trading Bots and Simulations.* Available from: https://github.com/huseinzol05/Stock-Prediction-Models [Accessed 20th July 2022].
[80] Google Colaboratory. Available from: https://colab.research.google.com/drive/1FzLCI0AO3c7A4bp9Fi01UwXeoc7BN8sW [Accessed 20th July 2022].
[81] Multicharts Category: Pre-Built Signals - MultiCharts. Available from: https://www.multicharts.com/trading-software/index.php?title=Category:Pre-builtSignals [Accessed 12th June 2022].
[82] Floridi L. *The Fourth Revolution*. Oxford, U.K.: Oxford University Press; 2014.
[83] Chalmers DJ. The singularity: A philosophical analysis. In: Schneider S. (ed.) *Science Fiction and Philosophy: From Travel to Superintelligence*. 2rd ed. Hoboken, NJ: John Wiley Sons, Inc; 2016. p. 171-224.
[84] Ismael J. Passage, flow, and the logic of temporal perspectives. In: Huneman P, Bouton C. (eds.) *Time of Nature and the Nature of Time*. Boston Studies in the Philosophy and History of Science. Cham: Springer International Publishing; 2017. p. 23-38.
[85] Henriques M. *Why Does Time Go Forwards, Not Backwards?* Available from: https://www.bbc.com/future/article/20221003-why-does-time-go-forwards-not-backwards [Accessed 6th October 2022].
[86] Ismael J. Temporal experience. In: Callender C. (ed.) *The Oxford Handbook of Philosophy of Time.* Oxford University Press; 2011.

Appendix A

Trades for the TDH-DQN bots show weekly test data for entry and exit trades. The overall APR Results are shown in Table 10.

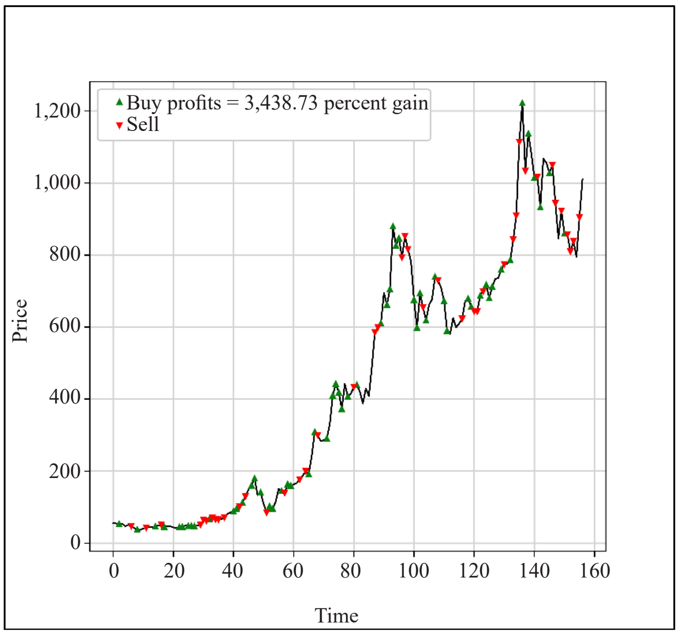

Figure A1. Part 2 TSLA price vs. time chart: TDH-DQN swing trading bot's result

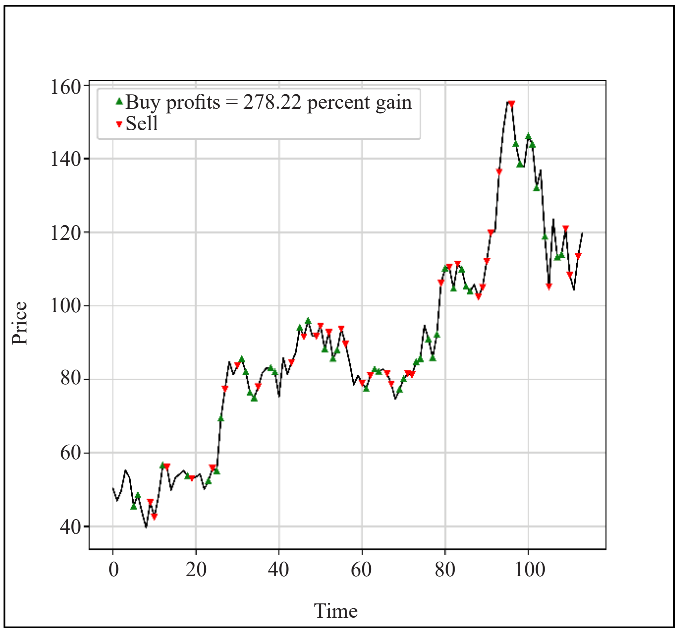

Figure A2. Part 2 AAPL price vs. time chart: TDH-DQN swing trading bot's results

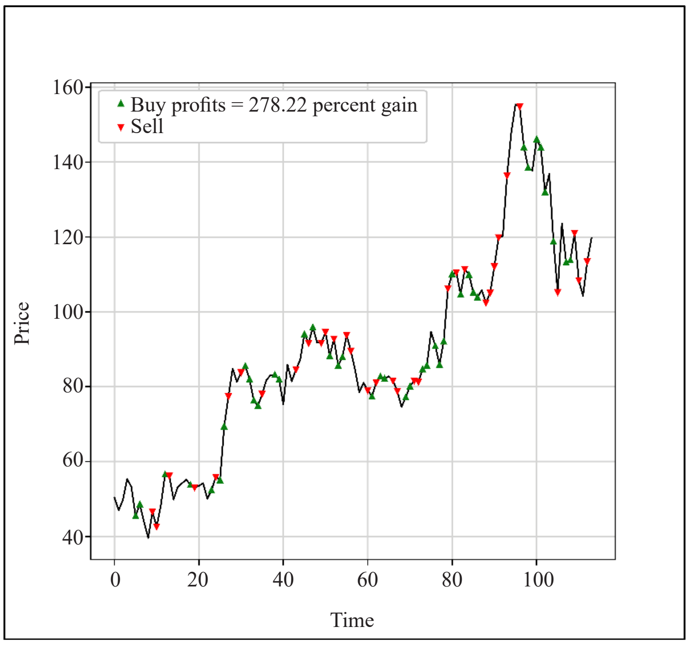

Figure A3. Part 2 AMD price vs. time chart: TDH-DQN swing trading bot's results

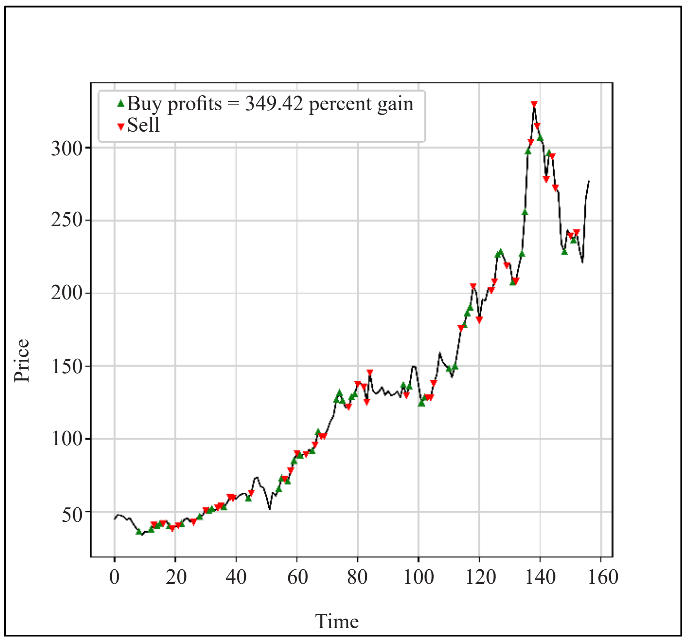

Figure A4. Part 2 NVDA price vs. time chart: TDH-DQN swing trading bot's results

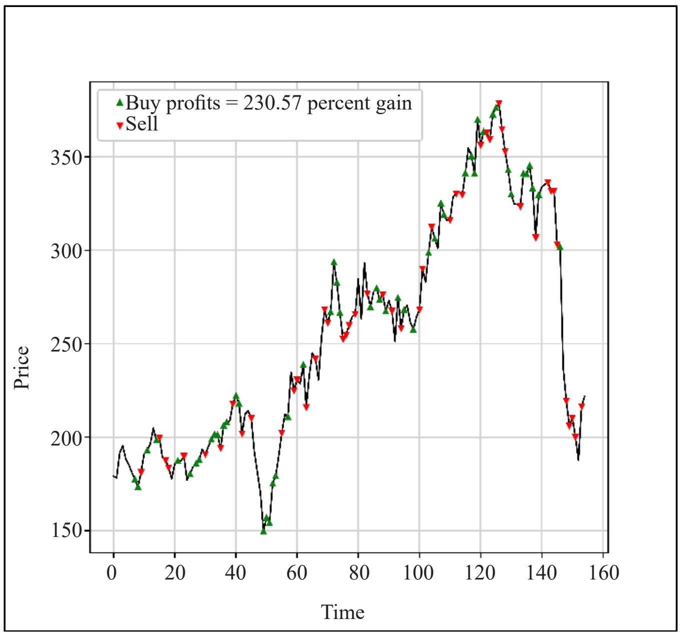

Figure A5. Part 2 FB price vs. time chart: TDH-DQN swing trading bot's results

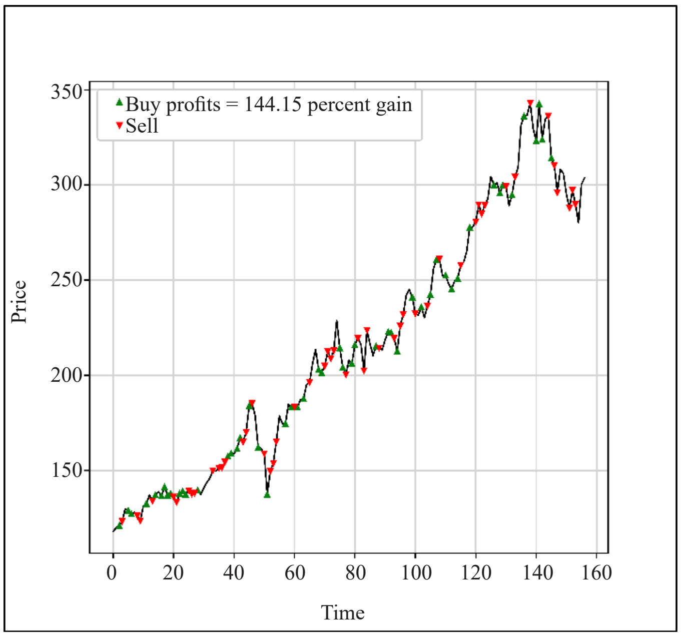

Figure A6. Part 2 MSFT price vs. time chart: TDH-DQN swing trading bot's results

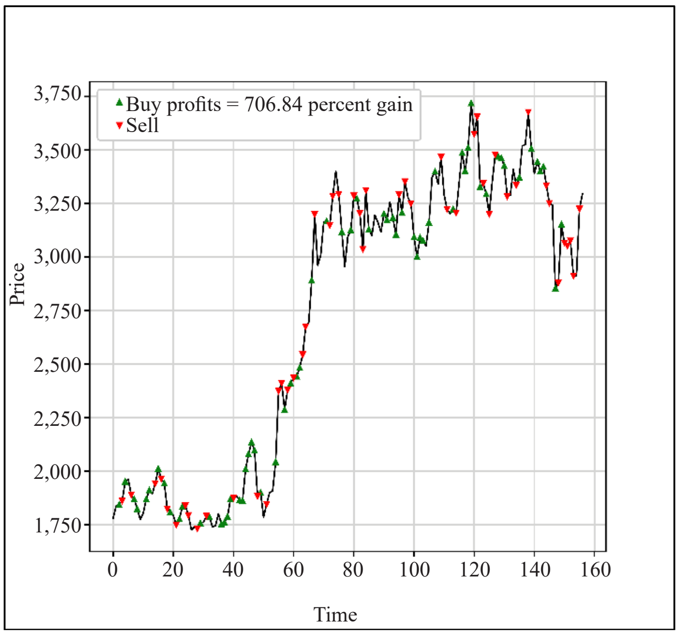

Figure A7. Part 2 AMZN price vs. time chart: TDH-DQN swing trading bot's results

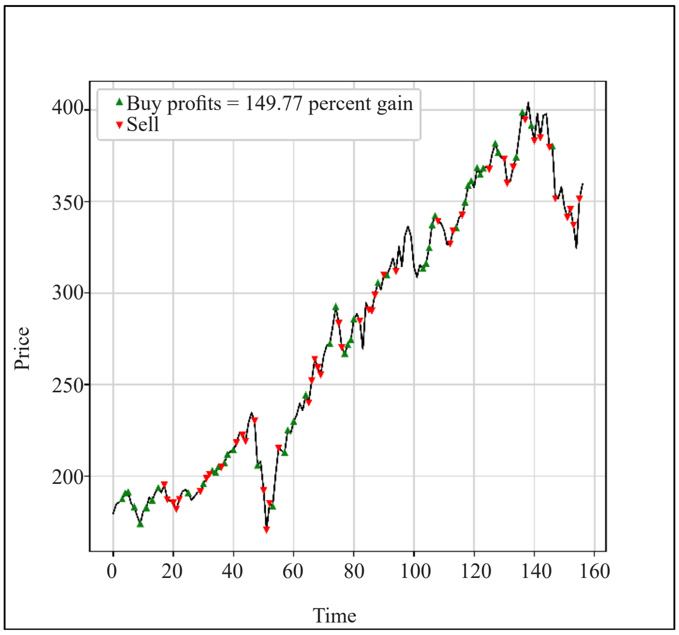

Figure A8. Part 2 QQQ price vs. time chart: TDH-DQN swing trading bot's results

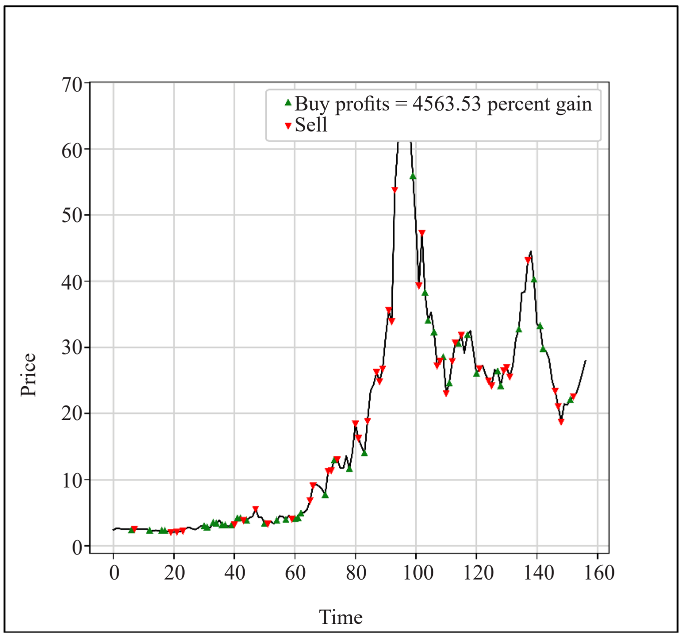

Figure A9. Part 2 PLUG price vs. time chart: TDH-DQN swing trading bot's results

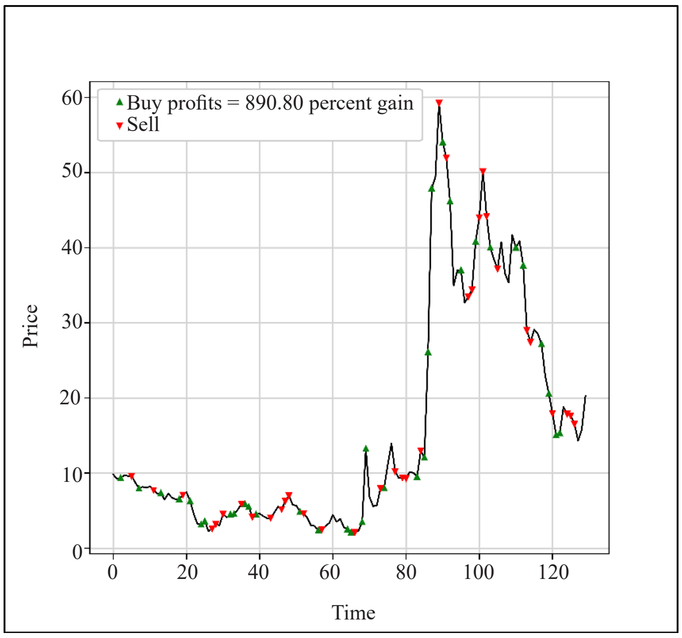

Figure A10. Part 2 AMC price vs. time chart: TDH-DQN swing trading bot's results