

Convolutional Attention in Betting Exchange Markets

Rui Gonçalves^a, Vitor Miguel Ribeiro^b, Roman Chertovskih^a, António Pedro Aguiar^a

^a SYSTEC-ARISE Research Center for Systems and Technologies, Faculty of Engineering, University of Porto

^b Department of Economics, School of Economics and Management, Porto University of Porto

Abstract

This study presents the implementation of a short-term forecasting system for price movements in exchange markets, using market depth data and a systematic procedure to enable a fully automated trading system. The case study focuses on the UK to Win Horse Racing market during the pre-live stage on the world's leading betting exchange, Betfair. Innovative convolutional attention mechanisms are introduced and applied to multiple recurrent neural networks and bi-dimensional convolutional recurrent neural network layers. Additionally, a novel padding method for convolutional layers is proposed, specifically designed for multivariate time series processing. These innovations are thoroughly detailed, along with their execution process. The proposed architectures follow a standard supervised learning approach, involving model training and subsequent testing on new data, which requires extensive pre-processing and data analysis. The study also presents a complete end-to-end framework for automated feature engineering and market interactions using the developed models in production. The key finding of this research is that all proposed innovations positively impact the performance metrics of the classification task under examination, thereby advancing the current state-of-the-art in convolutional attention mechanisms and padding methods applied to multivariate time series problems.

Keywords: Deep Learning, Betting Exchange, L3 Market Data, Classification.

JEL Classification: G17, D10, L10.

1. Introduction

Automated trading systems have revolutionized financial markets, with applications that span various domains, including exchange markets. One particularly interesting and evolving area is the betting exchange market, where predicting price movements can lead to profitable strategies. This study focuses on implementing a fully integrated short-term forecasting system to predict future price movements in exchange markets. To meet this purpose, Artificial Intelligence (AI) automated trading agents are developed to execute trades in the United Kingdom (UK) to Win Horse Racing market during the pre-live stage on Betfair, the world's largest betting exchange platform. These agents operate within a 10-minute window before a race begins, where odds (i.e., prices) are subject to speculation, having in mind the objective to generate profit by buying and selling bets at different odds.

The focus of this study is on the 10-minute window before a race begins, with the goal of accurately classifying the price movement in the final 2 minutes before the race begins, based on in-

*This work was financially supported by UID/00147 – Systems and Technologies Center (SYSTEC), with the support of the Associate Laboratory Advanced Production and Intelligent Systems (ARISE), LA/P/0112/2020 (DOI: 10.54499/LA/P/0112/2020), both funded by national funds through FCT/MCTES (PIDDAC).

Email addresses: rjpg@fe.up.pt (Rui Gonçalves), vsribeiro@fep.up.pt (Vitor Miguel Ribeiro), roman@fe.up.pt (Roman Chertovskih), apra@fe.up.pt (António Pedro Aguiar)

formation from previous minutes. There is no sliding window overlap, meaning each race serves as an individual example. By combining prediction models with trading strategies, this study evaluates the performance of the entire system and assesses the combined impact of individual components, as suggested by Gonçalves et al. (2013) and Gonçalves et al. (2019). A complete end-to-end framework has been developed, involving the preparation of complex market data, predictive model training, and the deployment of several Deep Learning (DL) Neural Network (NN) models within a fully automated trading system. The software implementation framework collects market depth data, performs Feature Engineering (FE), and operationalizes DL NN architectures for real-time use in trading simulations. Currently, the existing literature lacks end-to-end frameworks, with most research focusing on basic learning processes for prediction and/or classification tasks. In these studies, authors often claim the superiority of their models, yet fail to fully integrate them within a complete operational framework. Therefore, this study extends beyond indicator analysis, model building, and the typical train/test learning process to include a comprehensive evaluation of the overall performance of the trading system.

Moreover, even for the specific task of training and testing DL NN models, the aim of this study is to advance the understanding of whether novel convolutional attention mechanisms improve performance metrics (e.g., forecast accuracy) in the context of exchange markets and trading strategies. A specific objective of this study is to evaluate the performance of novel methodologies, including convolutional attention mechanisms, thereby contributing to the development of more robust, data-driven trading systems. In prediction and/or classification tasks addressed by existing literature, DL NN models possess inherent properties that make them particularly suitable for Multivariate Time Series (MTS) problems. Firstly, these models are robust to noise in input data and can support learning even in the presence of missing values (Dixon et al., 2015). Secondly, DL NN models do not impose strong assumptions about the underlying mapping function, allowing them to absorb both linear and non-linear relationships (Huck, 2009). This ability is crucial, as most real-life events involve complex, non-linear relationships (Huck, 2010). Thirdly, DL NN models exhibit generalization power, enabling them to recognize previously unobserved relationships in data after learning from a given set of inputs (Gonçalves et al., 2019). Fourthly, these models are flexible in their treatment of input data, not enforcing persistence of any specific distribution (Dorffner, 1996). Fifthly, they handle heteroskedasticity more effectively due to their ability to detect hidden relationships without the imposition of additional constraints (Gonçalves et al., 2023).

Unsurprisingly, a lively strand of research documents that DL NN architectures have been successfully implemented in various application fields (Hatcher & Yu, 2018), and financial exchange markets have recently started to benefit from the learning capability of these models. In general, researchers aim to use DL NN architectures to predict or classify trends and detect anomalies. Ding et al. (2015) assesses stock market price predictions by implementing a DL NN to learn event embeddings and a Convolutional Neural Network (CNN) for short-, medium-, and long-term analyses, improving accuracy and profit relative to basic NNs. Heaton et al. (2016) uses a DL auto-encoding technique based on Principal Component Analysis (PCA) for dimensionality reduction, allowing for feature extraction and defining a smart index for stocks. Similarly, Korczak & Hemes (2017) confirms that using a CNN within the H2O algorithmic trading framework significantly improved the average rate of return per transaction in the FOREX market. Additionally, Recurrent Neural Network (RNN)s in general and Long Short-Term Memory (LSTM) in particular are capable of learning temporal dependencies from context, being widely considered as benchmark models for MTS problems (Hochreiter & Schmidhuber, 1997).

Fischer & Krauss (2018) adopt LSTM networks for predicting stock price movements, showing that these layers had better predictive power than memory-free classification networks. In turn, Schnaubelt (2022) presents a Deep Reinforcement Learning (DRL) application to optimize execution

in cryptocurrency exchanges by learning optimal limit order placement strategies. Here, a specialized training environment is designed, incorporating a purpose-built reward function, market state features, and a virtual limit order exchange. Using 18 months of high-frequency data from major cryptocurrency exchanges – covering 300 million trades and 3.5 million order book states – DRL algorithms are compared against benchmarks. This study concludes that proximal policy optimization effectively learns superior order placement strategies, adapting aggressiveness based on execution probabilities influenced by trade and order imbalances. Moreover, Alfonso-Sánchez et al. (2024) apply DRL to automate optimal credit card limit adjustment policies. Using historical data from a Latin American super-app, the decision-making problem is framed as an optimization task balancing revenue maximization and provision minimization. An offline learning strategy is employed to train the DRL agent, with a Double Q-learning approach demonstrating superior performance over other strategies.

Kriebel & Stitz (2022) explore DL NN models and other techniques to extract credit-relevant information from user-generated text on Lending Club. Findings indicate that even short pieces of text significantly enhance credit default predictions. An information fusion analysis further confirms the value of textual data. DL generally outperforms other text-based approaches, and a comparison of six DL NN models architectures, including transformer models like BERT and RoBERTa, shows similar performance, suggesting that simpler methods, such as average embedding NNs, can be as effective as more complex models for credit scoring. Recently, Zhong et al. (2024) propose a distributed mean reversion online portfolio strategy using a stock correlation sub-network to address limitations in existing strategies, such as their lack of universality and the restriction on short selling. A theoretical analysis confirms its generalization and convergence rate. Empirical results demonstrate superior return performance compared to existing universal strategies while maintaining robustness against transaction costs, making it more suitable for real-time investment applications. In the context of exchange markets, Rzaev et al. (2025) evaluate the impact of adoption timing on cryptocurrency markets by decomposing total adoption into innovators (i.e., early adopters) and imitators (i.e., late adopters). Findings indicate that innovators drive the relationship between user adoption and cryptocurrency returns, enhancing price efficiency, whereas imitators contribute to price noise. Additionally, the adoption model effectively captures market phenomena such as herding behavior, improving cryptocurrency pricing forecasts. The proposed framework offers a valuable approach for studying market dynamics and can be applied across various domains in financial and operational research.

In addition to establishing a fully integrated end-to-end framework absent in existing literature, this study presents two key findings demonstrating the power and applicability of convolutional attention mechanisms and roll padding in exchange market forecasting and classification tasks. First, results indicate that convolutional attention mechanisms significantly improve forecasting accuracy by effectively capturing complex patterns and dependencies in market data, enhancing prediction performance. Second, findings show that integrating these technical innovations into automated trading systems strengthens decision-making capabilities, leading to more effective and profitable trading strategies in exchange markets.

The study is organized as follows. Section 2 presents the case study, including the trading and software implementation frameworks, as well as data pre-processing steps and feature engineering. Section 3 provides an overview of all DL NN architectures considered in this study, including the proposed technical innovations – roll padding and convolutional attention mechanisms. Results are presented in Section 4. Section 5 compiles managerial implications. Section 6 concludes.

2. Case Study

2.1. Betfair Trading Characteristics

The goal of this research is to accurately estimate changes in odds for buying and selling bets while attempting to guarantee a profit. The analysis focuses on the Betfair betting exchange, the largest platform of its kind globally, with a predominant customer base in the UK (Brown & Yang, 2017). Modeling exchange markets requires considering a platform where individuals and entities trade fungible items of value with low transaction costs, at prices determined by supply and demand. Extracted models represent multiple interactions among participants. In most modeling scenarios, unforeseen external factors can disrupt assumptions on which predictive systems rely. However, within this restricted time frame, the market under consideration operates as a closed-loop system, where only internal market data influence price fluctuations. Consequently, predictive accuracy depends exclusively on market data, so that this study is solely concerned with purely speculative markets.

Among real-world exchange markets that adhere to the closed-loop interaction property, betting exchanges serve as a prime example. These facilitate the trading of bookmaking contracts, structured as binary options (i.e., win or lose), where the payoff is either a fixed monetary amount or nothing, depending on the outcome of a future event (Chen et al., 2008; Schumaker et al., 2010). Betting exchanges primarily operate within sports markets but also offer trading opportunities on elections and other event-based markets. Analogous to financial markets, buying and selling operations correspond to betting for and against an outcome (i.e., Back and Lay bets). The proposed methodology applies to exchange markets that provide market depth access, also known as level 2 market data. Relevant examples include futures (e.g., Dorman Trading, Phillip Capital), forex (e.g., FXCM), securities (e.g., Euronext Bonds), betting exchanges (e.g., Betfair, Betdaq, Matchbook), and cryptocurrency exchanges (e.g., Coinbase, Bitmex). These markets share the same fundamental structure, enabling adaptation into the presented framework. In contrast, certain exchange types, such as contracts for difference (CFDs), which involve exchange virtualization and do not provide market depth data, cannot be considered.

Table 1 presents a snapshot example of a market depth view. This information is referred to throughout the manuscript as a Raw Data Frame (RDF). The “Price” column represents the ladder of possible transaction prices. The market buy and sell amounts are listed in the “Bid” and “Ask” columns, respectively. The “Buy” and “Sell” columns indicate the agent’s own orders awaiting execution. When buy and sell orders converge at the same price, a matched transaction occurs, with the corresponding transacted amount recorded in the “Volume” column. The yellow cell highlights the most recent matched price.

The Betfair exchange is selected as a case study due to its accessible and free Application Programming Interface (API), which facilitates the retrieval of raw market data. Bets are bought and sold at varying prices, commonly referred to as odds. The price dynamics enable the realization of Profit & Loss (PL) before the final event outcome is known. Depending on market sentiment, price movements can span a few or several ticks,¹ making price volatility a widely acknowledged characteristic. The UK To-Win horse racing market is distinguished by high liquidity and volatility levels, factors that are critical for aligning with the research objectives. In this study, market engagement is restricted to the 10-minute pre-race period. This time frame is strategically selected because, before a race begins, each runner’s price is predominantly influenced by speculative activity. Market movements during this period are driven solely by internal trading data, establishing a strongly closed-loop system. Given this atomistic property, the analysis focuses exclusively on purely speculative markets.

¹A tick denotes the smallest unit of a price change.

Buy	Bid	Price	Ask	Sell	Volume
		:			
		5,1			20
		5,0	250		93
		4,9			68
		4,8	263		24
		4,7	148	10,00	70
		4,6	349	5,00	76
	8	4,5			217
	2	4,4			23
10,00	10	4,3			4
	448	4,2			
	398	4,1			
	335	4,0			
		:			

Table 1: Snapshot of market depth RDT information.

Fig. 1 illustrates the average trading volume, liquidity (i.e., the total sum of unmatched amounts at the bid and ask prices), and volatility (i.e., the absolute number of tick variations per minute) across the observed sample of races, including all participating runners. From the tenth minute before race commencement to the race's official start, average trading volume increases by approximately 4.2 times, average liquidity rises by about 3.4 times, and average volatility escalates by nearly 1.6 times. From a dynamic perspective, this means that all key market variables exhibit an upward trend as the race start approaches.

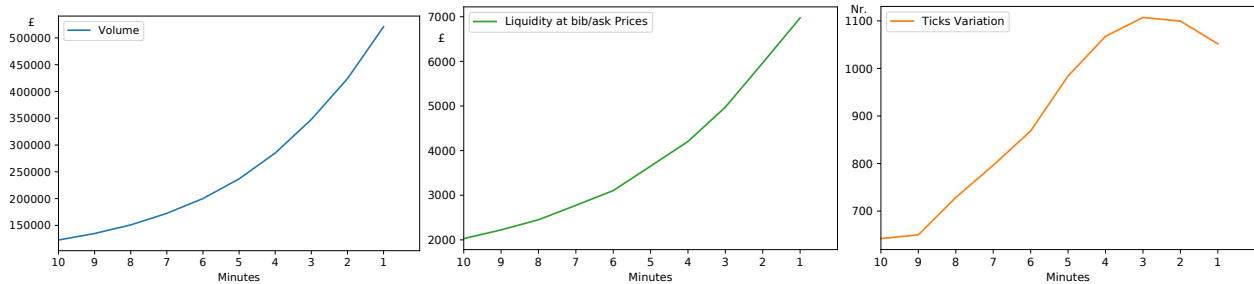


Figure 1: Average trading volume, average liquidity at the bid and ask price, and average number of ticks variation in absolute value per minute.

Raw data were collected directly from Betfair servers in real-time at a rate of two frames per second from September 1, 2014, to August 29, 2016. For conciseness, summary statistics on trading volume, liquidity, and volatility at the race level, analyzed on a per-minute basis within the 10-minute window before each race, are provided in Table 2. The dataset comprises a customized collection of UK To-Win horse racing markets, encompassing 15 to 30 daily races, with each race featuring between 3 and 25 competing horses. Differently from Gonçalves et al. (2019), market data is used to predict price variation for only the two final minutes before a race begins and not continuously throughout the 10 minutes pre-live. This form of trading, characterized by short market exposure and a strong reliance on real-time market data, aligns with day trading principles. When executed automatically over very short timeframes, such trading strategies fall under high-frequency trading (HFT). The methodologies implemented in this study belong to these trading categories. Access to market depth data is crucial in both high-frequency trading and day trading. Unlike long-term trading,

where broader trends are considered, short-term trading requires detailed real-time information on price movements at multiple levels. To obtain this data, traders – or automated trading agents – rely on a ladder view of the market, as illustrated in Table 1. All price forecasting data used in this study are derived from the continuously evolving price ladder.

	Mean	Std Dev	Min	Max
Volume				
1st min	520865	275541	14242	4858654
2nd min	423930	247598	10783	4713870
3rd min	347772	223834	8390	4569493
4th min	285066	201834	7014	4237354
5th min	236840	183575	5818	3944767
6th min	200076	168577	4514	3779854
7th min	172516	156206	2524	3554934
8th min	150905	145219	1898	3481841
9th min	134880	137576	1647	3450423
10th min	122693	131643	1503	3420282
Liquidity				
1st min	6974	12301	531	451241
2nd min	5967	13151	682	468516
3rd min	4975	12832	526	454538
4th min	4205	12544	393	453012
5th min	3652	12143	329	428307
6th min	3105	11137	285	399753
7th min	2774	10808	245	359948
8th min	2451	9825	212	331043
9th min	2224	9540	191	323377
10th min	2029	8935	171	292921
Volatility ⁺				
1st min	1052	1443	77	29899
2nd min	1100	1593	0	31323
3rd min	1107	1357	22	37893
4th min	1067	1524	0	29442
5th min	984	1412	20	33069
6th min	868	1437	29	49078
7th min	796	1191	4	30815
8th min	729	1162	7	27630
9th min	650	961	2	29686
10th min	642	1326	1	38439

Table 2: Pre-live betfair horse racing markets summary statistics.

Note: Total number of observations (i.e. races): 14421. Each line represents the x minute before the start of a race, $x = \{1st, \dots, 10th\}$. Units of measure are clarified in Fig.1 . Symbol ⁺ represents ticks variation in absolute value per minute.

To achieve PL, price movements in either direction are necessary. Price fluctuations follow the fundamental law of supply and demand. In financial markets, stock prices rise when demand for a company’s shares increases and fall when sellers outnumber buyers. The same mechanism applies to betting exchanges. If the majority of participants place Lay bets against a runner (e.g., a horse in a race or a football team), the fixed odds for winning the event will increase. Conversely, if bettors invest in favor of a runner’s victory, the odds will decline. The extent of these price movements, measured in ticks, depends on market pressure and investor sentiment, leading to either minor fluctuations or substantial shifts in odds.

2.2. Trading Implementation Framework

Betting markets are short-lived, yield easily quantifiable final payoffs for traded assets, and exhibit a degree of repetition, making them well-suited for efficiency testing. In betting exchanges, each market corresponds to a specific event, such as a tennis match or a horse race. Within an event, there are multiple runners (e.g., horses in a horse race), on which bets are placed. Two types of bets exist in betting exchanges: Back and Lay. A Back bet supports a runner to win, whereas a Lay bet wagers on the runner to lose. Bets are placed at specific prices, which correspond to implied probabilities. For example, a price of 2.0 represents a 50% probability of winning ($1/2 = 0.5$), a price of 1.01 implies a 99% probability ($1/1.01 = 0.99$), and a price of 100 corresponds to a 1% probability ($1/100 = 0.01$). Table 1 presents an example of unmatched bets placed at various price levels.

- Lay of £10.00 at 4.30 (Lay 10@4.3);
- Back of £10.00 at 4.70 (Back 10@4.7); and
- Back of £5.00 at 4.60 (Back 5@4.6).

If a bet is placed at a price that the market is willing to buy, it will be matched at the best available price. For instance, based on the market state in Table 1, placing a Back bet of 15@4.4 (on the blue side) will result in a match of £8.00 at 4.5 and £2.00 at 4.4, leaving the remaining £5.00 at 4.4 unmatched on the ask side, awaiting a Lay bet from another participant. The traded volume information will be updated accordingly. This mechanism determines how prices fluctuate in the market. Since this bet was matched at two different prices ($N = 2$), the global matched price of the bet can be calculated using the Eq. (1):

$$\text{Price Average} = \frac{\sum_{n=1}^N (\text{Price}_n \times \text{Amount}_n)}{\sum_{n=1}^N \text{Amount}_n} \quad (1)$$

If a Back bet is placed above the best available offer in the market (e.g., 4.5 in Table 1), such as 15@4.9, it will remain unmatched and will be queued in a First In First Out (FIFO) order with other Back bets at that price, waiting to be matched. Similarly, a Lay bet placed below the best available offer (i.e., a counter bet waiting to be matched) will also remain in the market until a matching Back bet is placed. Only unmatched or partially unmatched portions of bets can be canceled. The profit of a Back bet is calculated using Eq. (2), while the liability (i.e., the potential loss) of a Back bet is equal to the amount staked:

$$\text{Profit Back Bet} = \text{Amount Back} \times (\text{Price Back} - 1) \quad (2)$$

The liability or amount at risk in the event of a loss for a Lay bet is given by Eq. (3), while the profit of a Lay bet is equal to the amount of the bet itself. In summary, a Lay bet is the mirror image of a Back bet:

$$\text{Liability Lay Bet} = \text{Amount Lay} \times (\text{Price Lay} - 1) \quad (3)$$

Using combinations of Back and Lay bets, it is possible to secure a fixed PL before the final outcome of an event. An example of such a trade, where the event's result does not need to be known to secure a fixed PL, is as follows:

- Back of £2.00 at 2.12 (Back 2@2.12) Matched; and

- Lay of £2.00 at 2.10 (Lay 2@2.1) Matched;

For a bet to be matched, it must become the best available offer in the market and be countered by an opposing bet. When the runner wins, the profit from the Back bet minus the loss from the Lay bet is calculated as follows:

$$2 \times (2.12 - 1) - 2 \times (2.10 - 1) = 2.24 - 2.20 = 0.04$$

When the runner loses, the profit from the Lay bet minus the loss from the Back bet is: $2 - 2 = 0$. It is important to note that if a Back and Lay bet combination is placed with the same amount at different prices, a profit will occur if the Back price is higher than the Lay price, or a loss will occur if the Back price is lower than the Lay price, but only if the specified runner wins the event. If any other runner wins, and the amounts for the Back and Lay bets are equal, the PL will be zero. The agent can distribute the guaranteed PL from one runner across all other runners. To ensure an equal distribution of the PL across all possible outcomes, the amount required to close the trade must be recalculated. This process is known as *greening* or *hedging*. If a Back position is open in the market, the amount needed to close the position with the corresponding Lay bet is calculated using Eq. (4):

$$\text{Close Amount Lay} = \frac{\text{Price Open in Back}}{\text{Price Lay to Close}} \times \text{Amount Open in Back} \quad (4)$$

If a Lay position is open on the market, the amount to close the position with the corresponded Back is calculated using Eq. (5).

$$\text{Close Amount Back} = \frac{\text{Price Open in Lay}}{\text{Price Back to Close}} \times \text{Amount Open in Lay} \quad (5)$$

2.3. Software Implementation Framework

The developed software is based on an event-driven architecture (Zweigle et al., 2010; Kefalas et al., 2005; Deugo et al., 2001), where modules communicate through interfaces designed around architectural patterns that facilitate the production, detection, consumption, and reaction to events. Fig. 2 shows the primary modules and their interconnections. Parallel-processing techniques (Magee et al., 1994; Yau et al., 1995) are also incorporated, allowing for the instantiation of multiple Trading Agents, each with distinct policies, running concurrently and managing multiple trades simultaneously. This software framework we developed for automatic trading is called JBet².

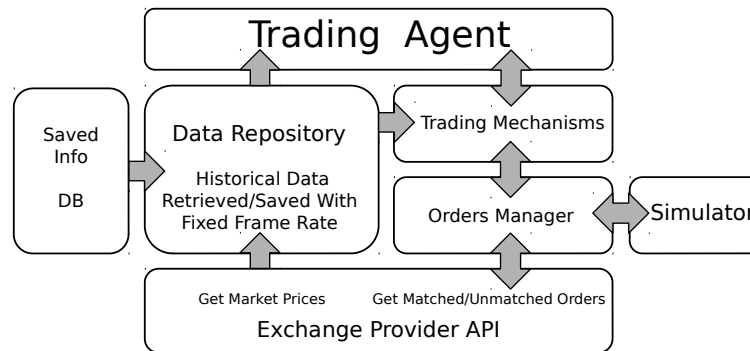


Figure 2: High level architecture for automated betting exchange.

²The JBet trading software framework source code can be consulted at <https://github.com/rjpg/JBet>.

Next, a description of each module is provided.

- **Exchange provider API:** This module serves as the interface between the trading system and the external betting exchange platform. It handles communication protocols and facilitates the retrieval and submission of data, such as market information, odds, and transaction details. The Exchange API ensures that the system can access real-time market data and execute trades in the betting exchange. The Betfair exchange API is used to access Betfair data (Betfair, 2012; Pitt et al., 2005), where Betfair acts as a service provider and other companies are clients. Its primary focus is speed and manageability, commonly employed in the development of trading software or software for tipsters. It can also be used to develop autonomous agents. In this framework, the Betfair API operates as a low-level communication interface between the framework and the exchange server. This layer interacts with the *Data Repository* module to provide data regarding the prices and volumes of each runner, and with the *Orders Manager* module to supply information on the states of bets and manage the placing and canceling of bets. According to Betfair (2012), main Betfair API services used for these actions include:³
 - *Data Repository*
 - Get Complete Market Prices; and
 - Get Market Traded Volume.
 - *Orders Manager*
 - Get Matched and Unmatched Bets;
 - Place Bets;
 - Update Bets; and
 - Cancel Bets.
- **Data Repository:** This module is responsible for gathering data, notifying listeners about new data or changes in the market state, saving data, and enabling the replay of saved data. The main interface of this module is shown in Listing 1. The *Market Update* event type simply notifies listeners that new data regarding the prices and volumes of the runners has arrived. Trading can occur both before and after the start of a race, which is reflected in the *Market Live* event type, activated when the market becomes in-play. The *Market Suspended* event type is triggered when the market is suspended, which may occur for various reasons depending on the type of event. For example, in a soccer match, the market is suspended after a goal until play resumes. Furthermore, markets are briefly suspended right after they go in-play. The *Data Repository* module can be instantiated with a new market by an external object, and when this occurs, the *Market New* event is triggered.⁴

Listing 1: Market Change Listener Interface

```
1 public interface MarketChangeListener {  
2  
3     public enum EventType {MarketUpdate , MarketLive , MarketClose , MarketSuspended , MarketNew}  
4  
5     public void MarketChange(MarketData md, MarketChangeListener.EventType marketEventType);  
6 }
```

³Depending on the type of licensing, a different number of calls per minute are permitted for each service.

⁴This module can be connected to the Betfair server via the Betfair API or through saved files, which can be used for data replay and simulation, as further explained below in this section.

- Orders Manager:** This module ensures the proper handling of bet information and manages all objects with a *BetListener* interface (i.e., *Trading Mechanisms*), notifying them about the state of their bets. Centralizing all bet processing is crucial to optimize the number of calls to the *Get Matched and Unmatched Bets* service, which is subject to limits. Additionally, in some cases, the Betfair API does not return the ID of a placed bet, leaving the program uncertain about the bet's placement. In such instances, the *Bets Manager* module tracks the bets without owner and correctly reassigns them (cf. Fig. 3, state *In Progress*). Fig. 3 presents the possible states and transitions of a bet within the framework. The *[Place]* and *[Cancel]* transitions are initiated by the agent or *Trading Mechanism* modules (i.e., client-side active actions). The *[SYS]* transitions are handled by the system, such as when the system automatically changes the state from *unmatched* to *matched* once the order is filled. This state machine serves as the foundational structure, extensively used by the *Trading Mechanisms* module to manage the market agent's position.

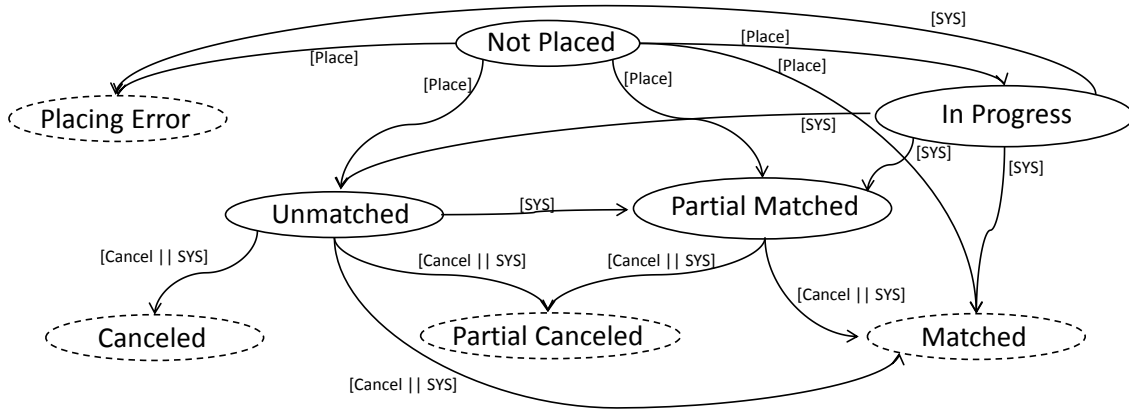


Figure 3: State machine for a given order.

- Trading Agents:** To instantiate a *Trading Agent* object, it must be extended from the abstract superclass *Bot*. This class implements all the interfaces and virtual methods necessary to interact with the *Data Repository* and *Trading Mechanisms* modules. *Trading Agent* objects are typically attached to a single market, observing a single runner, but they can also observe multiple markets and runners simultaneously, which is useful for techniques such as *dutching* and *bookmaking*. The *Trading Agent* object can also initialize *Trading Mechanisms* objects (i.e., trading processes) whenever it forms a conclusion about a runner's forecast. When a *Trading Mechanism* starts, it runs in parallel, and the *Trading Agent* is continuously informed about the state of the trade throughout its execution. At this higher abstraction level, it becomes easier to implement decision policies for interacting with the markets, ranging from simple rule-based decision policies to more complex methodologies, such as time series predictions.
- Trading Mechanisms:** This module is used to discipline the trader's attitude towards the market. In other words, these methods are designed to be implemented on a computer. They are executed after a decision regarding the market forecasting is made. Once the decision for the depreciation or appreciation of a runner is taken, a sequence of steps is initiated to maximize profit. This study focuses on three trading methodologies:

1. Scalping;

2. Swing; and
3. Trailing-Stop.

In the framework, these methods are implemented within the *Trading Mechanism* module (cf. Fig. 2). After a *Trading Agent* parametrizes and instantiates a *Trading Mechanism*, it runs in parallel and continuously informs the owning agent about the state of the trade. Ultimately, it will notify the agent when the trade is complete and provide the operation's PL.

- **Scalping:** It is a strategy used for very short-term trading, where the trader aims to make multiple small profits over time. These small profits accumulate, leading to a significant total gain. Scalping is most effective in markets with high liquidity and many active participants. It works particularly well in markets like Betfair, where the trader can take advantage of price fluctuations within short timeframes. The concept of scalping is simple: if a Back bet is placed at a certain price, a Lay bet must be placed at the next lower price, or vice versa for the opposite direction. The PL is determined by the difference – or spread – between the Back and Lay prices, as explained in Subsection 2.2. Betfair's betting exchange is ideal for this type of trading, especially in markets like horse racing, where liquidity is high, particularly just before the start of a race. Scalping involves trading in the market tick by tick, where one tick is a single step in the price scale of the ladder. For example, if a Back bet is placed at 2.12, one successful scalp would close the position with a Lay bet at 2.10 (i.e., one tick down). If a trade begins with a Back bet, it suggests that the price is expected to go down. Conversely, if a price is predicted to rise, the scalp begins with a Lay bet. Fig. 4 represents the state machine used to process a Back⇒Lay scalping strategy (i.e., predicting the price to drop). A Lay⇒Back scalp follows a mirror of this state machine. The Price Back Request (PBR) represents the price at which the agent enters the market, while the Price Back Now (PBN) is the current market price. If the price has already moved (i.e., $PBR \neq PBN$) when the order reaches the scalp module (i.e., the starting state), it is assumed that the opportunity has been lost (i.e., the prices have already dropped) or the prices have moved in the wrong predicted direction. In this case, the process ends without further action. If $PBR == PBN$, the position is opened in the market with a Back bet. After placing the bet, if it is not matched within a certain time-frame, the trade will end by canceling the bet, assuming that the price has already moved in the predicted direction, and the opportunity is lost. Otherwise, the system will try to close the position by placing a Lay bet. If the price moves down by one tick, the trade will close with a profit. If the price remains the same, the system will wait for a specified period. If the close bet has still not been matched after this waiting period, the system will attempt to close at the same price, resulting in a null profit/loss. If the price rises, the position will close “in emergency” with a loss. Listing 2 presents the declaration of the constructor method for the object that runs the scalping process in parallel. The `MarketData md` argument identifies the event (e.g., horse race, soccer game, tennis match), while the `RunnersData rd` argument specifies the runner to be traded. The `double entryAmount` represents the initial stake of the trade, and the `double entryPrice` is the price at which the position is opened in the market. The `int waitFramesNormal` defines the number of updates received from the *Data Repository* before attempting to close the position at the same price it was entered (i.e., with a zero profit/loss). Once the `int waitFramesNormal` period expires, the `int waitFramesEmergency` countdown begins. After it expires, the system will close the position at the best available offer to place the counter bet (i.e., an “emergency” close with a loss). The `Bot botOwner` is the agent that owns the trade and is used to keep track of the scalp's state. Finally, the `int direction` argument indicates the predicted direction of the price movement.

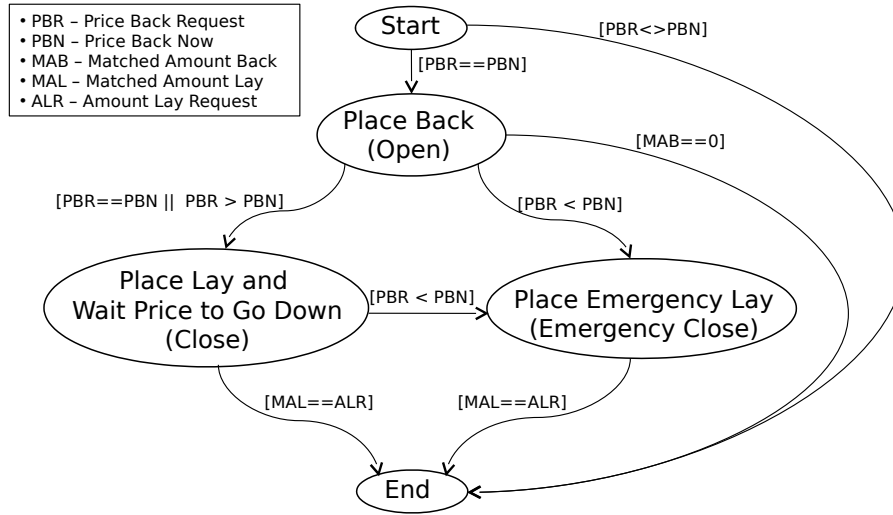


Figure 4: Simplified graph scheme for a Back⇒Lay scalp implementation.

Listing 2: Main parameters for Scalping mechanism

```

1 public Scalping(MarketData md,
2                 RunnersData rd,
3                 double entryAmount,
4                 double entryPrice,
5                 int waitFramesNormal,
6                 int waitFramesEmergency,
7                 Bot botOwner, int direction, ...);

```

- **Swing:** This trading strategy is very similar to scalping, with the main difference being the number of ticks the price must move before entering the close state (Carter, 2007). In the swing methodology, it is possible to define an offset number of ticks for both profit and loss. If the price remains within this offset interval, no action is taken. A swing with an offset of 1 tick for profit and 1 tick for loss is equivalent to scalping. Listing 3 describes the constructor for initializing the swing process. Besides the same parameters present in the scalping constructor, there are additional parameters: `int ticksUp` and `int ticksDown`, which represent the offset number of ticks to close the position in profit and loss, depending on the `direction` parameter. There are also two new arguments: `boolean frontLine` and `int waitFramesOpen`. These parameters are used when the agent does not want to enter the market immediately but prefers to wait until the market reaches the price specified in the `entryPrice` argument. If `waitFramesOpen` expires and the market does not match the entry bet, the trade process is canceled. If `frontLine = true`, the agent ignores the `waitFramesOpen` time and assumes that the agent wants to enter the market at the `entryPrice` where the counter offer is available.

Listing 3: Swing constructor example

```

1 public Swing(MarketData Market,
2              RunnersData rd,
3              double entryAmount,
4              double entryPrice,
5              boolean frontLine,
6              int waitFramesOpen,
7              int waitFramesNormal,
8              int waitFramesEmergency,
9              Bot botOwner,

```

```

10         int direction ,
11         int ticksUp ,
12         int ticksDown );

```

- **Trailing-Stop:** This methodology is used when the agent is aiming to capture a broader trend in a market but still wants to maintain a stop-loss condition if the trend reverses. The concept is straightforward: after a position is opened in the market, the close bet is set to close with a tick offset behind the current price and moves only when the price moves in the predicted direction. Eventually, the price will reverse, reaching the close price, at which point the order is placed to close the trade. Fig. 5 represents the state machine used to process this methodology when predicting a price drop (i.e., Back⇒Lay). The Price Lay to Close (PLC) represents the dynamic price, N ticks above the PBN. The state Update PLC N Ticks Above PBN repeatedly updates the price when [PBP > PBN] (i.e., when the runner price moves in the predicted direction – down in this case). The PLC is updated only when the PBN moves in the predicted direction. When the price reverses, it eventually reaches the PLC. When this happens, the close order (i.e., Lay bet) is placed. Finally, when [MAL = CAL], it means the close bet is fully matched, indicating that the price has moved in the reverse direction (i.e., up), filling the close bet completely and closing the trade. Listing 4 presents the constructor method of the object that runs the trailing-stop process in parallel. The `offset` represents the number of ticks by which the trailing stop follows the runner's price. Additionally, the `waitFramesNormal` parameter is included to control the duration of the trade, specifying the number of frames (or updates) after which the trade will close, should the price not move in the predicted direction within that time.

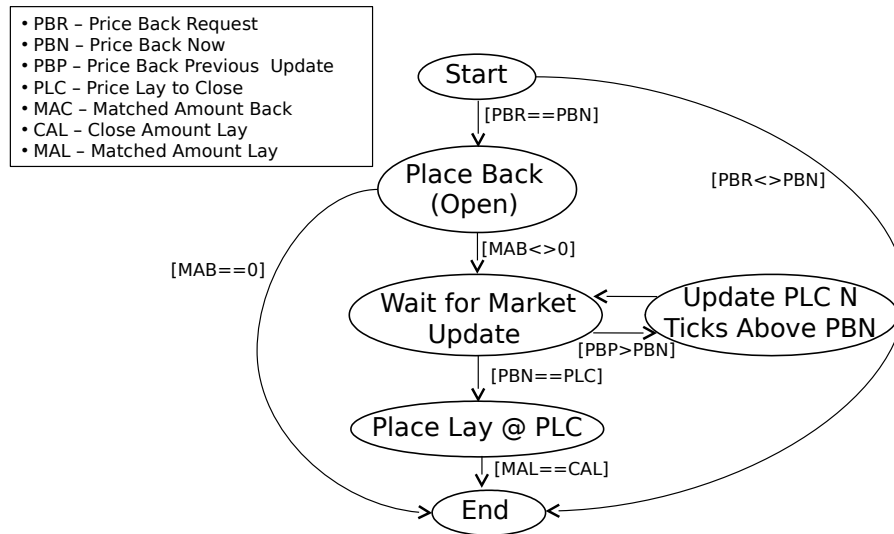


Figure 5: Simplified graph scheme for a Back⇒Lay Trailing-Stop implementation.

Listing 4: Trailing-Stop constructor example

```

1 public TrainlingStop (MarketData Market ,
2                       RunnersData rd ,
3                       double stakeSize ,
4                       double entryPrice ,
5                       boolean frontLine ,
6                       int waitFramesOpen ,
7                       int waitFramesNormal ,

```

```

8         int waitFramesEmergency ,
9         Bot botOwner ,
10        int direction ,
11        int offset );

```

- **Simulation:** This module addresses limitations involved in simulating markets of this type. The process involves the simulation of bet placement. There are two main problems when simulating a bet placement on the market:

- (I) The first major issue is the influence of the bet amount on the market. Unmatched bets do not appear in the real market, and matched bets do not consume or alter the available amounts in the real market. This issue is unavoidable because, in simulation, the actual amount of the bets is not placed (e.g., this limitation makes it impossible to simulate and test *Trading Agents* relying on *spoofing* methodologies).
- (II) The second problem is the simulation of the matching process. Bets of all traders in the market are placed in a *FIFO* queue for each price on the runner. It is impossible – since there is no data provided by the Betfair API – to know in which position our bet is in the queue. It is possible to make an approximate guess by observing the volume matched at the placement price and monitoring its evolution. However, due to the high-frequency nature of these markets, it is impossible to know the exact volume at the price when the placement order reaches the Betfair server. Moreover, it is impossible to determine if canceled bets were placed ahead or behind our bet, which compromises the volume monitoring approach to resolve this issue. For the framework described, we assume the worst-case scenario: bets are considered to be at the front of the queue when the volume transacted is equal to the amount that was in front of the order when it was placed. After that, we monitor the amount matched by tracking the volume variation. An order is considered fully matched when the volume variation reaches the order amount.

2.4. Data Collection and Feature Engineering

Raw data were collected twice per second. There are between 15 to 25 races per day, and data are collected only during the 10 minutes before a race starts. The goal is to process the collected data and update the models every month, as shown in Fig. 6. The raw data correspond to the data frames listed in Table 1, which were collected and then transformed into examples used to fit the models. These are organized into training and test datasets. The set of examples is constructed by going from the present to the past until the maximum number of examples defined for the training purpose is reached.

2.4.1. Rule-Base Filtering

This step required expert opinions on the specifics of this type of market (Gonçalves et al., 2019). Table 3 systematizes the decision tree that underlies the dynamics of each market. The combination of all properties generates 54 distinct categories (i.e., tree leaves), which are indexed to facilitate data processing. For instance, category 41 corresponds to a market dynamic characterized by the following properties:

root/nofavorite/mediumRunners/midleOdd/highLiquidity/ $\Rightarrow$ Model(41)

Out of the 54 categories, only 9 satisfy the minimum data requirements necessary to train the models, as these correspond to the more likely market states. To train a model from scratch, we define the minimum number of examples required as 1200. For the remaining categories, further studies must be conducted to explore the use of transfer learning. Transfer learning should be performed sequentially, moving from one category to the next based on similarity.

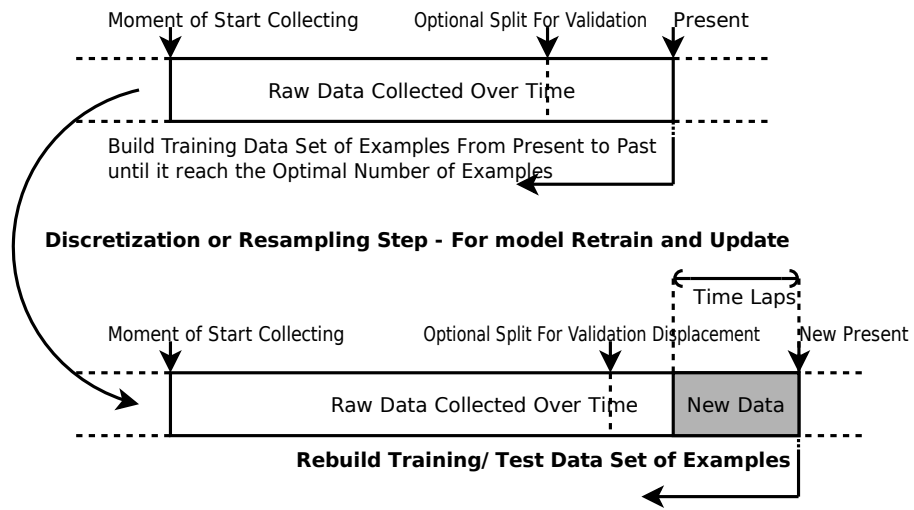


Figure 6: Add-in information to the raw dataset for global re-training of the models.

Favorite (1)	Runners (2)	Price (3)	Liquidity (4)
Yes	Few	High	High
No	Medium	Medium	Medium
	Many	Low	Low

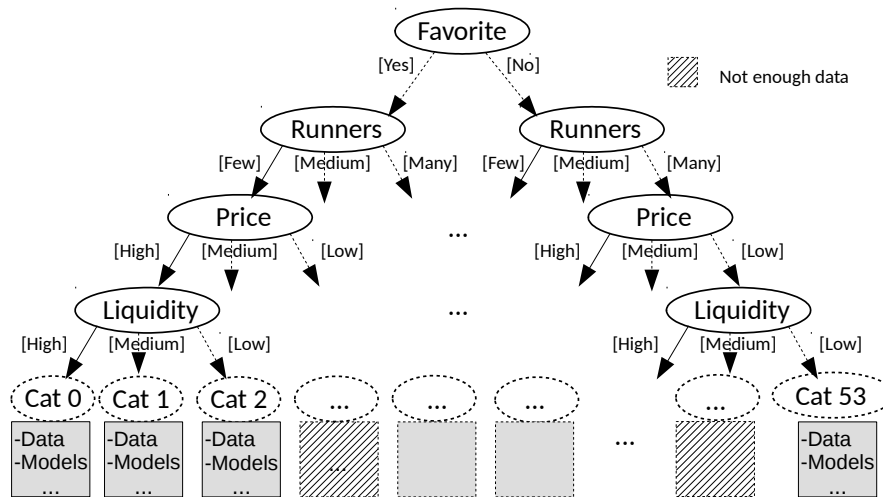


Table 3: Rule-based decision tree.

2.4.2. Input and Output Variables

For the input set, 512 RDFs are used, which corresponds to approximately 4.5 minutes of data to predict the price movement in the last 2 minutes before the start of a race. This means we need at least 6.5 minutes of pre-live market data for each race. Segments of 4 RDF are compressed into a single value to build indicators (cf. Fig. 8), resulting in 128 time steps. This data compression helps with computation and reduces the risk of overfitting. Each race constitutes one example for training. Nine indicators are used as model inputs, leading to the input format: $128 \text{ TimeSteps} \times 9 \text{ Variables}$. Fig. 7 shows 4 examples (i.e., 4 races with the evolution of the 9 indicators). The 9 indicators selected as input variables for the DL NN models are:

1. Integral of the price change of the runner in trade;
2. Integral of the price change of the competitor runner;
3. Liquidity variation on the ask side;
4. Liquidity variation on the bid side;
5. Volume variation and direction;
6. Price variation relative to the beginning of the sequence of the runner in trade;
7. Price variation relative to the beginning of the sequence of the competitor runner;
8. Weight Of Money (WoM) of the runner in trade; and
9. WoM of all other runners combined.

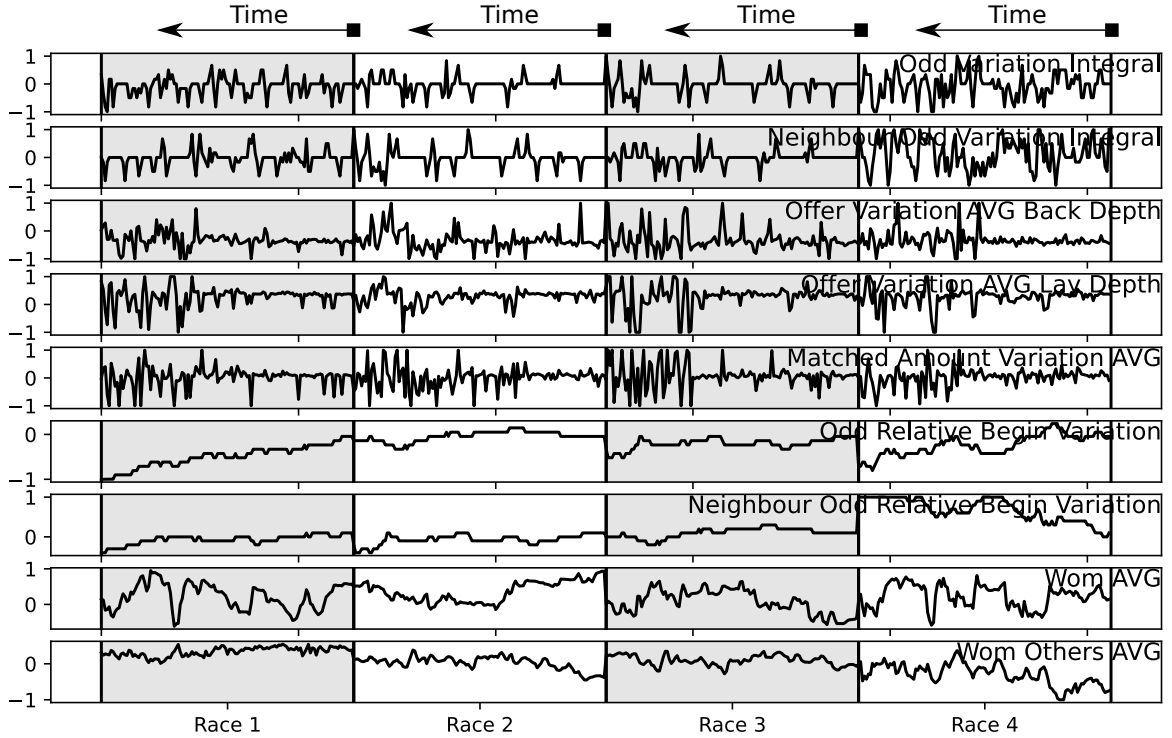


Figure 7: 4 races input examples, indicators evolution. Each input sample has 4.5 minutes of data.

Fig. 8 exemplifies a market state evolution corresponding to 3 RDFs in discrete time periods, compressed into one time segment value. It serves as a showcase for constructing the indicators considered in this case study. Selected indicators are also presented in Fig. 8. The first indicator corresponds to the integral of the price change of the runner subject to modeling, given by the integration of the ticks during the time segment. The second indicator corresponds to the integral of the price change of the competitor runner. In our case, the competitor runner is the one holding the closest price. In financial markets, this choice may be determined by expert opinion, i.e., another market with a strong positive or negative correlation. Since this is similar to the previous indicator, its graphical representation is omitted. Third and fourth indicators correspond to the variation in the amount on the Ask and Bid sides, respectively. Note that, for this example, it is assumed that the fourth past frame is equal to the third past frame. The fifth indicator highlights the *market strength*. It is given by the variation of the matched amounts, which provides information about the volume direction and strength. The sixth indicator corresponds to the price variation between the beginning of the entire sequence t_0 and the segment in processing t_i . For the numerical example exposed in Fig. 8, we assume that this 3rd RDF segment is the first of the 4.5-minute range. The seventh indicator is the same but applied to the competitor runner. The eighth indicator is the average WoM of the RDFs in the segment. The WoM is represented by a percentage value and shows when the market is balanced or unbalanced. The market is said to be balanced when the amount of money unmatched on each side of a selection is the same. This means the amount placed on the ask side must be approximately equal to that placed on the bid side.

(Past) Frame 3				Frame 2				Frame 1 (Present)			
Bid	Price	Ask	Volume	Bid	Price	Ask	Volume	Bid	Price	Ask	Volume
	:				:				:		
	4,8	263	24		4,8	263	24		4,8	263	24
	4,7	148	70		4,7	148	70		4,7	148	70
	4,6	349	76		4,6	349	76		4,6	349	76
8	4,5				4,5	92	8		4,5	92	8
2	4,4			2	4,4				4,4	98	2
10	4,3			10	4,3				4,3		
448	4,2			448	4,2			448	4,2		
	:				:				:		

Indicator 1 (and 2)			Indicator 3			Indicator 4		
Frame 3	Frame 2	Frame 1	Frame 3	Frame 2	Frame 1	Frame 3	Frame 2	Frame 1
4,6			0			0		
	4,5			+92			-8	
		4,4			+98			-2-10 = -12

$f = -3$ $\Sigma = +190$ $\Sigma = -20$

Indicator 5			Indicator 6 (and 7)			Indicator 8 (and 9)		
Frame 3	Frame 2	Frame 1	Frame 3	Frame 2	Frame 1	Frame 3	Frame 2	Frame 1
0			4,6			0.02		
	-8			4,5			0.43	
		-2			4,4			.45

$\Sigma = -10$ $Diff\ Ticks = -2$ $AVG(WoM) = 0.43$

Figure 8: Example of processing the 9 indicators given a segment of 3 RDF.

Note: In frame 1 of Fig. 8, the Lay amount of 10 at 4.3 disappears not due to the matching process, but rather to exemplify a cancellation of the amount.

The underlying logic is straightforward: when there is more unmatched money on the ask side than the bid side, the price decreases. The WoM pushes the price down. The same applies the other

way around. The WoM indicator is given by:

$$\text{WoM} = \frac{\text{Amounts Bid}}{\text{Amounts Bid} - \text{Amounts Ask}} \quad (6)$$

For the numerical example shown in Fig. 8, we consider only the depth of the best 3 prices around the transacted price. Depending on whether the price is high, medium, or low (Table 3), the depth used is 2, 3, and 4, respectively. For the ninth indicator, WoM is also applied but to all other runners. The logic is that if the ladder of unmatched amounts in all other runners is pressing the price in one direction, the runner in trade will be pressed in the opposite direction. Finally, the model output or *target* corresponds to the integral of the price variation, measured in ticks, for the last two minutes before the race starts. By compressing the data from multiple segments in this manner, we define a MTS problem with 128 time steps.

2.4.3. Frequency Distribution Histograms

To address outlier detection, we apply the truncation technique described in Deboeck (1994). Fig. 9 illustrates the automatic process of outlier truncation. Afterwards, the data are normalized into the interval $[-1, 1]$ through frequency analysis and histogram rescaling. The rescaling of maximum and minimum raw values involves truncating 10% of the histogram tails. This operation alters, but does not remove, the original examples. Only after these steps are the data fed as input for model training. This operation is systematically applied to all inputs for each category.⁵

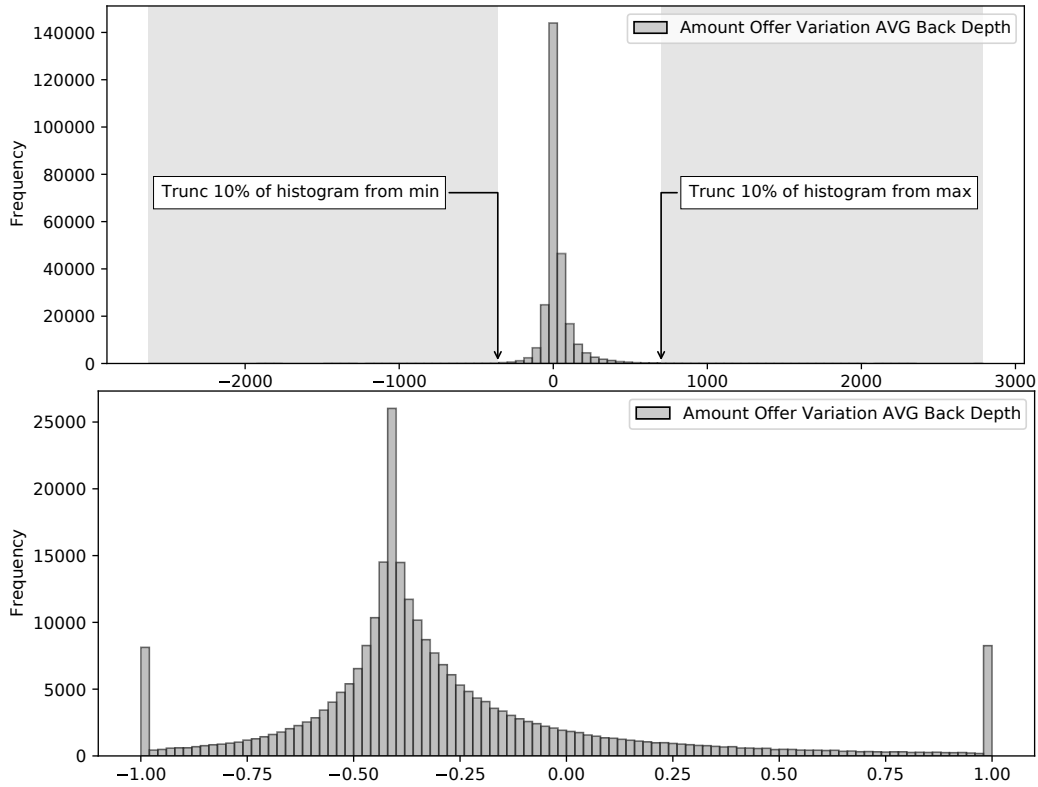


Figure 9: Example of histogram re-scaling with truncated tails at 10% level to find min-max values for input normalization. The illustration represents input #3 (Liquidity variation on the ask side) on category #41 of the rule-based system index.

⁵Fig. A1 in Appendix shows the result of this operation for all indicators used in this case study.

A similar technique, using histograms for data partitioning, is applied at the output level to transform the regression problem into a classification problem. An egalitarian distribution of examples across qualitative classes is ensured to avoid overfitting and/or biased models. The output for this case study is the integral (i.e., the area) of the tick variation of the runner’s price relative to the last 2 minutes before the race starts. When the integral is significantly negative, a “strong down” price change is considered, and the first qualitative class is established. If the numerical value falls into the second qualitative class, a “weak down” price change is assumed. In the third qualitative class, a “neutral” price change is considered. A “weak up” price change is assigned when the solution falls into the fourth qualitative class. Finally, a “strong up” price change occurs when the solution falls into the fifth qualitative class. This classification procedure is outlined in Fig. 10. Each class will contain approximately 20% of the examples in the training dataset. A “strong” movement prediction suggests activating a trailing-stop trading mechanism, while a “weak” movement prediction indicates the activation of a small swing trading strategy.

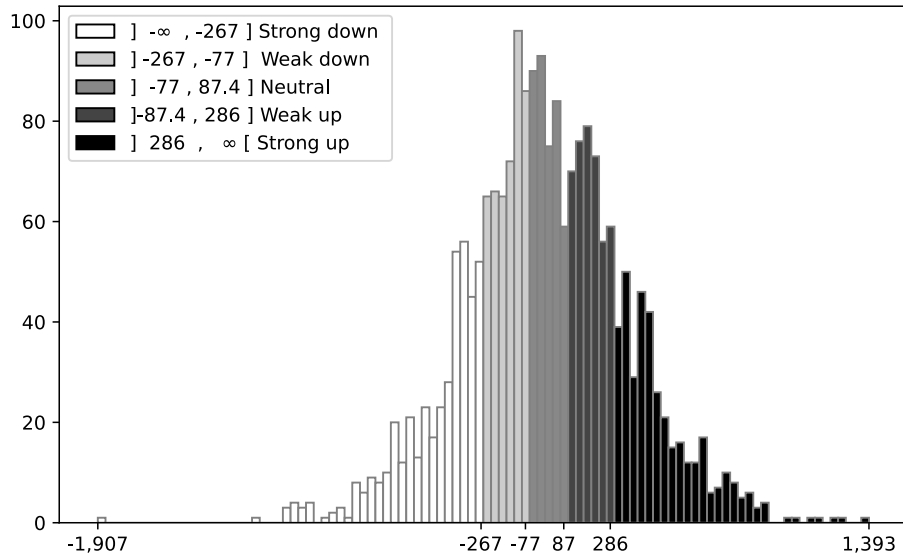


Figure 10: Example of the histogram for the qualitative classification of the output representing the integral of tick variation for the last 2 minutes: strong down (left white), weak down (light gray), neutral (gray), weak up (dark gray), and strong up (right black), respectively.

The choice of target and stop-loss prices in each category is based on the de-normalization of the output, as shown in the histogram presented in Fig. 10. The main idea behind this process is to adjust the parametrization of trading mechanisms – target and stop-loss – according to each category. The target price is determined as the average of the maximum tick variation for all examples within a given class during the predicting time. The stop-loss is then defined as 80% of the target price for swing trades and 60% of the target price for trailing-stop trades. Table 4 exemplifies how target and stop-loss prices are defined for each class within a given category.

Once the developed DL NN models are ready for production, they undergo final validation in the simulator described in Subsection 2.3. To provide a better understanding of the trading execution based on the model’s predictions, Table 5 presents log results of a single trading execution.

3. Methods: Applied Deep Learning Architectures

Let us now explain in detail the DL architectures that form the basis of this study, along with the proposed extensions.

Class	Mean of the ticks variation	Target	Stop-loss
Strong Up	6.44794	6	4
Weak Up	3.51428	4	3
Weak Down	-3.19424	3	2
Strong Down	-6.33173	6	4

Table 4: Example of trading mechanism parameters for a particular category.

Instantiation	
Runner category	<i>no favorite/mediumRunners/midleOdd/highLiquidit (Model #41)</i>
Model predicted probabilities	[0.14 , 0.19, 0.17 , 0.20 , 0.30]
Predicted class	5 th class: Strong Up
Bets/trade direction	Up: Lay (open) $\Rightarrow$ Back (close)
Trading mechanism	Trailing Stop (strong movement predicted)
Parameters (in ticks)	Stop-loss: 4 , Target: 6
Parameters (in odds)	Entry odd: 4.6 , Target odd: 5.2 , Stop odd: 4.2
Time parameters	20 frames open, 80 frames start close best price, 20 close emergency
Open amount stake	£3.00 (Lay)
Potential PL	Profit: £0.35 , Loss: - £0.28
Result	
Trade final state	CLOSED
Moved ticks	6
Open amount stake	£3.00 (Lay)
Effective open odd (price)	4.6
Close amount stake	£2.65 (Back)
Effective PL	£0.35
Effective close odd (price)	5.2

Table 5: Information example of one trading execution log line with the model prediction, category parameters, and results. Table A1 in the Appendix shows a segment of the output logs obtained.

3.1. CNN LeNet Based Models

LeCun et al. (1998) proposed a NN architecture for handwritten and machine-printed character recognition, called LeNet. This architecture, based on convolutional layers, is a straightforward CNN that is easy to understand. The LeNet-5 architecture consists of two sets of convolutional and average pooling layers, followed by a flattening operation and three dense layers (cf. Fig. 11). CNNs are also described in detail in LeCun et al. (2010). This simple model is introduced here as a formalization and should be interpreted as a basic benchmark case relative to alternative architectures. It is also used to identify improvements with the add-ons further described.

Consider a bi-dimensional input feature map x^l in layer l of size (H, W) and a stride δ of $(1, 1)$. The basic mathematics for the computation of a convolutional layer l to obtain the output feature map y^l with kernel K of size (k_H, k_W) can be expressed as:

$$y_{i,j}^l = \phi^l \left(\sum_{i=0}^{H-k_H} \sum_{j=0}^{W-k_W} K \cdot x_{i,j}^l \right) \quad (7)$$

where ϕ is the activation function. This equation represents the application of kernel K in the input map x^l at coordinates i, j in layer l . A bias term b is usually added to $y_{i,j}^l$, but it is omitted here for clarity. It is observed that the output is smaller than the input when the convolution kernel is larger than $(1, 1)$. If the input has size (H, W) and the kernel $K = (k_H, k_W)$, the resulting convolution has size $(H - k_H + 1, W - k_W + 1)$, which is smaller than the original input. Typically, this is not a concern for inputs with large dimensions (i.e., images) and small filters. However, it can be problematic with small input dimensions or when considering a high number of stacked convolutional layers. In such cases, the practical effect of large filter sizes and/or very deep CNNs on the size of the resulting feature map could lead to loss of information, causing the model to run out of data. The padding operation is introduced to address this issue.

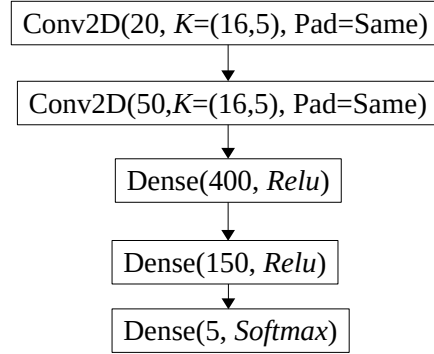


Figure 11: Base LeNet CNN 2D using same padding in convolutional layers

3.1.1. Traditional Padding Methods

Currently, the standard procedure to avoid the border effect problem consists of applying *same padding* (i.e., the inclusion of zeros outside of the input map). For each channel of the bi-dimensional input x , we insert zeros $k_H - 1/2$ rows above the first row and $k_H/2$ rows below the last row, as well as $k_W - 1/2$ columns to the left of the first column and $k_W/2$ columns to the right of the last column. This approach ensures that the convolution output size will be (H, W) , maintaining the same spatial extent as the input.

However, when analyzing a MTS problem, where the input feature map has a relatively small size in the variable component, the inclusion of zeros via same padding may weaken the learning

Method	Example of padded info		
	Pad	Input	Pad
Valid (None)		a b c d e f	
Same (Zero)	0 0 0 0	a b c d e f	0 0 0 0
Reflect (Mirror)	d c b a	a b c d e f	f e d c
Reflect101	e d c b	a b c d e f	e d c b
Constant n	n n n n	a b c d e f	n n n n
Tile 2	a b a b	a b c d e f	e f e f
Causal (Zero Left)	0 0 0 0	a b c d e f	
Wrap	c d e f	a b c d e f	a b c d

Table 6: Padding examples of size 4 for unidimensional input.

capability. This is because the learned kernel interacts with the zero values in the input during the dot product operation, potentially leading to erroneous generalization and less effective model training. According to Hamey (2015), there are other well-known padding methods commonly used in image processing environments. These use the information in input variables to fill in the borders. Table 6 provides examples of different padding methods.

3.1.2. A New Method: Roll Padding

Roll padding is an extension of wrap padding, specifically designed for MTS analysis. Wrap padding copies information from the opposite sides of the image, effectively mapping it onto a torus. This operation results in four copies of information under a bi-dimensional input: rows (columns) above the top (on the left) are duplicated from the bottom rows (right columns), respectively, and vice versa. While wrap padding is generally not useful for natural images, it is highly effective for computed images, such as Fourier transforms and polar coordinate transforms, where pixels on opposite borders are computationally adjacent.

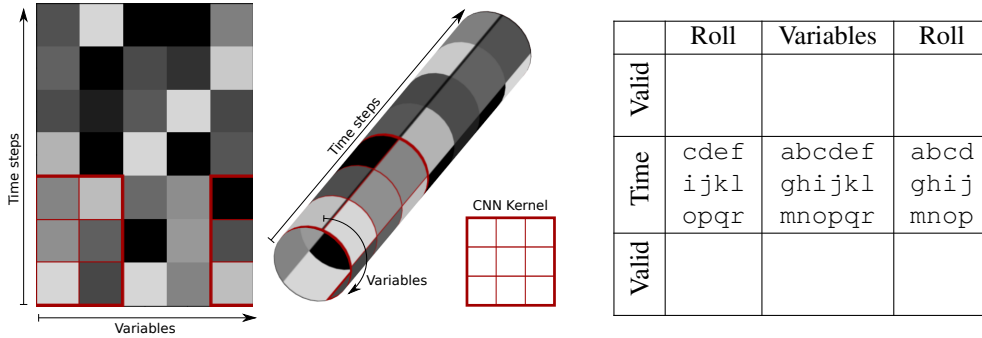


Figure 12: Roll padding scheme in MTS analysis.

As illustrated in Fig. 12, roll padding extends this concept by copying information from the opposite sides but only along a single dimension – input variables component in the bi-dimensional input map (i.e., $TimeSteps \times Variables$). In contrast, the time steps component remains unpadded (i.e., valid), transforming the structure into a cylinder rather than a torus. The reduction of the time steps component after multiple convolutions is generally not problematic, given that MTS problems often involve a high number of time steps. However, for cases where preserving temporal resolution is crucial, roll padding can be combined with other padding methods (e.g., causal padding), as demonstrated in Fig. 20 and Subsection 3.4.

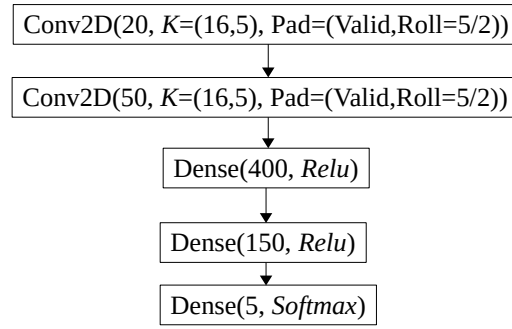


Figure 13: CNN 2D using valid padding in time steps component and roll padding, of size $\frac{K_H}{2}$, in the variables component.

As shown in Fig. 13, the structure of CNNs using roll padding is based on LeNet-5 (LeCun et al., 1998). Both CNNs share the same hyperparameters, with the only difference being the padding method applied. This allows us to directly assess the impact of roll padding on performance. In Fig. 13, we specify the padding method used in each dimension for the second CNN. The time steps component employs valid padding, while roll padding is applied to the variables component.

3.2. LSTM Based Models

In this study, the LSTM, originally proposed by Hochreiter & Schmidhuber (1997), consists of four layers, as illustrated in Fig. 14. Three of these layers use bidirectional LSTMs, while the final layer is a dense layer with a softmax activation function for classification. Similar to the model discussed in Subsection 3.1, this architecture serves as a benchmark to assess improvements when integrating additional methodologies. RNNs have been successfully applied to numerous sequential data tasks. Enhanced models, such as LSTMs, facilitate training on long sequences by addressing issues like vanishing gradients. However, despite these advancements, even the most sophisticated models face limitations, making it challenging for researchers to develop high-quality solutions for long-sequence data. Many MTS problems require establishing connections between distant input and output points across multiple layers, often spanning dozens of time steps. To effectively tackle such challenges, existing RNN architectures have had to be modified and adapted.

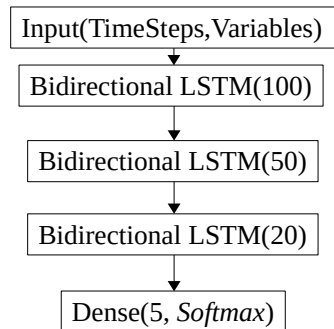


Figure 14: Stacked Bidirectional LSTMs. LSTMs based models baseline

3.2.1. Standard Attention

Attention is a mechanism designed to be integrated with RNNs, allowing the model to focus on specific parts of the input sequence when predicting certain parts of the output sequence. This capability enhances both the speed and robustness of convergence. The incorporation of attention mecha-

nisms has significantly improved performance across various tasks, making it an essential component of modern RNN networks.

A pivotal study on attention is Vaswani et al. (2017), which originally introduced the mechanism for machine translation tasks. However, attention has since been widely adopted across numerous application domains. Fundamentally, attention can be viewed as a residual block that multiplies its output with its own input, h_i , before reconnecting to the main NN pipeline through a weighted and scaled sequence. Scaling parameters, known as attention weights α_i , determine the importance of each input, while the resulting weighted sum is referred to as the context weight c_i for each sequence position i . Collectively, these form the context vector c , which has a sequence length of n . This operation is mathematically expressed as follows:

$$c_i = \sum_{i=0}^n \alpha_i h_i \quad (8)$$

The computation of α_i results from applying a softmax activation function to the input sequence x^l on layer l :

$$\alpha_i = \frac{\exp(x_i^l)}{\sum_{k=1}^n \exp(x_k^l)} \quad (9)$$

This means that input values of the sequence compete with each other for attention. Since attention scores are obtained through a softmax activation function, their sum is always 1, ensuring that scaling values in the attention vector α fall within the range $[0, 1]$. The mechanism described above is known as *soft attention* because it is a fully differentiable and deterministic process that integrates seamlessly into a backpropagation-based system. In this approach, gradients propagate through the attention block just as they do through the rest of the network.

Differently, *hard attention* does not use a weighted average. Instead, it treats α_i as a sampling probability that determines whether x_i is included in the context vector. This method replaces deterministic computation with stochastic sampling. To correctly compute gradient descent during backpropagation, *hard attention* employs the Monte Carlo method, where multiple sampling iterations are performed, and their results are averaged (Xu et al., 2015). The accuracy of this approach depends on the number and quality of the samples.

On the other hand, *soft attention* follows a simpler, more conventional backpropagation approach when computing gradients within the attention block. However, its accuracy depends on the assumption that a weighted average is a good representation of the relevant input areas. Both methods have their strengths and weaknesses. Currently, *soft attention* is more widely used due to its seamless integration with backpropagation, making it more efficient in practice. For this study, we exclusively use *soft attention*.

If attention is applied directly to the input before entering the *attention before*. Conversely, if attention is applied to the output sequence of the LSTM, it is called *attention after*, as clarified in Fig. 15. Since we work with MTS, a bi-dimensional dense layer for attention is used. To ensure that the attention mechanism is applied to the time step component of each sequence rather than the variable component, we perform a permutation operation both before and after this layer. It is relevant to note that when attention is applied after the LSTM, the recurrent layer must return its internal recursively generated sequences, which correspond to the number of units defined, denoted as NRC . This parameter is crucial within the attention block, as it determines how many sequences need to be processed.

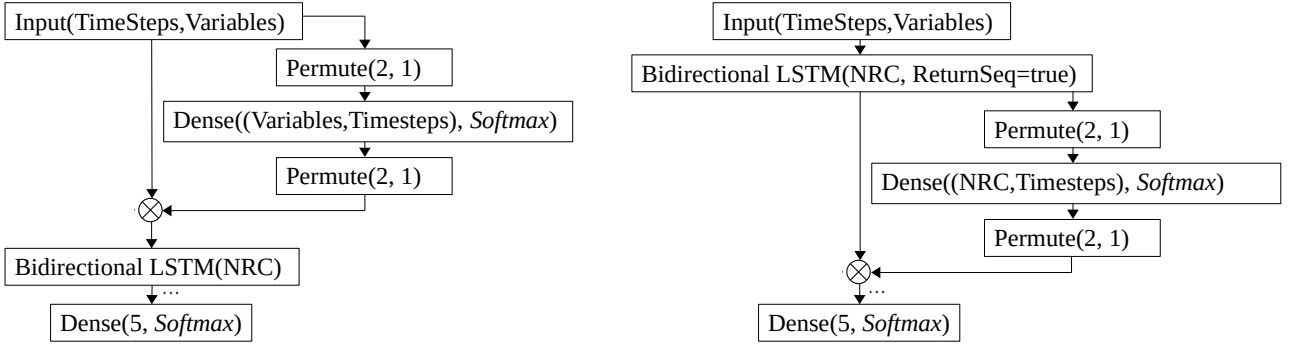


Figure 15: MTS attention before LSTMs on the left subplot and attention after LSTM on the right subplot.

3.2.2. Multi-Head Convolutional Attention

A key contribution of this study is the integration of convolutional layers within the attention block. The original design of attention mechanisms was primarily intended for text processing, where attention is assigned to each embedded word individually within long sequences. However, in MTS problems, which are inherently more continuous and less discrete than text, it can be beneficial to focus on patterns in small contiguous segments rather than on individual values. By incorporating convolutional layers within the attention block, we enable the model to capture local temporal patterns more effectively, enhancing its ability to recognize meaningful structures within the data.

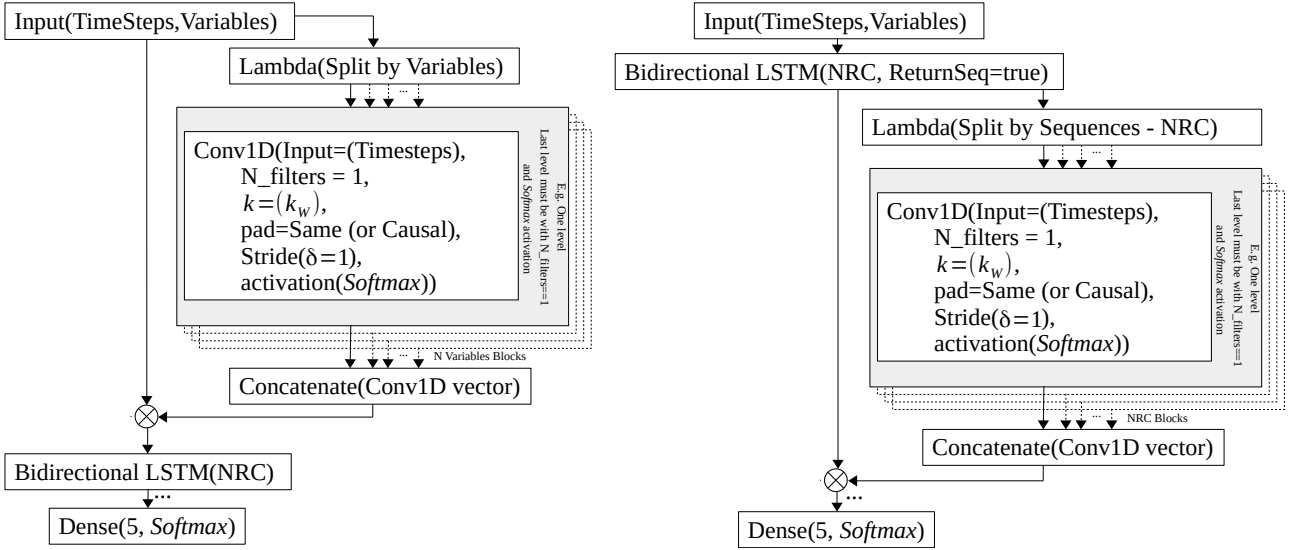


Figure 16: Attention using convolutional layers before and after LSTMs.

Fig. 16 shows the implementation process. The MTS is first split into individual time series using the Keras Lambda function. For each sequence, a path with 1D convolutional layers is created, and results are concatenated back together. In Fig. 16, only one convolutional filter per sequence is depicted (i.e., per variable of the MTS) if attention is applied before the LSTM, or per Number of Recursive Cell (NRC) generated sequence if applied after the LSTM. It is important to note that, before the concatenation operation, each path must return a one-dimensional vector with size $TimeSteps$. When concatenated with the other paths, this results in an attention weights feature map of size $TimeSteps \times Variables$. This map is then multiplied with h to obtain the 2D context map c .

To capture multiple small subsequence patterns (i.e., filters), we must stack multichannel 1D

convolution layers before the final attention layer. However, the last convolutional layer inside the attention block must output only a single channel, as explained earlier. An alternative way to enforce a 1D output vector for each path is to use the Keras *AveragePooling1D* layer, which averages the previous channels into one dimension. Additionally, the final single-channel 1D convolution output must use the softmax activation function to ensure that each value, in the resulting vector per variable, competes with each other, sums to 1, and has a scaling factor in the $[0, 1]$ range.

3.3. ConvLSTM2D Based Models

3.3.1. ConvLSTM2D for Segmented Time Series

The Bi-dimensional Convolutional LSTM (ConvLSTM2D) layer was proposed by Shi et al. (2015). The motivation for this structure was to predict future rainfall intensity based on sequences of meteorological images. By applying this layer in a NN architecture, they were able to outperform state-of-the-art algorithms for this task. The ConvLSTM2D is a recurrent layer, similar to the LSTM, but internal matrix multiplications are replaced with convolution operations. As a result, the data flowing through the ConvLSTM2D cells retains the input dimension, 3D in our case: $Segments \times TimeSteps \times Variables$, rather than being just a 2D map: $TimeSteps \times Variables$ (cf. Fig. 17-18). As explained in Subsection 3.1.2, we can also apply roll padding to this input on the variables component. ConvLSTM2D layers can be particularly useful in MTS that can be partitioned into segments, such as in the case study of household electric power consumption (Gonçalves et al., 2023). In this case, the time series will exhibit representative patterns for every day of the week, which can be grouped into a 2D map.

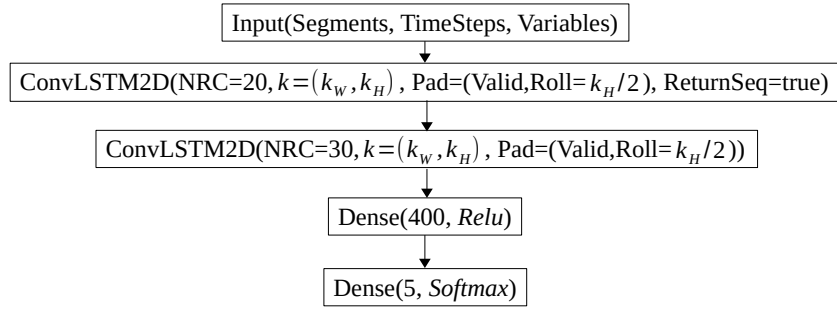


Figure 17: Base scheme for stacked ConvLSTM2D with roll padding on the variables component.

3.3.2. ConvLSTM2D Convolutional Attention with Roll Padding

When entering the attention block, after splitting by the input variable, the resulting 2D map to be processed by convolution layers will have a $Segments \times TimeSteps$ format. This means that the 2D kernels will attempt to capture patterns relating to contiguous time steps, as well as the same temporal steps across the previous and next segments. If segments represent days and time steps are divided by hours, a 2D kernel will capture attention patterns related to specific hours of the day, as well as similar periods in the preceding and following days. Moreover, if we have segments of seven days, roll padding can be applied to the segments component, allowing the kernel's border processing to correlate the first day of the week with the last day, especially if the data exhibits a weekly cyclical pattern, as shown in Fig. 19. If it is not desirable to correlate data between segments, a one-dimensional kernel should be defined (i.e., $2D\ K = (1, k_w)$, since we are working with a bi-dimensional convolution layer). Each 2D output map is obtained through a softmax activation. Each value in the resulting 2D map for each variable competes with the others, summing to 1, with a scaling factor in the range $[0, 1]$. Once all 2D maps are concatenated, the resulting α will be 3D and compatible for scaling the inputs h of the attention block to obtain c , as described in Eq. (8).

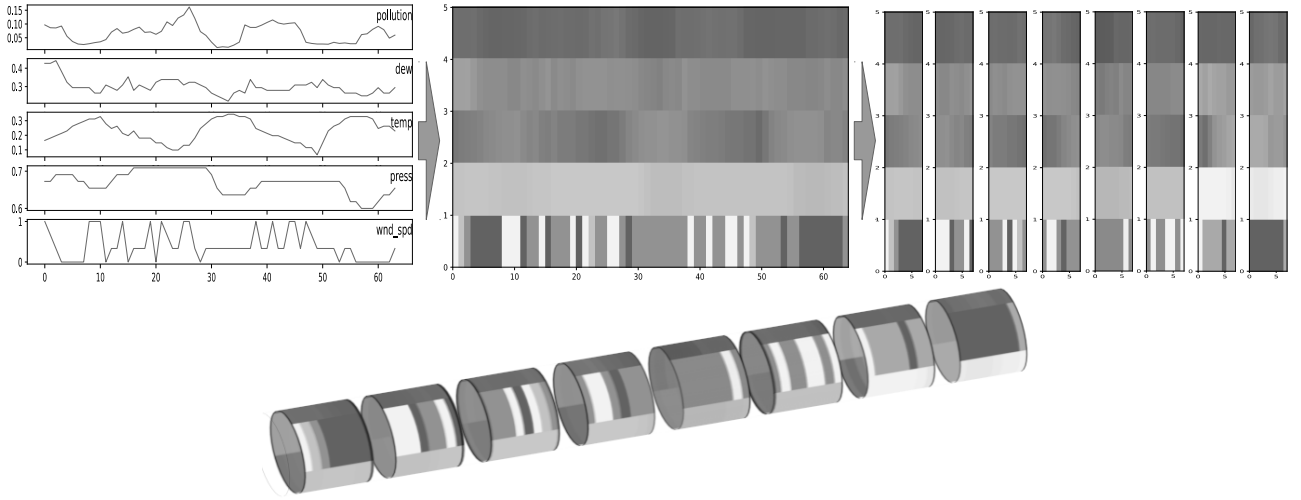


Figure 18: MTS input processing for ConvLSTM2D.

Note: The bottom plot describes the application of roll padding in the variables component for each segment.

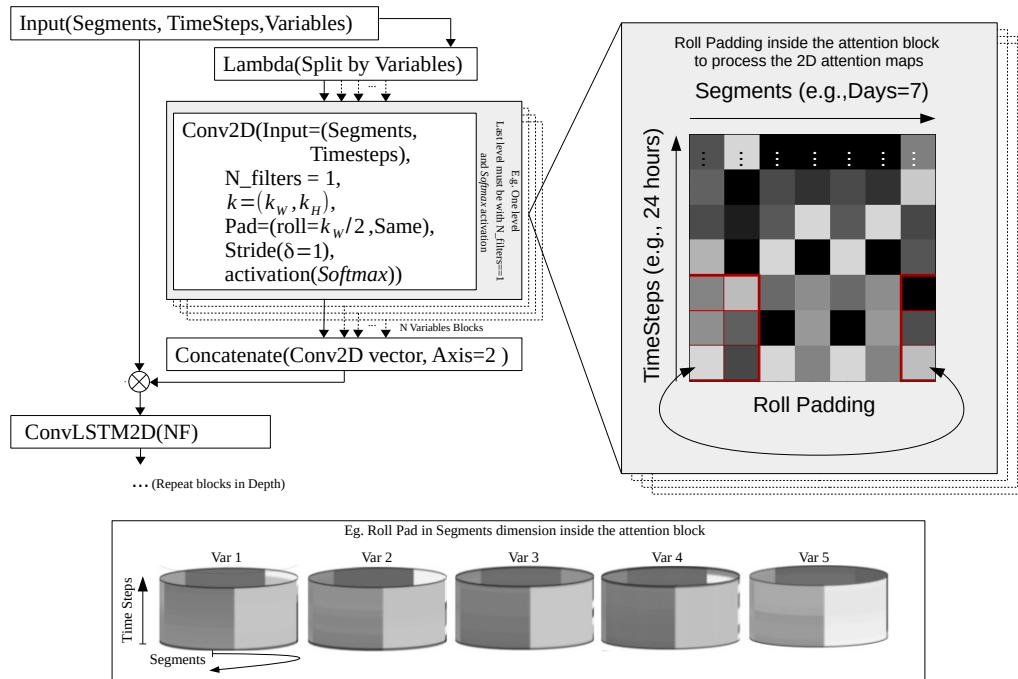


Figure 19: Attention using 2D convolutional layers before ConvLSTM2D.

Note: The bottom plot describes the application of roll padding in the segments component for each variable inside the attention block.

3.4. Multivariate WaveNet

3.4.1. WaveNet 1D with Multichannel Input

A relevant DL architecture that can be applied to MTS problems is WaveNet, developed by Google DeepMind (van den Oord et al., 2016). WaveNet was originally designed for audio signal generation. A key component in achieving this task was a sound classifier based on 1D convolutional layers. In the first two left subplots of Fig. 20, the output difference between using and not using causal padding in 1D convolutions can be observed. Causal padding ensures that the time steps of past information are preserved for the subsequent layer. In the context of MTS, this requires that the number of zeros to be added before the beginning of all sequences is equal to $k - 1$, where k represents the size of the one-dimensional kernel for 1D convolutions. It is important to note that, for MTS problems, each input variable is treated as a channel in 1D convolutions, and causal padding is uniformly applied across all channels.

WaveNet employs dilated convolutions to progressively expand the receptive field. In the time steps component, when using a dilation rate dr (i.e., for $dr > 1$), the causal padding size is given by $dr \times (k - 1)$. The residual block in the WaveNet architecture is executed multiple times according to a specified *depth* in the network, with $N = \{1, \dots, depth\}$. Within this block, the dilation dr of the *sigmoid* and *Tanh* convolutions applied to the time steps component increases exponentially according to the formula $dr = k^N$. As highlighted in Fig. 21, the third and final convolution within the residual block has $k = 1$ and $dr = 1$ to reduce dimensionality and manage model complexity. This layer is also referred to as the *channel-wise pooling layer*. The standard WaveNet uses 1D convolutions, which can be adapted for MTS problems by treating the input as a multichannel set of 1D sequences of variables. To explore the inclusion of roll padding in the variables component, the WaveNet architecture is extended to use 2D convolutions.

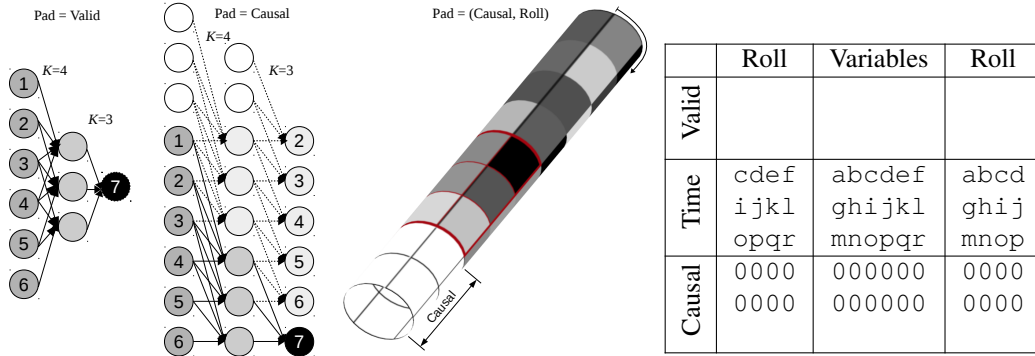


Figure 20: On the left subplot, comparison behavior between valid and causal padding. On the right subplot, combination of causal and roll padding scheme for MTS analysis with WaveNet 2D.

3.4.2. WaveNet Extended with 2D Convolutions and Roll Padding

Fig. 21 shows the extended WaveNet architecture, which operates with 2D maps instead of 1D multichannel inputs. The architecture preserves the standard WaveNet processing for the time steps component, using causal padding, while incorporating roll padding in the variables component. With the introduction of 2D convolutions, the kernel size is now defined as (k_H, k_W) . The k_H component (i.e., the time steps component) is processed, with both dr_H and causal padding applied to the time steps component, as described in Subsection 3.4.1. The combination of causal and roll padding is clarified in the last two right subplots of Fig. 20. In the second dimension (i.e., variables component), roll padding is applied, with a size determined by k_W . A roll padding size of $k_W/2$ is established,

copying the opposite $k_W/2$ columns from the input map. For simplicity, odd fixed sizes in k_W are assumed. No dilation rate is applied in the variables component (i.e., $dr_W = 1$). In summary, the time steps component is processed in line with the basic WaveNet approach, while the variables component is processed using standard convolutional layers with roll padding. Finally, after adding skip connections, three 2D convolutional layers are included. In this scheme, a stride $\delta = (\delta_H, \delta_W)$ with $\delta_H > 1$ and $\delta_W = 1$ is used to downsample only the time steps dimension, rather than using pooling layers. The final convolution layer has x filters ($x = 6$ in Fig. 21), generating x feature maps, to which global average pooling is applied. This allows us to directly apply softmax to the x resulting values for classification into x classes.

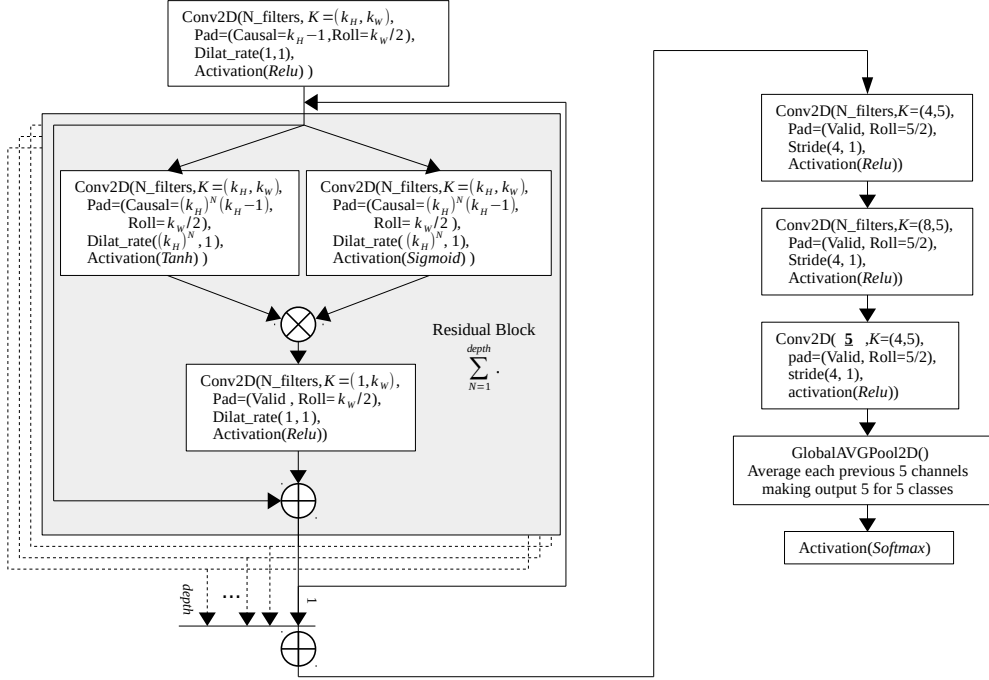


Figure 21: WaveNet 2D architecture for MTS classification using 2D convolutions with causal padding in the time steps component and roll padding in the variables component.

4. Results

An end-to-end analysis was conducted for this case study, as a complete framework was established, covering everything from data gathering to market interaction. Table 7 shows that the best result was achieved using the LSTM-based model with multi-head attention, processing the attention weights of each variable time steps with Conv1D^6 . LSTM models outperformed the other alternatives presented. This suggests that the inherent nature of the LSTM architecture is well-suited to this type of data. Additionally, it is evident that the inclusion of roll padding improves the accuracy of both simple CNN and WaveNet models compared to their base versions. However, due to the relatively small sample size in this case study, these CNN-based models appear to be highly sensitive to overfitting. Further investigation of smaller CNN-based solutions with better convergence is recommended.

Table 8 presents the best DL model, which achieves an accuracy of 30.92%. Although this is only 11 percentage points above the baseline, it is important to consider that the goal of this problem is to

⁶Fig. A2 in Appendix shows the DL digram implementation for this best model.

Model	Min	Max	Mean	Variance
CNN	0.2342	0.2488	0.2381	3.6756e-05
CNN ROLL	0.2391	0.2536	0.2430	3.9674e-05
LSTM	0.2826	0.2874	0.2840	4.6675e-06
LSTM Std. Att.	0.2681	0.2826	0.2720	3.9674e-05
LSTM Att. Conv1D	0.2971	0.3092	0.3005	2.5088e-05
ConvLSTM2D	0.2584	0.2801	0.2671	7.1763e-05
ConvLSTM2D Att. Conv2D	0.2608	0.2705	0.2642	1.3419e-05
WaveNet	0.2512	0.2681	0.2589	7.1180e-05
WaveNet2D ROLL	0.2826	0.2922	0.2864	1.4518e-05

Table 7: Accuracies obtained by 5 repetitive runs of the fitting process for each model applied to the case study.

generate a positive PL. Even with some incorrect predictions, a positive PL can still be obtained. For instance, predicting a movement to the weak up class when the actual movement is a strong up class can still result in a positive PL.

		Predicted					Recall(%)	ACC (%)
		Strong Down	Weak Down	Neutral	Weak Up	Strong Up		
Real	Strong Down	32	7	11	9	10	46.38	30.92
	Weak Down	39	9	17	10	11	10.47	
	Neutral	27	6	23	9	18	27.71	
	Weak Up	23	13	19	9	27	9.89	
	Strong Up	14	8	15	10	38	44.71	
Precision (%)		23.70	20.93	27.06	19.15	36.54		

Table 8: Confusion matrix for LSTM-based model with Conv1D multi-head attention on the validation dataset.

Table 8 confirms in green color all cases with a positive PL. Similarly, cases that generate a negative PL are indicated by the red color. The marginal convergence of the DL model causes predicted values to approximate the main diagonal, drawing them towards the green cells and away from the red cells, which allows for a profitable model even with relatively low accuracy. For this specific case study, with 20% of the data used for validation, the model predicts 173 trades with expected positive PL and 98 trades with expected negative PL, resulting in a total of 75 predicted trades with expected positive PL. To fully assess its potential, the model needs to be evaluated in a production environment, as numerous factors can influence trade execution. This is done using a final test dataset of 30 days, which was not used during the modeling phase, to test the model's performance in a real-world scenario.

Fig. 22 presents the cumulative PL from the execution of trades in simulation for the final test dataset in one category. The left-hand subplot displays the absolute PL achieved with stakes of £3.00 and £100.00, while the right-hand subplot shows the relative PL, calculated as a percentage of the initial investment, for the same stakes. By executing the same predicted trades, it is evident that using a stake of £3.00 generates a higher return on investment (ROI) compared to using £100.00. This is

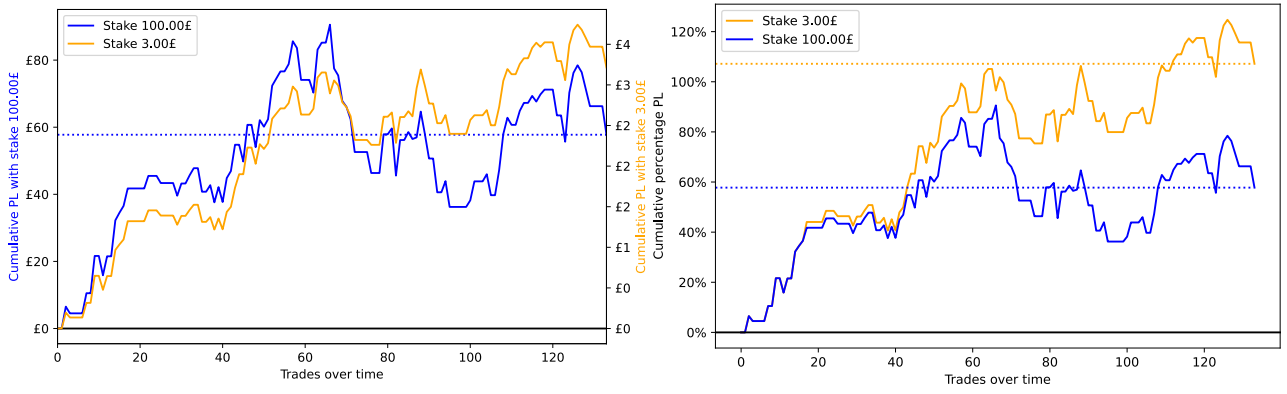


Figure 22: Evolution of the PL during 30 days of trading using the best model, of one category (#41), with stakes of £3.00 and £100.00. On the left subplot, absolute PL values. On the right subplot, relative PL values in relation to the investment stake.

attributed to the market’s absorption capacity. Future studies should focus on evaluating the optimal stake policy to maximize the absolute PL.

Stake	£3.00	£100.00
Trades	134	134
Greens	54	50
Reaches target	14	13
Closes]null;target[	40	37
Swings	14	13
Trailing-Stops	40	37
Reds	36	39
Reaches stop-loss	13	14
Breaks stop-loss	3	5
Closes]null;stop-loss[	23	25
Swings	10	11
Trailing-Stops	26	28
Null	44	45
Positive ticks	134	129
Swings	31	28
Trailing-Stops	103	101
Negative ticks	87	100
Swings	23	26
Trailing-Stops	64	74

Table 9: Global trading simulation results with the model in production on the final test dataset.

Table 9 summarizes the number of executed trades, greens, reds, positive ticks, and negative ticks for the best DL model. The number of trades refers to the total instances when a trading mechanism is instantiated. Greens represent the number of trades that close in profit, while reds indicate the number of trades that close in loss. The sum of greens and reds does not always equal the total number of trades, as there are cases where the trading mechanism closes a position at the same entry price, resulting in no profit or loss. This can occur when the trade reaches its timeout exposure and closes at the entry price. Additionally, if the opening bet is not matched during the opening time,

the result is also null. Positive ticks represent the total ticks that result from profitable trades, while negative ticks account for the ticks resulting in losses. These are the key metrics used to evaluate the effectiveness of the trading policy.

Table 10 presents the confusion matrix for the model’s predictions on the final test dataset, which generates the data in Table 9. The results are consistent; the number of trades instantiated corresponds to the number of times a class that results in an action is predicted, i.e., 134. The total expected number of green trades in Table 10 is 66, and reds are 41, which differs from the actual result in Table 9. This discrepancy can be attributed to instances where the model predicts a direction, but the actual class is Neutral, leading to trades with small positive or negative outcomes. Additionally, the classes are feature-engineered from the raw values representing the integral of price evolution during the prediction period. While this integral can indicate a consistent movement in one direction, rapid, aggressive price movements in the opposite direction can occur, contributing to the discrepancy between the expected and actual outcomes. As a plausible indicator of model resilience, the overall result ratios remain similar in both the production phase (Table 10) and the validation dataset confusion matrices (Table 8).

		Predicted					Recall(%)	ACC (%)
		Strong Down	Weak Down	Neutral	Weak Up	Strong Up		
Real	Strong Down	16	3	2	2	4	59.26	28.26
	Weak Down	12	6	1	5	11	17.14	
	Neutral	14	4	1	3	6	3.57	
	Weak Up	10	2	0	3	8	13	
	Strong Up	2	5	0	5	13	52	
Precision (%)		29.63	30	25	16.67	30.95		

Table 10: Confusion matrix for the LSTM-based model with Conv1D multi-head attention, i.e., best model, on the final test dataset.

5. Managerial implications

Results of this study demonstrate the feasibility and potential of leveraging DL techniques, particularly LSTM-based models with multi-head attention and Conv1D processing, for forecasting short-term price movements in the Betfair UK to Win Horse Racing market. The systematic approach to feature engineering, model training, and production testing provides valuable insights into predictive performance and practical implementation of such models in real-world trading environments. Therefore, managers should consider implementing LSTM-based models with multi-head attention and Conv1D processing, for forecasting price movements, as these models are effective at capturing sequential dependencies in market data, crucial for accurately predicting short-term price fluctuations.

The study confirms that LSTM-based architectures outperform alternative models, indicating their structure is well-suited for capturing sequential dependencies in market depth data. The inclusion of multi-head convolutional attention mechanisms further enhances the model’s ability to focus on key patterns in time-series data, improving accuracy. Hence, managers should prioritize integrating this

DL technique over simpler ones, as they have been proven to deliver superior predictive performance, giving a competitive edge in dynamic markets. Additionally, multi-head convolutional attention mechanisms allow to refine trading algorithms, enabling them to better identify critical patterns in market data, which improves prediction accuracy. Additionally, the introduction of roll padding in CNN-based models contributes to additional performance gains. This implies that, from a managerial perspective, investing in continuous model refinement and newer techniques will ensure predictive performance remains robust, especially when adapting to changing market conditions.

Beyond the classification accuracy of 30.92%, which is significantly higher than the 20% expected from random choices, this research emphasizes the model's ability to generate a positive PL. Results show that the model consistently predicts more trades with expected positive PL than those with expected negative PL, reinforcing its potential for profitable trading. Consequently, it is important to focus not just on accuracy but also on profitability metrics, ensuring that the model's predictions result in real-world gains. Aligning trading strategies with profitability measures and risk-adjusted returns will help in translating predictive success into tangible financial outcomes.

Furthermore, analysis of trade outcomes over a 30-day final test dataset reveals the model maintains a favorable expected PL, though further refinements could improve its performance. Results also highlight the importance of market absorption capacity in determining optimal stake size. Managers should consider adaptive stake sizing strategies based on liquidity constraints, as smaller stakes of £3.00 yield higher returns compared to larger stakes of £100.00. This outcome confirms the influential role of liquidity in exchange markets, which impacts trade execution and profitability. Developing flexible stake-sizing strategies that adapt to market conditions will ensure that trading is optimized without overexposing the system to risks caused by liquidity issues.

Another key observation is the discrepancy between expected and actual numbers of positive and negative trades. This is mainly due to instances where the model predicts a direction but encounters volatile market behavior that diverges from the expected trend. These inconsistencies reinforce the importance of real-time market dynamics, order execution efficiency, and slippage in trading performance. As such, managers should ensure that real-time market conditions, such as volatility and execution efficiency, are continuously monitored to reduce slippage and improve trade execution. Improving order routing systems and adjusting trades dynamically could minimize discrepancies between predicted and actual outcomes.

6. Conclusions

This study advances automated trading strategies by integrating innovative convolutional attention mechanisms and a specialized padding method for time series forecasting. Specifically, it introduces novel enhancements to DL NN architectures, including roll padding, multi-head attention with Conv1D for LSTM-based models, and multi-head attention with Conv2D and roll padding for ConvLSTM2D-based models. Unlike previous studies, the practical implementation extends beyond these technical developments. By focusing on the UK to Win Horse Racing market during the pre-live stage of the world's leading betting exchange, this research proposes a comprehensive end-to-end framework for predicting price movements while emphasizing the importance of model robustness in real-world trading environments. These enhancements significantly improve the learning process, demonstrating the potential of DL approaches when combined with domain expertise in betting exchange markets. At the same time, the study acknowledges challenges such as market volatility, data limitations, and model interpretability. Ultimately, this research opens new avenues for future studies in automated trading systems and contributes to the broader field of financial and sports market analytics.

This study presents several important findings. Firstly, LSTM-based models with multi-head attention are more effective than alternative architectures when it comes to predicting short-term price movements. Secondly, while roll padding enhances CNN-based models, it does not completely address the overfitting challenges arising from limited training data. Thirdly, focusing solely on classification accuracy is not an adequate measure of trading performance; instead, the ability to generate a net positive PL is a more meaningful benchmark for success. Lastly, smaller stake sizes tend to yield higher returns, emphasizing the importance of liquidity and market absorption in optimizing trading strategies.

As such, in terms of scholarly implications, this study introduces a data-driven framework for predicting price changes in betting exchange markets, reducing dependence on expert-driven approaches. From a managerial perspective, findings suggest that managers should prioritize refining and leveraging DL techniques, particularly LSTM models with multi-head convolutional attention mechanisms and roll padding, while optimizing trading strategies for profitability, liquidity constraints, and execution efficiency. Continuous model evaluation and adaptability to market dynamics remain critical for sustaining competitive advantages in real-time trading environments.

Despite the effort to make a valuable contribution, the study is not without its limitations. Future research could refine model convergence, optimize stake policies, and improve adaptability to real-world market fluctuations. Further work may involve expanding the dataset to enhance model generalization, developing adaptive trading strategies based on real-time market conditions, and exploring alternative techniques to dynamically optimize trade execution. Additionally, integrating other risk management strategies to mitigate potential losses will be crucial for the practical deployment of such systems in high-frequency trading environments.

References

- Alfonso-Sánchez, S., Solano, J., Correa-Bahnsen, A., Sendova, K. P., & Bravo, C. (2024). Optimizing credit limit adjustments under adversarial goals using reinforcement learning. *European Journal of Operational Research*, 315, 802–817.
- Betfair (2012). *Sports API Reference Guide - v1.101*. The Sports Exchange API Documentation. Betfair.
- Brown, A., & Yang, F. (2017). The role of speculative trade in market efficiency: Evidence from a betting exchange. *Review of Finance*, 21, 583–603.
- Carter, J. F. (2007). *Mastering the Trade: Proven Techniques for Profiting from Intraday and Swing Trading Setups*. American Media International.
- Chen, Y., Goel, S., & Pennock, D. M. (2008). Pricing combinatorial markets for tournaments. In *Proceedings of the 40th annual ACM symposium on Theory of Computing STOC '08* (pp. 305–314). New York, NY, USA: ACM. doi:10.1145/1374376.1374421.
- Deboeck, G. (1994). *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets*. Wiley Finance. Wiley.
- Deugo, D., Weiss, M., & Kendall, E. (2001). Reusable patterns for agent coordination. In *in: Omicini, A., Coordination of Internet Agents* (pp. 347–368). Springer.
- Ding, X., Zhang, Y., Liu, T., & Duan, J. (2015). Deep learning for event-driven stock prediction. In *Ijcai* (pp. 2327–2333).
- Dixon, M., Klabjan, D., & Bang, J. H. (2015). Implementing deep neural networks for financial market prediction on the intel xeon phi. In *Proceedings of the 8th Workshop on High Performance Computational Finance* (p. 6). ACM.

- Dorffner, G. (1996). Neural networks for time series processing. In *Neural Network World*. Citeseer.
- Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270, 654–669.
- Gonçalves, R., Ribeiro, V. M., & Pereira, F. L. (2023). Variable split convolutional attention: A novel deep learning model applied to the household electric power consumption. *Energy*, 274, 127321.
- Gonçalves, R., Ribeiro, V. M., Pereira, F. L., & Rocha, A. P. (2019). Deep learning in exchange markets. *Information Economics and Policy*, 47, 38–51.
- Gonçalves, R., Rocha, A., & Pereira, F. (2013). High level architecture for trading agents in betting exchange markets. In *Advances in Information Systems and Technologies* (pp. 497–510). Springer Berlin Heidelberg volume 206 of *Advances in Intelligent Systems and Computing*. URL: http://dx.doi.org/10.1007/978-3-642-36981-0_46. doi:10.1007/978-3-642-36981-0_46.
- Hamey, L. G. C. (2015). A functional approach to border handling in image processing. In *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)* (pp. 1–8). doi:10.1109/DICTA.2015.7371214.
- Hatcher, W. G., & Yu, W. (2018). A survey of deep learning: Platforms, applications and emerging research trends. *IEEE Access*, 6, 24411–24432.
- Heaton, J., Polson, N., & Witte, J. H. (2016). Deep learning in finance.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9, 1735–1780.
- Huck, N. (2009). Pairs selection and outranking: An application to the s&p 100 index. *European Journal of Operational Research*, 196, 819–825.
- Huck, N. (2010). Pairs trading and outranking: The multi-step-ahead forecasting case. *European Journal of Operational Research*, 207, 1702–1716.
- Kefalas, P., Holcombe, M., Eleftherakis, G., & Gheorghe, M. (2005). Formal development of reactive agent-based systems. In *Encyclopedia of Information Science and Technology, First Edition* (pp. 1201–1204). IGI Global.
- Korczak, J., & Hemes, M. (2017). Deep learning for financial time series forecasting in a-trader system. In *Computer Science and Information Systems (FedCSIS), 2017 Federated Conference on* (pp. 905–912). IEEE.
- Kriebel, J., & Stitz, L. (2022). Credit default prediction from user-generated text in peer-to-peer lending using deep learning. *European Journal of Operational Research*, 302, 309–323.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86, 2278–2324. doi:10.1109/5.726791.
- LeCun, Y., Kavukcuoglu, K., & Farabet, C. (2010). Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (pp. 253–256). doi:10.1109/ISCAS.2010.5537907.
- Magee, J., Dulay, N., & Kramer, J. (1994). A constructive development environment for parallel and distributed programs. In *Proceedings of 2nd International Workshop on Configurable Distributed Systems* (pp. 4–14). doi:10.1109/IWCDS.1994.289940.

- van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv:1609.03499*.
- Pitt, L. F., Watson, R. T., & Shapiro, D. M. (2005). Www. betfair. com: World wide wagering. *Communications of the Association for Information Systems (Volume 15)*, 15, 149–161.
- Rzayev, K., Sakkas, A., & Urquhart, A. (2025). An adoption model of cryptocurrencies. *European Journal of Operational Research*, 323, 253–266.
- Schnaubelt, M. (2022). Deep reinforcement learning for the optimal placement of cryptocurrency limit orders. *European Journal of Operational Research*, 296, 993–1006.
- Schumaker, R. P., Solieman, O. K., & Chen, H. (2010). Sports data mining. In *Integrated Series in Information Systems* 26. Springer.
- Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., kin Wong, W., & chun Woo, W. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. *arXiv:1506.04214*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *arXiv:1706.03762*.
- Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning* (pp. 2048–2057).
- Yau, S. S., Bae, D.-H., & Wang, J. (1995). An architecture-independent software development approach for parallel processing systems. In *Computer Software and Applications Conference, 1995. COMPSAC 95. Proceedings., Nineteenth Annual International* (pp. 370–375). IEEE.
- Zhong, Y., Xu, W., Li, H., & Zhong, W. (2024). Distributed mean reversion online portfolio strategy with stock network. *European Journal of Operational Research*, 314, 1143–1158.
- Zweigle, O., Kappeler, U. P., Haussermann, K., & Levi, P. (2010). Event based distributed real-time communication architecture for multi-agent systems. In *5th International Conference on Computer Sciences and Convergence Information Technology* (pp. 503–510). doi:10.1109/ICCIT.2010.5711108.

Appendix

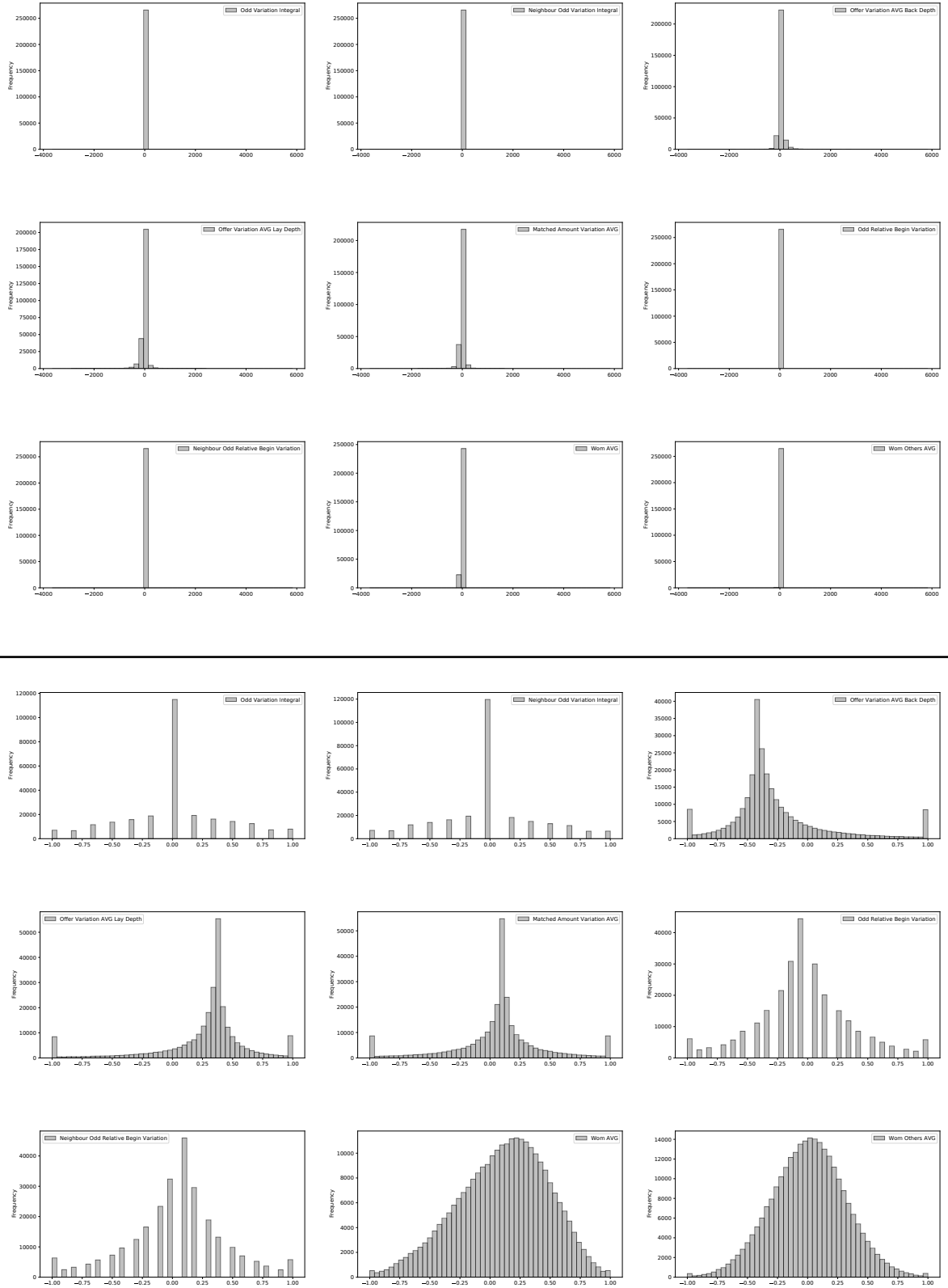


Figure A1: Example of histogram re-scaling with truncated tails at 10% level to find min-max values for input normalization. The illustration represents inputs on category #41 of the rule-based system index of the betting exchange horse race markets case study. Top 9 subplots are raw data and bottom 9 subplots are the result of applying this technique.

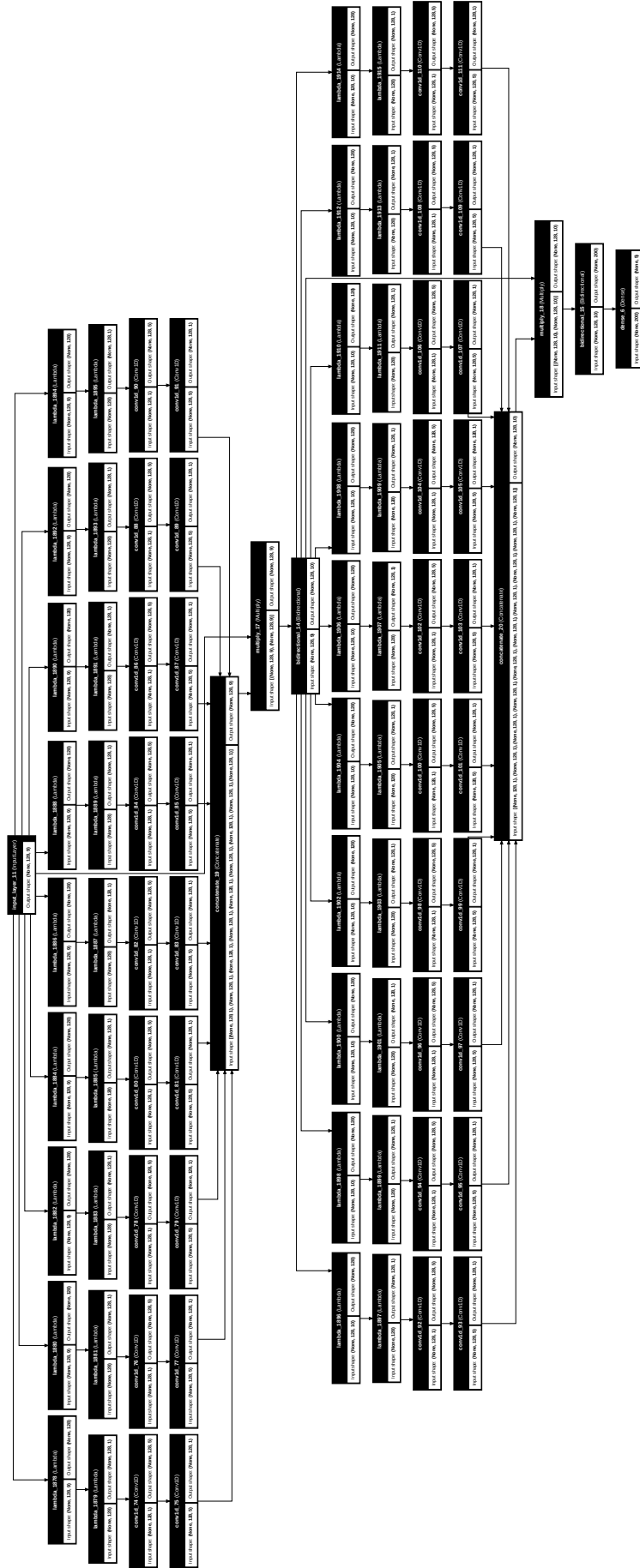


Figure A2: Best model diagram - Multihead Convolutional Attention 1D with bidirectional LSTM. We plot the model with only 2 blocks in depth for simplicity; the model used had 3 blocks with 3 stacked bidirectional LSTMs. Also, in this diagram, the first bidirectional LSTM only uses 5 recurrent cells (returning 10 sequences due to bidirection) for simplicity. For the case study, 50, 20, 5 recurrent cells were used in depth, respectively.

P&L	TM	END_STATE	EVENT	RUNNER	VOLUME	N_R	ENTR	TARG	STO	DIR	T_P	T_L	PT_P	PT_L	O_ODD	C_ODD	O_AM	CAT_AM
£0.00	2	NOT_OPEN	Hum_2nd_Sep	Nightster	38189.27	8	5.5	6.2	5.1	LB	6	4	£11.29	-£7.84	0	0	£0.00	£0.00
£0.00	2	NOT_OPEN	Hum_2nd_Sep	Tectonic	18415.76	8	4.5	3.95	4.9	BL	6	4	£13.92	-£8.16	0	0	£0.00	£0.00
£6.52	1	CLOSED	FloL_2nd_Sep	MiccooL_Bannanas	23465.96	6	4.9	4.6	5.2	BL	3	3	£6.52	-£5.77	4.9	4.6	£100.00	£106.52
-£2.00	1	CLOSED	Hum_2nd_Sep	King_Of_Paradise	15901.31	6	5.1	5.5	4.7	LB	4	4	£7.27	-£8.51	5.1	5	£100.00	£102.00
£0.00	1	NOT_OPEN	FloL_2nd_Sep	Cubichon	22716.31	7	4.4	4.1	4.7	BL	3	3	£7.32	-£6.38	0	0	£0.00	£0.00
£0.00	1	CLOSED	FloL_2nd_Sep	Men_Dont_Cry	24814.14	7	4.5	4.9	4.1	LB	4	4	£8.16	-£9.76	4.5	4.5	£100.00	£100.00
£0.00	2	NOT_OPEN	Hum_2nd_Sep	George_Fenton	20976.69	9	4.9	4.3	5.3	BL	6	4	£13.95	-£7.55	0	0	£0.00	£0.00
£6.00	1	CLOSED	Brig_2nd_Sep	Admiralofthesea	20249.86	9	5.3	5	5.6	BL	3	3	£6.00	-£5.36	5.3	5	£100.00	£106.00
£0.00	1	CLOSED	Muss_3rd_Sep	Mishaal	21280.96	9	5	4.7	5.3	BL	3	3	£6.38	-£5.66	5	5	£100.00	£100.00
£11.11	2	CLOSED	Good_3rd_Sep	Deeds_Not_Words	21824.36	6	6	5.4	6.8	BL	6	4	£11.11	-£11.76	6	5.4	£100.00	£111.11
£0.00	1	CLOSED	Good_3rd_Sep	Argent_Knight	45269.49	9	5.6	5.3	5.9	BL	3	3	£5.66	-£5.08	5.6	5.6	£100.00	£100.00
-£5.77	2	CLOSED	Good_3rd_Sep	Minority_Interest	15054.53	10	4.9	4.3	5.3	BL	6	4	£13.95	-£7.55	4.9	5.2	£100.00	£94.23
£5.66	2	CLOSED	Good_3rd_Sep	Swift_Blade	19614.89	10	5.6	5	6	BL	6	4	£12.00	-£6.67	5.6	5.3	£100.00	£105.66
£0.00	2	NOT_OPEN	Ling_3rd_Sep	Prospera	30635.19	7	4.8	5.4	4.4	LB	6	4	£11.11	-£9.09	0	0	£0.00	£0.00
£10.71	2	CLOSED	Ling_3rd_Sep	Rock_God	29429.46	7	5	5.6	4.6	LB	6	4	£10.71	-£8.70	5	5.6	£100.00	£89.29
£2.27	2	CLOSED	Bath_4th_Sep	Kakapuka	31543.94	7	4.5	3.95	4.9	BL	6	4	£13.92	-£8.16	4.5	4.4	£100.00	£102.27
£2.04	2	CLOSED	Bath_4th_Sep	Dreams_Of_Glory	31085.05	7	4.8	5.4	4.4	LB	6	4	£11.11	-£9.09	4.8	4.9	£100.00	£97.96
£5.20	2	CLOSED	Bath_4th_Sep	Devon_Diva	6343.06	6	4.3	3.85	4.7	BL	6	4	£11.69	-£8.51	4.3	4	£69.27	£74.47
£0.00	2	CLOSED	Kemp_4th_Sep	For_Posternity	17928.36	8	4.6	4	5	BL	6	4	£15.00	-£8.00	4.6	4.6	£100.00	£100.00
£0.00	2	NOT_OPEN	Salis_5th_Sep	Mysterious_Man	29961.6	6	4.2	3.8	4.6	BL	6	4	£10.53	-£8.70	0	0	£0.00	£0.00
£0.00	2	NOT_OPEN	Salis_5th_Sep	New_Rich	14194.09	6	4.2	3.8	4.6	BL	6	4	£10.53	-£8.70	0	0	£0.00	£0.00
£3.74	1	CLOSED	Salis_5th_Sep	Catchanova	19254.96	7	4.2	3.95	4.5	BL	3	3	£6.33	-£6.67	0	0	£0.00	£0.00
-£0.00	1	CLOSED	Salis_5th_Sep	South_Cape	17128.67	7	4.3	4.7	3.95	LB	4	4	£8.51	-£8.86	4.3	4.47	£100.00	£96.26
-£0.00	1	CLOSED	Newe_6th_Sep	Red_Pike	17883.92	10	4.3	4.7	3.95	LB	4	4	£8.51	-£8.86	4.3	4.3	£100.00	£100.00
£0.00	2	CLOSED	Newe_6th_Sep	Noble_Asset	33583.39	9	4.3	4.9	3.95	LB	6	4	£12.24	-£8.86	4.3	4.3	£100.00	£100.00
-£2.13	2	CLOSED	Newe_6th_Sep	Pure Impressions	11972.8	9	4.6	4	5	BL	6	4	£15.00	-£8.00	4.6	4.7	£100.00	£97.87
£0.00	1	CLOSED	Hayd_6th_Sep	Asdpan_Sam	20740.66	8	4.3	4.7	3.95	LB	4	4	£8.51	-£8.86	4.3	4.3	£93.25	£93.25
£0.00	1	NOT_OPEN	Hayd_6th_Sep	Anomaly	19408.38	8	5.8	5.5	6.2	BL	3	3	£5.45	-£6.45	0	0	£0.00	£0.00
£0.00	2	NOT_OPEN	Kemp_7th_Sep	Masterstroke	26429.99	10	6	5.4	6.8	BL	6	4	£11.11	-£11.76	0	0	£0.00	£0.00
-£3.77	2	CLOSED	Ascot_7th_Sep	Stevenson_Sur	10016.69	7	5.1	4.5	5.5	BL	6	4	£13.33	-£7.27	5.1	5.3	£100.00	£96.23
£3.64	2	CLOSED	Ascot_7th_Sep	Forgive	19005.23	8	5.3	5.9	4.9	LB	6	4	£10.17	-£8.16	5.3	5.5	£100.00	£96.36
£0.00	1	NOT_OPEN	Wolv_7th_Sep	Lord_Buffhead	48688.37	10	5.8	6.4	5.4	LB	4	4	£9.38	-£7.41	0	0	£0.00	£0.00
£2.44	2	CLOSED	Font_8th_Sep	Brough_Academy	13833.73	7	4.2	3.8	4.6	BL	6	4	£10.53	-£8.70	4.2	4.1	£100.00	£102.44
£2.13	2	CLOSED	Font_8th_Sep	Chilworth_Screamer	18831.11	7	4.8	4.2	5.2	BL	6	4	£14.29	-£7.69	4.8	4.7	£100.00	£102.13
-£0.00	1	CLOSED	Font_8th_Sep	The_Tracey_Shuffle	44132.1	7	4.1	4.7	3.85	LB	6	4	£12.77	-£6.49	4.1	4.1	£100.00	£100.00
-£7.02	1	CLOSED	Font_8th_Sep	Ovilla	47113.2	9	5.3	5	5.6	BL	3	3	£6.00	-£5.36	5.3	5.7	£100.00	£92.98
£0.00	2	NOT_OPEN	Hum_9th_Sep	Brimham_Boy	19783.33	9	6	7.2	5.6	LB	6	4	£16.67	-£7.14	0	0	£0.00	£0.00
£1.96	1	CLOSED	Hum_9th_Sep	Tiny_Tenor	10597.64	6	5	5.4	4.6	LB	4	4	£7.41	-£8.70	5	5.1	£100.00	£98.04
-£5.13	2	CLOSED	Perth_9th_Sep	Rathmoyle_House	15105.65	6	4.1	4.7	3.85	LB	6	4	£12.77	-£6.49	4.1	3.9	£100.00	£105.13
£4.55	2	CLOSED	Hum_9th_Sep	Getaway_Car	43158.07	7	4.6	4	5	BL	6	4	£15.00	-£8.00	4.6	4.4	£100.00	£104.55
-£4.44	2	CLOSED	Brig_9th_Sep	Aftecchino	34162.89	6	4.3	3.85	4.7	BL	6	4	£11.69	-£8.51	4.3	4.5	£100.00	£95.56
£7.14	1	CLOSED	Brig_9th_Sep	Kamchutka	8723.94	6	4.5	4.2	4.8	BL	3	3	£7.14	-£6.25	4.5	4.2	£100.00	£107.14
£2.08	2	CLOSED	Leic_10th_Sep	Tithius	32783.13	7	4.9	4.3	5.3	BL	6	4	£13.95	-£7.55	4.9	4.8	£100.00	£102.08
£7.84	2	CLOSED	Worc_10th_Sep	Gud_Day	16815.93	6	4.7	5.3	4.3	LB	6	4	£11.32	-£9.30	4.7	5.1	£100.00	£92.16
£0.00	2	NOT_OPEN	Bev_10th_Sep	Hadaj	19770.65	8	4.3	3.85	4.7	BL	6	4	£11.69	-£8.51	0	0	£0.00	£0.00
-£5.00	1	CLOSED	Bev_10th_Sep	Bondi_Beach_Boy	16922.4	8	5.7	5.4	6	BL	3	3	£5.56	-£5.00	5.7	6	£100.00	£95.00
£10.94	2	CLOSED	Bev_10th_Sep	Dubai_Dynamo	20403.79	8	5.7	6.6	5.3	LB	6	4	£13.64	-£7.55	5.7	6.4	£100.00	£89.06
£0.00	2	NOT_OPEN	Bev_10th_Sep	StunIt_Canania	35090.79	9	5.1	4.5	5.5	BL	6	4	£13.33	-£7.27	0	0	£0.00	£0.00
-£6.67	2	CLOSED	Utox_11th_Sep	Teak	19665.05	7	5.6	5	6	BL	6	4	£12.00	-£6.67	5.6	6	£100.00	£93.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Table A1: Logs segment showing the output fields of trading mechanisms (TM 2 - Trailing-stop and 1 - Swing) and parametrization according to the DL models prediction, during several races. The base stake used is £100.00.