UNIVERISTY OF PIRAEUS - DEPARTMENT OF INFORMATICS

Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

MSc «Informatics»

ΠΜΣ «Πληροφορική»

MSc Thesis

Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**Extracting Market Events from Limit Order Book Data: A Data-Driven Approach to Financial Feature Engineering** Εξαγωγή Συμβάντων Αγοράς από Δεδομένα "Limit Order Book": Μια Δεδομενοκεντρική Προσέγγιση για τη Μηχανική Χαρακτηριστικών στη Χρηματοοικονομική Ανάλυση
Student's Name-Surname: Ονοματεπώνυμο Φοιτητή:	**Stamatopoulos Iason** Σταματόπουλος Ιάσων
Father's Name: Πατρώνυμο:	**Georgios** Γεώργιος
Student's ID No: Αριθμός Μητρώου:	ΜΠΠΛ20077
Supervisor: Επιβλέπων:	**Dionisios Sotiropoulos, Assistant Professor** Διονύσιος Σωτηρόπουλος, Επίκουρος Καθηγητής

January 2025 / Ιανουάριος 2025

3-Member Examination Committee

Τριμελής Εξεταστική Επιτροπή

Dionisios Sotiropoulos	**Evangelos Sakkopoulos**	**George Tsihrintzis**
Assistant Professor	**Associate Professor**	**Professor**
Διονύσιος Σωτηρόπουλος	Ευάγγελος Σακκόπουλος	Γεώργιος Τσιχριντζής
Επίκουρος Καθηγητής	Αναπληρωτής Καθηγητής	Καθηγητής

Στην οικογένειά μου

Abstract

Financial markets have evolved into highly complex systems, where the intricate dynamics of trading are captured in the limit order book (LOB), a critical component of market microstructure. This thesis explores the challenge of analyzing LOB data to infer meaningful patterns and predictive signals, leveraging machine learning techniques to address the inherent complexity and non-linearity of the data. Our work focuses on bridging the gap between raw LOB snapshots and the more detailed market by order (MBO) data, which offers richer insights but is often unavailable or underutilized in academic research.

To this end, we developed a greedy algorithm to reconstruct synthetic MBO data from high-frequency LOB snapshots. This reconstructed dataset serves as the foundation for constructing a feature vector set inspired by prior studies, incorporating inferred market event data to compute time-sensitive features aimed at capturing temporal dynamics. The resulting features aim to provide a deeper understanding of market activity and support applications such as alpha signal prediction, a crucial element in financial modeling.

By simplifying access to granular market information and demonstrating the feasibility of extracting temporal dynamics from raw LOB data, this thesis lays the groundwork for future work in predictive modeling and feature validation. Experimental results demonstrate the potential of this method to contribute to trading strategies and market analysis while highlighting avenues for further exploration and optimization.

Περιληψη

Οι χρηματοπιστωτικές αγορές έχουν εξελιχθεί σε εξαιρετικά πολύπλοκα συστήματα, όπου οι περίπλοκες δυναμικές των συναλλαγών καταγράφονται στο βιβλίο οριακών εντολών (LOB), ένα κρίσιμο στοιχείο της μικροδομής της αγοράς. Η παρούσα διπλωματική εργασία διερευνά την πρόκληση της ανάλυσης δεδομένων LOB για την εξαγωγή ουσιαστικών προτύπων και προβλεπτικών σημάτων, αξιοποιώντας τεχνικές μηχανικής μάθησης για την αντιμετώπιση της εγγενούς πολυπλοκότητας και μη γραμμικότητας των δεδομένων. Το έργο μας εστιάζει στη γεφύρωση του χάσματος μεταξύ των στιγμιοτύπων του LOB και των λεπτομερέστερων ανά-εντολή δεδομένων (MBO - Market By Order), τα οποία προσφέρουν πιο πλούσιες πληροφορίες αλλά συχνά δεν είναι διαθέσιμα ή δεν αξιοποιούνται στην ακαδημαϊκή έρευνα.

Για αυτόν τον σκοπό, αναπτύξαμε έναν άπληστο αλγόριθμο για την ανακατασκευή συνθετικών δεδομένων MBO από στιγμιότυπα υψηλής συχνότητας του LOB. Αυτό το ανακατασκευασμένο σύνολο δεδομένων χρησιμεύει ως βάση για τη δημιουργία ενός διανύσματος χαρακτηριστικών εμπνευσμένου από προηγούμενες μελέτες, ενσωματώνοντας τα συνθετικά δεδομένα για τον υπολογισμό χρονικών χαρακτηριστικών που στοχεύουν στη σύλληψη χρονικών δυναμικών. Τα χαρακτηριστικά που προκύπτουν αποσκοπούν στην καλύτερη κατανόηση της δραστηριότητας της αγοράς και στην υποστήριξη εφαρμογών όπως η πρόβλεψη σημάτων alpha, ένα κρίσιμο στοιχείο στη χρηματοοικονομική μοντελοποίηση.

Με την απλοποίηση της πρόσβασης σε λεπτομερείς πληροφορίες της αγοράς και τη διαπίστωση της δυνατότητας εξαγωγής χρονικών δυναμικών από ακατέργαστα δεδομένα LOB, αυτή η διπλωματική εργασία θέτει τις βάσεις για μελλοντική έρευνα στη μοντελοποίηση και επικύρωση χαρακτηριστικών. Τα πειραματικά αποτελέσματα αποδεικνύουν την αποτελεσματικότητα αυτής της μεθόδου να συνεισφέρει σε στρατηγικές συναλλαγών και ανάλυση της αγοράς, ενώ υπογραμμίζουν τις δυνατότητες περαιτέρω διερεύνησης και βελτιστοποίησης.

Contents

Chapter 1 Introduction

Adam Smith claimed that man has an intrinsic *"propensity to truck, barter, and exchange one thing for another"* [1]. This quote carries a profound meaning on the nature of human interactions and our tendencies to engage in mutually beneficial exchanges and transactions. The manifestation of this inclination has evolved dramatically over centuries—from the ancient sailing ships that brought silk from China to Rome in the second century, to the rise and fall of the Portuguese monopoly in the spice trade in the sixteenth century, and to the modern era, where computer chips from Taiwan and feta from Greece traverse the globe seamlessly [2]. This evolution highlights an ongoing integration of national economies into a global economic system. The continuous progression of trade has also seen the establishment of regulated exchanges, creating structured environments for the trading of goods, currencies, and financial assets. In recent years, trade has advanced yet again, reaching new speeds and efficiencies with the advent of algorithmic High-Frequency Trading (HFT).

High-Frequency Trading is a highly specialized strategy in which financial firms leverage advanced technology and complex algorithms to execute trades at extraordinary speeds, often within microseconds or nanoseconds. This process allows HFT firms to capitalize on fleeting market inefficiencies or tiny price movements. By executing vast numbers of trades with extremely short holding periods, typically milliseconds to seconds, HFT aims to capture small profits on each trade, which can aggregate into significant gains. This approach has transformed modern markets, adding liquidity and increasing the speed of transactions, although it has also introduced complexities and debates over fairness and market stability. HFT is particularly complex as the discipline rests at the confluence of two already complicated areas of study: high-frequency finance and computer science. Very few academic institutions offer programs that prepare students to be simultaneously competent in both areas. Most finance-trained people do not understand computer programming, and most computer scientists do not have a grasp on the required highly academic finance [3].

To develop such complex systems, it is essential to study the processes and inner workings of financial markets, a field known as market microstructure [4]. Market microstructure research focuses on the mechanics of price formation, trading rules and behaviors, the structure of exchanges, and the factors influencing liquidity, transaction costs, and information flow. By analyzing these elements, researchers aim to understand how they collectively impact price discovery, market efficiency, and participant behavior within the marketplace. A fundamental aspect of market microstructure is the limit order book (LOB). A LOB is an electronic record of all buy (bid) and sell (ask) orders for a specific financial asset, organized by price levels. It displays the quantities available at each price for buyers and sellers and is typically structured so that the highest bid and the lowest ask, representing the best prices for buying and selling, appear at the top. As new orders are placed, modified, or canceled, the LOB updates, reflecting the dynamic nature of trading activity and allowing for price discovery, liquidity assessment, and trade execution insights [5].

The vast volume and complexity of data generated by the limit order books has introduced challenges in analyzing and predicting market behavior. Traditional models often fall short in capturing the highly non-linear and intricate patterns present in LOB data. Deep learning, a subset of machine learning [6], has shown particular promise for modeling such complex, high-dimensional data. With its ability to model non-linear relationships and learn directly from raw, large-scale data, deep learning seems especially suited to handling the structures within the LOB. This capacity makes deep learning a powerful tool for limit order book analysis, enabling the extraction of meaningful patterns and predictive signals from the vast amount of information generated in real-time trading.

One especially valuable application of deep learning in this context is the prediction of alpha signals (also known as return predictors), which are key to generating insights on expected returns. Accurate prediction of alpha signals is a major focus for financial firms such as hedge funds and banks, where the ideal outcome is to produce high-frequency price forecasts directly from raw limit order book data. However, LOB data is inherently limited in granularity, as it is derived from a broader and more detailed data feed known as market by order (MBO) data. MBO data includes all individual orders, providing a richer view of market activity, but it is largely neglected in current academic literature [7]. Combining LOB and MBO data has the potential to improve forecasting accuracy, as shown by the feature vector sets of by Alec N. Kercheval and Juan Zhang [8], which is was used in many relevant studies to create datasets for training deep learning models.

In this thesis, we aim to bridge this gap by developing a method to infer MBO data from high-frequency limit order book snapshots. To achieve this, we developed a greedy algorithm that deterministically maps consecutive LOB snapshots to a set of inferred trade orders, effectively creating a synthetic MBO dataset from the raw data. By using both MBL data (Market by Limit - Raw Book) and an inferred MBO data, we constructed the proposed feature vector set. This approach could simplify access to one of the most granular sources of market microstructure information, provided that future experiments demonstrate that the resulting features effectively capture key patterns and produce results comparable in quality to those obtained with actual MBO data.

Chapter 2 Limit Order Books

2.1 Origins and Current Landscape

"The bookkeeper of the Amsterdam chamber of the VOC,[1] *Barent Lampe, was seated behind an imposing volume with deckle-edged pages and a vellum cover, the Amsterdam chamber's subscription share register...Under the watchful eye of the notary, the bookkeeper checked each entry and added up the amounts...Altogether, 1,143 investors subscribed to the initial capital of the Company's Amsterdam chamber"* [9, pp. 7–8]. Through Lodewijk Petram's vivid narration in his book on the origins of stock exchanges, we witness the first recorded instance of investment accounting in 1602. The Dutch East India Company initially required shareholders to hold their shares for at least ten years before cashing out. However, realizing this might be too lengthy, they added a clause to the first page of the share register: *"Conveyance or transfer [of shares] may be done through the bookkeeper of this chamber"*. This addition quickly paved the way for the first recorded stock trades. In March 1603, Jan Allertsz tot Londen sold one share to Maria van Egmont and another to Mrs. van Barssum. The use of this subscription register, enabling organized ledgers and share transfers, can be seen as the earliest foundation of what would eventually become the modern order books.

Modern market microstructure evolved into using two main trading mechanisms: quote-driven (or dealer) markets and limit order book (LOB) systems. Initially, markets relied on designated market makers, such as dealers or specialists, to provide liquidity and ensure market continuity. These dealers maintained buy and sell quotes to facilitate trades, playing a central role in price stability. This quote-driven model underpinned much of the microstructure literature before the 1990s and was crucial in markets with lower liquidity or fewer participants [10].

However, advancements in telecommunications and computing in the 1970s and 1980s initiated a shift toward electronic trading systems. A key milestone was the establishment of NASDAQ in 1971, the first fully electronic stock exchange, which marked the beginning of a broad transition toward LOB systems. By the 1990s, LOBs had become prevalent across major financial markets, operating without designated market makers. Instead, participants post limit orders—commitments to trade at specified prices—which remain in the LOB until matched or canceled, while market orders execute immediately at the best available prices. This shift to LOBs has brought numerous benefits, including lower trading costs, improved transparency, and efficient price discovery through automated electronic networks, making it the standard in today's high-frequency, competitive trading environment [10].

[1] Vereenigde Oost-Indische Compagnie (Dutch East India Company)

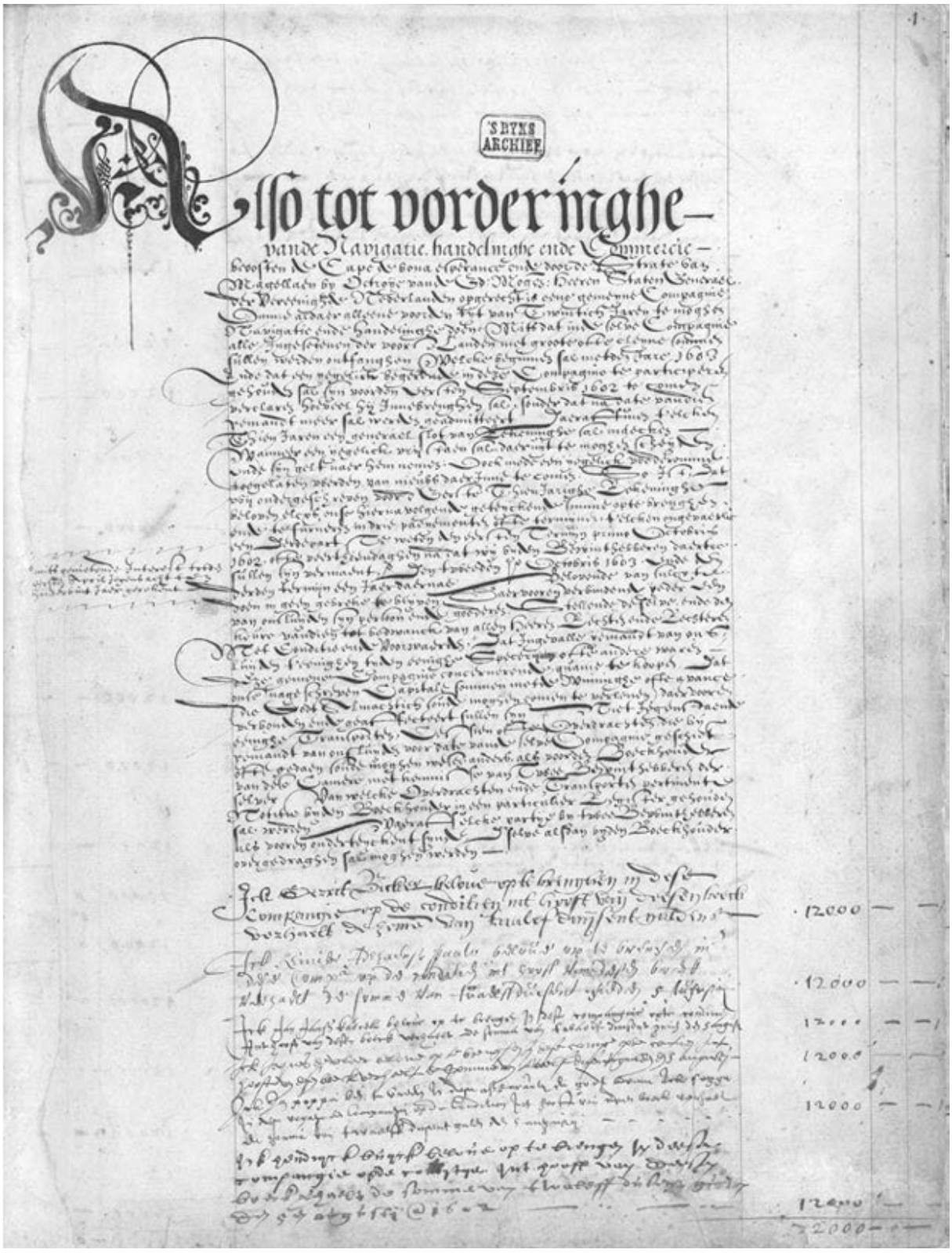


Figure 2.1 | First page of the share register of the Dutch East India Company's Amsterdam chamber [9, p. 11].

2.2 Definitions and Basic Structure

In order to rigorously analyze and model the limit order book (LOB) and its behavior in financial markets, it is essential to establish precise mathematical definitions and formal descriptions of its components and dynamics. Most of the definitions and mathematical structures presented in the following sections are derived from the thorough survey of Gould et al. [11], which synthesizes findings from a wide range of empirical and theoretical research on LOBs. When examples are needed, we will use BTCUSDT[2] as the illustrative asset, as its limit order book data will also serve as the basis for our later experiments. At first, we need to define the building blocks of a LOB which are the orders. A market participant can post a buy or sell order.

Box 2.1 | Buy and Sell Orders

An order $x = (p_x, \omega_x, t_x)$ submitted at time t_x with price p_x and size ω_x ($\omega_x > 0$ for sell orders and $\omega_x < 0$ for buy orders) is the commitment to sell or buy up to $|\omega_x|$ units of the traded asset at a price no less, or no greater, respectively, than p_x.

Prices and sizes of orders in a LOB come with their respective measurement parameters: *tick size* for price and *lot size* for quantity. Together, these are referred to as the *resolution parameters* of the LOB:

Box 2.2 | Resolution Parameters

The tick size π of a LOB is the smallest price interval between different orders within it. All orders must be submitted with a price that is specified to the accuracy of π.

The lot size σ of a LOB is the smallest amount of the asset that can be traded within it. All orders must be submitted with a size $\omega_x \in \{\pm k\sigma | k = 1, 2, \ldots\}$.

For example, for the pair BTCUSDT in the Binance Exchange, the tick size is $\pi = 0.01$ USDT and the lot size is $\sigma = 0.00001$ BTC (Updated on 2022-12-14). This indicates that a trader cannot place an order of volume lower than 0.00001 and the specified price of his order should have precision of at most 2 decimal places.

When a trade is submitted, the trade-matching algorithm of the LOB checks whether it can match it to the already available orders of the opposite side. If this is the case, the trade immediately gets fulfilled. If not, it becomes active and remains active until it gets fulfilled from another trade or it is canceled. We will delve further into the specific rules governing the matching process, but first, defining active orders is essential to clearly establish what constitutes a LOB at any given moment.

Box 2.3 | Limit Order Book at Time t

A LOB $\mathscr{L}(t)$ is the set of all active orders in market at time t

[2] BTCUSDT refers to the trading pair for Bitcoin (BTC) against Tether (USDT), a stablecoin pegged to the US dollar, on cryptocurrency exchanges

The evolution of a LOB $\mathscr{L}(t)$ over time is a càdlàg[3] process [12]. This property arises because for a limit order $x = (p_x, \omega_x, t_x)$ that becomes active, it holds that $x \in \mathscr{L}(t)$ and for $t' \longrightarrow t_x$ holds that $x \notin \mathscr{L}(t')$, or more concisely, $x \notin \lim_{t' \uparrow t_x} \mathscr{L}(t')$.

Box 2.4 | Càdlàg Process

A sample function x on a well-ordered set T is càdlàg if it is continuous from the right and limited from the left at every point. That is, for every $t_0 \in T$, $t \downarrow t_0$ implies $x(t) \longrightarrow x(t_0)$, and for $t \uparrow t_0$, $\lim_{t \uparrow t_0} x(t)$ exists, but need not be $x(t_0)$. A stochastic process X is càdlàg if almost all its sample paths are càdlàg.

The set of active orders in $\mathscr{L}(t)$ can be considered as two distinct sets of ordered queues, one with active buy orders $\mathscr{B}(t)$ and one with active sell orders $\mathscr{A}(t)$. The total sum of the sizes of orders for a specified price and time is called bid-side depth and ask-side depth respectively.

Box 2.5 | Bid-Side Depth and Ask-Side Depth

$$n^b(p,t) := \sum_{\{x \in \mathscr{B}(t) | p_x = p\}} \omega_x, \tag{2.1}$$

$$n^a(p,t) := \sum_{\{x \in \mathscr{A}(t) | p_x = p\}} \omega_x \tag{2.2}$$

We can also calculate the mean bid-side and ask-side depth at a given price level between two consecutive LOB snapshots. This metric provides a useful tool for analyzing liquidity dynamics over short intervals.

Box 2.6 | Mean Bid-Side Depth and Mean Ask-Side Depth between t_1 and t_2

$$\overline{n}^b(p,t_1,t_2) := \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} n^b(p,t)dt, \tag{2.3}$$

$$\overline{n}^a(p,t_1,t_2) := \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} n^a(p,t)dt \tag{2.4}$$

Some fundamental terms that play a crucial role in market analysis are the bid price, ask price, mid-price, and bid-ask spread. The latter two, in particular, are frequent targets in forecasting research, as mid-price prediction [13] and spread-crossing forecasts [14] can lead to effective trading strategies with promising results.

The bid price is the highest stated price within all active buy orders $\mathscr{B}(t)$ and ask price is the lowest stated price within active sell orders $\mathscr{A}(t)$.

[3] Acronym of the french phrase *"continue à droite, limite à gauche"*

Box 2.7 | Bid Price and Ask Price

$$b(t) := \max_{x \in \mathscr{B}(t)} p_x, \tag{2.5}$$

$$a(t) := \min_{x \in \mathscr{A}(t)} p_x \tag{2.6}$$

The mid-price is the average of the current highest bid price and the lowest ask price, representing the fair market value of an asset. The bid-ask spread is the difference between the highest price buyers are willing to pay (bid) and the lowest price sellers are willing to accept (ask), reflecting the transaction cost and market liquidity.

Box 2.8 | Mid Price and Bid-Ask Spread

$$m(t) := \frac{a(t) + b(t)}{2}, \tag{2.7}$$

$$s(t) := a(t) - b(t) \tag{2.8}$$

Many times, it is useful to consider prices in both sets relative to the bid price and ask price in each case. For a buy order price the bid-relative price is $\delta^b(p) := b(t) - p$ and for a sell order price the ask-relative price is $\delta^a(p) := p - a(t)$. So, for any given order we have:

Box 2.9 | Relative Price of an Order $x = (p_x, \omega_x, t_x)$

$$\delta^x := \begin{cases} \delta^b(p), & \textit{if the order is a buy order} \\ \delta^a(p), & \textit{if the order is a sell order} \end{cases} \tag{2.9}$$

Now, we can express the depths and their means in terms of relative prices, just as we did with the absolute prices.

Box 2.10 | Bid-Side Depth and Ask-Side Depth at relative price p

$$N^b(p,t) := \sum_{\{x \in \mathscr{B}(t) | \delta_x = p\}} \omega_x, \tag{2.10}$$

$$N^a(p,t) := \sum_{\{x \in \mathscr{A}(t) | \delta_x = p\}} \omega_x \tag{2.11}$$

Box 2.11 | Mean Bid-Side Depth and Mean Ask-Side Depth at relative price p between t_1 and t_2

$$\overline{N}^b(p, t_1, t_2) := \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} N^b(p,t) dt, \tag{2.12}$$

$$\overline{N}^a(p, t_1, t_2) := \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} N^a(p, t) dt \tag{2.13}$$

By aggregating individual depths over a time interval, we can capture a broader view of liquidity on both sides of the order book. This leads to the mean bid-side and ask-side relative depth profiles.

Box 2.12 | Relative Depth Profiles

The mean bid-side relative depth profile between times t_1 and t_2 is the set of all ordered pairs $(p, \overline{N}^b(p, t_1, t_2))$.

The mean ask-side relative depth profile between times t_1 and t_2 is the set of all ordered pairs $(p, \overline{N}^a(p, t_1, t_2))$.

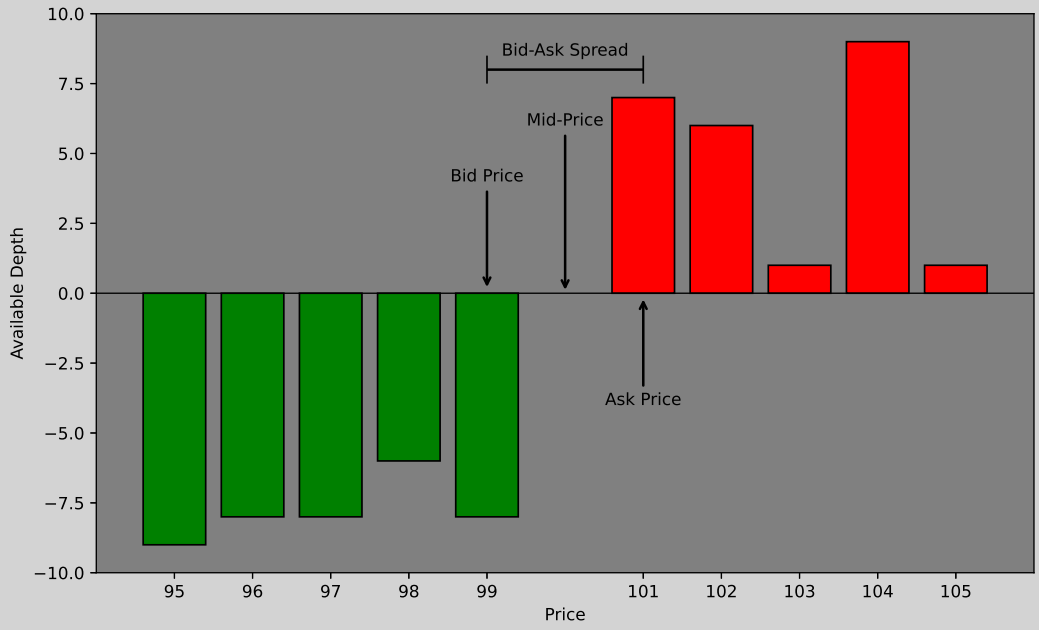


Figure 2.2 | LOB Depth Profile

2.3 The Trade-Matching Algorithm

The matching rules of the LOB characterize its evolution over time. There are three distinct cases of the application of a new buy or sell order x with price p_x at time t_x in relation to $b(t)$ and $a(t)$.

1. When $p_x \leq b(t)$ or $p_x \geq a(t)$, then x is a limit order that becomes active upon arrival and it does not alternate $b(t)$ or $a(t)$.

2. When $b(t) < p_x < a(t)$, then x is a limit order that becomes active upon arrival resulting in a new bid-price or ask-price, $b(t_x) = p_x$ or $a(t_x) = p_x$, respectively.

3. When $p_x \geq a(t)$ or $p_x \leq b(t)$, then x is a market order that matched immediately to one or more active sell or buy orders upon arrival, respectively.The price in which the match occurs is not necessarily equal to the price of the incoming order. The actual price p_x of the active order, as well as the resulting $a(t)$ or $b(t)$ depend on the available depths $n^a(a(t), t)$ or $n^b(b(t), t)$ and the size of the incoming order ω_x. To elaborate further:

Box 2.13 | The New Bid-Price $b(t_x)$ Upon Arrival of a Sell Market Order x

$$b(t_x) = max(p_x, q), \qquad \text{where } q = \arg\max_{k'} \sum_{k=k'}^{b(t)} \left| n^b(k,t) \right| > \omega_x \tag{2.14}$$

Box 2.14 | The New Ask-Price $a(t_x)$ Upon Arrival of a Buy Market Order x

$$a(t_x) = \min(p_x, q), \qquad \text{where } q = \arg\min_{k'} \sum_{k=k'}^{a(t)} n^a(k,t) > |\omega_x| \tag{2.15}$$

Box 2.15 | Price of a Sell Market Order x

$$p_x = \begin{cases} b(t) & \text{if } \omega_x \leq \left| n^b(b(t),t) \right|, \\ \dfrac{\sum_{k \leq q} k \cdot \left| n^b(k,t) \right|}{\omega_x} & \text{where } q = \arg\max_{k'} \sum_{k=k'}^{b(t)} \left| n^b(k,t) \right| > \omega_x. \end{cases} \tag{2.16}$$

Box 2.16 | Price of a Buy Market Order x

$$p_x = \begin{cases} a(t) & \text{if } \omega_x \leq n^a(a(t),t), \\ \dfrac{\sum_{k \geq q} k \cdot n^a(k,t)}{|\omega_x|} & \text{where } q = \arg\min_{k'} \sum_{k=k'}^{a(t)} n^a(k,t) > |\omega_x|. \end{cases} \tag{2.17}$$

To put it simply, regarding the third case:

- **Incoming Market Sell Order**
  If the order size ω_x is small enough to be fulfilled at the current best bid price, then this best bid price remains the price of the sell order. If the order size exceeds the available depth at the best bid, the algorithm identifies the next price levels where it can find enough depth to

fulfill the order. This might require moving down the bid prices until enough cumulative depth is reached. In the same manner, If the sell order size ω_x is less than or equal to the depth at the best bid price $b(t)$, the entire order is executed at that price $b(t)$. If the sell order requires more depth than what is available at the best bid, the price of execution becomes a weighted average of multiple bid prices, where each price is weighted by the depth available at that level, continuing down the bid side until enough depth is gathered.

- **Incoming Market Buy Order**
  If the buy order size ω_x is small enough to be filled at the current best ask price, then this best ask price becomes the price of the buy order. If the order size is larger than the available depth at the best ask, the algorithm moves up the ask prices, accumulating depth across multiple price levels until it can fulfill the buy order. This might require moving up the ask prices until enough cumulative depth is reached. Similarly, If the order size ω_x is smaller than or equal to the depth at the best ask price $a(t)$, the order is executed entirely at that price $a(t)$. If the order size exceeds the best ask depth, the price of execution is calculated as a weighted average across multiple ask prices, where each price level is weighted by the depth consumed at that level, continuing up the ask side until enough depth is gathered.

In practice, when placing an order on exchanges and trading platforms (*Figure 2.3*), traders specify the type of order (either a limit order or a market order). For a limit order, the trader sets both the size (quantity of the asset to be traded) and the price at which they wish to buy or sell. This ensures the order will only execute if the market reaches their specified price, giving the trader more control over the execution price but without a guarantee that the trade will be filled immediately. On the other hand, when selecting a market order, the trader only specifies the size of the order. The execution price is not pre-set. Instead, it will be determined at the moment the order is matched against the best available bid or ask prices in the market. This allows for immediate execution, but the final price is based on current market conditions and may fluctuate slightly.

In financial analysis, a common metric used to assess price evolution over time is known as returns. Returns measure the relative change in price between two points in time, providing insight into the growth or decline of an asset's value [15]. This metric is fundamental in finance as it allows for the comparison of performance across assets and periods, enabling analysis of trends, volatility, and risk.

Box 2.17 | Returns

The bid-price return between times t_1 and t_2 is:

$$R^b(t_1, t_2) := \frac{b(t_2) - b(t_1)}{b(t_1)} \tag{2.18}$$

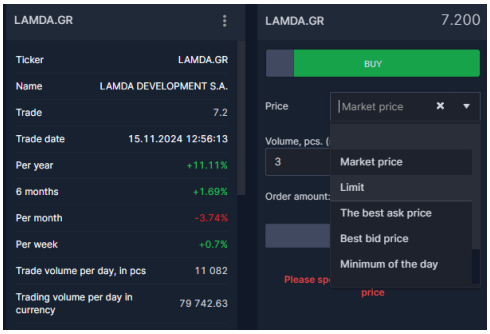
Similarly, we define the ask-price return between times t_1 and t_2 ($R^a(t_1, t_2)$) and the mid-price return between times t_1 and t_2 ($R^m(t_1, t_2)$).

Box 2.18 | Logarithmic Returns

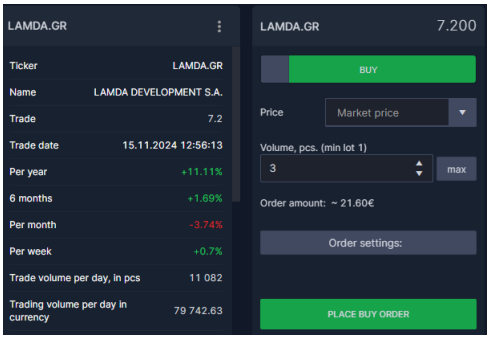
The bid-price logarithmic return between times t_1 and t_2 is:

$$r^b(t_1, t_2) := \log\left(\frac{b(t_2)}{b(t_1)}\right) \tag{2.19}$$

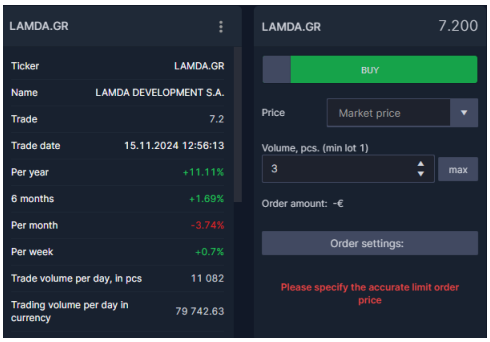
Similarly, we define the ask-price logarithmic return between times t_1 and t_2 ($r^a(t_1, t_2)$) and the mid-price return logarithmic between times t_1 and t_2 ($r^m(t_1, t_2)$).



(a) Option for placing a Market or Limit Order.



(b) Only the size is necessary for the market order.
The amount is just an approximation of the execution price.



(c) The size and the price are mandatory to place a limit order.

Figure 2.3 | Example of market and limit order placement from a real trading platform.

2.4 Priority Mechanisms

As discussed in the previous section, priority in the execution of an incoming trade is given to active orders at the best bid or best ask prices. However, we have yet to address the concept of priority within an individual price level. This refers to the specific order in which active orders at a single price level are executed. In other words, when multiple orders exist at the same price level, it becomes important to understand the sequence in which these orders are fulfilled.

Typically, exchanges use a first-in, first-out (FIFO) or time-priority rule, whereby the earliest order placed at that price level is executed first. This priority mechanism is commonly called **price-time**. It effectively incentivises traders to place limit orders [16]. Without a time-based priority system, traders would have little motivation to reveal their intentions by submitting limit orders sooner than absolutely necessary.

Another priority mechanism frequently used in futures markets is **pro-rata** [17]. Under this system, when multiple orders are competing at the same price level, each active order receives a portion of the trade based on its share of the total available depth. For example, if the total depth at a price level σ consists of two active orders, with one representing $1/3$ and the other $2/3$ of σ, then an incoming buy market order of size 3σ would be split such that one unit σ matches with the smaller order and 2σ matches with the larger order. This priority system presents a unique challenge for traders, as submitting limit orders with larger sizes than the desired trade quantity becomes a viable strategy to secure a greater allocation in pro-rata priority order books [18].

Using price-time (FIFO) versus pro-rata priority produces distinct market results, particularly affecting liquidity, order depth, and price stability. FIFO prioritizes orders based on arrival time, creating a tight order queue where speed is rewarded, leading to shallower order books but higher price stability and efficiency, with fewer price reversals due to lower cancellation rates. In contrast, pro-rata distributes fills based on order size, encouraging larger orders and resulting in greater market depth but with higher variability, as traders tend to over-quote to capture partial fills. This variability leads to more frequent price reversals and slightly reduced price efficiency, as price levels shift more often due to the dynamic queue behavior in pro-rata markets. Thus, FIFO generally fosters faster, more stable price discovery, while pro-rata allows for deeper liquidity at the cost of increased price volatility [18].

Another, not so frequently used priority mechanism is **price-size**. That is, breaking the ties by selecting the active order of the largest size over the rest. The first major exchange that introduced a price-size trading platform was NASDAQ in 2010 (*NASDAQ OMX PSX*).

2.5 Hidden Liquidity

Hidden liquidity has become a prominent feature of modern equity markets, allowing traders to conceal all or part of their orders from public view. This results in market liquidity being composed of both a visible (displayed) and an invisible (non-displayed) component. While non-displayed orders typically lose priority to displayed orders at the same price level, their invisibility offers significant strategic advantages. Traders can place partially or totally hidden orders to mask their intentions, reducing the risk of adverse price movements caused by opportunistic trading behavior. However, this opacity also limits the ability of market participants to assess the true depth of the market and

can obscure the best available prices, as hidden depth may exist within the spread. The rise of hidden liquidity, including mechanisms such as iceberg orders and trading in dark pools, reflects an increasing preference for privacy in trading. This shift has introduced complexities in price discovery and has raised concerns about market transparency, prompting regulatory scrutiny regarding its impact on overall market efficiency [19] [20].

One manifestation of hidden liquidity is the use of **iceberg orders**, which allow traders to display only a small portion of their total order while keeping the remainder hidden. Institutional traders often employ iceberg orders to minimize market impact when executing large trades. By revealing only a fraction of their order, they can avoid signaling their intentions to the market, which could lead to unfavorable price movements. This approach is especially valuable in public markets where transparency can work against large traders, merely exposing a large sell or buy order can create downward or upward pressure on prices, respectively. To mitigate this risk, traders often "slice and dice" their large orders, entering them in smaller increments over time [21]. Alternatively, some institutional participants turn to off-exchange venues, like dark pools, to further conceal their activity.

Dark pools, a type of private exchanges, have seen significant growth in recent years. These venues enable traders to place orders that are entirely invisible to the broader market until execution. By keeping orders hidden, dark pools reduce market impact and associated liquidation costs, making them particularly attractive for executing large trades. Unlike traditional limit order books, dark pools lack intrinsic price discovery; instead, their execution prices are typically derived from publicly quoted prices on exchanges. This linkage allows for potential price manipulation on public exchanges to indirectly influence dark pool execution prices. While dark pools offer advantages in minimizing market impact, they have also drawn scrutiny from regulators due to their lack of transparency and potential for misuse [22].

Hidden liquidity significantly complicates the modeling of limit order books because a snapshot $\mathscr{L}(t)$ at any given time t may not capture the full picture of market activity [11]. The visible orders displayed in the limit order book represent only a portion of the total liquidity available, as hidden or iceberg orders and trades executed in dark pools remain obscured. This incomplete representation introduces challenges in understanding true market depth, assessing supply and demand dynamics, and accurately predicting price movements. Consequently, models based solely on visible LOB data may overlook key liquidity components, leading to potential biases or inaccuracies in market behavior predictions. For this reason, there is research that focused on detecting hidden liquidity events, such as studies on identifying iceberg orders [23].

The table below illustrates how different trading mechanisms, in regards to hidden liquidity, could impact the market differently. Inspired by similar comparative analyses in the literature (Fig. 4.7 [21]), it highlights the potential outcomes for price and volume depending on the mechanism used, under common initial market conditions.

Figure 2.4 | Comparison of trading mechanisms for BTCUSDT using hypothetical scenarios. The current bid and ask prices are $34,950 and $35,050 with depth 1 BTC.

	Public Order Book	**Iceberg Orders**	**Dark Pool**
Order Placement	Place two limit orders to sell 2 BTC at $35,000 and $35,050	Place a limit order to sell 4 BTC at $35,000, displaying only 1 BTC, with the rest hidden	Place a sell order for 4 BTC in a dark pool, indicating willingness to match at the midpoint of the bid-ask spread
Possible Consequences	Sell pressure cancels the bid, a new offer at $34,950 arrives, and quotes shift to $34,900-$34,950	The smaller displayed sell order does not trigger a price drop, and buy orders gradually fill both the visible and hidden portions	A buy order for 4 BTC in the dark pool matches the sell order at the midpoint of $35,000
Possible Outcome	BTC sells below $35,000 due to sell pressure	BTC sells for $35,000 after time passes and new buy orders arrive	BTC sells for $35,000

Chapter 3 Foundational Research and Key Contributions

3.1 Large-Scale Application of Deep Learning to LOBs and Universal Features of Price Dynamics

In his work [24], Justin Sirignano explored the potential of deep learning in modeling the dynamics of limit order books (LOBs). At the time, LOB data posed significant challenges due to its high-dimensional, nonlinear structure and immense volume. The research aimed to demonstrate the advantages of neural networks for capturing the complexities of price movements and risk modeling within this environment. The dataset used in this study encompassed LOB data from nearly 500 stocks listed on the NASDAQ and S&P 500 indices over a period extending from January 2014 to August 2015. The data included around 50 terabytes of raw event-level information, which was preprocessed into structured snapshots for training and testing. The neural networks were trained to predict the joint distribution of the best bid and ask prices at future times.

A key innovation was the introduction of a spatial neural network architecture tailored to exploit local spatial structures within the LOB. This approach outperformed traditional methods such as logistic regression and even standard neural networks, particularly in handling deep levels of the LOB and accurately modeling extreme market events. By leveraging a cluster of 50 GPUs for training, Sirignano demonstrated that neural networks could efficiently process vast datasets while achieving superior out-of-sample prediction accuracy. The results revealed that neural networks not only improved forecasting of future price distributions but also showed potential for advancing risk management by better estimating value-at-risk. This study highlighted the scalability and adaptability of deep learning for financial applications, setting the stage for subsequent research in this domain.

Building on his earlier work, Sirignano et al [25] tried to tackle the question of universality of price formation mechanisms across financial assets. Using a high-frequency database containing billions of transactions for US equities, this study provided evidence of a universal, stationary relationship between order flow history and price changes.

This study employed a recurrent neural network (RNN) architecture with Long Short-Term Memory (LSTM) units [26], specifically designed to capture the temporal dependencies inherent in LOB data. This architecture effectively modeled historical dependencies by incorporating multiple layers of LSTMs, followed by a fully connected layer of rectified linear units (ReLUs). The network's output was a probability distribution predicting the direction of the next price move, generated using a softmax activation function. A significant innovation in this research was the network's use of rich historical data as input, including the order book's state over many lags. This approach allowed the model to incorporate path-dependence and nonlinear relationships between historical order flow and price changes. The training utilized an extensive high-frequency dataset comprising billions of

transactions and quotes for approximately 1,000 US equities, processed using the LOBSTER engine to reconstruct limit order book states. The sheer scale of the data required parallel training across 500 GPU nodes.

Key takeaways from the study are:

- **Nonlinearity:** The study confirmed the nonlinear dependence of price changes on historical order flow, demonstrating that deep learning models significantly outperformed linear models like vector autoregression.
- **Universality:** The model identifies universal features in the relationship between order flow and price changes that are consistent across all stocks analyzed. This ability uncovers the presence of common underlying mechanisms in price formation, irrespective of the specific characteristics of individual stocks.
- **Stationarity:** The model demonstrates remarkable stability in price forecasting accuracy over time, maintaining consistent performance even when tested a year out of sample. This suggests the existence of a stationary relationship between order flow and price changes, as observed in the long-term stability of the predictions.This finding is particularly striking given that prices themselves are typically considered non-stationary, meaning their statistical properties, such as mean and variance, evolve over time.
- **Path-Dependence:** The inclusion of historical order flow and price information greatly enhanced prediction performance, indicating that price dynamics are influenced by long-range dependencies.

These findings emphasize the ability of deep learning to uncover features that are not only robust across various stocks and time periods but also hold significant practical value for trading and risk management. The model's capacity to generalize effectively to out-of-sample stocks demonstrates its versatility, making it particularly useful in real-world scenarios. For instance, it could be applied to newly listed financial instruments, where historical data is limited, offering valuable insights and predictions even in the absence of extensive training data.

3.2 A Feature Vector Set Combining MBL and MBO Data

Alec N. Kercheval and Yuan Zhang in 2015 [8] attempted to model the limit order book dynamics of equity financial instruments applying SVMs [27]. They were inspired by the work of Fletcher and Shaw-Taylor [28] whose work yield promising results using similar approach on the foreign exchange pair EUR/USD. They captured high frequency limit order book data from five NASDAQ-listed stocks and proposed a four-phase design framework for building machine learning models. These models were trained to capture price fluctuations, or more generally, the book dynamics, in the form certain metrics.

The chosen metrics were the mid-price (2.7) movement and the bid-ask spread (2.8) crossing. The latter refers to the event when a market order interacts with the limit order book in such way that it causes the best bid or best ask price to cross over the other side of the bid-ask spread. Those are very common choices in the relevant literature because their successful and consistent prediction

can easily lead to profitable trading strategies. There are three cases regarding the direction of both. Consider an example of bid and ask prices for BTCUSDT at time t $\{\,67416\,\$,\ 67416.34\,\$\,\} \xrightarrow[mid-price]{} 67416.17\,\$$:

Box 3.1 | Mid-Price Movement

1. *Upward movement*: At $t + \Delta t$ the mid-price increases to $\{\,67416.22\,\$,\ 67416.36\,\$\,\} \rightarrow 67416.29\,\$$.
2. *Downward movement*: At $t + \Delta t$ the mid-price decreases to $\{\,67415.92\,\$,\ 67416.02\,\$\,\} \rightarrow 67415.97\,\$$.
3. *Stationary*: At $t + \Delta t$ the mid-price remains constant or, in reality, we usually set a threshold between which the mid-price is considered stationary.

Box 3.2 | Bid-Ask Spread Crossing

1. *Upward price spread crossing*: At $t + \Delta t$ the best bid price $\{\,67416.5\,\}$ is greater than the best ask price $\{\,67416.34\,\}$ at t .
2. *Downward price spread crossing*: At $t + \Delta t$ the best ask price $\{\,67415.78\,\}$ is less than the best bid price $\{\,67416\,\}$ at t.
3. *No price spread crossing*: No occurrence of the above cases ($p^{ask}_{t+\Delta t} \geq p^{bid}_{t}$ and $p^{bid}_{t+\Delta t} \leq p^{ask}_{t}$).

Their four-phase process consisted of the feature extraction, model training, model validation and unseen data classification. A simplified overview of their framework architecture is depicted below:

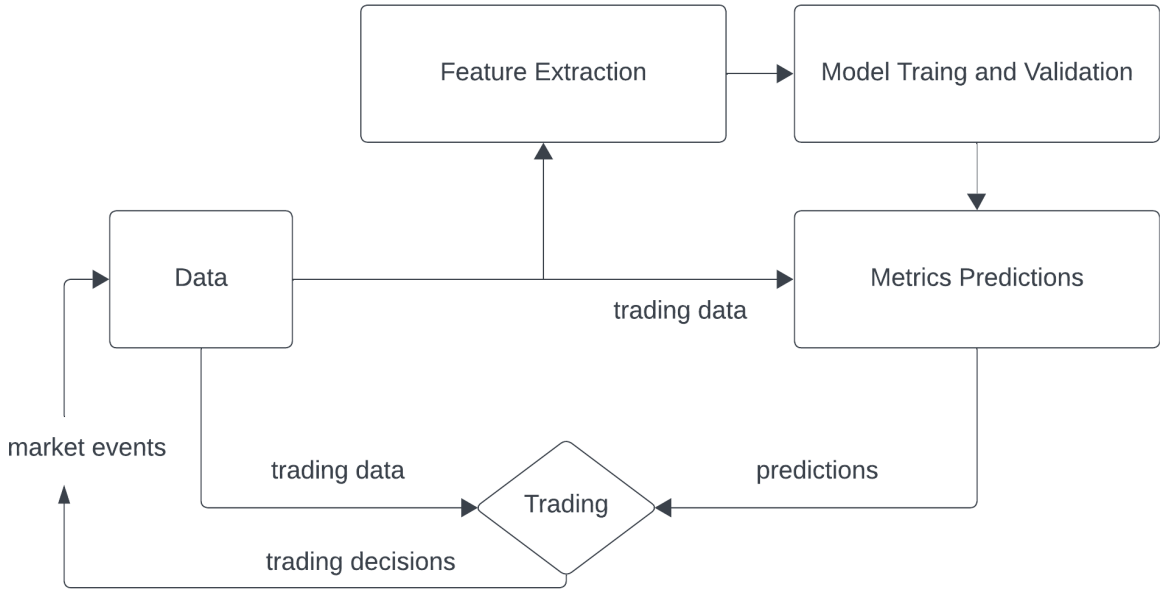


Figure 3.1 | Forecasting Process Architecture

An influential aspect of their work is the feature representation. The authors constructed a feature dataset that incorporated not only raw Market-by-Limit (MBL) data but also enriched it with order-messaging information. This additional information included price and volume derivatives, intensities of different event types and accelerations. Instead of relying solely on data from a single LOB snapshot at each time point, they aimed to capture the temporal dynamics of the LOB's previous states, allowing the model to better understand the evolution of the market activity over time.

The inclusion of these time-sensitive parameters significantly improved model performance, yielding a boost of 3.2% to 6.3% in precision, accuracy, and F1 score metrics [29] compared to models using only raw data. Notably, their feature set has been adopted in other studies [30], [13] and serves as a foundation for much of the subsequent research in this area. In this thesis, we will also aim to utilize a similar feature set, with the key difference being that we will attempt to infer Market-by-Order (MBO) data to generate the time-sensitive parameters. A detailed explanation of each feature vector and its construction will be provided in a later chapter as part of our implementation.

Box 3.3 | Feature Vector Sets

Basic set

$$v_1 = \{P_i^{ask}, V_i^{ask}, P_i^{bid}, V_i^{bid}\}_{i=1}^{n}$$

Time-insensitive set

$$v_2 = \{(P_i^{ask} - P_i^{bid}), (P_i^{ask} + P_i^{bid})/2\}_{i=1}^{n}$$
$$v_3 = \{|P_{i+1}^{ask} - P_i^{ask}|, |P_{i+1}^{bid} - P_i^{bid}|\}_{i=1}^{n}, \quad \text{where} \quad P_{n+1} = P_1$$
$$v_4 = \{\frac{1}{n}\sum_{i=1}^{n} P_i^{ask}, \frac{1}{n}\sum_{i=1}^{n} P_i^{bid}, \frac{1}{n}\sum_{i=1}^{n} V_i^{ask}, \frac{1}{n}\sum_{i=1}^{n} V_i^{bid}\}$$
$$v_5 = \{\sum_{i=1}^{n}(P_i^{ask} - P_i^{bid}), \sum_{i=1}^{n}(V_i^{ask} - V_i^{bid})\}$$

Time-sensitive set

$$v_6 = \{\mathrm{d}P_i^{ask}t, \mathrm{d}P_i^{bid}t, \mathrm{d}V_i^{ask}t, \mathrm{d}V_i^{bid}t\}_{i=1}^{n}$$
$$v_7 = \{\lambda_{\Delta t}^{la}, \lambda_{\Delta t}^{lb}, \lambda_{\Delta t}^{ma}, \lambda_{\Delta t}^{mb}, \lambda_{\Delta t}^{ca}, \lambda_{\Delta t}^{cb}\}$$
$$v_8 = \{\mathbf{1}_{\lambda_{\Delta t}^{la} > \lambda_{\Delta T}^{la}}, \mathbf{1}_{\lambda_{\Delta t}^{lb} > \lambda_{\Delta T}^{lb}}, \mathbf{1}_{\lambda_{\Delta t}^{ma} > \lambda_{\Delta T}^{ma}}, \mathbf{1}_{\lambda_{\Delta t}^{mb} > \lambda_{\Delta T}^{mb}}\}$$
$$v_9 = \{\mathrm{d}\lambda^{ma}t, \mathrm{d}\lambda^{lb}t, \mathrm{d}\lambda^{mb}t, \mathrm{d}\lambda^{la}t\}$$

3.3 A Public Benchmark Dataset and an Experimental Protocol of Research Methods in Mid-Price Forecasting

Ntakaris et al.[30] introduced the first publicly available LOB-ITCH[1] dataset (F1-2010) for research in machine learning for prediction of mid-price movement. The authors derived limit order book data for 5 Finish stocks from the Helsinki exchange operated by NASDAQ NORDIC for 10 consecutive trading days. Additionally to the release of the dataset, they described an experimental protocol for the evaluation of the performance of research methods and different models applied to mid-price forecasting.

They used data only from 10:30 to 18:00, excluding auction periods, in order to ensure consistency and reliability in the analysis of market dynamics. Auction periods, such as pre-opening and post-closing auctions, follow different operational mechanisms compared to continuous trading hours. These periods are characterized by concentrated trading activity aimed at determining opening and closing prices, leading to structural breaks in the limit order book dynamics. By excluding these intervals, the researchers focus on the regular trading period where market conditions are stable and comparable, avoiding biases or anomalies introduced by auction-specific behaviors. This approach ensures that the dataset accurately represents the continuous trading phase with minimal noise from irregular auction-driven activity.

The order book data was processed into three distinct datasets using different normalization strategies: z-score, min-max, and decimal precision. These pre-normalization techniques not only enhanced the utility of the data but also ensured the anonymity of the original dataset. The extracted features were generated following the previously discussed methodology outlined by Kercheval and Zhang[8], which employs an event-based inflow of trading information. They build each data point as a vector containing details of the last 10 consecutive events, decoupling the dataset from time-based inflow to account for the irregular intervals between trading events. To facilitate classification tasks, five integer labels were assigned to each vector, representing the average percentage change in mid-price over the next 1, 2, 3, 5, and 10 consecutive events. The labels correspond to three movement states: 1 for upward movement (percentage change ≥ 0.002), 2 for stationarity (percentage change between -0.00199 and 0.00199), and 3 for downward movement (percentage change ≤ -0.002).

The prediction framework utilized an expanding window cross-validation format (Figure 3.2). Specifically, the 10-day trading dataset was divided into 9 folds. The first fold used the data from the first day for training and the second day for testing. Each subsequent fold expanded the training dataset by including an additional day, while the test dataset remained fixed as the next day in the sequence. For instance, the final fold used data from the first nine days for training and the 10th day for testing. Performance was evaluated across all folds using standard metrics, including accuracy, recall, precision, and F1 score.

They utilized two regression models using the proposed dataset to set the performance baselines for its evaluation: Ridge Regression (RR) and Single Hidden Layer Feedforward Neural Network (SLFN). These methods are chosen because they represent two distinct paradigms of predictive modeling: linear and nonlinear approaches. Ridge Regression is a linear model that is computationally efficient, especially for high-dimensional datasets like the one presented, and offers regularization to

[1]ITCH protocol is a data feed specification provided by exchanges like NASDAQ offering granular event-level data of an instrument's limit order book.

prevent overfitting. On the other hand, the SLFN represents a nonlinear modeling approach, capable of capturing more complex relationships and interactions within the data. By comparing these two methods, the study establishes a foundational understanding of the dataset's predictive capabilities and highlights the potential for improvement with more sophisticated models.

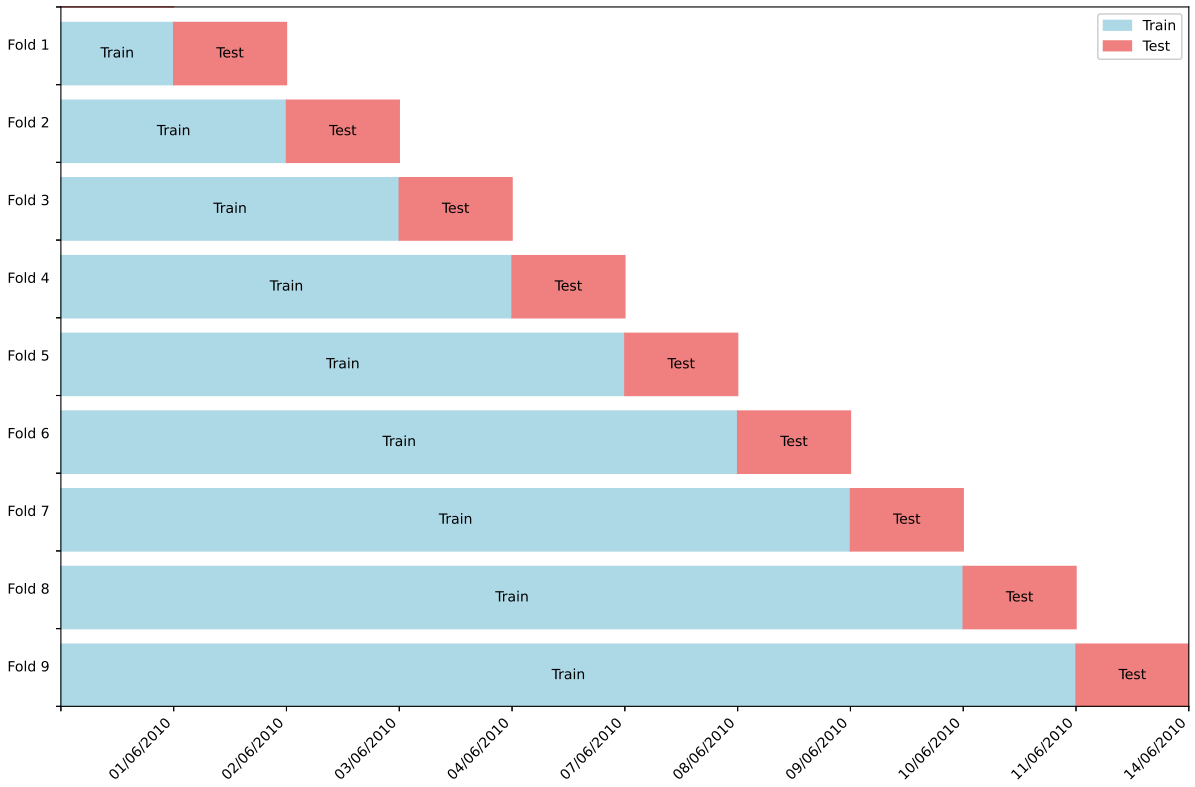


Figure 3.2 | Expanding Window Cross-Validation Setup

3.4 A Structured Approach to Stationary Features and Appropriate Models

Tsantekidis et al. [31] addressed the challenge of price non-stationarity by developing features designed to be robust against distribution shifts. They also investigated the practical question of which types of machine learning models are best suited to leverage these features effectively. Their analysis primarily utilized the F1-2010 benchmark dataset, introduced by Ntakaris et al. [30]. Additionally, they incorporated a simpler dataset comprising FOREX price data for 66 currency pairs.

Non-stationarity of prices presents a significant challenge when modeling financial data with machine learning. In essence, non-stationarity implies that the statistical properties of price data, such as mean and variance, change over time. This dynamic nature creates several issues for predictive modeling:

- ***Shifitng Data Distributions:*** Machine learning models typically assume that the training and testing data come from the same underlying distribution. However, non-stationarity violates

this assumption, as the patterns and relationships in price data evolve over time. This leads to models that perform well on historical data but fail to generalize to new, unseen data.

- ***Noise and Volatility:*** Financial markets are inherently noisy and volatile, with price movements influenced by various factors such as macroeconomic news, market sentiment, and order flow dynamics. These fluctuations exacerbate the non-stationarity problem, making it difficult for models to distinguish between meaningful signals and random noise.
- ***Feature Instability:*** In non-stationary data, the importance of features may change over time. This instability undermines the ability of models to maintain consistent predictive performance across different time periods.
- ***Overfitting Risks:*** Non-stationarity increases the risk of overfitting, as models may capture short-term trends or patterns that are not representative of the long-term behavior of the market.

An existing approach to address this problem involves normalizing the data from each trading day using the mean and standard deviation computed from the previous day's data. However, the authors proposed a novel method by introducing a set of stationary features. They transformed prices into their percentage differences relative to the current mid-price and utilized cumulative sums of order sizes (depth) at each level of the limit order book. Additionally, they incorporated the percentage changes of the mid-price from the previous time step. Features like these proportional differences relative to the mid-price are less sensitive to price-level shifts, while utilizing price relations from prior time steps incorporates temporal dynamics into the feature dataset. Below, we present an organized overview of these features:

Box 3.4 | Price Level Differences

$$p'^{(i)}(t) = \frac{p^{(i)}(t)}{m(t)} - 1 \tag{3.1}$$

where i is the price level and $m(t)$ is the mid-price (2.7).

Box 3.5 | Mid Price Change

$$m'(t) = \frac{m(t)}{m(t-1)} - 1 \tag{3.2}$$

Box 3.6 | Depth Size Cumulative Sum

$$v'^{(k)}(t) = \sum_{i=1}^{k} v^{(i)}(t) \tag{3.3}$$

where k is the price level. This equation is equivalent to the equations (2.1) and (2.2).

The authors analyzed and justified the use of CNNs (*Convolutional Neural Networks*) and LSTMs (*Long Short Term Memory*) networks for this kind of data. CNNs were chosen for their ability to capture the temporal dynamics after the input data was formed as a tensor of temporally ordered features. The same ability is also true to LSTMs because of their the capacity to retain information from previous states by using gated functions which control how much of the input and the previous state will be included in the activation of the network. They proposed a hybrid architecture of those types of models, where the CNN layers act as a feature extractor of the initial limit order book time series producing a new time series of the same length. The LSTM layer is then applied on the time series and produce the final label for each time-step.

This novel approach was rigorously evaluated against standalone CNN and LSTM architectures, as well as simpler models such as SVM and MLP. The models were tested across various prediction horizons using performance metrics including recall, precision, F1-score, and Cohen's kappa. The combined CNN-LSTM model consistently demonstrated greater stability and superior performance, outperforming the other approaches in nearly every metric and across all prediction horizons. The tests were conducted separately using the raw values of the LOB snapshots and the stationary feature set. In all cases, the stationary feature set outperformed the raw values, demonstrating its effectiveness in enhancing model performance.

3.5 State of the Art - DeepLOB

Zihao Zhang et al. [32] expanded upon the hybrid CNN-LSTM architecture introduced by [31], introducing a novel approach to feature extraction and architectural design. Their model, named DeepLOB, integrates a sophisticated Inception Module [33] within the CNN, enabling it to automatically extract multi-scale spatial features from raw LOB data without relying on handcrafted features. Additionally, their innovative framework incorporates a transfer learning strategy to test the model's universality across different stocks. DeepLOB is widely regarded as the state of the art in the field, achieving remarkable performance with accuracy levels nearing 80% across multiple prediction horizons in the case of the F1-2010 benchmark dataset.

The authors choose to not rely on handcrafted features seen in earlier studies, opting instead for a data-driven machine learning approach that integrates feature discovery directly into the modeling workflow. This approach not only streamlines the process by eliminating the need for separate preprocessing steps but also uncovers features that may not be apparent to human designers. They used raw limit order book data, consisting of price and volume information from the top 10 levels of the book (40 features in total), as input to a Convolutional Neural Network (CNN). A key innovation was the use of an Inception Module within the CNN architecture. This module combines multiple convolutional filters of varying sizes, enabling the network to capture market dynamics across different time scales. This design is analogous to applying multiple moving averages simultaneously, allowing the model to identify both short-term and long-term trends in the data. The output of the Inception Module was then passed to an LSTM network, which captures the temporal relationships. The final output of the network classified mid-price movements into one of three categories: upward, downward, or stationary, representing the direction of the mid-price movement. Below, we present a schematic representation of their model architecture.

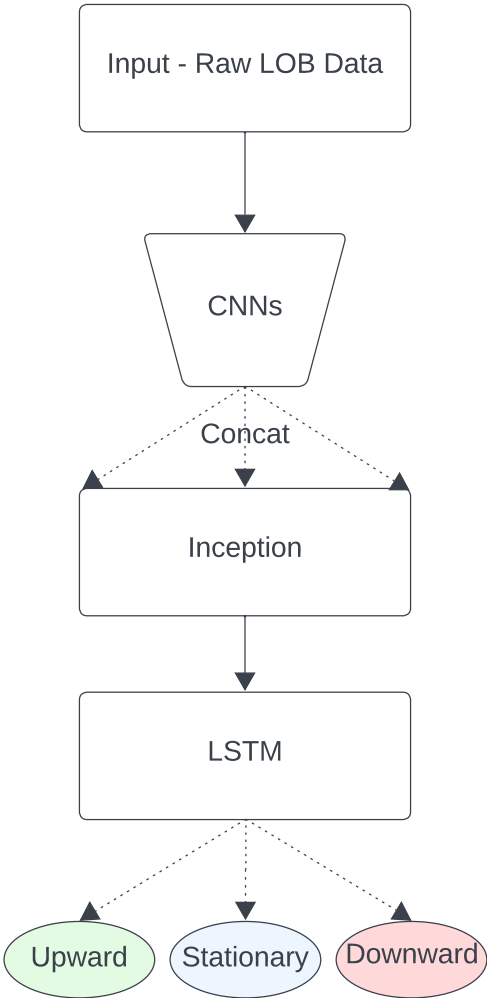


Figure 3.3 | CNN-LSTM with Inception Module architecture

The model was evaluated using the F1-2010 benchmark dataset [30], which, as previously discussed, was specifically created for this type of problem. However, the authors argued that while this dataset is valuable, it has certain limitations. It spans only 10 days of trading data and focuses on a set of relatively illiquid stocks. To address these issues, they employed a second dataset consisting of one year of order book data for five highly liquid stocks[2] from the London Stock Exchange (*LSE*). These stocks provided a richer source of information due to their high liquidity.

The model outperformed all existing algorithms and models on both datasets. An even more remarkable finding emerged when the model trained on the five LSE stocks was directly applied (*transfer learning*) to five different[3] LSE stocks. It demonstrated robust performance, suggesting that the model was able to capture some degree of universality in price formation mechanisms. This result is particularly noteworthy as it indicates the potential for the model to generalize across different financial instruments. The results for these two sets of stocks are presented below.

To address and demystify the "black-box" nature of deep learning models, the study employed the LIME (Local Interpretable Model-Agnostic Explanations) method as part of a sensitivity analysis.

[2] Lloyds Bank (*LLOY*), Barclays (*BARC*), Tesco (*TSCO*), BT, and Vodafone (*VOD*)
[3] HSBC, Glencore (*GLEN*), Centrica (*CNA*), BP, and ITV

Table 3.1 | LSE Dataset Performance

Prediction Horizon	Accuracy %	Precision %	Recall %	F1 %
Training Set: LLOY, BARC, TSCO, BT, VOD				
$k = 20$	70.17	70.17	70.17	70.15
$k = 50$	63.93	63.43	63.93	63.49
$k = 100$	61.52	60.73	61.52	60.65
Transfer Learning Set: GLEN, HSBC, CNA, BP, ITV				
$k = 20$	68.62	68.64	68.63	68.48
$k = 50$	63.44	62.81	63.45	62.84
$k = 100$	61.46	60.68	61.46	60.77

This approach aims to provide interpretable insights into individual predictions by analyzing the relationship between input components and the model's final outputs. By identifying the most influential input features, the authors sought to enhance understanding of how specific elements, such as price and volume dynamics, contribute to price formation. These identified influences were then compared to established domain knowledge, offering a means to validate the model's alignment with known patterns of market behavior and price dynamics.

Chapter 4 Data Extraction and Persistence

4.1 The BTCUSDT Pair from the Binance Exchange

We selected the BTCUSDT financial instrument as the focus of our study, representing the exchange rate of Bitcoin against Tether, a stablecoin tied to the value of the US dollar. Its combination of accessibility, liquidity, and prominence within the cryptocurrency ecosystem makes it an excellent choice for experiments related to our current theme of study. The data was sourced from Binance Exchange, the largest cryptocurrency exchange globally by trading volume.

A major advantage of this selection is that Binance provides free and open access to a comprehensive suite of web services for interacting with various components of the exchange. These services include integration capabilities through multiple protocols such as REST, FIX, and WebSocket, along with a dedicated testing environment (*Testnet*) that allows users to test trading strategies with virtual accounts. Furthermore, several free programming libraries serve as wrappers for Binance's web services, consolidating functionalities and abstracting low-level details to create a more user-friendly interface. For this study, we utilized the python-binance library, the most popular and well-maintained Python wrapper for Binance. Its thorough documentation and active maintenance make it a reliable choice for our requirements.

Another significant benefit of Bitcoin is its high liquidity, commanding the largest percentage of market capitalization within the cryptocurrency space. This high liquidity translates into large order volumes and frequent order book updates, resulting in an information-rich dataset. Such datasets are vital for studies of this nature, as discussed in previous chapters, where the richness of information contributes to more accurate and robust modeling of market dynamics.

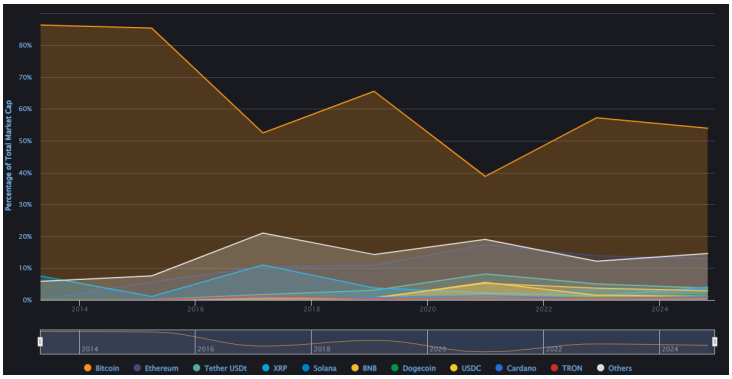


Figure 4.1 | Percentage breakdown of the top ten cryptocurrencies relative to the total market capitalization of all assets. [34]

Although Binance is the largest cryptocurrency exchange by trading volume, the cryptocurrency market remains highly fragmented, presenting challenges for comprehensive analysis. Fragmentation refers to the dispersal of trading activity across numerous exchanges, each with its own order books, trading rules, and liquidity pools. This dispersion can lead to inefficiencies, as price discovery is often incomplete, and discrepancies between exchanges may arise due to differences in trading volumes, participant behaviors, or latency. Regulatory fragmentation further compounds this issue, with varying legal and compliance requirements across jurisdictions, creating additional barriers to a unified market. For instance, some exchanges are restricted from operating in certain regions, leading to pockets of isolated liquidity. Despite these challenges, our selection of Binance mitigates some of these issues, as it commands a dominant share of global cryptocurrency trading and represents one of the most active and liquid platforms. While fragmentation still exists, working with data from the largest exchange ensures that our study captures a significant portion of the market activity.

Another notable drawback is the inherent volatility of cryptocurrency markets. High volatility introduces significant noise in the datasets, complicating the task of predictive modeling. Unlike traditional financial assets, cryptocurrencies often experience rapid and substantial price fluctuations within short time frames, complicating the development of accurate forecasting models. This volatility stems from several factors, including limited supply, speculative trading, regulatory developments, and market sentiment.

In particular, market sentiment plays a pivotal role in driving sudden and unpredictable price changes. Positive developments, such as the approval of Bitcoin ETFs, have been shown to significantly boost prices, reflecting increased investor confidence and market enthusiasm. In contrast, negative sentiment can have a detrimental impact, as evidenced by sharp declines triggered by regulatory crackdowns or public statements from influential figures. For example, announcements or comments by high-profile individuals on social media platforms can lead to rapid price corrections, underscoring the sensitivity of cryptocurrency markets to external influences. Such dynamics highlight the challenges of modeling and forecasting in an environment where sentiment-driven volatility can outweigh fundamental and technical factors.

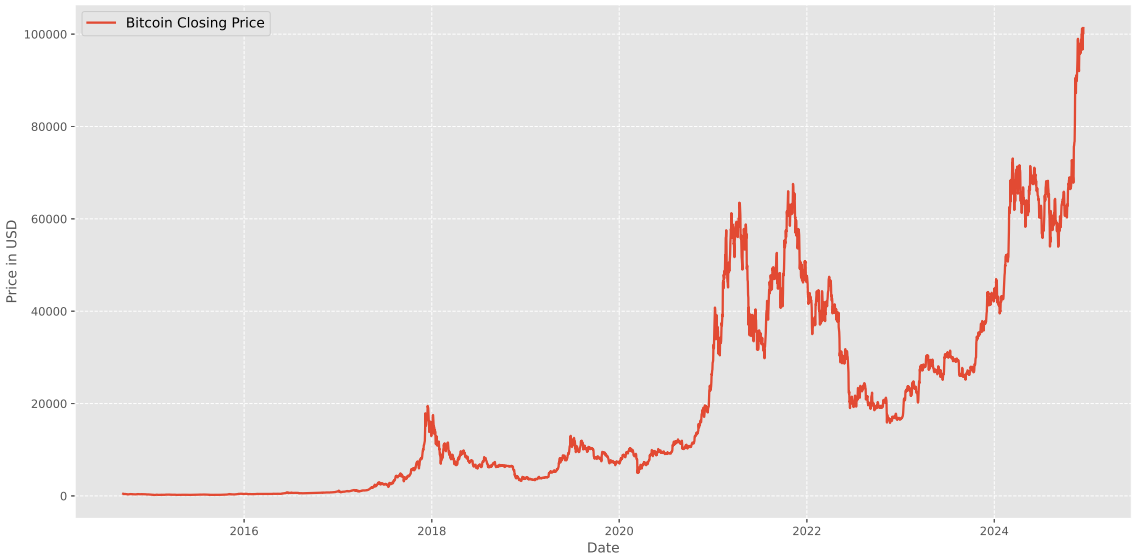


Figure 4.2 | Bitcoin value over time.

4.2 Binance WebSocket Depth Stream

The protocol that best fits our case amongst those provided by the Binance API is the WebSocket protocol, defined in RFC 6455 [35]. WebSocket allows for an upgraded HTTP request that enables bidirectional, asynchronous communication through a single connection between the client and server with minimal resource overhead [36]. This makes WebSocket particularly suitable for scenarios requiring high-frequency data streams, as it eliminates the need for repetitive request-response cycles typical of REST architecture. In our case, where we need to access high-frequency limit order book data, relying on REST would be inefficient and cumbersome. With WebSocket, a single communication channel is opened, enabling live streaming of changes in the limit order book in near real-time without the constant overhead of creating and tearing down connections.

The Binance API offers a specialized stream for our requirements, known as the Depth Stream, specifically designed for managing a local order book. This stream provides incremental updates, reporting only the price and volume changes for affected levels of the order book. To ensure that the client has a consistent and accurate view of the order book, it is the developer's responsibility to first fetch the entire order book via Binance's REST API as an initial snapshot. The Depth Stream then maintains this local order book by applying the incremental updates received from the stream. This approach is highly efficient and minimizes computational overhead, as it avoids repeatedly downloading the entire order book. The Depth Stream offers two options for update intervals: 100 milliseconds or 1000 milliseconds. To maximize granularity and capture market dynamics as precisely as possible, we chose the 100-millisecond interval.

The DepthCacheManager in the python-binance library is a specialized class designed to manage and maintain a real-time local order book for a specified trading pair by utilizing Binance's WebSocket Depth Stream. This manager simplifies the otherwise complex process of handling incremental updates to the order book while ensuring data consistency and reliability. It continuously listens to the WebSocket stream for real-time updates, received at intervals specified by the ws_interval parameter (100ms or 1000ms), providing updated bid and ask prices as market activity occurs. Additionally, to address potential discrepancies due to network delays or dropped messages, the manager periodically fetches a full snapshot of the order book using Binance's REST API (by default every 1800 seconds). The DepthCacheManager is ideal for applications requiring high-frequency trading or live market analysis, as it abstracts the intricacies of reconciling WebSocket data with REST API snapshots. Its integration as an asynchronous context manager further ensures efficient resource management, automatically opening and closing WebSocket connections within the defined context, while offering methods such as get_bids() and get_asks() to access the top levels of the bid and ask books in a sorted manner.

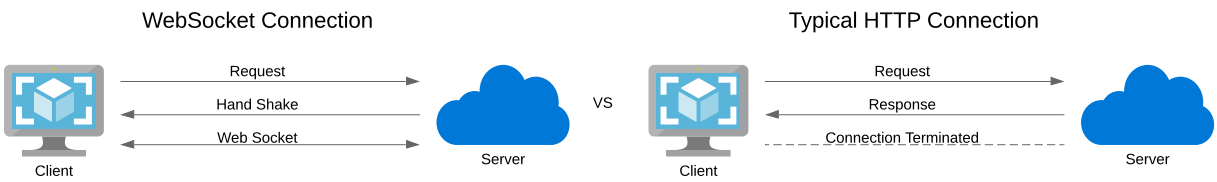


Figure 4.3 | WebSocket and HTTP connections.

4.3 Limit Order Book Data Streaming

We will now demonstrate the practical application of the DepthCacheManager from the python-binance library to retrieve real-time snapshots of the BTCUSDT limit order book. In this example, we will focus on capturing only the top 10 bids and asks from the order book. This choice is not merely for the sake of simplicity but reflects the actual level of detail we will use in our experiments. According to existing literature [37], the best bid and ask levels play a dominant role in price discovery and formation, while the deeper levels of the book are estimated to contribute significantly less, often as little as 20%. By concentrating on the top levels, we align our methodology with established research practices, ensuring both relevance and efficiency in our data analysis.

```python
from time import time
from datetime import datetime
from binance import AsyncClient, DepthCacheManager
import asyncio

async def stream_lob(min = 1):
    client = await AsyncClient.create()
    dcm = DepthCacheManager(client, 'BTCUSDT', ws_interval = 100)

    if min < 1:
        min = 1

    now = time()
    print("Streaming started at "
          + datetime.fromtimestamp(now).strftime('%Y/%m/%d:%H:%M:%S.%f'))
    stop_at = now + min * 60
    print("Estimated end time: "
          + datetime.fromtimestamp(stop_at).strftime('%Y/%m/%d:%H:%M:%S'))

    async with dcm as dcm_socket:
        while True:
            try:
                depth_cache = await dcm_socket.recv()
                print("symbol {}".format(depth_cache.symbol))
                print("top 10 bids")
                print(depth_cache.get_bids()[:10])
                print("top 10 asks")
                print(depth_cache.get_asks()[:10])
                print("last update time {}".format(depth_cache.update_time))
                print("\n")
                now = time()
                if  now > stop_at:
                    print("Streaming ended at "
                          + datetime.fromtimestamp(now).strftime('%Y/%m/%d:%H:%M:%S.%f'))
                    break
            except Exception as ex:
                print(f"An exception occurred: {ex}")
```

```
                break

    await client.close_connection()

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(stream_lob(min = 1))
```

This code demonstrates the implementation of a real-time order book streaming application using the python-binance library. It employs the DepthCacheManager class to maintain a local representation of the limit order book for the trading pair BTCUSDT. To use DepthCacheManager, it is essential to provide an instance of AsyncClient which is a specialized client designed for asynchronous interaction with the Binance API. Additionally, we specify the trading pair of interest (BTCUSDT) and set the WebSocket update interval (ws_interval) to 100 milliseconds.

The script is structured to stream data for an arbitrary number of minutes as determined by the user through the min parameter, with a default runtime of one minute. This parameter is essential to define a condition for terminating the stream and gracefully closing the connection to Binance's WebSocket API. The main loop utilizes the asynchronous context manager functionality to manage the lifecycle of the WebSocket connection efficiently. Within the loop, incremental updates to the order book are received and processed, displaying the top 10 bid and ask levels, along with the last update time. This design not only handles data streaming but also ensures proper resource cleanup by automatically closing the WebSocket connection once the user-defined streaming time has elapsed.

Below are the first three snapshots streamed using the script. The bids and asks are presented as two distinct arrays, each containing two-element subarrays representing price-volume pairs. These arrays are sorted, with bids arranged from the highest to the lowest price and asks from the lowest to the highest price. The update time is highlighted in the output and displayed in milliseconds since the Unix epoch. In particular, the time difference between consecutive updates is exactly 100 milliseconds, as expected given the specified ws_interval. However, as we will observe in the next section, this timing consistency will not always hold. Outliers can occur, particularly during extended streaming sessions. These deviations may be attributed to the periodic full update of the limit order book, network latency, connection instability, or server-side delays. Additional reasons could include temporary congestion in data transmission or local hardware limitations affecting processing speed.

```
Streaming started at 2024/12/14:12:16:05.903802
Estimated end time: 2024/12/14:12:17:05
symbol BTCUSDT
top 10 bids
[[101399.99, 3.88737], [101399.98, 0.0001], [101399.97, 0.0001], [101399.73, 0.48746], [101399.65, 0.79332], [101399.64, 0.0001], [101399.63, 0.0001], [101399.62, 0.0002], [101399.61, 0.36447], [101399.6, 0.0395]]
top 10 asks
[[101400.0, 0.21658], [101400.93, 0.00023], [101400.94, 0.00018], [101400.95, 0.0001], [101401.98, 0.00018], [101402.09, 0.0001], [101402.38, 0.00034], [101402.39, 0.00417], [101402.4, 0.0001], [101402.41, 0.07406]]
last update time 1734171367414


symbol BTCUSDT
top 10 bids
[[101399.99, 3.78678], [101399.98, 0.0001], [101399.97, 0.0001], [101399.73, 0.48746], [101399.65, 0.79332], [101399.64, 0.0001], [101399.63, 0.0001], [101399.62, 0.0002], [101399.61, 0.36447], [101399.6, 0.0395]]
top 10 asks
[[101400.0, 0.21658], [101400.93, 0.00023], [101400.94, 0.00018], [101400.95, 0.0001], [101401.98, 0.00018], [101402.09, 0.0001], [101402.38, 0.00034], [101402.39, 0.00417], [101402.4, 0.0001], [101402.41, 0.01193]]
last update time 1734171367514


symbol BTCUSDT
top 10 bids
[[101399.99, 3.78678], [101399.98, 0.0001], [101399.97, 0.0001], [101399.73, 0.48746], [101399.65, 0.79332], [101399.64, 0.0001], [101399.63, 0.0001], [101399.62, 0.0002], [101399.61, 0.36447], [101399.6, 0.0395]]
top 10 asks
[[101400.0, 0.21658], [101400.93, 0.00023], [101400.94, 0.00018], [101400.95, 0.0001], [101401.98, 0.00018], [101402.09, 0.0001], [101402.38, 0.0001], [101402.39, 0.00417], [101402.4, 0.0001], [101402.41, 0.06966]]
last update time 1734171367614
```

Figure 4.4 | BTCUSDT limit order book real-time streaming.

4.4 Data Persistence

Although streaming real-time data to the console is useful for initial testing and prototyping, it is not sufficient for most practical applications. Effective data handling and persistence are critical priorities, especially when working with massive datasets, as discussed in the previous chapter. In real-world scenarios, managing such large volumes of data typically requires specialized tools and robust infrastructure. However, in our smaller-scale case, we can demonstrate a simpler approach to data persistence by storing the data in commonly used formats, such as CSV files, or directly in a database. A slight modification to the previous Python code is enough to implement this functionality.

For saving limit order book snapshots to the file system, we can use the popular pandas library. It allows us to create a DataFrame to organize the data and provides built-in functionality to export it to CSV format. By incorporating the additional code provided below, we can replace the logic that simply prints data to the console with a mechanism to collect snapshots in a global DataFrame. At the end of the streaming process, this DataFrame can be sorted and saved as a CSV file. This approach not only ensures that the data is properly persisted but also provides a simple yet effective example of data management.

```
lob_snapshots_df = pd.DataFrame(columns=[
    "depth_update_time",
    "bid1_p", "bid1_v", "bid2_p", "bid2_v", "bid3_p", "bid3_v",
    "bid4_p", "bid4_v", "bid5_p", "bid5_v",
    "bid6_p", "bid6_v", "bid7_p", "bid7_v", "bid8_p", "bid8_v",
    "bid9_p", "bid9_v", "bid10_p", "bid10_v",
    "ask1_p", "ask1_v", "ask2_p", "ask2_v", "ask3_p", "ask3_v",
    "ask4_p", "ask4_v", "ask5_p", "ask5_v",
    "ask6_p", "ask6_v", "ask7_p", "ask7_v", "ask8_p", "ask8_v",
    "ask9_p", "ask9_v", "ask10_p", "ask10_v"
])

def insert_snapshot(update_time, bids, asks):
    global lob_snapshots_df

    bids_flat = [item for sublist in bids for item in sublist]
    asks_flat = [item for sublist in asks for item in sublist]

    data = [update_time] + bids_flat + asks_flat

    lob_snapshots_df = pd.concat([lob_snapshots_df if not lob_snapshots_df.empty else None,
                                  pd.DataFrame([data], columns=lob_snapshots_df.columns)],
                                  ignore_index=True)

def dump_to_csv():
    global lob_snapshots_df
    lob_snapshots_df.sort_values(by="depth_update_time", inplace=True)
    file_name = f"{lob_snapshots_df['depth_update_time']
                   .iloc[0]}__{lob_snapshots_df['depth_update_time'].iloc[-1]}.csv"
```

```
30
31      raw_dir = os.path.join(os.path.dirname(__file__), '..', '..', 'data', 'raw')
32      os.makedirs(raw_dir, exist_ok=True)
33
34      file_path = os.path.join(raw_dir, file_name)
35
36      lob_snapshots_df.to_csv(file_path, index=False, float_format='%.5f')
```

For the database implementation, we chose to use PostgreSQL, a popular and free relational database management system known for its reliability and flexibility. The first step in this approach is to design the database schema, starting with a table to store raw limit order book snapshots. This table serves as the foundation for capturing the high-frequency updates from the Depth Stream. As the implementation progresses, we can expand the schema by adding more tables to store derived data, such as market events. The structure of the snapshots table is straightforward, as shown below, with each column representing a price or volume level from the bid and ask sides, along with the timestamp for each update.

```
1   create table public.snapshots
2   (
3       depth_update_time bigint  not null,
4       bid1_p            numeric not null,
5       bid1_v            numeric not null,
6       bid2_p            numeric not null,
7       bid2_v            numeric not null,
8                     ...
9                     ...
10                    ...
11      bid9_p            numeric not null,
12      bid9_v            numeric not null,
13      bid10_p           numeric not null,
14      bid10_v           numeric not null,
15      ask1_p            numeric not null,
16      ask1_v            numeric not null,
17      ask2_p            numeric not null,
18      ask2_v            numeric not null,
19                    ...
20                    ...
21                    ...
22      ask9_p            numeric not null,
23      ask9_v            numeric not null,
24      ask10_p           numeric not null,
25      ask10_v           numeric not null
26  );
```

Once the table is created, we can modify the asynchronous loop in the Python code to insert a new row into this table every time an update event is received. By doing so, each incremental

update to the limit order book is persisted directly into the database in real-time. This approach demonstrates the integration of streaming data with a relational database, a common practice in financial applications and other data-intensive domains.

```
def insert_snapshot(update_time, bids, asks):

    connInfo = CommonUtils.get_dbconnection_info('appsettings.json')

    conn = psycopg2.connect(**connInfo)

    cursor = conn.cursor()

    insert_query = """INSERT INTO snapshots (
    depth_update_time,
    bid1_p, bid1_v, bid2_p, bid2_v, bid3_p, bid3_v, bid4_p, bid4_v, bid5_p, bid5_v,
    bid6_p, bid6_v, bid7_p, bid7_v, bid8_p, bid8_v, bid9_p, bid9_v, bid10_p, bid10_v,
    ask1_p, ask1_v, ask2_p, ask2_v, ask3_p, ask3_v, ask4_p, ask4_v, ask5_p, ask5_v,
    ask6_p, ask6_v, ask7_p, ask7_v, ask8_p, ask8_v, ask9_p, ask9_v, ask10_p, ask10_v)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
            %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""

    bids_flat = [item for sublist in bids for item in sublist]
    asks_flat = [item for sublist in asks for item in sublist]

    data = (update_time,) + tuple(bids_flat) + tuple(asks_flat)

    cursor.execute(insert_query, data)

    conn.commit()
    cursor.close()
    conn.close()
```

After successfully extracting and inserting data into the database, we can perform a quick consistency check on our raw dataset by analyzing the intervals between consecutive snapshots. The denser the intervals, the better the data quality for high-frequency analysis. This can be achieved by querying the database to calculate the total count of consecutive snapshots for each time difference and their respective percentages relative to the entire dataset.

```
WITH time_differences AS (
    SELECT
        depth_update_time,
        depth_update_time - LAG(depth_update_time) OVER (ORDER BY depth_update_time)
        AS time_diff
    FROM
        snapshots
)
SELECT
```

```
    time_diff,
    COUNT(*) AS occurrence_count,
    TO_CHAR(100.0*cast(COUNT(*) as float) / (select count(*) from snapshots),'999D99999999%')
        as per_total
FROM
    time_differences
WHERE
    time_diff IS NOT NULL
GROUP BY
    time_diff
ORDER BY
    occurrence_count DESC;
```

The output of this query, when applied to two 4-hour streaming sessions, are summarized in the tables below to provide an example of the data's temporal distribution.

Table 4.1 | Streaming Result Temporal Distribution

Time Difference (ms)	Total Occurrences	Total Occurrences %
2024-06-19 *07:39:57.822 → 11:39:55.973*		
100	131264	93.1162
101	6908	4.9004
200	2284	1.6202
201	275	0.1951
300	137	0.0972
99	54	0.0383
301	21	0.0149
400	19	0.0135
500	2	0.0014
102	1	0.0007
401	1	0.0007
98	1	0.0007
2024-07-22 *18:39:41.752 → 22:39:41.035*		
100	135052	94.3977
101	7165	5.0081
200	652	0.4557
201	75	0.0524
99	66	0.0461
300	40	0.0280
400	8	0.0056
301	7	0.0049
102	1	0.0007

The results show that the 100ms interval is consistent in over 98% and 99% of the raw snapshots across these sessions, indicating highly reliable data capture with minimal irregularities.

Chapter 5 Synthetic MBO Data

5.1 A Greedy Algorithm to Infer Market Events

In an ideal scenario where consecutive limit order book snapshots are captured with an infinitesimally small time interval Δt, given that market events are applied sequentially, it would be possible to record every order individually as it is executed. This would allow us to precisely infer the type of order, its price level, and its volume. However, in real-world settings, as in our dataset, the time interval between snapshots, while small (100 milliseconds in our case), is not close to this ideal. In practice, snapshots can differ significantly in both price levels and volumes, reflecting the rapid and dense market activity that occurs, particularly in highly liquid markets. These disparities make it challenging to accurately trace every intermediate market event between two snapshots.

Given this reality, the paths between one snapshot and the next can involve an arbitrary number of market events, resulting in an effectively infinite number of possible transitions. To address this, we propose a set of deterministic rules that simplify each transition by identifying the path with the minimal number of market events necessary to account for the observed changes. This approach, which embodies the "greedy" principle, prioritizes the selection of the most straightforward and efficient sequence of market events to explain each transition. It is essential to clarify that this approach can be effective because we are not striving to reconstruct the exact ground truth, but to produce features that effectively encapsulate the underlying market dynamics.

The first step in our approach is to construct a mapping array that encodes the transition between snapshots, focusing solely on prices. For each price in a given snapshot, we perform a binary search against the array of prices in the subsequent snapshot. The results of this search are stored in the mapping array, with slight modifications to enhance readability and ensure symmetry for both bids and asks. As a result, each snapshot is associated with two mapping arrays: one for bids and one for asks. The length of these arrays corresponds to the total number of levels in the order book under consideration, which, in our case, is consistently 10 levels.

The Binary Search algorithm that we applied returns the index of the specified price if it exists in the next snapshot. If the price is not found and is less than one or more prices in the next snapshot array, the algorithm returns a negative number, which is the bitwise complement of the index of the first element larger than the price under test. If it is not found and is greater than all the prices in the next snapshot, the algorithm returns the negative of the array's length. This allows us to determine the following:

- When a previous price exists in the next snapshot and its corresponding index.
- When a previous price does not exist while being greater than every other price.

- When a previous price does not exist while being less than every other price.
- When a previous price does not exist but falls between the values of the next snapshot's prices, and more precisely, its ordered position relative to the new prices.

We manipulate the results of the binary search so the mapping array for both bids and asks for a snapshot will have the following encoding:

- index of existing prices in the next snapshot (1-based indexing).
- -128 (signed byte minimum) for prices that are lower (greater) from every new bid (ask).
- 127 (signed byte maximum) for prices that are greater (lower) from every new bid (ask).
- -(index) of the next larger (smaller) price of the new snapshot for all the non-existing prices that fall between the next snapshot prices

Box 5.1 | Example of decoding a bid map: [1,2,3,4,5,6,7,8,10, -128]

The previously first price (first index of the mapping array) is still in the first position (value of mapping array = 1) in the next snapshot. The same holds for the second, third, fourth, fifth, sixth, seventh, and eighth prices. We can see that the previously ninth bid (ninth element of the mapping array) is now the tenth price in the next snapshot (value of mapping array = 10). Additionally, the previously lowest price (tenth index) is not present in the next snapshot and is lower than all the new prices (indicated by the value -128). The algorithm then follows the "easiest route to truth". It interprets the missing price (previously ninth) as a new limit order. This new limit order is positioned between the previously eighth and ninth prices, so it naturally takes the ninth spot. Consequently, the previously ninth and tenth prices are shifted down.

Using the above logic is only part of the solution. After determining the relationship between the prices in the previous and next snapshots during a transition, it is also necessary to examine the volumes in both cases. Consider an example of such a scenario, as illustrated in the figure below:

```
************************************************************
************************************************************
Lob Transition From: 2024-07-22:10.39.29.429 - 1721687969429
             To:   2024-07-22:10.39.29.529 - 1721687969529
********
* BIDS *
********
1.  Price: 67581.78, Volume: 6.73713   ------>   1.  Price: 67581.78, Volume: 6.73723
2.  Price: 67581.74, Volume: 0.21090   ------>   2.  Price: 67581.74, Volume: 0.21090
3.  Price: 67581.73, Volume: 0.59164   ------>   3.  Price: 67581.73, Volume: 0.59164
4.  Price: 67581.49, Volume: 0.37340   ------>   4.  Price: 67581.49, Volume: 0.37340
5.  Price: 67580.63, Volume: 0.05118   ------>   5.  Price: 67580.63, Volume: 0.05118
6.  Price: 67580.09, Volume: 0.26042   ------>   6.  Price: 67580.09, Volume: 0.26042
7.  Price: 67580.08, Volume: 0.33845   ------>   7.  Price: 67580.08, Volume: 0.33845
8.  Price: 67580.00, Volume: 0.08180   ------>   8.  Price: 67580.00, Volume: 0.08180
9.  Price: 67579.83, Volume: 0.21090   ------>   9.  Price: 67579.95, Volume: 0.08277
10. Price: 67579.72, Volume: 0.00016   ------>   10. Price: 67579.83, Volume: 0.21090

MAP = [1,2,3,4,5,6,7,8,10,-128]

1. LimitOrder - Buy 0.00010 @ 67581.78
2. LimitOrder - Buy 0.08277 @ 67579.95
```

Figure 5.1 | Scenario of bid map: [1,2,3,4,5,6,7,8,10, -128]

As previously stated, we infer that the ninth level of the next snapshot represents a new buy limit order with a price equal to the price at that level and a volume equal to the total depth at that level. Additionally, we observe that the volume of the best bid increases after the transition. In this case, we infer that there was also a limit buy order at the best bid price, with a volume equal to the difference between the volumes before and after the transition. We will rigorously analyze all cases in the following sections.

5.2 Trivial Cases

A very common scenario occurs when the price levels remain unchanged. This corresponds to a mapping array of [1,2,3,4,5,6,7,8,9,10], where the 1-based indices of the array match their values at every position. In such cases, we can make the following inferences:

- Any increase in volume indicates a new limit order, with the volume equal to the difference between the current and previous volumes at the same price level.
- A decrease in volume can result from one of two events:
  1. If it occurs at the top price level, it represents a market order.
  2. If it occurs at lower price levels, it can only indicate a canceled limit order.
- No volume change in any level of the book indicates no market events.

Let's test all of the above cases using simulated transition data.

- **MAP = [1,2,3,4,5,6,7,8,9,10] with no volume changes:**

```
********
* BIDS *
********
1.  Price: 10000.00, Volume: 10.00000   ------>   1.  Price: 10000.00, Volume: 10.00000
2.  Price: 9000.00, Volume: 10.00000    ------>   2.  Price: 9000.00, Volume: 10.00000
3.  Price: 8000.00, Volume: 10.00000    ------>   3.  Price: 8000.00, Volume: 10.00000
4.  Price: 7000.00, Volume: 10.00000    ------>   4.  Price: 7000.00, Volume: 10.00000
5.  Price: 6000.00, Volume: 10.00000    ------>   5.  Price: 6000.00, Volume: 10.00000
6.  Price: 5000.00, Volume: 10.00000    ------>   6.  Price: 5000.00, Volume: 10.00000
7.  Price: 4000.00, Volume: 10.00000    ------>   7.  Price: 4000.00, Volume: 10.00000
8.  Price: 3000.00, Volume: 10.00000    ------>   8.  Price: 3000.00, Volume: 10.00000
9.  Price: 2000.00, Volume: 10.00000    ------>   9.  Price: 2000.00, Volume: 10.00000
10. Price: 1000.00, Volume: 10.00000    ------>   10. Price: 1000.00, Volume: 10.00000

MAP = [1,2,3,4,5,6,7,8,9,10]
```

Figure 5.2 | [1,2,3,4,5,6,7,8,9,10] - No market events.

As expected, the 10 levels of the order book are identical in both prices and volumes, indicating that no market events occurred.

- **MAP = [1,2,3,4,5,6,7,8,9,10] with increased volume:**

```
********
* ASKS *
********
1.  Price: 11000.00, Volume: 10.00000   ------>   1.  Price: 11000.00, Volume: 11.00000
2.  Price: 12000.00, Volume: 10.00000   ------>   2.  Price: 12000.00, Volume: 10.00000
3.  Price: 13000.00, Volume: 10.00000   ------>   3.  Price: 13000.00, Volume: 11.00000
4.  Price: 14000.00, Volume: 10.00000   ------>   4.  Price: 14000.00, Volume: 10.00000
5.  Price: 15000.00, Volume: 10.00000   ------>   5.  Price: 15000.00, Volume: 10.00000
6.  Price: 16000.00, Volume: 10.00000   ------>   6.  Price: 16000.00, Volume: 10.00000
7.  Price: 17000.00, Volume: 10.00000   ------>   7.  Price: 17000.00, Volume: 10.00000
8.  Price: 18000.00, Volume: 10.00000   ------>   8.  Price: 18000.00, Volume: 11.00000
9.  Price: 19000.00, Volume: 10.00000   ------>   9.  Price: 19000.00, Volume: 10.00000
10. Price: 20000.00, Volume: 10.00000   ------>   10. Price: 20000.00, Volume: 10.00000

MAP = [1,2,3,4,5,6,7,8,9,10]

1. LimitOrder - Sell 1.00000 @ 11000.00
2. LimitOrder - Sell 1.00000 @ 13000.00
3. LimitOrder - Sell 1.00000 @ 18000.00
```

Figure 5.3 | [1,2,3,4,5,6,7,8,9,10] - New Limit Orders.

The algorithm correctly infers that there are three sell limit orders corresponding to volume differences at the first, third, and eighth levels, with sizes equal to the differences in depth at each respective level.

- **MAP = [1,2,3,4,5,6,7,8,9,10] with decreased volume at best price:**

```
********
* ASKS *
********
1.  Price: 11000.00, Volume: 10.00000   ------>   1.  Price: 11000.00, Volume: 8.00000
2.  Price: 12000.00, Volume: 10.00000   ------>   2.  Price: 12000.00, Volume: 10.00000
3.  Price: 13000.00, Volume: 10.00000   ------>   3.  Price: 13000.00, Volume: 10.00000
4.  Price: 14000.00, Volume: 10.00000   ------>   4.  Price: 14000.00, Volume: 10.00000
5.  Price: 15000.00, Volume: 10.00000   ------>   5.  Price: 15000.00, Volume: 10.00000
6.  Price: 16000.00, Volume: 10.00000   ------>   6.  Price: 16000.00, Volume: 10.00000
7.  Price: 17000.00, Volume: 10.00000   ------>   7.  Price: 17000.00, Volume: 10.00000
8.  Price: 18000.00, Volume: 10.00000   ------>   8.  Price: 18000.00, Volume: 10.00000
9.  Price: 19000.00, Volume: 10.00000   ------>   9.  Price: 19000.00, Volume: 10.00000
10. Price: 20000.00, Volume: 10.00000   ------>   10. Price: 20000.00, Volume: 10.00000

MAP = [1,2,3,4,5,6,7,8,9,10]

1. MarketOrder - Buy 2.00000 @ 11000.00
```

Figure 5.4 | [1,2,3,4,5,6,7,8,9,10] - New Market Order.

There was a reduction in the total depth at the best ask price, indicating a market buy order with a size equal to the volume that was removed.

- **MAP = [1,2,3,4,5,6,7,8,9,10] with decreased volume at lower prices:**

```
********
* ASKS *
********
1.  Price: 11000.00, Volume: 10.00000   ------>   1.  Price: 11000.00, Volume: 10.00000
2.  Price: 12000.00, Volume: 10.00000   ------>   2.  Price: 12000.00, Volume: 9.00000
3.  Price: 13000.00, Volume: 10.00000   ------>   3.  Price: 13000.00, Volume: 10.00000
4.  Price: 14000.00, Volume: 10.00000   ------>   4.  Price: 14000.00, Volume: 8.00000
5.  Price: 15000.00, Volume: 10.00000   ------>   5.  Price: 15000.00, Volume: 7.50000
6.  Price: 16000.00, Volume: 10.00000   ------>   6.  Price: 16000.00, Volume: 10.00000
7.  Price: 17000.00, Volume: 10.00000   ------>   7.  Price: 17000.00, Volume: 10.00000
8.  Price: 18000.00, Volume: 10.00000   ------>   8.  Price: 18000.00, Volume: 10.00000
9.  Price: 19000.00, Volume: 10.00000   ------>   9.  Price: 19000.00, Volume: 9.95000
10. Price: 20000.00, Volume: 10.00000   ------>   10. Price: 20000.00, Volume: 10.00000

MAP = [1,2,3,4,5,6,7,8,9,10]

1. CancelOrder - Sell 1.00000 @ 12000.00
2. CancelOrder - Sell 2.00000 @ 14000.00
3. CancelOrder - Sell 2.50000 @ 15000.00
4. CancelOrder - Sell 0.05000 @ 19000.00
```

Figure 5.5 | [1,2,3,4,5,6,7,8,9,10] - Cancellations.

All the missing volume at each level is interpreted as a cancellation of an existing order, with the canceled volume equal to the difference in depth.

- **MAP = [1,2,3,4,5,6,7,8,9,10] with mixed events:**

```
********
* BIDS *
********
1.  Price: 10000.00, Volume: 10.00000   ------>   1.  Price: 10000.00, Volume: 9.00000
2.  Price: 9000.00, Volume: 10.00000    ------>   2.  Price: 9000.00, Volume: 11.00000
3.  Price: 8000.00, Volume: 10.00000    ------>   3.  Price: 8000.00, Volume: 8.00000
4.  Price: 7000.00, Volume: 10.00000    ------>   4.  Price: 7000.00, Volume: 10.00000
5.  Price: 6000.00, Volume: 10.00000    ------>   5.  Price: 6000.00, Volume: 10.00000
6.  Price: 5000.00, Volume: 10.00000    ------>   6.  Price: 5000.00, Volume: 10.00000
7.  Price: 4000.00, Volume: 10.00000    ------>   7.  Price: 4000.00, Volume: 9.50000
8.  Price: 3000.00, Volume: 10.00000    ------>   8.  Price: 3000.00, Volume: 12.00000
9.  Price: 2000.00, Volume: 10.00000    ------>   9.  Price: 2000.00, Volume: 10.00000
10. Price: 1000.00, Volume: 10.00000    ------>   10. Price: 1000.00, Volume: 10.00000

MAP = [1,2,3,4,5,6,7,8,9,10]

1. MarketOrder - Sell 1.00000 @ 10000.00
2. LimitOrder - Buy 1.00000 @ 9000.00
3. CancelOrder - Buy 2.00000 @ 8000.00
4. CancelOrder - Buy 0.50000 @ 4000.00
5. LimitOrder - Buy 2.00000 @ 3000.00
```

Figure 5.6 | [1,2,3,4,5,6,7,8,9,10] - All types of orders.

We observe that the aggregation of cases works flawlessly.

5.3 Extreme Cases

Extreme cases refer to scenarios where none of the prices from the previous snapshot match those in the new snapshot. These cases can be categorized into two distinct types, which are symmetrical for bids and asks:

1. **All new bids are greater than the previous bids (or, conversely, all new asks are lower than the previous asks) - MAP = [-128,-128,-128,-128,-128,-128,-128,-128,-128,-128]:** This scenario suggests that an equal number of new limit orders were placed, surpassing the previous bids or asks with better (more competitive) prices.

2. **All new bids are lower than the previous bids (or all new asks are higher than the previous asks) - MAP = [127,127,127,127,127,127,127,127,127,127]:** This situation indicates the occurrence of a very large market order that consumed the entire volume of the first 10 levels. As a result, the prices and volumes observed in the next snapshot have shifted upward from the deeper levels of the order book.

- **MAP = [-128,-128,-128,-128,-128,-128,-128,-128,-128,-128] for bids:**

```
********
* BIDS *
********
1.  Price: 10000.00, Volume: 10.00000   ------>   1.  Price: 20000.00, Volume: 1.00000
2.  Price: 9000.00, Volume: 10.00000    ------>   2.  Price: 19000.00, Volume: 2.00000
3.  Price: 8000.00, Volume: 10.00000    ------>   3.  Price: 18000.00, Volume: 3.00000
4.  Price: 7000.00, Volume: 10.00000    ------>   4.  Price: 17000.00, Volume: 4.00000
5.  Price: 6000.00, Volume: 10.00000    ------>   5.  Price: 16000.00, Volume: 5.00000
6.  Price: 5000.00, Volume: 10.00000    ------>   6.  Price: 15000.00, Volume: 6.00000
7.  Price: 4000.00, Volume: 10.00000    ------>   7.  Price: 14000.00, Volume: 7.00000
8.  Price: 3000.00, Volume: 10.00000    ------>   8.  Price: 13000.00, Volume: 8.00000
9.  Price: 2000.00, Volume: 10.00000    ------>   9.  Price: 12000.00, Volume: 9.00000
10. Price: 1000.00, Volume: 10.00000    ------>   10. Price: 11000.00, Volume: 10.00000

MAP = [-128,-128,-128,-128,-128,-128,-128,-128,-128,-128]

1. LimitOrder - Buy 1.00000 @ 20000.00
2. LimitOrder - Buy 2.00000 @ 19000.00
3. LimitOrder - Buy 3.00000 @ 18000.00
4. LimitOrder - Buy 4.00000 @ 17000.00
5. LimitOrder - Buy 5.00000 @ 16000.00
6. LimitOrder - Buy 6.00000 @ 15000.00
7. LimitOrder - Buy 7.00000 @ 14000.00
8. LimitOrder - Buy 8.00000 @ 13000.00
9. LimitOrder - Buy 9.00000 @ 12000.00
10. LimitOrder - Buy 10.00000 @ 11000.00
```

Figure 5.7 | [-128,-128,-128,-128,-128,-128,-128,-128,-128,-128] - Bid Side.

As we can observe, the algorithm generates an entirely new set of ten buy limit orders that, due to their more competitive prices, occupy the top levels of the order book. Additionally, we infer that the previous ten orders were simply shifted downward to the lower levels of the book.

- **MAP = [-128,-128,-128,-128,-128,-128,-128,-128,-128,-128] for asks:**

```
********
* ASKS *
********
1.  Price: 11000.00, Volume: 10.00000   ------>   1.  Price: 1000.00, Volume: 1.00000
2.  Price: 12000.00, Volume: 10.00000   ------>   2.  Price: 2000.00, Volume: 2.00000
3.  Price: 13000.00, Volume: 10.00000   ------>   3.  Price: 3000.00, Volume: 3.00000
4.  Price: 14000.00, Volume: 10.00000   ------>   4.  Price: 4000.00, Volume: 4.00000
5.  Price: 15000.00, Volume: 10.00000   ------>   5.  Price: 5000.00, Volume: 5.00000
6.  Price: 16000.00, Volume: 10.00000   ------>   6.  Price: 6000.00, Volume: 6.00000
7.  Price: 17000.00, Volume: 10.00000   ------>   7.  Price: 7000.00, Volume: 7.00000
8.  Price: 18000.00, Volume: 10.00000   ------>   8.  Price: 8000.00, Volume: 8.00000
9.  Price: 19000.00, Volume: 10.00000   ------>   9.  Price: 9000.00, Volume: 9.00000
10. Price: 20000.00, Volume: 10.00000   ------>   10. Price: 10000.00, Volume: 10.00000

MAP = [-128,-128,-128,-128,-128,-128,-128,-128,-128,-128]

1. LimitOrder - Sell 1.00000 @ 1000.00
2. LimitOrder - Sell 2.00000 @ 2000.00
3. LimitOrder - Sell 3.00000 @ 3000.00
4. LimitOrder - Sell 4.00000 @ 4000.00
5. LimitOrder - Sell 5.00000 @ 5000.00
6. LimitOrder - Sell 6.00000 @ 6000.00
7. LimitOrder - Sell 7.00000 @ 7000.00
8. LimitOrder - Sell 8.00000 @ 8000.00
9. LimitOrder - Sell 9.00000 @ 9000.00
10. LimitOrder - Sell 10.00000 @ 10000.00
```

Figure 5.8 | [-128,-128,-128,-128,-128,-128,-128,-128,-128,-128] - Ask Side.

It behaves symmetrically on the ask side, where the more competitive prices correspond to a new set of lower price levels.

- **MAP = [127,127,127,127,127,127,127,127,127,127] for bids:**

```
********
* BIDS *
********
1.  Price: 10000.00, Volume: 10.00000   ------>   1.  Price: 900.00, Volume: 1.00000
2.  Price: 9000.00, Volume: 10.00000    ------>   2.  Price: 800.00, Volume: 2.00000
3.  Price: 8000.00, Volume: 10.00000    ------>   3.  Price: 700.00, Volume: 3.00000
4.  Price: 7000.00, Volume: 10.00000    ------>   4.  Price: 600.00, Volume: 4.00000
5.  Price: 6000.00, Volume: 10.00000    ------>   5.  Price: 500.00, Volume: 5.00000
6.  Price: 5000.00, Volume: 10.00000    ------>   6.  Price: 400.00, Volume: 6.00000
7.  Price: 4000.00, Volume: 10.00000    ------>   7.  Price: 300.00, Volume: 7.00000
8.  Price: 3000.00, Volume: 10.00000    ------>   8.  Price: 200.00, Volume: 8.00000
9.  Price: 2000.00, Volume: 10.00000    ------>   9.  Price: 100.00, Volume: 9.00000
10. Price: 1000.00, Volume: 10.00000    ------>   10. Price: 50.00, Volume: 10.00000

MAP = [127,127,127,127,127,127,127,127,127,127]

1. MarketOrder - Sell 100.00000 @ 5500.00
```

Figure 5.9 | [127,127,127,127,127,127,127,127,127,127] - Bid Side.

The final snapshot is filled with less favorable prices, leading us to infer that a large sell market order consumed the entire volume of the top ten levels. The volume of this market order is equal to the sum of the depths from all ten levels, while the resulting price is the weighted average of those levels' prices, with the respective volumes serving as weights (2.16).

- **MAP = [127,127,127,127,127,127,127,127,127,127] for asks:**

```
********
* ASKS *
********
1.  Price: 11000.00, Volume: 10.00000   ------>   1.  Price: 21000.00, Volume: 1.00000
2.  Price: 12000.00, Volume: 10.00000   ------>   2.  Price: 22000.00, Volume: 2.00000
3.  Price: 13000.00, Volume: 10.00000   ------>   3.  Price: 23000.00, Volume: 3.00000
4.  Price: 14000.00, Volume: 10.00000   ------>   4.  Price: 24000.00, Volume: 4.00000
5.  Price: 15000.00, Volume: 10.00000   ------>   5.  Price: 25000.00, Volume: 5.00000
6.  Price: 16000.00, Volume: 10.00000   ------>   6.  Price: 26000.00, Volume: 6.00000
7.  Price: 17000.00, Volume: 10.00000   ------>   7.  Price: 27000.00, Volume: 7.00000
8.  Price: 18000.00, Volume: 10.00000   ------>   8.  Price: 28000.00, Volume: 8.00000
9.  Price: 19000.00, Volume: 10.00000   ------>   9.  Price: 29000.00, Volume: 9.00000
10. Price: 20000.00, Volume: 10.00000   ------>   10. Price: 30000.00, Volume: 10.00000

MAP = [127,127,127,127,127,127,127,127,127,127]

1. MarketOrder - Buy 100.00000 @ 15500.00
```

Figure 5.10 | [127,127,127,127,127,127,127,127,127,127] - Ask Side.

The same applies to the ask side, where a buy market order was submitted with a total volume equal to the sum of the depths across the top ten levels. The resulting price is the weighted average of these levels' prices, with the volumes serving as weights (2.17).

5.4 General Cases

Here, we address all cases where at least one price level has changed, but the scenario does not qualify as an extreme case. We will divide these into subcases based on the expected type of order inferred from the observed transitions.

MARKET ORDERS

There are two scenarios that lead to the inference of a market order:

1. Similar to the trivial case, there is a decrease in volume at the previously top price level, but in this case, the new snapshot places it at any level. This suggests that a market order occurred, followed by the submission of new limit orders with better prices, which shifted the top levels.
2. There are consecutive price levels missing from the top downwards. This indicates the presence of a market order with a volume larger than the depth at the top price level, continuing to consume volume from subsequent levels. In this case, the market order's total volume is calculated as the sum of the depths of all missing price levels, starting from the top and moving downward. Additionally, any volume difference at the first existing level is accounted for as the final piece of the consumed size.

- **An example of the first case of market order with MAP = [2,3,4,5,6,7,8,9,10,-128]:**

```
********
* BIDS *
********
1.  Price: 10000.00, Volume: 10.00000   ------>   1.  Price: 11000.00, Volume: 10.00000
2.  Price: 9000.00, Volume: 10.00000    ------>   2.  Price: 10000.00, Volume: 9.00000
3.  Price: 8000.00, Volume: 10.00000    ------>   3.  Price: 9000.00, Volume: 10.00000
4.  Price: 7000.00, Volume: 10.00000    ------>   4.  Price: 8000.00, Volume: 10.00000
5.  Price: 6000.00, Volume: 10.00000    ------>   5.  Price: 7000.00, Volume: 10.00000
6.  Price: 5000.00, Volume: 10.00000    ------>   6.  Price: 6000.00, Volume: 10.00000
7.  Price: 4000.00, Volume: 10.00000    ------>   7.  Price: 5000.00, Volume: 10.00000
8.  Price: 3000.00, Volume: 10.00000    ------>   8.  Price: 4000.00, Volume: 10.00000
9.  Price: 2000.00, Volume: 10.00000    ------>   9.  Price: 3000.00, Volume: 10.00000
10. Price: 1000.00, Volume: 10.00000    ------>   10. Price: 2000.00, Volume: 10.00000

MAP = [2,3,4,5,6,7,8,9,10,-128]

1. MarketOrder - Sell 1.00000 @ 10000.00
2. LimitOrder - Buy 10.00000 @ 11000.00
```

Figure 5.11 | [2,3,4,5,6,7,8,9,10,-128] - Bid Side.

As depicted in the figure above, the previously first price is now in the second position with a reduced volume. Here, we infer that a sell market order occurred, with a volume equal to the missing size. The mapping clearly reveals what occurred. Each previous price corresponding to the array indices has been shifted one position lower, resulting in a new best bid. Additionally, the previously lowest bid is now outside the scope of the top ten levels, indicated by the value -128.

- **An example of the second case of market order with MAP = [127,127,127,1,2,3,4,5,6,7]:**

```
********
* BIDS *
********
1.  Price: 10000.00, Volume: 10.00000   ------>   1.  Price: 7000.00, Volume: 5.00000
2.  Price: 9000.00, Volume: 10.00000    ------>   2.  Price: 6000.00, Volume: 10.00000
3.  Price: 8000.00, Volume: 10.00000    ------>   3.  Price: 5000.00, Volume: 10.00000
4.  Price: 7000.00, Volume: 10.00000    ------>   4.  Price: 4000.00, Volume: 10.00000
5.  Price: 6000.00, Volume: 10.00000    ------>   5.  Price: 3000.00, Volume: 10.00000
6.  Price: 5000.00, Volume: 10.00000    ------>   6.  Price: 2000.00, Volume: 10.00000
7.  Price: 4000.00, Volume: 10.00000    ------>   7.  Price: 1000.00, Volume: 10.00000
8.  Price: 3000.00, Volume: 10.00000    ------>   8.  Price: 800.00, Volume: 10.00000
9.  Price: 2000.00, Volume: 10.00000    ------>   9.  Price: 700.00, Volume: 10.00000
10. Price: 1000.00, Volume: 10.00000    ------>   10. Price: 600.00, Volume: 10.00000

MAP = [127,127,127,1,2,3,4,5,6,7]

1. MarketOrder - Sell 35.00000 @ 8714.29
```

Figure 5.12 | [127,127,127,1,2,3,4,5,6,7] - Bid Side.

Here, we observe an upward shift in prices, as the top three price levels are missing. The best bid is now the previously fourth price level, but with reduced volume. This is interpreted as a large

market order that consumed the entire volume of the first three levels and partially consumed the volume at the fourth level, which now sits at the top.

CANCELLATIONS

There are again two scenarios in the case of cancellations:

1. A decrease in the volume at a lower price level, which was not previously identified due to a large market order consuming volumes, will be interpreted as a cancellation of an order. The canceled volume is equal to the missing volume, regardless of where the price level now resides in the new snapshot.

2. A price level is entirely missing, but this was not identified earlier due to a large market order consuming volumes. In this case, we must carefully distinguish between levels missing due to cancellations and levels that disappeared as a result of downward price shifting caused by new limit orders. If the disappearance is due to price shifting, no cancellation should be reported, as we follow the greedy approach to infer the most straightforward sequence of market events.

- **An example of the first case of cancellation with MAP = [127,127,127,1,2,3,4,5,6,7]:**

```
********
* BIDS *
********
1.  Price: 10000.00, Volume: 10.00000   ------>   1.  Price: 7000.00, Volume: 5.00000
2.  Price: 9000.00, Volume: 10.00000    ------>   2.  Price: 6000.00, Volume: 10.00000
3.  Price: 8000.00, Volume: 10.00000    ------>   3.  Price: 5000.00, Volume: 10.00000
4.  Price: 7000.00, Volume: 10.00000    ------>   4.  Price: 4000.00, Volume: 10.00000
5.  Price: 6000.00, Volume: 10.00000    ------>   5.  Price: 3000.00, Volume: 10.00000
6.  Price: 5000.00, Volume: 10.00000    ------>   6.  Price: 2000.00, Volume: 10.00000
7.  Price: 4000.00, Volume: 10.00000    ------>   7.  Price: 1000.00, Volume: 9.00000
8.  Price: 3000.00, Volume: 10.00000    ------>   8.  Price: 800.00, Volume: 10.00000
9.  Price: 2000.00, Volume: 10.00000    ------>   9.  Price: 700.00, Volume: 10.00000
10. Price: 1000.00, Volume: 10.00000    ------>   10. Price: 600.00, Volume: 10.00000

MAP = [127,127,127,1,2,3,4,5,6,7]

1. MarketOrder - Sell 35.00000 @ 8714.29
2. CancelOrder - Buy 1.00000 @ 1000.00
```

Figure 5.13 | [127,127,127,1,2,3,4,5,6,7] - Bid Side.

It is shown that the tenth price still exists in the ending price levels but with a decreased depth by one. We infer that this decrease corresponds to the cancellation of an order with a volume of one.

- **An example of the second case of cancellation with MAP = [127,127,127,1,2,3,4,5,6,-6]:**

```
********
* BIDS *
********
1.  Price: 10000.00, Volume: 10.00000   ------>   1.  Price: 7000.00, Volume: 5.00000
2.  Price: 9000.00, Volume: 10.00000    ------>   2.  Price: 6000.00, Volume: 10.00000
3.  Price: 8000.00, Volume: 10.00000    ------>   3.  Price: 5000.00, Volume: 10.00000
4.  Price: 7000.00, Volume: 10.00000    ------>   4.  Price: 4000.00, Volume: 10.00000
5.  Price: 6000.00, Volume: 10.00000    ------>   5.  Price: 3000.00, Volume: 10.00000
6.  Price: 5000.00, Volume: 10.00000    ------>   6.  Price: 2000.00, Volume: 10.00000
7.  Price: 4000.00, Volume: 10.00000    ------>   7.  Price: 900.00, Volume: 10.00000
8.  Price: 3000.00, Volume: 10.00000    ------>   8.  Price: 800.00, Volume: 10.00000
9.  Price: 2000.00, Volume: 10.00000    ------>   9.  Price: 700.00, Volume: 10.00000
10. Price: 1000.00, Volume: 10.00000    ------>   10. Price: 600.00, Volume: 10.00000

MAP = [127,127,127,1,2,3,4,5,6,-6]

1. MarketOrder - Sell 35.00000 @ 8714.29
2. CancelOrder - Buy 10.00000 @ 1000.00
```

Figure 5.14 | [127,127,127,1,2,3,4,5,6,-6] - Bid Side.

In this similar example, the price level of 1000 is entirely missing, while price levels with worse prices have shifted upward from below the tenth level to occupy its position. This indicates a cancellation of an order with a size equal to the total depth of the missing price level.

- **An example of omitting a cancellation due to downward shifting with MAP = [3,4,5,6,7,8,9,10,-128-128]:**

```
********
* BIDS *
********
1.  Price: 10000.00, Volume: 10.00000   ------>   1.  Price: 12000.00, Volume: 10.00000
2.  Price: 9000.00, Volume: 10.00000    ------>   2.  Price: 11000.00, Volume: 10.00000
3.  Price: 8000.00, Volume: 10.00000    ------>   3.  Price: 10000.00, Volume: 10.00000
4.  Price: 7000.00, Volume: 10.00000    ------>   4.  Price: 9000.00, Volume: 10.00000
5.  Price: 6000.00, Volume: 10.00000    ------>   5.  Price: 8000.00, Volume: 10.00000
6.  Price: 5000.00, Volume: 10.00000    ------>   6.  Price: 7000.00, Volume: 10.00000
7.  Price: 4000.00, Volume: 10.00000    ------>   7.  Price: 6000.00, Volume: 10.00000
8.  Price: 3000.00, Volume: 10.00000    ------>   8.  Price: 5000.00, Volume: 10.00000
9.  Price: 2000.00, Volume: 10.00000    ------>   9.  Price: 4000.00, Volume: 10.00000
10. Price: 1000.00, Volume: 10.00000    ------>   10. Price: 3000.00, Volume: 10.00000

MAP = [3,4,5,6,7,8,9,10,-128,-128]

1. LimitOrder - Buy 10.00000 @ 12000.00
2. LimitOrder - Buy 10.00000 @ 11000.00
```

Figure 5.15 | [127,127,127,1,2,3,4,5,6,-6] - Bid Side.

Here, we observe that the two bottom price levels are missing because new limit orders with better prices at the top have pushed the existing levels two positions down. As a result, we do not interpret the absence of the price levels at 2000 and 1000 as cancellations, since their disappearance is due to the natural downward shifting caused by the introduction of new limit orders.

LIMIT ORDERS

Again, we design our algorithm to handle two cases, which are essentially the opposite of cancellations.

1. An increase in volume at any price level is interpreted as a new limit order, with the order's volume equal to the difference in depth (the additional volume).

2. Any new price level that did not exist in the previous snapshot but appears in the current snapshot is a new limit order. However, care must be taken to distinguish these from price levels that have been upwardly shifted from deeper levels due to cancellations at upper levels or large market orders.

- **An example of the first case of limit order with MAP = [127,127,127,1,2,3,4,5,6,7,8]:**

```
********
* ASKS *
********
1.  Price: 11000.00, Volume: 10.00000   ------>   1.  Price: 13000.00, Volume: 12.00000
2.  Price: 12000.00, Volume: 10.00000   ------>   2.  Price: 14000.00, Volume: 10.00000
3.  Price: 13000.00, Volume: 10.00000   ------>   3.  Price: 15000.00, Volume: 10.00000
4.  Price: 14000.00, Volume: 10.00000   ------>   4.  Price: 16000.00, Volume: 10.00000
5.  Price: 15000.00, Volume: 10.00000   ------>   5.  Price: 17000.00, Volume: 10.00000
6.  Price: 16000.00, Volume: 10.00000   ------>   6.  Price: 18000.00, Volume: 10.00000
7.  Price: 17000.00, Volume: 10.00000   ------>   7.  Price: 19000.00, Volume: 10.00000
8.  Price: 18000.00, Volume: 10.00000   ------>   8.  Price: 20000.00, Volume: 10.00000
9.  Price: 19000.00, Volume: 10.00000   ------>   9.  Price: 21000.00, Volume: 10.00000
10. Price: 20000.00, Volume: 10.00000   ------>   10. Price: 22000.00, Volume: 10.00000

MAP = [127,127,1,2,3,4,5,6,7,8]

1. MarketOrder - Buy 20.00000 @ 11500.00
2. LimitOrder - Sell 2.00000 @ 13000.00
```

Figure 5.16 | [127,127,127,1,2,3,4,5,6,7,8] - Ask Side.

The previously third price level is now at the top, with an increase in depth by 2. This indicates that a new limit order was placed at the price level of 13000 with a size of 2.

- **An example of the second case of limit order with MAP = [1,2,3,5,6,7,8,9,10,-128]:**

```
********
* ASKS *
********
1.  Price: 11000.00, Volume: 10.00000   ------>   1.  Price: 11000.00, Volume: 10.00000
2.  Price: 12000.00, Volume: 10.00000   ------>   2.  Price: 12000.00, Volume: 10.00000
3.  Price: 13000.00, Volume: 10.00000   ------>   3.  Price: 13000.00, Volume: 10.00000
4.  Price: 14000.00, Volume: 10.00000   ------>   4.  Price: 13500.00, Volume: 5.00000
5.  Price: 15000.00, Volume: 10.00000   ------>   5.  Price: 14000.00, Volume: 10.00000
6.  Price: 16000.00, Volume: 10.00000   ------>   6.  Price: 15000.00, Volume: 10.00000
7.  Price: 17000.00, Volume: 10.00000   ------>   7.  Price: 16000.00, Volume: 10.00000
8.  Price: 18000.00, Volume: 10.00000   ------>   8.  Price: 17000.00, Volume: 10.00000
9.  Price: 19000.00, Volume: 10.00000   ------>   9.  Price: 18000.00, Volume: 10.00000
10. Price: 20000.00, Volume: 10.00000   ------>   10. Price: 19000.00, Volume: 10.00000

MAP = [1,2,3,5,6,7,8,9,10,-128]

1. LimitOrder - Sell 5.00000 @ 13500.00
```

Figure 5.17 | [1,2,3,5,6,7,8,9,10,-128] - Ask Side.

As we observe the testing transition, the absence of the value 4 in the mapping array, along with the value 5 appearing at the fourth index, indicates that the previously fourth best ask has been shifted one level down. The same is true for the subsequent price levels, with the final value of -128 signaling that the previously last ask price has been pushed out of the visible top 10 levels. This suggests the presence of a new sell limit order at the price corresponding to the fourth level in the ending snapshot, with a volume equal to the total depth at that level.

- **An example of omitting a limit order due to upward shifting with MAP = [1,2,3,4,-4,-4,5,6,7,8]:**

```
********
* ASKS *
********
1.  Price: 11000.00, Volume: 10.00000   ------>   1.  Price: 11000.00, Volume: 10.00000
2.  Price: 12000.00, Volume: 10.00000   ------>   2.  Price: 12000.00, Volume: 10.00000
3.  Price: 13000.00, Volume: 10.00000   ------>   3.  Price: 13000.00, Volume: 10.00000
4.  Price: 14000.00, Volume: 10.00000   ------>   4.  Price: 14000.00, Volume: 10.00000
5.  Price: 15000.00, Volume: 10.00000   ------>   5.  Price: 17000.00, Volume: 10.00000
6.  Price: 16000.00, Volume: 10.00000   ------>   6.  Price: 18000.00, Volume: 10.00000
7.  Price: 17000.00, Volume: 10.00000   ------>   7.  Price: 19000.00, Volume: 10.00000
8.  Price: 18000.00, Volume: 10.00000   ------>   8.  Price: 20000.00, Volume: 10.00000
9.  Price: 19000.00, Volume: 10.00000   ------>   9.  Price: 21000.00, Volume: 10.00000
10. Price: 20000.00, Volume: 10.00000   ------>   10. Price: 22000.00, Volume: 10.00000

MAP = [1,2,3,4,-4,-4,5,6,7,8]

1. CancelOrder - Sell 10.00000 @ 15000.00
2. CancelOrder - Sell 10.00000 @ 16000.00
```

Figure 5.18 | [1,2,3,4,-4,-4,5,6,7,8] - Ask Side.

Finally, even though the price levels for 21000 and 22000 appear new in the ending snapshot, we do not infer the presence of any new limit orders. This is because their appearance results from upward shifting caused by two cancellations at higher levels.

5.5 Storing Transitions and Events

We need to finalize the database schema introduced in the previous chapter to store the transitions and the events produced by the algorithm. The second table, following the snapshots table, will hold the transitions. It will include the transition serial number, which indicates the sequential order of transitions, as well as references to the starting and ending snapshots. Additionally, it will store the time interval between the two snapshots, the total number of extracted events, the mapping arrays that represent the relationship between price levels of the snapshots, and a human-readable text representation similar to the descriptive format used in the figures in previous sections.

```
create table public.transitions
(
    serial_no     integer   not null
        constraint transition_pkey
            primary key,
    from_snapshot bigint    not null,
    to_snapshot   bigint    not null,
    time_diff     integer   not null,
    events_count  integer   not null,
    bid_map       integer[] not null,
    ask_map       integer[] not null,
    display_text  text      not null
);
```

Next, we need a table to store the events produced by the algorithm. This table will include a reference to the transition it belongs to, along with integer code values to represent the type of the event (limit, cancel, or market), the side of the event (buy or sell), and the effect side (bid or ask). Additionally, the table will store the price and volume of the event, as well as a human-readable text representation for clarity and ease of interpretation.

```
create table public.events
(
    for_transition integer     not null
        constraint events_transitions_fk
            references public.transitions,
    serial_no      integer     not null,
    event_type     integer     not null, --enum
    event_side     integer     not null, --enum
    effect_side    integer     not null, --enum
    price          numeric     not null,
    volume         numeric     not null,
    display_text   varchar(50) not null,
    primary key (for_transition, serial_no)
);
```

We will also create an enumeration table to store the code values and their corresponding descriptions for event types, event sides, and effect sides.

```
create table public.enums
(
    name        varchar(50) not null,
    value       integer     not null,
    description varchar(50) not null
);

insert into public.enums (name, value, description) values ('event_type', 10, 'Limit Order');
insert into public.enums (name, value, description) values ('event_type', 20, 'Market Order');
insert into public.enums (name, value, description) values ('event_type', 30, 'Cancel Order');

insert into public.enums (name, value, description) values ('event_side', 10, 'Buy');
insert into public.enums (name, value, description) values ('event_side', 20, 'Sell');

insert into public.enums (name, value, description) values ('effect_side', 10, 'Bid');
insert into public.enums (name, value, description) values ('effect_side', 20, 'Ask');
```

Lastly, we will organize all this information into a database view that consolidates the most important details from our results. This view will serve as a structured summary, making it easy to query and analyze. Additionally, it provides a convenient way to export the data into formats such as CSV files, if needed, for further processing or sharing.

```
create view events_export as
select
    transitions.from_snapshot,
    transitions.to_snapshot,
    et.description AS event_type,
    es.description AS event_side,
    ef.description AS effect_side,
    events.price, events.volume, events.display_text
from
    events
join
    transitions on events.for_transition = transitions.serial_no
join
    enums et ON events.event_type = et.value AND et.name = 'event_type'
join
    enums es ON events.event_side = es.value AND es.name = 'event_side'
join
    enums ef ON events.effect_side = ef.value AND ef.name = 'effect_side'
order by
    events.for_transition, events.serial_no;
```

Let us now examine the results of the test transition (Figure 5.20) presented below as they are stored in our database.

from_snapshot	to_snapshot	event_type	event_side	effect_side	price	volume	display_text
0	100	Cancel Order	Sell	Ask	15000	10	CancelOrder - Sell 10.00000 @ 15000.00
0	100	Cancel Order	Sell	Ask	16000	10	CancelOrder - Sell 10.00000 @ 16000.00
0	100	Limit Order	Buy	Bid	12000	10	LimitOrder - Buy 10.00000 @ 12000.00
0	100	Limit Order	Buy	Bid	11000	10	LimitOrder - Buy 10.00000 @ 11000.00

Figure 5.19 | Database View Output.

```
********
* BIDS *
********
1.  Price: 10000.00, Volume: 10.00000   ------>   1.  Price: 12000.00, Volume: 10.00000
2.  Price: 9000.00, Volume: 10.00000    ------>   2.  Price: 11000.00, Volume: 10.00000
3.  Price: 8000.00, Volume: 10.00000    ------>   3.  Price: 10000.00, Volume: 10.00000
4.  Price: 7000.00, Volume: 10.00000    ------>   4.  Price: 9000.00, Volume: 10.00000
5.  Price: 6000.00, Volume: 10.00000    ------>   5.  Price: 8000.00, Volume: 10.00000
6.  Price: 5000.00, Volume: 10.00000    ------>   6.  Price: 7000.00, Volume: 10.00000
7.  Price: 4000.00, Volume: 10.00000    ------>   7.  Price: 6000.00, Volume: 10.00000
8.  Price: 3000.00, Volume: 10.00000    ------>   8.  Price: 5000.00, Volume: 10.00000
9.  Price: 2000.00, Volume: 10.00000    ------>   9.  Price: 4000.00, Volume: 10.00000
10. Price: 1000.00, Volume: 10.00000    ------>   10. Price: 3000.00, Volume: 10.00000

MAP = [3,4,5,6,7,8,9,10,-128,-128]

1. LimitOrder - Buy 10.00000 @ 12000.00
2. LimitOrder - Buy 10.00000 @ 11000.00
********
* ASKS *
********
1.  Price: 11000.00, Volume: 10.00000   ------>   1.  Price: 11000.00, Volume: 10.00000
2.  Price: 12000.00, Volume: 10.00000   ------>   2.  Price: 12000.00, Volume: 10.00000
3.  Price: 13000.00, Volume: 10.00000   ------>   3.  Price: 13000.00, Volume: 10.00000
4.  Price: 14000.00, Volume: 10.00000   ------>   4.  Price: 14000.00, Volume: 10.00000
5.  Price: 15000.00, Volume: 10.00000   ------>   5.  Price: 17000.00, Volume: 10.00000
6.  Price: 16000.00, Volume: 10.00000   ------>   6.  Price: 18000.00, Volume: 10.00000
7.  Price: 17000.00, Volume: 10.00000   ------>   7.  Price: 19000.00, Volume: 10.00000
8.  Price: 18000.00, Volume: 10.00000   ------>   8.  Price: 20000.00, Volume: 10.00000
9.  Price: 19000.00, Volume: 10.00000   ------>   9.  Price: 21000.00, Volume: 10.00000
10. Price: 20000.00, Volume: 10.00000   ------>   10. Price: 22000.00, Volume: 10.00000

MAP = [1,2,3,4,-4,-4,5,6,7,8]

1. CancelOrder - Sell 10.00000 @ 15000.00
2. CancelOrder - Sell 10.00000 @ 16000.00
```

Figure 5.20 | A transition sample.

Chapter 6 Experimental Results

6.1 Feature Construction

This section represents the culmination of our efforts and the realization of the final objective which is the generation of handcrafted features akin to those proposed in [8]. However, our approach differs significantly, as we rely exclusively on raw snapshots of the limit order book and reverse-engineer the market events that occurred to construct the feature vector set presented in their study. The resulting feature set, illustrated in Box 3.2, demonstrates the feasibility of reconstructing high-quality, informative features without direct access to detailed event-level data.

We conducted experiments using raw limit order book data from a 10-hour streaming session, which exhibits the statistical characteristics outlined below. In this context, static transitions are defined as those in which there are no changes in price levels, although volumes may vary.

Table 6.1 | Streaming Result Statistics - Transitions

	Total	**Total %**
10-hour session started at 2024-12-17 16:31:09.714		
Transitions	359441	100
Static Transitions	245241	68.23
Transitions - With Price Movement	114200	31.77
Transitions - With No Change	41999	11.68

Table 6.2 | Streaming Result Statistics - Inferred Market Events

	Count
10-hour session started at 2024-12-17 16:31:09.714	
Market Events	1201035
Average Events Per Transition	3.34
Average Bid Events Per Transition	1.64
Average Ask Events Per Transition	1.70

We will now provide a detailed explanation of the feature vector set presented in Box 3.2, along with the assumptions made to adapt it to our specific data context.

BASIC SET

This feature vector set is the most straightforward, consisting of the raw bid and ask prices and volumes for the top 10 levels of the limit order book.

bid1_p	bid2_p	bid3_p	bid4_p	bid5_p	bid6_p	bid7_p	bid8_p	bid9_p	bid10_p	...	ask1_v	ask2_v	ask3_v	ask4_v	ask5_v	ask6_v	ask7_v	ask8_v	ask9_v	ask10_v
106481.19	106481.18	106480.75	106480.74	106480.73	106480.72	106480.71	106480.53	106480.52	106480.15	...	2.66967	0.00010	0.00015	0.00010	0.00010	0.09739	0.00010	0.00010	0.00010	0.00010
106481.19	106481.18	106480.75	106480.74	106480.73	106480.72	106480.71	106480.53	106480.52	106480.15	...	2.66939	0.00010	0.00015	0.00010	0.00010	0.09739	0.00010	0.00010	0.00010	0.00010
106481.19	106481.18	106480.75	106480.74	106480.73	106480.72	106480.71	106480.53	106480.52	106480.15	...	2.47973	0.00010	0.00015	0.00010	0.00010	0.09739	0.00010	0.00010	0.00010	0.00010
106481.19	106481.18	106480.75	106480.74	106480.73	106480.72	106480.71	106480.53	106480.52	106480.15	...	2.94882	0.00010	0.00015	0.00010	0.00010	0.09739	0.00010	0.00010	0.00010	0.00010
106481.19	106481.18	106480.75	106480.74	106480.73	106480.72	106480.71	106480.53	106480.52	106480.15	...	3.32346	0.00010	0.00015	0.00010	0.00010	0.09739	0.00010	0.09401	0.00010	0.00010
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
107538.32	107538.06	107538.05	107538.03	107538.02	107537.80	107537.79	107537.78	107537.56	107537.54	...	3.17615	0.00010	0.03730	0.00015	0.00038	0.00010	0.00017	0.00015	0.00015	0.00010
107538.32	107538.06	107538.05	107538.03	107538.02	107537.80	107537.79	107537.78	107537.56	107537.54	...	2.93872	0.00010	0.00010	0.00015	0.00038	0.00010	0.00017	0.00015	0.00015	0.00010
107538.32	107538.06	107538.05	107538.03	107538.02	107537.80	107537.79	107537.78	107537.56	107537.54	...	1.16166	0.00015	0.00015	0.00038	0.00010	0.00017	0.00015	0.00015	0.00010	0.35527
107538.32	107538.06	107538.05	107538.03	107538.02	107537.80	107537.79	107537.78	107537.56	107537.54	...	1.43729	0.00015	0.00015	0.00038	0.00010	0.00017	0.00015	0.00015	0.00010	0.35527
107538.32	107538.06	107538.05	107538.03	107538.02	107537.80	107537.79	107537.78	107537.56	107537.54	...	1.25089	0.00015	0.00015	0.00038	0.00010	0.00017	0.00015	0.00015	0.00010	0.35527

Figure 6.1 | Raw limit order book prices and volumes

TIME-INSENSITIVE SET

This group of features captures the relationships between prices and volumes within the current time snapshot. It comprises four distinct vector sets. The first focuses on the bid-ask spread and mid-price, providing a measure of the immediate market state. The second examines the differences between consecutive price levels on each side of the order book. The third set calculates the averages of prices and volumes. Lastly, the fourth set considers the accumulated cross-side differences in prices and volumes, highlighting disparities between the bid and ask sides. These features are independent of time dynamics and instead provide a snapshot-based perspective of the order book's structure.

spread_1	mid_price_1	spread_2	mid_price_2	spread_3	mid_price_3	spread_4	mid_price_4	spread_5	mid_price_5	spread_6	mid_price_6	spread_7	mid_price_7	spread_8	mid_price_8	spread_9	mid_price_9	spread_10	mid_price_10
0.01	106481.195	0.03	106481.195	0.47	106480.985	0.52	106481.000	0.58	106481.020	0.60	106481.02	0.62	106481.020	0.81	106480.935	0.94	106480.99	1.40	106480.850
0.01	106481.195	0.03	106481.195	0.47	106480.985	0.52	106481.000	0.58	106481.020	0.60	106481.02	0.62	106481.020	0.81	106480.935	0.94	106480.99	1.40	106480.850
0.01	106481.195	0.03	106481.195	0.47	106480.985	0.52	106481.000	0.58	106481.020	0.60	106481.02	0.62	106481.020	0.81	106480.935	0.94	106480.99	1.40	106480.850
0.01	106481.195	0.03	106481.195	0.47	106480.985	0.52	106481.000	0.58	106481.020	0.60	106481.02	0.62	106481.020	0.81	106480.935	0.94	106480.99	1.40	106480.850
0.01	106481.195	0.03	106481.195	0.47	106480.985	0.52	106481.000	0.58	106481.020	0.60	106481.02	0.62	106481.020	0.81	106480.935	0.94	106480.99	1.40	106480.850
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0.01	107538.325	0.33	107538.225	0.35	107538.225	0.59	107538.325	0.77	107538.405	1.40	107538.50	1.43	107538.505	1.81	107538.685	2.14	107538.63	2.22	107538.650
0.01	107538.325	0.33	107538.225	0.35	107538.225	0.59	107538.325	0.77	107538.405	1.40	107538.50	1.43	107538.505	1.81	107538.685	2.14	107538.63	2.22	107538.650
0.01	107538.325	0.46	107538.290	0.57	107538.335	0.76	107538.410	1.18	107538.610	1.42	107538.51	1.80	107538.690	1.92	107538.740	2.20	107538.66	2.45	107538.765
0.01	107538.325	0.46	107538.290	0.57	107538.335	0.76	107538.410	1.18	107538.610	1.42	107538.51	1.80	107538.690	1.92	107538.740	2.20	107538.66	2.45	107538.765
0.01	107538.325	0.46	107538.290	0.57	107538.335	0.76	107538.410	1.18	107538.610	1.42	107538.51	1.80	107538.690	1.92	107538.740	2.20	107538.66	2.45	107538.765

Figure 6.2 | Bid-Ask Spreads and Mid Prices

bid_pdif_1	ask_pdif_1	bid_pdif_2	ask_pdif_2	bid_pdif_3	ask_pdif_3	bid_pdif_4	ask_pdif_4	bid_pdif_5	ask_pdif_5	bid_pdif_6	ask_pdif_6	bid_pdif_7	ask_pdif_7	bid_pdif_8	ask_pdif_8	bid_pdif_9	ask_pdif_9	bid_pdif_10	ask_pdif_10
0.01	0.01	0.43	0.01	0.01	0.04	0.01	0.05	0.01	0.01	0.01	0.01	0.18	0.01	0.01	0.12	0.37	0.09	1.04	0.35
0.01	0.01	0.43	0.01	0.01	0.04	0.01	0.05	0.01	0.01	0.01	0.01	0.18	0.01	0.01	0.12	0.37	0.09	1.04	0.35
0.01	0.01	0.43	0.01	0.01	0.04	0.01	0.05	0.01	0.01	0.01	0.01	0.18	0.01	0.01	0.12	0.37	0.09	1.04	0.35
0.01	0.01	0.43	0.01	0.01	0.04	0.01	0.05	0.01	0.01	0.01	0.01	0.18	0.01	0.01	0.12	0.37	0.09	1.04	0.35
0.01	0.01	0.43	0.01	0.01	0.04	0.01	0.05	0.01	0.01	0.01	0.01	0.18	0.01	0.01	0.12	0.37	0.09	1.04	0.35
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0.26	0.06	0.01	0.01	0.02	0.22	0.01	0.17	0.22	0.41	0.01	0.02	0.01	0.37	0.22	0.11	0.02	0.06	0.78	1.43
0.26	0.06	0.01	0.01	0.02	0.22	0.01	0.17	0.22	0.41	0.01	0.02	0.01	0.37	0.22	0.11	0.02	0.06	0.78	1.43
0.26	0.19	0.01	0.10	0.02	0.17	0.01	0.41	0.22	0.02	0.01	0.37	0.01	0.11	0.22	0.06	0.02	0.23	0.78	1.66
0.26	0.19	0.01	0.10	0.02	0.17	0.01	0.41	0.22	0.02	0.01	0.37	0.01	0.11	0.22	0.06	0.02	0.23	0.78	1.66
0.26	0.19	0.01	0.10	0.02	0.17	0.01	0.41	0.22	0.02	0.01	0.37	0.01	0.11	0.22	0.06	0.02	0.23	0.78	1.66

Figure 6.3 | Price Differences

bid_p_mean	bid_v_mean	ask_p_mean	ask_v_mean	pdiff_acc	vdiff_acc
106480.722	0.182532	106481.320	0.276791	5.98	0.94259
106480.722	0.182532	106481.320	0.276763	5.98	0.94231
106480.722	0.187872	106481.320	0.257797	5.98	0.69925
106480.722	0.187872	106481.320	0.304706	5.98	1.16834
106480.722	0.187773	106481.320	0.351561	5.98	1.63788
...	...	...	...	...	...
107537.895	0.243740	107539.000	0.321475	11.05	0.77735
107537.895	0.261973	107539.000	0.294012	11.05	0.32039
107537.895	0.339697	107539.172	0.151828	12.77	-1.87869
107537.895	0.340945	107539.172	0.179391	12.77	-1.61554
107537.895	0.340853	107539.172	0.160751	12.77	-1.80102

Figure 6.4 | Mean Prices and Volume - Accumulated Differences

TIME-SENSITIVE SET

Here, we introduce the inferred data integration with raw snapshot data to compute time-dependent parameters designed to capture temporal dependencies. This group consists of four feature vector sets, each targeting a specific aspect of market dynamics over time.

The first feature set computes the average time derivatives of prices and volumes over the past 1 second, capturing short-term changes. The second calculates the average intensity of each event type, representing the recent arrival rate within the same 1-second window. The third set uses an indicator function to signal whether a specific type of limit or market order has intensified in the recent past compared to the longer term. Specifically, the recent short-term period is defined as the last 10 seconds, while the long-term period spans 900 seconds. This feature is set to 1 if the short-term average intensity exceeds the long-term intensity. Finally, the fourth feature set measures the acceleration of limit or market orders, computed as the derivative of event intensity over the past 1 second.

These features rely on an important assumption due to the lack of detailed event timestamps within transitions. Specifically, we assume that each snapshot has a fixed temporal distance of exactly 100 milliseconds. This assumption is justified by the smoothness of the interval distribution and allows us to approximate time fractions by counting snapshots. For instance, to calculate metrics for the past 1 second, we simply use the last 10 snapshots. This assumption enables consistent and practical computation of temporal features without requiring precise event timing.

bid1_p_deriv_avg_1s	bid1_v_deriv_avg_1s	ask1_p_deriv_avg_1s	ask1_v_deriv_avg_1s	bid2_p_deriv_avg_1s	bid2_v_deriv_avg_1s	ask2_p_deriv_avg_1s	ask2_v_deriv_avg_1s	bid3_p_deriv_avg_1s	bid3_v_deriv_avg_1s	...	ask8_p_deriv_avg_1s
12.55	-0.77074	12.55	0.69485	12.27	-0.19559	14.37	0.42398	12.27	4.202000e-02	...	14.50
12.55	-0.69229	12.55	0.64703	12.27	-0.19559	13.39	0.06733	12.27	4.202000e-02	...	13.35
12.55	-0.78499	12.55	1.53031	12.27	-0.19559	13.39	0.06733	12.27	4.202000e-02	...	13.35
12.55	-0.48787	12.55	2.22138	12.27	-0.19559	13.39	0.06733	12.27	4.202000e-02	...	13.34
12.55	-2.38925	12.55	3.09065	12.27	-0.19564	13.39	0.06733	12.27	-8.000000e-05	...	13.34
...	...	...	...	...	...	...	...	...	...	...	...
0.00	1.21791	0.00	0.41096	0.03	0.00008	0.00	0.00000	0.03	-6.172840e-15	...	0.00
0.00	1.21691	0.00	-0.27817	0.03	0.00008	0.00	0.00000	0.03	-6.172840e-15	...	0.00
0.00	1.99420	0.00	-2.14822	0.03	0.00008	0.13	0.00005	0.03	-6.172840e-15	...	0.11
0.00	2.00755	0.00	-1.63571	0.03	0.00008	0.13	0.00005	0.03	-6.172840e-15	...	0.11
0.00	2.01255	0.00	-1.92811	0.03	0.00008	0.13	0.00005	0.03	-6.172840e-15	...	0.11

Figure 6.5 | Price and Volume Derivatives

lambda_lb	lambda_la	lambda_mb	lambda_ma	lambda_cb	lambda_ca	i_lb	i_la	i_mb	i_ma	lambda_lb_deriv	lambda_la_deriv	lambda_mb_deriv	lambda_ma_deriv
25	19	1	3	4	7	1	1	1	1	-4.7	-2.4	-1.6	1.6
25	20	2	3	4	6	1	1	1	1	0.1	-1.8	-1.5	1.0
15	21	3	2	5	6	1	1	1	1	3.3	-0.4	-1.3	0.2
8	14	4	3	5	5	1	1	1	1	3.6	1.2	-0.8	-0.8
7	13	4	3	5	5	1	1	1	1	2.7	1.5	-0.2	-1.6
...	...	...	...	...	...	...	...	...	...	...	...	...	...
10	17	2	3	5	2	0	1	1	0	-4.0	4.4	-1.8	-1.5
10	16	2	3	5	2	0	1	1	0	-2.9	1.8	-2.1	-0.8
9	26	3	2	5	1	0	1	1	0	-2.0	-0.3	-2.3	0.0
8	27	3	1	4	7	0	1	1	0	-1.1	-2.1	-2.5	0.5
6	28	3	0	5	8	1	1	1	0	-0.6	-3.5	-2.6	0.6

Figure 6.6 | **Average Intensity - Intensity Indicators - Accelerations**

6.2 Class Labeling Methodology

It somewhat escapes the scope of this thesis to delve deeply into labelling strategies, as they often depend on the specific goals of what one opts to predict. However, for the sake of completeness and to showcase how a fully constructed dataset could look, we will briefly discuss a common approach for mid-price prediction labelling. For this example, we refer to the methodology employed in [30] for creating the F1-2010 dataset.

In this approach, the number of labels corresponds to the future horizons for which we aim to predict the mid-price. These horizons can be defined in terms of time intervals, ticks, or the number of events in the future. In our case, we define the horizons in terms of how many snapshots ahead we want to predict. For every row in our feature dataset, we compute the mid-price at each specified horizon and calculate its percentage change relative to the current mid-price.

Next, we apply a threshold to this percentage change to classify movements as upward, downward, or stationary. As suggested in [8], a good dataset should aim for an approximate class distribution of 1:1:2 for upward, downward, and stationary signals. For our experiments, we set a threshold of $\pm 0.01\%$ to distinguish the classes. Specifically, a change equal to or above 0.01% is classified as an upward movement (label 1), a change between -0.01% and 0.01% is considered stationary (label 2), and a change equal to or below -0.01% is classified as a downward movement (label 3).

In our experiments, we used horizons of 50 and 100 forward snapshots. The class distributions for these horizons are presented below. It is important to note, however, that the chosen thresholds and horizons may not always be optimal for real-world applications. For example, longer horizons could be challenging for models to predict accurately, and small percentage changes may not account for transaction costs and fees, potentially limiting the profitability of trading strategies. Therefore, careful consideration is needed when selecting parameters for labelling in practical scenarios.

Table 6.3 | Label Distribution

Horizon	Upward	Downward	Stationary
1st hour of 10-hour session started at 2024-12-17 16:31:09.714			
50	7766	6151	13083
100	8047	8297	10656

Chapter 7 Conclusion and Future Work

The central goal of this thesis has been to explore the potential of starting from raw limit order book (LOB) snapshots to infer meaningful market dynamics, a stepping stone toward constructing effective alpha predictors. This challenge is often viewed as a kind of "holy grail" in quantitative finance, as it seeks to directly extract predictive signals from raw data. By leveraging the structural characteristics of LOB data, we have attempted to infer market events and the temporal dynamics underlying price formation. This methodology enables the generation of features that capture not just the static state of the market but also its evolution over time.

The work presented demonstrates the feasibility of reverse-engineering market events and transitions from raw snapshots. This is achieved through the development of a robust greedy algorithm that resolves the sequential paths between snapshots, identifying the simplest yet most likely transitions. From these inferred transitions, we constructed feature sets designed to encapsulate temporal dynamics, offering a more comprehensive view of market activity. Such features, especially those derived from the inferred time-sensitive parameters, hold significant promise for predictive modeling. By including rich information such as price derivatives, order intensities, and event accelerations, we aimed to provide a foundation for models to better understand market dynamics and potentially improve forecasting capabilities.

As for future work, the natural progression of this research involves extensive experimentation in predictive modeling. This step will evaluate the constructed feature sets against various baselines, such as raw LOB features or other handcrafted features created without synthetic reconstruction. Furthermore, comparisons should be made with methodologies that utilize automated feature extraction techniques, such as those found in deep learning models like CNNs or hybrid CNN-LSTM architectures. The goal is to quantify the added value of inferred features and determine their practical utility in real-world applications.

Another area ripe for exploration is the extension of the dataset and model applicability. While this thesis focused on the BTCUSDT pair from Binance, applying the framework to other assets, exchanges, and asset classes could reveal the universality or specificity of the inferred features. Transfer learning techniques could also be employed to test the adaptability of models trained on one dataset to different markets.

In conclusion, this thesis represents a step forward in the effort to harness raw LOB data for predictive modeling, bridging the gap between granular market microstructure information and higher-level alpha predictors.

Bibliography

[1] Adam Smith. *The Wealth of Nations: An Inquiry into the Nature and Causes of the Wealth of Nations*. University of Chicago Press, 1776.

[2] William J. Bernstein. *A Splendid Exchange: How Trade Shaped The World*. Atlantic Books, 1st edition, 2008.

[3] Irene Aldridge. *High-frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems*. John Wiley & Sons, 2nd edition, 2013.

[4] Maureen O' Hara. *Market Microstructure Theory*. Wiley, 1st edition, 1998.

[5] Thierry Foucault, Marco Pagano, and Ailsa Roell. *Market Liquidity: Theory, Evidence, and Policy*. Oxford University Press, 1st edition, 2013.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 1st edition, 2016.

[7] Zihao Zhang, Bryan Lim, and S. Zohren. Deep learning for market by order data. *Applied Mathematical Finance*, 28:79–95, 2021.

[8] Alec N. Kercheval and Yuan Zhang. Modelling high-frequency limit order book dynamics with support vector machines. *Quantitative Finance*, 15(8):1315–1329, 2015.

[9] Lodewijk Petram. *The World's First Stock Exchange*. Columbia Business School Publishing, 1st edition, 2014.

[10] Abhinava Tripathi, Vipul, and Alok Dixit. Limit order books: a systematic review of literature. *Qualitative Research in Financial Markets*, 12(4):505–541, 2020.

[11] Martin D. Gould, Mason A. Porter, Stacy Williams, Mark McDonald, Daniel J. Fenn, and Sam D. Howison. Limit order books. *Quantitative Finance*, 13(11):1709–1742, 2013.

[12] Cosma Shalizi. Stochastic Processes (Advanced Probability II), 36-754, 2007. URL https://www.stat.cmu.edu/~cshalizi/754/notes/all.pdf. Course notes, Carnegie Mellon University.

[13] Paraskevi Nousi, Avraam Tsantekidis, Nikolaos Passalis, Adamantios Ntakaris, Juho Kanniainen, Anastasios Tefas, Moncef Gabbouj, and Alexandros Iosifidis. Machine learning for forecasting mid price movement using limit order book data. *IEEE Access*, 7:64722–64736, 2018.

[14] Luca Cattivelli and Davide Pirino. A sharp model of bid–ask spread forecasts. *International Journal of Forecasting*, 35(4):1211–1225, 2019.

[15] Perry J. Kaufman. *Basic Concepts and Calculations*, chapter 2, pages 25–77. John Wiley Sons, Ltd, 2012. ISBN 9781119202561.

[16] Christine A. Parlour. Price dynamics in limit order markets. *The Review of Financial Studies*, 11(4):789–816, 2015.

[17] Jonathan Field and Jeremy Large. Pro-rata matching and one-tick futures markets. CFS Working Paper Series 2008/40, Center for Financial Studies (CFS), 2008.

[18] Richard Haynes and Esen Onur. Precedence rules in matching algorithms. *Journal of Commodity Markets*, 19:100109, 2020.

[19] Robert Bloomfield, Maureen O'Hara, and Gideon Saar. Hidden liquidity: Some new light on dark trading. *The Journal of Finance*, 70(5):2227–2274, 11 2015.

[20] Stefan Frey and Patrik Sandås. The impact of hidden liquidity in limit order books. CFS Working Paper Series 2008/48, Center for Financial Studies (CFS), 2008.

[21] Deniz Ozenbas, Michael Pagano, Robert Schwartz, and Bruce Weber. *Liquidity, Markets and Trading in Action: An Interdisciplinary Perspective*. Springer, 01 2022.

[22] Florian Klöck, Alexander Schied, and Yuemeng Sun. Price manipulation in a market impact model with dark pool. *Applied Mathematical Finance*, 24:417 – 450, 2011.

[23] Hugh L. Christensen and Robert J Woodmansey. Prediction of hidden liquidity in thelimit order book of globex futures. *The Journal of Trading*, 8:68 – 95, 2013.

[24] Justin Sirignano. Deep learning for limit order books. *Quantitative Finance*, 19:1–22, 11 2018.

[25] Justin Sirignano and Rama Cont. Universal features of price formation in financial markets: perspectives from deep learning. *Quantitative Finance*, 19(9):1449–1459, 2019. doi: 10.1080/14697688.2019.1622295.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9: 1735–1780, 12 1997.

[27] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3): 273–297, September 1995.

[28] Tristan Fletcher and John Shawe-Taylor. Multiple kernel learning with fisher kernels for high frequency currency prediction. *Computational Econonomics*, 42(2):217–240, August 2013.

[29] David Powers. Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2:2229–3981, 01 2011.

[30] Adamantios Ntakaris, Martin Magris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. *Journal of Forecasting*, 37(8):852–866, 2018.

[31] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Using deep learning for price prediction by exploiting stationary limit order book features. *Applied Soft Computing*, 93:106401, 2020.

[32] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, 2019.

[33] Umberto Michelucci. *Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection*. Apress, 1st edition, 2019.

[34] Top crypto assets by market cap percentage, 2024. URL https://www.binance.com/en/altcoins/tradable. Accessed: 2024-12-13.

[35] Web socket protocol, 2024. URL https://www.rfc-editor.org/rfc/rfc6455.html. Accessed: 2024-12-13.

[36] Andrew Lombardi. *WebSocket: Lightweight Client-Server Communications*. O' Reilly Media, 1st edition, 2015.

[37] Charles Cao, Oliver Hansch, and Xiaoxin Wang. The information content of an open limit-order book. *Journal of Futures Markets*, 29(1):16–41, 2009.