UNIVERSITY OF GOTHENBURG

Deep Learning for High-Frequency Price Prediction in Cryptocurrency Markets

Master's thesis in Mathematical Statistics

Noah Trägårdh

DEPARTMENT OF MATHEMATICAL SCIENCES
UNIVERSITY OF GOTHENBURG
GOTHENBURG, SWEDEN 2025

Master's thesis 2025

Deep Learning for High-Frequency Price Prediction in Cryptocurrency Markets

Noah Trägårdh

UNIVERSITY OF GOTHENBURG

Department of Mathematical Sciences
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Deep Learning for High-Frequency Price Prediction in Cryptocurrency Markets
Noah Trägårdh

Supervisor: Carl Lindberg, Department of Mathematical Sciences
Examiner: Peter Helgesson, Department of Mathematical Sciences

Master's Thesis 2025
Department of Mathematical Sciences
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 10 00

Typeset in LaTeX
Gothenburg, Sweden 2025

Abstract

The rapid rise of high-frequency trading in cryptocurrency markets has intensified the need for robust predictive models capable of navigating extreme volatility, fragmented liquidity, and the unique microstructure of digital asset exchanges. This thesis investigates the effectiveness of deep learning architectures, specifically convolutional neural networks (CNNs), long short-term memory networks (LSTMs), and hybrid CNN-LSTM models, in forecasting short-horizon mid-price log-returns from limit order book data in cryptocurrency markets. Drawing on a large scale, tick-level dataset from ByBit across three major cryptocurrency pairs, the study evaluates the performance of these neural networks. Each model is evaluated on its ability to predict a vector of ten future returns, using both raw book features and engineered stationary signals. Results indicate that deep learning models consistently outperform linear benchmarks in terms of out-of-sample R^2 and directional accuracy, capturing complex non-linear dependencies present in the data. The analysis also highlights the impact of input feature transformations on predictive power and model robustness. These findings provide insights into the adaptability of deep neural architectures for high-frequency price prediction in continuously evolving, highly volatile financial environments, and underscore the potential for advanced machine learning techniques to enhance market understanding in the cryptocurrency domain.

Acknowledgements

I would like to thank my supervisor, Carl, for his valuable guidance and for giving me the freedom to shape the direction of my thesis work. I am also grateful to my examiner, Peter, for being helpful and assisting with the logistics throughout the process. Your support has been much appreciated.

Any mistakes and errors are unfortunately those of my own.

Noah Trägårdh, Gothenburg, 2025

Notation

Symbol	Description
$\mathbb{R}$	The set of real numbers
F	Input feature dimension
M	Output dimension (number of horizons)
N	Number of samples in dataset
$\mathbf{x}, \mathbf{x}_t$	Input feature vector at time t, $\mathbf{x} \in \mathbb{R}^F$
$\mathbf{X}$	Mini-batch or matrix of input vectors, $\mathbf{X} \in \mathbb{R}^{F \times m}$
$\mathcal{X}$	Tensor
f_θ	Neural network mapping with parameters θ
θ	Collection of trainable parameters
$W^{(\ell)}, \mathbf{W}^{(\ell)}$	Weight matrix of layer ℓ, $\mathbf{W}^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$
$b^{(\ell)}, \mathbf{b}^{(\ell)}$	Bias vector of layer ℓ, $\mathbf{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$
$h^{(\ell)}, \mathbf{h}^{(\ell)}$	Hidden activation at layer ℓ
L	Total number of layers
ϕ	Activation function
$\ell(\hat{y}, y)$	Loss function comparing prediction $\hat{y}$ and target y
$\mathcal{L}(\theta)$	Empirical risk: average training loss
g_t, m_t, v_t	Adam optimizer moment estimates
$\delta^{(\ell)}$	Backpropagated error signal at layer ℓ
$\mathbf{X}_{t-W:t-1}$	Look-back window of W consecutive inputs
p_t	Mid-price at time t
$r_{t,k}$	Log-return target at horizon k at time t
Δt	Average event time between price changes
h_k	Forecast horizon interval k
η_t	Learning rate at iteration t
β_1, β_2	Adam optimizer decay rates
λ	L_2 weight decay coefficient
$\delta\tau$	Latency buffer (ms)
$\mathcal{D}_{\text{train/valid/test}}$	Index sets for training, validation, and test samples
$N_{\text{train/valid/test}}$	Number of samples in each split
$R^2_{\text{OOS},k}$	Out-of-sample R^2 at horizon k

Contents

1 Introduction

A limit order book (LOB) is an electronic trading system used in modern financial markets to match buy and sell orders. Traders submit market orders, which are executed immediately at the best available price, or limit orders, which specify a maximum buy price or minimum sell price. The bid-ask spread, defined as the difference between the highest buy price and the lowest sell price, reflects real time supply and demand dynamics. The mid-price, calculated as the average of these two values, serves as a reference point for market participants. Unlike quote-driven systems where market makers set prices, limit order books rely on self organized price formation through the submission, amendment, and cancellation of orders. This dynamic process makes them valuable tools for analyzing liquidity, price discovery, and microstructural behavior, while also enabling participants to observe real time order queues [12, 27].

Advances in machine learning have enabled increasingly sophisticated analysis of limit order book data. For example, Kolm et al. showed that order flow imbalance, derived from LOB data, can predict short-term price movements across different time horizons, with models using stationary features often outperforming those relying solely on raw data [38, 27]. In parallel, end-to-end deep learning approaches have emerged: studies such as Zhang et al. [52] and Tsantekidis et al. [50] applied hybrid convolutional neural network-long short-term memory (CNN-LSTM) models to forecast mid-price returns, demonstrating these architectures strength in capturing both spatial patterns within the order book and temporal dependencies in order flow. While these techniques have been effective in equity markets, they may also be well suited to address challenges inherent to cryptocurrency markets, which operate continuously with high volatility and fragmented liquidity across exchanges [6, 41].

Cryptocurrency markets present unique challenges and opportunities for high-frequency forecasting. Unlike traditional financial markets with defined trading hours and overnight closures, cryptocurrency exchanges operate 24/7, demanding continuous adaptation to changing conditions. The microstructure of cryptocurrency trading features larger price fluctuations and persistent order flow activity, characteristics that differentiate these markets from equities [6, 41]. Short-term return modeling in this environment serves not only immediate trading objectives but also provides insight into fundamental price formation processes, market resilience after large shocks, and the relationship between order flow and price impact. Such insights can help optimize execution algorithms, inform risk management systems, and deepen understanding of market efficiency [30, 4].

The growing adoption of machine learning methods, particularly deep learning, offers new avenues for modeling complex dynamics in limit order books. Traditional econometric approaches often struggle to capture the intricate non-linearities and spatiotemporal dependencies present in high-frequency data. Deep learning models, with their capacity to automatically learn hierarchical representations from raw data, provide promising alternatives for extracting predictive signals from the wealth of information contained in order flow sequences [51]. By applying these techniques to cryptocurrency markets, which are characterized by their unique volatility, evolving microstructure, and absence of natural trading breaks, this thesis seeks to contribute to our understanding of how predictive models adapt to less structured, continuously evolving financial ecosystems. The work builds on established methodologies in equity markets while addressing the distinct challenges posed by crypto assets, ultimately advancing the intersection of market microstructure research and machine learning [38, 52, 50].

1.1 Research Objectives & Scope

This thesis investigates how well deep learning architectures can adapt to the unique challenges of high-frequency cryptocurrency limit order book data. While previous research has shown that deep networks can be effective for price prediction in traditional equity markets [38, 52, 50], their performance in cryptocurrency markets, which are defined by continuous trading, fragmented liquidity across exchanges, and frequent periods of heightened volatility, remains relatively unexplored.

The main objectives of this study are to collect tick-level limit order book data from major cryptocurrency pairs, construct relevant features using both raw book states and engineered stationary signals, and evaluate the predictive power of deep learning models over a range of short time horizons. The models are benchmarked against a standard linear baseline following the approach of Kolm et al. [38], with out-of-sample R^2 serving as the main metric for model comparison.

The research questions addressed in this work are as follows.

- How effective are deep learning models in predicting short-term mid-price returns in cryptocurrency markets?
- How do these models compare to a traditional linear baseline in this context?
- How does applying a stationary transformation to the limit order book data affect model performance?

There are several important limitations to consider regarding the scope of this work. The analysis is restricted to three liquid cryptocurrency pairs over a three month window, which may limit the generalizability of the results to other assets or market conditions. Feature construction is based only on the top 10 levels of the order book on each side, excluding deeper levels and other potentially informative signals.

All model evaluation is conducted on historical data in an offline setting, without any live trading or consideration of real world market impact, meaning that predictive performance observed here does not account for transaction costs or slippage. Additionally, while several deep learning architectures are tested, other modern approaches and more extensive hyperparameter tuning may yield different results. The study focuses exclusively on forecasting over short time frames, ranging from milliseconds to a couple of seconds, so the findings may not generalize to longer term prediction tasks.

1.2 Thesis Structure

The thesis structure is organised as follows. Chapter 2 introduces the cryptocurrency market landscape, reviews the microstructure of limit order books, and contrasts classical time-series models with modern machine-learning architectures for high-frequency price prediction. Chapter 3 details the data collection and preprocessing pipeline, the formulation of the multi-horizon forecasting task, and the specifications of each modelling approach. Going from ARX baselines to CNN, LSTM, and the hybrid CNN-LSTM network.

Chapter 4 delivers the empirical results, reporting how each forecasting approach performs across the selected assets and time horizons. Finally, chapter 5 draws together these findings, offering overarching conclusions, discussing their implications, and suggesting future exploration.

2
Theoretical Foundations

In this section, we establish the formal framework and core concepts that underline our study of high-frequency price dynamics in cryptocurrency markets. We begin by characterizing the distinctive microstructure of digital-asset trading venues, then introduce the mathematical representation of limit order books and the statistical foundations of short-horizon forecasting, before outlining the deep learning models that enable the extraction of complex non-linear patterns from market data.

2.1 Cryptocurrency Markets

Cryptocurrencies constitute a comparatively young asset class whose trading takes place on specialised venues with micro-structural properties that differ markedly from those of traditional exchanges [6]. While the underlying blockchain protocols guarantee asset issuance and security [45], our focus is on the *trading environment* in which high-frequency price formation occurs.

Beyond their novelty, cryptocurrency markets have grown at a high pace, evolving from niche platforms with limited liquidity to multi trillion dollar ecosystems engaging both retail and institutional participants worldwide. Unlike conventional financial assets, crypto instruments are issued and settled natively on distributed ledgers, enabling permissionless global access and real time settlement but also introducing new operational risks [6, 8]. The rapid pace of innovation, proliferation of new trading venues, and diverse regulatory regimes have led to an environment where market conventions, infrastructure standards, and trading behaviours are far from settled [6]. This persistent market activity creates both challenges and opportunities for liquidity providers, traders, and researchers alike.

Understanding the microstructure of these markets allows for interpreting price dynamics, evaluating execution strategies, and assessing systemic risks. The following sections outline aspects that set crypto trading apart from traditional financial markets, focusing on continuous operation, fragmented liquidity, heightened volatility, and the distinctive mix of market participants and venues.

2.1.1 Continuous & Fragmented Liquidity

Crypto markets operate continuously, 24 hours a day and 7 days a week, without scheduled halts. The absence of an overnight cooling-off window implies that both information arrival and order-flow shocks are temporally unbounded. Liquidity therefore follows a rolling pattern that peaks during the business hours of Asia, Europe, and North America [6]. For empirical work this continuity raises non-trivial issues, such as the arbitrary choice of a daily cut-off when computing returns, and it demands time-series models able to accommodate intraday seasonality without the benefit of market closures.

No single venue enjoys a monopoly over trading in any major cryptocurrency. Instead, different centralized exchanges such as Binance, Coinbase or ByBit maintain independent limit order books for largely the same assets. Because these order books operate in isolation, temporary price discrepancies can emerge across venues. During the 2017 boom, for example, Bitcoin traded at premiums of up to 40% on South Korean exchanges, a phenomenon widely referred to as the Kimchi premium [42]. Although arbitrageurs eventually compress these gaps, frictions such as blockchain settlement delays, withdrawal fees and regional capital controls prevent prices from converging instantaneously [6]. From a modeling perspective, a consolidated order book of an aggregation of depth across all venues, is therefore essential for accurately assessing liquidity and establishing reliable reference prices.

2.1.2 Volatility & Data Quality Risks

Major cryptocurrency assets routinely exhibit daily price ranges of several percent, an order of magnitude greater than those typically seen in large cap equities or major FX pairs [6, 20]. This elevated price activity is reflected in annualized volatility, denoted σ_{ann}, which is defined using the log-return series

$$r_t = \ln \frac{P_t}{P_{t-1}}, \qquad \bar{r} = \frac{1}{N}\sum_{t=1}^{N} r_t, \qquad \sigma_{\text{sample}} = \sqrt{\frac{1}{N-1}\sum_{t=1}^{N}(r_t - \bar{r})^2},$$

with annualized volatility computed as

$$\sigma_{\text{ann}} = \sigma_{\text{sample}}\sqrt{P},$$

where P is the number of return intervals per year.

For Bitcoin, the most traded of all cryptocurrencies, the annualized volatility appears to have stabilized at somewhere between 60-80%, which is high relative to traditional assets [33]. This is largely driven by speculative sentiment, shifting regulatory outlooks, and the absence of circuit breakers in the exchanges [49].

The relative youth of cryptocurrency exchanges adds additional sources of risk. Microstructural noise is heightened by immature infrastructure. Here heavy surges in order flow have repeatedly caused matching engine outages or degraded API performance, leaving gaps in both executable liquidity and historical data [8]. Cybersecurity incidents remain an ongoing concern, where it is estimated that 5-10% of the Bitcoin supply has been lost to exchange hacks since its start [28, 34].

Finally, data quality is sometimes compromised by unregulated venues inflating reported volumes via wash trading.[1] These uncertainties increase risk relative to mature equity or futures markets.

2.1.3 Market Participants

In traditional equity markets, trading volume is dominated by institutional investors such as mutual funds, pension funds, and proprietary trading firms, whereas retail activity is typically intermediated by brokers. By contrast, cryptocurrency markets have historically exhibited much higher direct retail participation as individuals can trade via exchanges or broker applications with comparatively few entry barriers [37]. Retail traders are prone to behavioural biases and momentum-chasing. Empirical evidence shows that retail investors who trade equities tend to act contrarian: buying into price dips and selling into rallies [9], while the same individuals, when trading cryptocurrencies, display momentum-oriented behaviour, purchasing during price ascents and selling during declines [6, 37]. This behaviour amplifies short-term trends and volatility.

Over the last years, institutional players including hedge funds, high-frequency trading firms, and corporate treasuries have entered the market, bringing larger capital bases and more sophisticated strategies [6]. Industry estimates indicate that, by Q1 2024, institutional accounts were responsible for 82 % of the total traded volume [17]. Their presence has deepened liquidity and partially mitigated retail-driven inefficiencies, yet the participant mix remains distinctive. High-frequency retail sentiment coexists with institutional flow.

2.1.4 The Different Trading Venues

Cryptocurrency trading occurs predominantly on two venue types: centralized exchanges (CEXs) and decentralized exchanges (DEXs), whose architectures imply distinct microstructural characteristics [47, 15].

CEX platforms such as Binance, ByBit, and Coinbase maintain custodial wallets and an internal limit order book with a millisecond matching engine. The off-chain ledger design enables high throughput and a rich order type set [3]. The trade-off is counterparty risk where historical failures e.g. Mt. Gox in 2014 and FTX in 2022 illustrate the need to trust the exchange as both custodian and operator.

DEXs execute trades via smart contracts on public blockchains. Most employ automated market-maker protocols popularised by Uniswap, where deterministic pricing curves replace the limit order book [1]. Although users retain self-custody and the venue is censorship-resistant, every trade waits for block confirmation, yielding latency several orders of magnitude higher than CEX engines. A block confirmation is when network nodes validate and include a transaction in a mined block, and on Ethereum this takes roughly 12-15 seconds per confirmation [24, 23]. The full

[1]Wash trading refers to the practice in which a trader, or colluding parties, simultaneously buys and sells the same asset to create artificial trading volume and misleading price activity without changing the trader's net position [32].

transparency of pending transactions further exposes traders to front-running and sandwich attacks [53].

2.2 Limit Order Books

At the heart of modern electronic markets lies the *limit order book*. This is a transparent, automated system for matching buy and sell interests in real time. By organizing all outstanding orders into price ranked queues, LOBs allow markets to function without centralized market makers, with prices instead emerging dynamically from the interaction of competing traders. As traders submit, cancel, or execute orders, the available supply and demand are reflected in market prices [27, 12].

2.2.1 Order Structure & Price Formation

Every order in an LOB carries four essential attributes, direction $\varepsilon_x \in \{-1, +1\}$ for sell/buy, limit price p_x, volume v_x, and submission time t_x. These define an order tuple

$$x = (\varepsilon_x, p_x, v_x, t_x), \tag{2.1}$$

which determines whether the order rests in the book as a *limit order* or executes immediately as a *market order*.

Limit orders represent patient trading intentions. A buy limit order, for instance, enters the LOB queue if its price p_x is below the current best ask $a(t)$, waiting to be matched with future sell orders. Conversely, market orders demand immediacy in that a buy market order executes at the best available ask price $a(t)$, consuming liquidity from the sell side. The distinction between these order types is crucial: limit orders provide liquidity, while market orders consume it.

The LOB's state at any moment t is defined by its *best bid* $b(t)$ and *best ask* $a(t)$, calculated as

$$b(t) = \max\{p_x \mid x \in \mathcal{B}(t)\}, \quad a(t) = \min\{p_x \mid x \in \mathcal{A}(t)\},$$

where $\mathcal{B}(t)$ and $\mathcal{A}(t)$ denote the sets of active buy and sell orders, respectively. The difference between these prices defines the *bid-ask spread*

$$s(t) = a(t) - b(t), \tag{2.2}$$

a measure of transaction costs and liquidity. The spread narrows when limit orders cluster near the best quotes and widens during volatile periods.

A commonly used, yet untradable benchmark for fair value, is the *mid-price*, which is defined as

$$m(t) = \frac{1}{2}(b(t) + a(t)). \tag{2.3}$$

To illustrate these concepts, figure 2.1 shows a snapshot of a limit order book.

Participants must conform to the *tick size* ϑ which sets the minimum price increment, requiring p_x to be multiples of ϑ, while the *lot size* v_0 defines the smallest

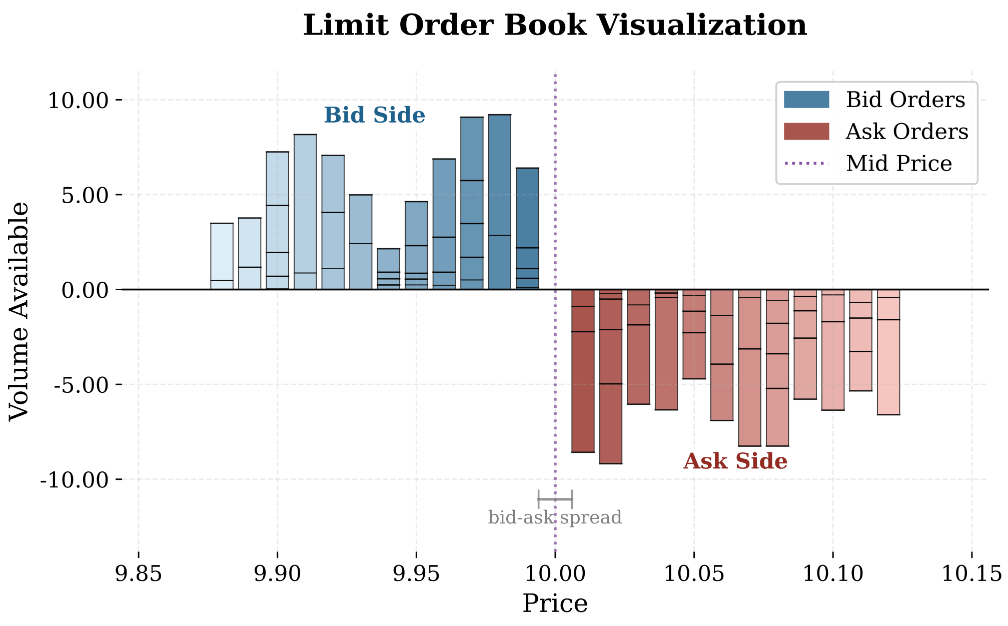

Figure 2.1: Snapshot of the limit order book at time t. Bars above (below) the horizontal axis represent the bid-side (ask-side) depth. The total height of each bar equals the volume accumulated at that price level, while the thin black segments within a bar indicate the individual order sizes that compose that volume. The vertical dashed line marks the mid-price $m(t)$, and the arrow highlights the bid-ask spread $s(t)$ between the best bid $b(t)$ and best ask $a(t)$.

tradable volume, with v_x as integer multiples of v_0. These parameters influence price granularity and liquidity, where smaller ticks allow precise pricing but may fragment liquidity via queue competition, while lot sizes restrict trade size flexibility, affecting order-splitting strategies.

2.2.2 Order Flow & Price Dynamics

When a new order (2.1) reaches the matching engine, the book is updated under a price-to-time priority rule. Orders are first ranked by price and, within each price level, executed in strict arrival order. Posting a limit order at a level that already contains resting volume therefore places the newcomer behind the existing queue. Therefore, a trader must quote a better price, one tick above the best bid for a buy or one tick below the best ask for a sell, to gain immediate queue priority.

Suppose a buy limit order arrives with $p_x > b(t)$. Because the price improves on the current best bid, the order is displayed at p_x and instantly becomes the new best bid, compressing the spread $s(t)$ (2.2). Had the same order been submitted at $p_x = b(t)$, it would instead be appended to the tail of the bid queue at that price and would execute only after all earlier bids at $b(t)$ had been satisfied or withdrawn.

Conversely, consider a sell market order of size v_x. Market orders demand immediacy, so the engine removes liquidity starting at $b(t)$ and, if necessary, continues down the book until the cumulative depth can absorb the entire volume contained in v_x.

Denote by $V_+(k,t)$ the quantity bid at level k. The post-trade best bid is

$$b(t^+) = \max_{p \leq b(t)} \Big\{ p \,\Big|\, \sum_{k=p}^{b(t)} V_+(k,t) \geq v_x \Big\},$$

and the resulting *slippage*, $b(t) - b(t^+)$, provides a quantification of how much worse the realized traded price is compared to the pre-trade expected price. That is, shallow books amplify slippage, whereas deep books dampen it.

Hence, limit orders placed inside the spread improve quoted prices but face execution risk, in that they may not be filled if the market moves away or if there is significant volume already queued ahead. In contrast, market orders guarantee fills at the cost of slippage. The tension between patience for price and urgency for execution generates the high-frequency dynamics of quotes, spreads, and transaction prices.

2.2.3 LOB Transformation

The empirical distributions of quote levels, queue depths, and event counts change throughout the trading day and across market regimes. Consequently, their unconditional mean and variance are time-varying. Because many statistical and machine learning methods assume these quantities remain constant, the raw book variables are first mapped through a transformation that records increments instead of levels.

A transformation that is used in the literature is the *order-flow imbalance* (OFI) introduced by Kolm et al.[38]. For price level i at time t let $b_{t,i}$ and $a_{t,i}$ denote the bid and ask quotes, and let $v_{t,i}^{(b)}$ and $v_{t,i}^{(a)}$ be the corresponding displayed volumes. The bid side and ask side order flows are

$$\mathrm{bOF}_{t,i} = \begin{cases} v_{t,i}^{(b)}, & b_{t,i} > b_{t-1,i}, \\ v_{t,i}^{(b)} - v_{t-1,i}^{(b)}, & b_{t,i} = b_{t-1,i}, \\ -\,v_{t-1,i}^{(b)}, & b_{t,i} < b_{t-1,i}, \end{cases} \qquad \mathrm{aOF}_{t,i} = \begin{cases} -\,v_{t,i}^{(a)}, & a_{t,i} < a_{t-1,i}, \\ v_{t,i}^{(a)} - v_{t-1,i}^{(a)}, & a_{t,i} = a_{t-1,i}, \\ v_{t-1,i}^{(a)}, & a_{t,i} > a_{t-1,i} \end{cases} \tag{2.4}$$

and the imbalance is defined as

$$\mathrm{OFI}_{t,i} \;=\; \mathrm{bOF}_{t,i} \;-\; \mathrm{aOF}_{t,i}. \tag{2.5}$$

A positive value of $\mathrm{OFI}_{t,i}$ indicates net buying pressure, generated either by the addition of bid volume or by the removal of ask-side volume. A negative value indicates net selling pressure, generated either by the addition of ask volume or by the withdrawal of bid-side volume. Because OFI is expressed in first differences, its mean and variance remain comparatively stable over time, and empirical studies report that it enhances the accuracy of short-horizon return forecasts [38, 50].

Technical Challenges for High-Frequency Models

Several market characteristics create unique challenges for high-frequency price prediction in cryptocurrencies. Liquidity fragmentation means that order book depth on any single exchange may be limited, causing significant price impact from large orders. During market stress, this can trigger cascading effects as prices diverge across venues until arbitrageurs restore equilibrium [35].

The 24/7 trading environment requires models to account for time varying liquidity and volatility patterns without regular recalibration periods. Rapid shifts in market conditions can occur at any time, demanding approaches that respond to changing dynamics across global trading sessions.

Exchange infrastructure limitations introduce additional complexities. Centralized exchanges occasionally experience outages or API slowdowns during high volume periods, disrupting data feeds and order execution. Without circuit breakers or coordinated trading halts, extreme price movements can develop and reverse rapidly, creating data patterns rarely seen in traditional markets.

These distinctive market microstructure features form the environment in which high-frequency price prediction models must operate. Understanding these characteristics is essential for developing effective deep learning approaches in cryptocurrency markets.

2.3 Short-Horizon Forecasting

Early studies of intra-day price prediction applied linear filters to high-frequency returns and basic order book summaries [2]. These methods assume that past returns and book metrics relate to future price changes in a constant, linear way. In practice, the limit order book show pronounced nonlinear dependencies and its statistical behaviour varies over time [14, 48]. Neural networks can learn such nonlinear relationships directly from the data. Zhang et al. [52] demonstrated that models trained on raw order book snapshots uncover predictive patterns without manual feature design. Kolm et al. [38] found that replacing raw order book states with engineered, stationary inputs, most notably the order-flow imbalance in (2.5), produces consistent improvements in multi horizon mid-price forecasts in equity markets. Together, these results show that both end-to-end representation learning and crafted feature transformations enhance the accuracy of short-horizon price predictions.

2.3.1 Classical Time-Series Models

Classical statistical models have provided the bedrock for financial forecasting due to their transparency and computational efficiency. Methods such as autoregressive (AR), ARMA, and ARIMA model linear dependencies in time series under stationarity assumptions, enabling practitioners to forecast future values based on past observations [13]. In the context of high-frequency markets, these frameworks have been further developed to incorporate exogenous variables related to market microstructure. For example, the ARX, autoregressive with exogenous inputs model leverages both historical price data and features from the limit order book or other external signals to improve predictive performance [19, 52].

However, as financial markets have evolved, so too have the demands on modeling approaches. Traditional linear and econometric models, including AR, ARX, and their variants, often require substantial feature engineering and are limited in capturing the nonlinear dynamics, regime shifts, and high dimensionality of modern limit order books [52]. Similarly, volatility models such as ARCH and GARCH are effective for modeling time-varying volatility and variance clustering, but cannot account for the complex dependencies present in high-frequency trading environments [21, 10]. Other methods, such as state-space models, autoregressive conditional duration models, and Hawkes processes, offer further insights but generally rely on strong parametric assumptions that can hinder adaptability to the rapidly changing structure of electronic markets [22, 7].

We introduce the ARX model as our primary classical baseline before proceeding to deep learning architectures that automatically learn hierarchical nonlinear features from raw order book data.

We denote by $r_t \in \mathbb{R}$ the single horizon log-return at event time t, and by $\mathbf{x}_t \in \mathbb{R}^m$ the feature vector. The simplest linear regression reads

$$r_t = w_0 + \mathbf{v}^\top \mathbf{x}_t + \varepsilon_t, \tag{2.6}$$

where w_0 is an intercept, $\mathbf{v}$ collects feature weights, and ε_t is zero mean noise.

An autoregressive model of order n_y replaces the feature term by a sum over the past n_y returns. That is,

$$r_t = w_0 + \sum_{i=1}^{n_y} w_i \, r_{t-i} + \varepsilon_t. \tag{2.7}$$

To leverage both return persistence and lagged market signals we combine equations (2.6) and (2.7) into the ARX framework of orders n_y and n_x [13]

$$r_t = w_0 + \sum_{i=1}^{n_y} w_i \, r_{t-i} + \sum_{j=0}^{n_x-1} \mathbf{v}_j^\top \, \mathbf{x}_{t-j} + \varepsilon_t. \tag{2.8}$$

The first summation blends the past n_y returns while the second incorporates up to n_x lagged feature vectors. Each $\mathbf{v}_j$ weights the information seen j steps ago.

2.3.2 Deep Learning Models

The limit order book is naturally expressed as a series of matrices whose rows index price levels and whose entries record the corresponding bid and ask volumes; taken over time, this sequence forms a high dimensional spatio-temporal signal [18, 16]. Convolutional neural networks (CNNs) exploit shared linear filters to detect translation-invariant spatial dependencies within each matrix, making them an appropriate tool for modelling cross-level interactions in a single snapshot [40]. Long short-term memory (LSTM) networks introduce gated recurrence that selectively preserves or discards past information, allowing the model to describe temporal dependence across successive snapshots [31].

Neural Networks

We begin with the simplest form of a neural network, which is a function f_θ that transforms an input vector $\mathbf{x} \in \mathbb{R}^F$ into an output $f_\theta(\mathbf{x}) \in \mathbb{R}^M$. The collection of all trainable parameters, weights and biases, is denoted by θ, and may contain millions of real values once the network is large [43].

At the heart of this mapping is the *neuron*, a computational unit that first computes a weighted sum of its inputs and then applies a non-linear activation [26]. Concretely, given an input $\mathbf{x}$, a neuron with weight vector $\mathbf{w} \in \mathbb{R}^F$ and bias $b \in \mathbb{R}$ forms the scalar

$$z \;=\; \mathbf{w}^\top \mathbf{x} \;+\; b.$$

This scalar z is then passed through a fixed non-linear activation function $\phi\colon \mathbb{R} \to \mathbb{R}$. A widely used choice is the *rectified linear unit* (ReLU)

$$\phi(z) \;=\; \max\{0, z\}, \tag{2.9}$$

introduced to improve the performance and trainability of neural networks by mitigating the vanishing gradient problem [44]. Other activation functions, such as $\phi(z) = \tanh(z)$, are also common.

The non-linear activation function is what allows neural networks to learn and represent complicated patterns. If we remove all non-linearities, even a very deep neural network could only learn simple, straight-line relationships, making it no more powerful than a single-layer model [26].

Multiple neurons operating on the same input form a *layer*. If the layer has d_1 neurons, we collect their weight vectors into a matrix $\mathbf{W}^{(1)} \in \mathbb{R}^{d_1 \times F}$ and their biases into $\mathbf{b}^{(1)} \in \mathbb{R}^{d_1}$. The layer then computes, in one step,

$$\mathbf{h}^{(1)} = \phi\Big(\mathbf{W}^{(1)}\,\mathbf{x} \;+\; \mathbf{b}^{(1)}\Big),$$

where ϕ (2.9) is applied elementwise to the vector argument. The output $\mathbf{h}^{(1)} \in \mathbb{R}^{d_1}$ is called a *hidden representation*, because it maps the original input $\mathbf{x}$ into a new set of internal features that the network learns automatically to solve the task at hand. These features are ”hidden” in the sense that they are not specified in advance, but are discovered during training [26].

To build a *feed-forward network*, illustrated in figure 2.2, we stack L layers of neurons as described above, not counting the input. We set $\mathbf{h}^{(0)} = \mathbf{x}$ and then, for each $\ell = 1, \ldots, L$, compute

$$\mathbf{h}^{(\ell)} = \phi\Big(\mathbf{W}^{(\ell)}\,\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}\Big). \tag{2.10}$$

After L layers the vector $\mathbf{h}^{(L)}$ is our final output $f_\theta(\mathbf{x})$. The integer L is the network's *depth*, and the sequence of layer sizes $\Big(d_1, d_2, \ldots, d_L\Big)$ is referred to as its *widths*. Each successive layer composes an affine map with a non-linearity (2.9), enabling the network to approximate functions of increasing complexity and capture hierarchical patterns in the data [26].

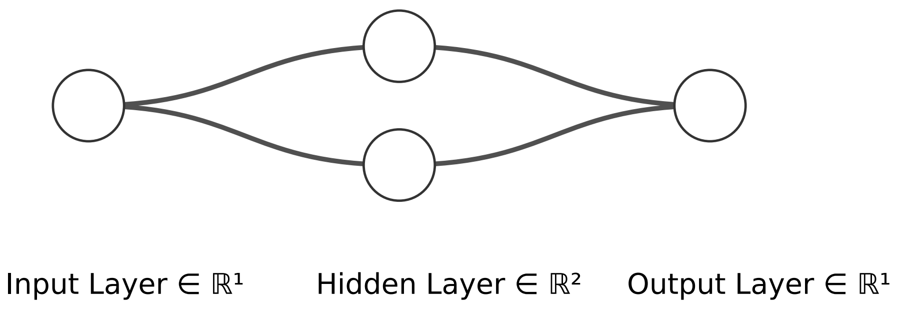

Figure 2.2: A simple feed-forward network as described in equation. (2.10). The input $\mathbf{x} \in \mathbb{R}^1$ (left) is mapped to a hidden representation $\mathbf{h}^{(1)} \in \mathbb{R}^2$ (middle), then to the output $f_\theta(\mathbf{x}) = \mathbf{h}^{(2)} \in \mathbb{R}^1$ (right). Each connection represents a learned weight, and each neuron applies an affine transform followed by a non-linear activation function ϕ. Here, the network has depth $L = 2$ and widths $(d_1, d_2) = (2, 1)$.

Training Neural Networks

When the network architecture (number of layers, number of neurons per layer, and activation functions) is fixed, the only free variables are the weight matrices and bias vectors

$$\theta = \left\{\mathbf{W}^{(\ell)},\, \mathbf{b}^{(\ell)}\right\}_{\ell=1}^{L},$$

which together collect all trainable parameters of the model.

These parameters are learned by minimizing the empirical risk over the training data [26]. Given a dataset of N samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, with inputs $\mathbf{x}_i \in \mathbb{R}^F$ and targets $y_i \in \mathbb{R}^M$, we define the average loss

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell\Big(f_\theta(\mathbf{x}_i), y_i\Big), \tag{2.11}$$

where ℓ is a loss function that measures the prediction error. For regression tasks, we typically use the mean-squared error

$$\ell(\hat{y}, y) = \tfrac{1}{2}\|\hat{y} - y\|_2^2, \tag{2.12}$$

[1]Diagram generated with NN-SVG (Alex Lenail).

while for binary classification the cross-entropy loss is common

$$\ell(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}),$$

where $\hat{y}$ is interpreted as the predicted probability of class 1.

Evaluating this loss over all N samples can be computationally expensive for large datasets. Instead, we adopt *stochastic gradient descent* (SGD), which at iteration t samples a mini-batch $B_t \subset \{1, \ldots, N\}$ of size $m \ll N$ and computes the average loss over this mini-batch

$$\mathcal{L}_{B_t}(\theta) = \frac{1}{m} \sum_{i \in B_t} \ell\Big(f_\theta(\mathbf{x}_i), y_i\Big).$$

The parameters θ are then updated by moving in the negative direction of the mini-batch gradient,

$$\theta \leftarrow \theta - \eta_t \nabla_\theta \mathcal{L}_{B_t}(\theta),$$

where $\eta_t > 0$ is the *learning rate*. With appropriate choices of learning rate, this procedure converges to a local minimum of $\mathcal{L}(\theta)$ [11].

Using a single learning rate for all parameters can be inefficient, as each parameter may have different optimal step sizes. The *Adam* optimizer addresses this by adaptively adjusting the learning rate for each parameter using estimates of the first and second moments of the gradients [36].

Let $\mathbf{g}_t = \nabla_\theta \mathcal{L}_{B_t}(\theta_t)$ denote the gradient at iteration t. Adam maintains two moving averages

$$\begin{aligned} \mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t \\ \mathbf{v}_t &= \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)(\mathbf{g}_t \odot \mathbf{g}_t) \end{aligned}$$

where $\odot$ denotes elementwise multiplication. Both $\mathbf{m}_0$ and $\mathbf{v}_0$ are initialized to zero.

To correct for bias towards zero at initialization, Adam uses the bias-corrected estimates

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}, \qquad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}.$$

The parameters are then updated as

$$\theta_{t+1} = \theta_t - \eta_t \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon}, \tag{2.13}$$

where $\beta_1, \beta_2 \in (0, 1)$ are decay rates, and ε is a small constant to avoid division by zero.

Both SGD and Adam rely on *backpropagation* to compute gradients of the loss with respect to all network parameters. Backpropagation is an algorithm that recursively applies the chain rule to propagate error signals backward through the layers of the network [26].

Let the pre-activation and post-activation at layer ℓ be

$$a^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}, \qquad \mathbf{h}^{(\ell)} = \phi(a^{(\ell)}),$$

where ϕ is the activation function applied elementwise. We define the *error signal* at layer ℓ as

$$\delta^{(\ell)} = \frac{\partial \mathcal{L}}{\partial a^{(\ell)}},$$

which quantifies how much a small change in the pre-activation $a^{(\ell)}$ at this layer would affect the overall loss $\mathcal{L}$. In other words, the error signal tells us how much each neuron's output at layer ℓ contributes to the final loss, and forms the basis for computing parameter updates.

The backpropagation algorithm computes these error signals recursively using the chain rule

$$\begin{aligned} \delta^{(L)} &= \nabla_{\mathbf{h}^{(L)}}\ell \;\odot\; \phi'(a^{(L)}), \\ \delta^{(\ell)} &= \left(\mathbf{W}^{(\ell+1)}\right)^{\top}\delta^{(\ell+1)} \;\odot\; \phi'(a^{(\ell)}), \qquad \ell = L-1,\dots,1, \end{aligned} \tag{2.14}$$

where $\odot$ denotes elementwise multiplication and ϕ' is the derivative of the activation function [26].

Once the error signals $\delta^{(\ell)}$ are computed for each layer, the gradients with respect to the weights and biases are given by

$$\nabla_{\mathbf{W}^{(\ell)}}\mathcal{L}_{B_t} = \delta^{(\ell)}\left(\mathbf{h}^{(\ell-1)}\right)^{\top}, \qquad \nabla_{\mathbf{b}^{(\ell)}}\mathcal{L}_{B_t} = \delta^{(\ell)}.$$

Because each layer only needs to communicate its error signal $\delta^{(\ell)}$ to the preceding layer, the computational cost of the backward pass is comparable to that of the forward pass, making backpropagation efficient for training deep networks.

In very deep networks, repeatedly applying the chain rule during backpropagation can cause gradients to either shrink or grow exponentially as they are propagated backward through the layers. Specifically, consider the recursion

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(\ell)}} = \left(\mathbf{W}^{(\ell+1)}\right)^{\top}\left(\phi'\left(a^{(\ell+1)}\right) \odot \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(\ell+1)}}\right).$$

At each layer, the gradient with respect to the activations is multiplied by the weight matrix and by the derivative of the activation function. If the spectral norms (largest singular values) of the weight matrices and the maximum value of $|\phi'|$ are each less than one, then repeated multiplication will cause the gradients to decay exponentially as we move backward through the network, this is known as the *vanishing gradient* problem. Conversely, if these quantities exceed one, the gradients can grow exponentially and become unstable, this is known as *exploding gradients* [26].

Regularization

Neural networks with many parameters are prone to overfitting, in which the model learns idiosyncrasies of the training set rather than general patterns that generalize to unseen data. Several regularization strategies are commonly employed to improve out-of-sample performance.

One effective approach is *weight decay* (also known as L^2 regularization), where the empirical risk in equation. (2.11) is augmented with an L^2-penalty on the weights [26]. Concretely, we minimize

$$\mathcal{L}(\theta) + \frac{\lambda}{2} \sum_{\ell=1}^{L} \left\| \mathbf{W}^{(\ell)} \right\|_F^2 ,$$

where $\lambda > 0$ is the regularization parameter that controls the strength of the penalty. The Frobenius norm is defined as

$$\|\mathbf{W}\|_F^2 = \sum_{i,j} W_{ij}^2 = \|\mathrm{vec}(\mathbf{W})\|_2^2 ,$$

which is the Euclidean norm of the matrix entries viewed as a vector.

During gradient-based optimization, this penalty adds an extra term $-\eta_t \lambda \, \mathbf{W}^{(\ell)}$ to the update of each weight matrix, discouraging large weights and thereby limiting model complexity [26].

Dropout is another regularization technique that helps prevent overfitting by introducing noise into the network during training [26]. Specifically, dropout randomly zeroes out a fraction p of the hidden unit activations in each forward pass, making the network less reliant on any particular set of features.

During training, if $\mathbf{h} \in \mathbb{R}^d$ is the pre-dropout activation vector at a given layer, dropout applies a random mask $\mathbf{r} \in \{0,1\}^d$ with

$$r_i \sim \mathrm{Bernoulli}(1-p) \quad \text{(independently for each } i\text{)}, \qquad \tilde{\mathbf{h}} = \mathrm{diag}(\mathbf{r})\, \mathbf{h},$$

so that, on average, only $(1-p)d$ units remain active per layer. This thinning effect forces the network to learn redundant representations and discourages co-adaptation of neurons.

At test time, the full network is used, so no dropout is applied, but the activations are multiplied by $(1-p)$ to maintain the same expected output as during training.

A third regularization strategy is *early stopping*, which helps prevent overfitting by monitoring the model's performance on unseen data. During training, the model is updated on mini-batches while periodically evaluating the loss on a held-out validation set. If the validation loss does not improve for a specified number of epochs, which is the patience parameter, training is halted and the parameters from the epoch with the lowest validation loss are retained. Early stopping prevents the network from overfitting the training set and descending into sharp or spiky minima that fit noise rather than true patterns in the data [26].

Convolutional Neural Networks

Convolutional layers extend the standard fully connected (dense affine) map by exploiting the spatial structure present in data such as images or limit order book snapshots (see Section 2.2) [40]. While the feed-forward networks (2.10) take input vectors $\mathbf{x} \in \mathbb{R}^F$, convolutional neural networks (CNNs) generalize this to structured tensor inputs $\mathcal{X} \in \mathbb{R}^{H \times W \times C}$, which encode spatial and channel information [26].

The convolutional layer learns K small filters, also known as kernels, each of spatial size $r \times s$ and spanning all C channels, collected in the weight tensor $\mathbf{W} \in \mathbb{R}^{r \times s \times C \times K}$, with associated biases $\mathbf{b} \in \mathbb{R}^K$. To compute an output value at location (i, j) for the k-th filter, the layer extracts a local patch $\mathcal{X}_{i:i+r-1,\, j:j+s-1,:}$, multiplies it elementwise with the k-th filter, sums all products, adds the bias b_k, and applies a non-linear activation function ϕ as defined in equation (2.9)

$$A_{i,j,k} = \sum_{u=1}^{r} \sum_{v=1}^{s} \sum_{c=1}^{C} \mathbf{W}_{u,v,c,k}\, \mathcal{X}_{i+u-1,\, j+v-1,\, c} + b_k, \tag{2.15}$$

$$Z_{i,j,k} = \phi(A_{i,j,k}). \tag{2.16}$$

This operation is analogous to the affine transformation in equation (2.10), but with local weight sharing and translation equivariance, meaning that the same rsC filter weights are applied at every spatial location, reducing the number of parameters compared to a fully connected layer.

Sliding the filter window across all valid spatial locations produces an output tensor $Z \in \mathbb{R}^{H' \times W' \times K}$, where H' and W' depend on the choice of padding and stride. The resulting tensor Z thus serves as a learned hidden representation, analogous to $\mathbf{h}^{(1)}$ in fully connected networks, but now encoding spatial features.

Stacking multiple convolutional layers increases the *receptive field*, that is, the portion of the input that affects a single activation in the output. For example, two consecutive 3×3 filters with unit stride together cover a 5×5 region of the input. Using small filters and occasional strided layers enables the network to capture both local details and more global patterns, without a rapid increase in the number of parameters [26].

Pooling operations, such as *max pooling*, further reduce the spatial size of the feature maps and increase robustness to small input shifts. Max pooling divides Z into non-overlapping blocks of size $p \times p$ and retains only the largest entry in each block

$$P_{i,j,c} = \max_{1 \le u \le p} \max_{1 \le v \le p} Z_{(i-1)p+u,\, (j-1)p+v,\, c}, \tag{2.17}$$

so that the pooled tensor has shape $\lfloor H'/p \rfloor \times \lfloor W'/p \rfloor \times K$.

A deep stack of convolution, activation, padding, stride, and pooling layers allows the network to learn hierarchical, multi-scale representations of the input. The final feature maps provide a compact and expressive description of the input grid, suitable for further processing by fully connected or recurrent layers.

Recurrent Networks and Long Short-Term Memory

Recurrent neural networks (RNNs) process sequential data by maintaining an internal state that evolves as new inputs arrive [26]. Unlike feed-forward networks (2.10), which process each example independently, RNNs incorporate temporal dependencies by feeding part of their previous output back into the computation at the next time step.

A vanilla recurrent cell receives an input vector $\mathbf{x}_t \in \mathbb{R}^F$ at time t and updates its hidden state $\mathbf{h}_t \in \mathbb{R}^d$ by

$$\mathbf{h}_t = \phi\left(\mathbf{W}_{xh}\,\mathbf{x}_t + \mathbf{W}_{hh}\,\mathbf{h}_{t-1} + \mathbf{b}_h\right), \tag{2.18}$$

where $\mathbf{W}_{xh} \in \mathbb{R}^{d\times F}$, $\mathbf{W}_{hh} \in \mathbb{R}^{d\times d}$ are weight matrices, $\mathbf{b}_h \in \mathbb{R}^d$ is a bias, and ϕ is a non-linearity as defined in (2.9). The output $\mathbf{y}_t \in \mathbb{R}^M$ is then

$$\mathbf{y}_t = \mathbf{W}_{hy}\,\mathbf{h}_t + \mathbf{b}_y, \qquad \mathbf{W}_{hy} \in \mathbb{R}^{M\times d},\ \mathbf{b}_y \in \mathbb{R}^M.$$

Repeated application of $\mathbf{W}_{hh}$ can cause gradients to vanish or explode (see 2.3.2), making it hard to learn long-range dependencies [26].

Long short-term memory (LSTM) cells address this issue by maintaining a separate cell state $\mathbf{c}_t \in \mathbb{R}^d$, which is updated through element-wise linear operations [26]. At each time step, the LSTM computes four gating vectors: an input gate $\mathbf{i}_t$, forget gate $\mathbf{f}_t$, output gate $\mathbf{o}_t$, and a candidate update $\tilde{\mathbf{c}}_t$. With weight matrices $\mathbf{W}_{x\bullet} \in \mathbb{R}^{d\times F}$, $\mathbf{W}_{h\bullet} \in \mathbb{R}^{d\times d}$, and bias vectors $\mathbf{b}_\bullet \in \mathbb{R}^d$,

$$\begin{aligned}
\mathbf{i}_t &= \sigma\Big(\mathbf{W}_{xi}\,\mathbf{x}_t + \mathbf{W}_{hi}\,\mathbf{h}_{t-1} + \mathbf{b}_i\Big),\\
\mathbf{f}_t &= \sigma\Big(\mathbf{W}_{xf}\,\mathbf{x}_t + \mathbf{W}_{hf}\,\mathbf{h}_{t-1} + \mathbf{b}_f\Big),\\
\mathbf{o}_t &= \sigma\Big(\mathbf{W}_{xo}\,\mathbf{x}_t + \mathbf{W}_{ho}\,\mathbf{h}_{t-1} + \mathbf{b}_o\Big),\\
\tilde{\mathbf{c}}_t &= \tanh\Big(\mathbf{W}_{xc}\,\mathbf{x}_t + \mathbf{W}_{hc}\,\mathbf{h}_{t-1} + \mathbf{b}_c\Big).
\end{aligned} \tag{2.19}$$

The logistic sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2.20}$$

outputs values in $(0,1)$, so that each entry in the gate vectors acts as a soft switch, multiplying the relevant signal by a value between 0 (completely blocked) and 1 (fully passed). The cell and hidden states are then updated by

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \qquad \mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),$$

where $\odot$ denotes element-wise multiplication. The forget gate controls how much old information is retained, the input gate controls how much new information is written, and the output gate determines what information is output at each step [26]. Because the cell state $\mathbf{c}_t$ is updated by element-wise products and sums, the gradient with respect to $\mathbf{c}_{t-1}$,

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \mathbf{f}_t$$

remains well-conditioned.

Thus, when $\mathbf{f}_t$ is close to one, information and gradients flow through time with little weakening, helping to prevent the vanishing gradient problem that limits simple recurrent networks [26].

At each time step, the LSTM updates its hidden state $\mathbf{h}_t$ and cell state $\mathbf{c}_t$ using the current input $\mathbf{x}_t$ and the previous hidden and cell states. This allows the network to retain relevant information over long sequences and to selectively forget or incorporate new data as needed. By processing a sequence $\mathbf{x}_1, \dots, \mathbf{x}_T$ in this way, the LSTM builds internal representations $(\mathbf{h}_t, \mathbf{c}_t)$ that capture both short-term fluctuations and long-term dependencies [26].

CNN–LSTM

The spatial patterns in each individual book snapshot and the temporal dynamics across successive snapshots both carry information about imminent price moves. Convolutional layers excel at detecting local structures within a single snapshot as described in section 2.3.2, while recurrent layers, such as LSTMs, described in section 2.3.2, capture how these patterns evolve and interact over time. By feeding the CNN's spatial encodings into an LSTM, we obtain a unified model that first compresses each high dimensional snapshot into a compact feature vector and then models the temporal dependencies among those vectors, thereby leveraging both spatial and temporal signals for improved short-horizon forecasting.

Concretely, let

$$\mathcal{X}_t \in \mathbb{R}^{H \times W \times C}$$

be the raw input at time t. The final convolutional block produces a tensor of feature maps,

$$Z_t^{(L)} = \mathrm{Conv}^{(L)}\Big(\mathcal{X}_t\Big) \in \mathbb{R}^{H_L \times W_L \times K_L},$$

where the convolution operation follows equation (2.15). To input this into the LSTM, we transform $Z_t^{(L)}$ into a fixed length vector $\mathbf{z}_t$ by either flattening

$$\mathbf{z}_t = \mathrm{vec}\Big(Z_t^{(L)}\Big) \in \mathbb{R}^{H_L W_L K_L}$$

or applying global average pooling over the spatial dimensions

$$\mathbf{z}_t = \frac{1}{H_L W_L} \sum_{i=1}^{H_L} \sum_{j=1}^{W_L} Z_{t,i,j,:} \in \mathbb{R}^{K_L}.$$

This vector $\mathbf{z}_t$ then serves as the input to an LSTM, which updates its hidden and cell states according to the recurrence (2.18)

$$(\mathbf{h}_t, \mathbf{c}_t) = \mathrm{LSTM}\Big(\mathbf{z}_t,\, \mathbf{h}_{t-1},\, \mathbf{c}_{t-1}\Big), \qquad \mathbf{h}_0 = \mathbf{0},\; \mathbf{c}_0 = \mathbf{0}.$$

After processing a sequence of T snapshots, the final hidden state $\mathbf{h}_T \in \mathbb{R}^d$ summarizes both the spatial features and their temporal evolution. A single step forecast is then made by

$$\hat{y} = \sigma\Big(\mathbf{W}_o \mathbf{h}_T + \mathbf{b}_o\Big), \qquad \mathbf{W}_o \in \mathbb{R}^{1 \times d},\; \mathbf{b}_o \in \mathbb{R},$$

where σ is the logistic sigmoid function (2.20).

3
Methodology

The methodology closely follows Kolm et al. [38], adapting their multi-horizon return forecasting approach to the continuous trading environment of cryptocurrency markets. At a high level, the workflow consists of data preprocessing, rolling-window model training, and an out-of-sample evaluation that measures forecasting performance via an out-of-sample R^2. We formulate the prediction task as a multi-horizon regression problem where models simultaneously forecast log-returns at ten future time points, implement several neural architectures alongside a linear ARX baseline for comparison, and maintain strict separation between training, validation, and test sets to ensure realistic performance assessment.

3.1 Forecasting Task

We frame our forecasting problem as predicting ten future mid-price log-returns simultaneously. At each event time t, we collect these into the vector

$$\mathbf{r}_t \;=\; \Big(r_{t,1},\, r_{t,2},\, \dots,\, r_{t,10}\Big)^{\top}. \tag{3.1}$$

This setup allows the model to learn patterns that influence both very short-term and slightly longer-term outcomes.

Because cryptocurrency markets operate continuously, we measure the total period of interest as exactly 24 hours in milliseconds

$$T \;=\; 24 \times 60 \times 60 \times 1000 \;=\; 86{,}400{,}000 \text{ ms}.$$

Within that window we observe N nonzero mid-price (2.3) changes. We define the average time between such changes as

$$\Delta t \;=\; \frac{T}{N}.$$

To space our ten forecast points evenly, we set

$$h_k \;=\; \frac{k}{5}\,\Delta t, \qquad k = 1, 2, \dots, 10. \tag{3.2}$$

This choice places the first horizon at $0.2 \times \Delta t$ of the average interval and the last at two full intervals, $2.0 \times \Delta t$, covering a range of microstructural dynamics.

To guarantee all inputs are available before forecasting, we introduce a fixed latency buffer of $\delta\tau = 50$ms. This buffer is larger than typical network round-trip and computation delays of 20-40ms reported in practice [5], and thus ensures that predictions are made strictly using information available up to time t, with no lookahead bias.

Each return target is the buffered log-return, rescaled to basis point units

$$r_{t,k} \;=\; 10\,000\;\ln \frac{p_{t+\delta\tau+h_k} + \varepsilon}{p_{t+\delta\tau} + \varepsilon}. \tag{3.3}$$

Here p_t is the mid-price at time t, and ε is a small offset to prevent division by zero. Multiplying by 10,000 converts the natural log value into basis points, in line with the conventions for spread, fees, and risk in financial markets. This scaling also ensures that the neural network's output is well conditioned and directly interpretable. For example, a prediction of +12 corresponds to an expected rise of twelve basis points or 0.12%.

Our inputs consist of a lookback window of $W = 100$ consecutive feature vectors,

$$\mathbf{x}_{t-W},\; \mathbf{x}_{t-W+1},\; \dots,\; \mathbf{x}_{t-1}.$$

When using raw limit order book features, each vector $\mathbf{x}_\tau \in \mathbb{R}^{40}$, as we include 10 bid prices, 10 bid volumes, 10 ask prices, and 10 ask volumes at each time step. When using engineered order flow imbalance features (2.5), each vector $\mathbf{x}_\tau \in \mathbb{R}^{10}$. In all cases, updates are included in chronological order without downsampling, preserving full microstructural information.

Prior to training, each feature channel is standardized using statistics computed only from the training data. For feature i we compute its mean and standard deviation as

$$\mu_i \;=\; \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{t \in \mathcal{D}_{\text{train}}} x_{t,i}, \qquad \sigma_i \;=\; \sqrt{\frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{t \in \mathcal{D}_{\text{train}}} (x_{t,i} - \mu_i)^2}$$

and transform each observation as

$$\hat{x}_{t,i} \;=\; \frac{x_{t,i} - \mu_i}{\sigma_i}.$$

The same μ_i and σ_i are applied to the validation and test sets to prevent information leakage from future data into feature scaling.

The training process creates time-based windows where each window is split chronologically into three consecutive periods: 21 days for fitting the model parameters, followed by 7 days for validation to tune any hyperparameters and prevent overfitting, and finally 7 days for out-of-sample testing to evaluate real world performance. This ensures that the model learns from one historical period and is always evaluated on a strictly future segment. This process is visualized in figure 3.1.

Each architecture is trained by minimising the mean-squared error on the training set

$$\mathcal{L}(\theta) \;=\; \frac{1}{N_{\text{train}}} \sum_{t \in \mathcal{D}_{\text{train}}} \Big\| g_\theta\big(\hat{X}_{t-W:t-1}\big) - \mathbf{r}_t \Big\|_2^2,$$

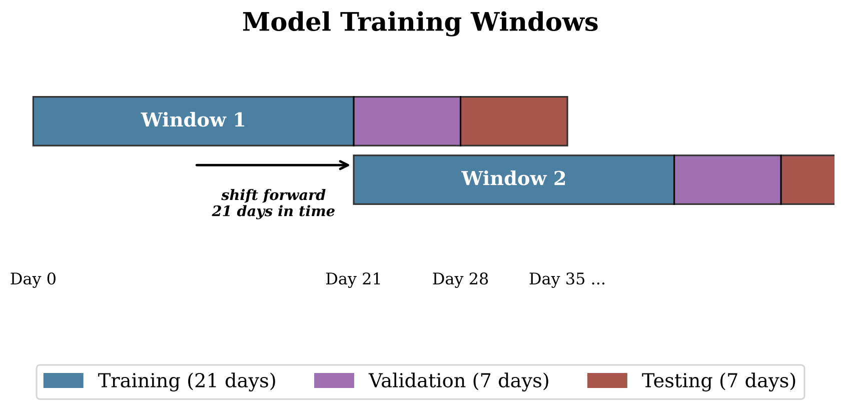

Figure 3.1: Each window consists of 21 days for model fitting (blue), 7 days for validation (purple), and 7 days for testing (red). Window 2 begins at day 21, creating an overlap pattern with a 21-day forward shift in time.

where g_θ is the neural network with parameters θ, $\hat{X}_{t-W:t-1}$ is the matrix of the last 100 standardised vectors, and $N_{\text{train}} = |\mathcal{D}_{\text{train}}|$. We employ the Adam optimizer with initial learning rate $\eta = 10^{-3}$ and L^2 weight decay $\lambda = 10^{-5}$.

We prevent overfitting by applying early stopping based on the validation loss. After each epoch we compute the average validation loss. If it fails to decrease by at least 10^{-4} for three consecutive epochs we terminate training and restore the best performing parameters.

To see how much value the neural networks add we compare them with a reasonable alternative, a constant mean predictor, built separately for every horizon. This benchmark assumes that the best forecast one can make, in the absence of any market structure, is the historical mean of the return at that horizon. Using only the training data we estimate that mean as

$$\bar{r}_k = \frac{1}{N_{\text{train}}} \sum_{t\in\mathcal{D}_{\text{train}}} r_{t,k},$$

so the benchmark always outputs the same number $\bar{r}_k$ regardless of the input features.

Because the benchmark has no parameters to tune we measure its accuracy directly on the held out test set,

$$\text{MSE}_{\text{bench},k} = \frac{1}{N_{\text{test}}} \sum_{t\in\mathcal{D}_{\text{test}}} \left(r_{t,k} - \bar{r}_k\right)^2.$$

A learning model with parameters θ yields its own test error $\text{MSE}_{\text{model},k}$. To summarise the relative improvement we report the out-of-sample coefficient of determination

$$R^2_{\text{OOS},k} = 1 - \frac{\text{MSE}_{\text{model},k}}{\text{MSE}_{\text{bench},k}}. \tag{3.4}$$

An $R^2_{\text{OOS},k}$ of zero means the network explains no more variance than the constant mean forecast, a value between zero and one means it reduces squared error by that fraction, and a negative value means it performs worse than simply predicting the historical mean. This scale lets us compare models of very different complexity at each horizon using a single, dimensionless metric.

3.2 Implementation of Models

To uncover the spatio-temporal structure in our high-frequency limit order book data, we develop and compare three neural architectures: a purely convolutional model, a purely recurrent model based on LSTM cells, and a hybrid CNN-LSTM. All three are trained to predict a 10-step ahead return vector from 100 consecutive book snapshots, and their designs follow the principles laid out by Kolm et al. [38] and Zohren et al. [52]. Below we describe each in turn and all model architectures are summarized in table 3.1.

3.2.1 Convolutional Network

The convolutional network inputs the most recent $W = 100$ book messages and arranges them in a tensor

$$\mathcal{X} \in \mathbb{R}^{100\times d\times 1},$$

where the first dimension is time, the second dimension is the feature dimension d, and the third is the channel dimension. If using raw limit order book price and volume features, $d = 40$, corresponding to the 20 bid and 20 ask levels, each with price and volume. If using the order flow imbalance features (2.5), then $d = 10$.

The first convolutional block applies 32 filters of spatial size 1×2 and stride 1×2. When $d = 40$, this stride fuses the price-volume pair at each level and reduces the lateral dimension from 40 to 20. This is followed by a temporal convolution with a 4×1 kernel, unit stride, and zero padding along the time dimension so that the temporal length of 100 is preserved. The resulting feature maps have shape

$$100 \times d' \times 32,$$

where $d' = d/2$ after the first lateral convolution.

A second convolutional block with the same 1×2 filters and stride reduces the lateral dimension further ($20 \to 10$ if $d = 40$ or $10 \to 5$ if $d = 10$), followed by another temporal convolution. The resulting feature maps have shape

$$100 \times d'' \times 32,$$

where $d'' = d'/2$.

The third block aggregates information across all remaining levels with a $1 \times d''$ convolution, stride 1×1, producing feature maps of shape

$$100 \times 1 \times 32.$$

To detect patterns at multiple scales, the network applies an inception module with three parallel branches

- a 1×1 convolution followed by a 3×1 convolution, 64 filters in each layer,
- a 1×1 convolution followed by a 5×1 convolution, 64 filters in each layer,
- a 3×1 max-pooling followed by a 1×1 convolution, 64 filters.

Concatenating the branch outputs along the channel axis yields a tensor of shape

$$100 \times 1 \times 192.$$

Flattening this tensor produces the feature vector $\mathbf{z} \in \mathbb{R}^{19{,}200}$ (for $d = 40$), which is then fed through two fully connected layers of sizes 512 and 256, each followed by a ReLU activation (2.9) and dropout for regularization. The final output layer has size 10, yielding the multi-horizon return predictions (3.1),

$$\hat{\mathbf{r}}_t = \mathbf{W}_2\, \phi\Big(\mathbf{W}_1 \mathbf{z} + \mathbf{b}_1\Big) + \mathbf{b}_2 \in \mathbb{R}^{10},$$

where $\mathbf{W}_1, \mathbf{W}_2$ and $\mathbf{b}_1, \mathbf{b}_2$ are the weights and biases of the dense layers, and $\phi(\cdot)$ is the ReLU activation function defined in (2.9).

For $d = 10$ (when using OFI features), the convolutional blocks operate analogously but on reduced lateral dimensions at each step, and the final flattened vector has dimension $100 \times 1 \times 192 = 19,200$ as above.

3.2.2 LSTM Network

The LSTM model uses a two-layer LSTM, each with a hidden size of 128, to process the last $W = 100$ messages as an ordered sequence

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{100} \in \mathbb{R}^d,$$

where the input dimension d depends on the feature set: if raw limit order book price and volume features are used, $d = 40$ as in the convolutional network. If the order flow imbalance features (2.5) are used, then $d = 10$. In both cases, feature vectors are presented in strict chronological order without downsampling. At every step t, the long short-term memory cell updates its hidden state $\mathbf{h}_t \in \mathbb{R}^{128}$ and cell state $\mathbf{c}_t \in \mathbb{R}^{128}$ according to the gating equations (2.19).

After processing all 100 events, the model retains only the final hidden state

$$\mathbf{h}_{100} \in \mathbb{R}^{128},$$

which serves as a compressed representation of the entire lookback window. This vector is then passed through two fully connected layers of sizes 256 and 128, each followed by ReLU activation (2.9) and dropout. The final output layer has size 10 and produces the multi-horizon return predictions (3.1)

$$\hat{\mathbf{r}} = \mathbf{W}_2\, \phi\Big(\mathbf{W}_1\, \mathbf{h}_{100} + \mathbf{b}_1\Big) + \mathbf{b}_2 \in \mathbb{R}^{10},$$

where $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{b}_1$, and $\mathbf{b}_2$ are the weights and biases of the dense layers, and $\phi(\cdot)$ denotes the ReLU function.

3.2.3 CNN–LSTM Hybrid

The hybrid network chains a convolutional front-end (section 2.3.2) to an LSTM back-end (section 2.3.2), so that local spatial patterns are extracted first and their temporal evolution is modeled second. We describe the architecture for raw limit order book features $d = 40$, and with engineered order flow imbalance (2.5) feature, the input dimension is reduced to $d = 10$.

The processing unfolds in four stages.

1. ***Block 1****: Level-wise fusion and temporal convolution.* A 1×2 convolution with 32 filters and stride 1×2 fuses the price-volume pair at each book level (reducing from 40 to 20 features), followed by a 4×1 temporal convolution with unit stride and padding along time. The output shape is $100 \times 20 \times 32$.
2. ***Block 2****: Cross-side fusion and temporal convolution.* A 1×2 convolution with 32 filters and stride 1×2 combines bid and ask sides (reducing from 20 to 10), followed by a 4×1 temporal convolution. Output: $100 \times 10 \times 32$.
3. ***Block 3****: Level aggregation.* A 1×10 convolution with 32 filters aggregates all levels, yielding $100 \times 1 \times 32$.
4. ***Block 4****: Inception module.* Three parallel branches:
   - 1×1 convolution then 3×1 convolution (64 filters each),
   - 1×1 then 5×1 convolution (64 filters each),
   - 3×1 max-pooling then 1×1 convolution (64 filters).

   Concatenating branch outputs yields a $100 \times 1 \times 192$ tensor.

The $100 \times 1 \times 192$ tensor is reshaped to a sequence of length 100, each element a 192-dimensional vector, $z_t \in \mathbb{R}^{192}$, $t = 1, \dots, 100$. This sequence is processed by a 2-layer LSTM with 64 hidden units. After the final step, the hidden state $\mathbf{h}_{100} \in \mathbb{R}^{64}$ summarizes all spatial features and their dynamics. This vector is passed through two fully connected layers of sizes 256 and 128, each with ReLU activation (2.9) and dropout, then to the final output layer of size 10, yielding the multi-horizon return prediction (3.1)

$$\hat{\mathbf{r}} = \mathbf{W}_o\, \phi\Big(\mathbf{W}_2\, \phi(\mathbf{W}_1\, \mathbf{h}_{100} + \mathbf{b}_1) + \mathbf{b}_2\Big) + \mathbf{b}_o \in \mathbb{R}^{10},$$

where $\phi(\cdot)$ denotes the ReLU function.

3.2.4 ARX Model

As a linear benchmark, we implement an autoregressive model with exogenous variables (ARX) at each forecast horizon k. The ARX framework is a standard approach in time series econometrics and serves as a natural baseline for evaluating more complex deep learning architectures [13, 38], as introduced in section 2.3.1.

Denote by $r_{t,k}$ the buffered return at time t and forecast horizon k (3.1). While a complete ARX(n_y, n_x) model includes both autoregressive terms (lagged returns) and exogenous variables (order book features), we focus here on the exogenous-only variant, ARX$(0, W)$, to isolate the predictive contribution of the order book features, rather than the effect of lagged returns. Specifically, for each horizon k we estimate

$$r_{t,k} = w_{0,k} + \sum_{j=0}^{W-1} \mathbf{v}_{j,k}^{\top} \mathbf{x}_{t-j} + \varepsilon_{t,k}, \tag{3.5}$$

where $\{\mathbf{v}_{j,k}\}_{j=0}^{W-1} \subset \mathbb{R}^d$ are coefficient vectors for each lagged feature vector (with $d = 40$ for raw limit order book features or $d = 10$ for OFI features (2.5)), $w_{0,k} \in \mathbb{R}$ is an intercept, and $\varepsilon_{t,k}$ is a zero-mean noise term. The parameter $W = 100$ matches the lookback window used by the neural networks, ensuring a fair comparison.

Parameters are estimated independently for each forecast horizon k using ordinary least squares on the training set. This results in $W \times d + 1$ parameters per forecast horizon, for example, $4,001$ parameters for $d = 40$. While this is much fewer than in the deep neural models, the ARX still captures linear dependencies across time and features. The same input features and data splits are used as in the neural models, allowing for direct comparison of linear and nonlinear prediction performance.

3.2.5 Training Procedure

All models are trained by minimizing the empirical risk (2.12) over the training set

$$\mathcal{L}(\theta) = \frac{1}{N_{\text{train}}} \sum_{t \in \mathcal{D}_{\text{train}}} \left\| g_\theta\left(\hat{\mathbf{X}}_{t-W:t-1}\right) - \mathbf{r}_t \right\|_2^2.$$

We use the Adam optimizer with decay rates $\beta_1 = 0.9$, $\beta_2 = 0.999$ (2.13), and He normal initialization for layers with ReLU activation (2.9). Convolutional and linear layer weights are initialised with He normal initialisation, which preserves activation variance under ReLU (2.9) and mitigates vanishing or exploding gradients in deep networks [29]. In LSTM layers, input-to-hidden weights use Xavier initialisation to maintain balanced signal propagation through sigmoid and tanh gates [25], while hidden-to-hidden recurrent weights use orthogonal initialisation to preserve the norm of the hidden state across time steps and prevent gradient decay or explosion [26]. All biases are set to zero to break symmetry without introducing bias. To guard against overfitting we apply dropout at rate 0.5 in the CNN and CNN-LSTM fully connected layers, and 0.2 in the standalone LSTM fully connected layers, which discourages co-adaptation of units and improves generalisation [26], and employ early stopping with patience set to 3 epochs based on validation loss, thereby halting training once further improvements become negligible [26].

Model	Layer	Kernel Units	Stride Hidden Size	Output Shape
CNN				
	Conv1 (fuse)	$1 \times 2,\ 32$	1×2	$100 \times 20 \times 32$
	Conv2 (temp)	$4 \times 1,\ 32$	1×1	$100 \times 20 \times 32$
	Conv3 (fuse)	$1 \times 2,\ 32$	1×2	$100 \times 10 \times 32$
	Conv4 (temp)	$4 \times 1,\ 32$	1×1	$100 \times 10 \times 32$
	Conv5 (agg)	$1 \times 10,\ 32$	1×1	$100 \times 1 \times 32$
	Inception	—	—	$100 \times 1 \times 192$
	FC1	512 units	—	512
	FC2	256 units	—	256
	Output	10 units	—	10
LSTM				
	Input	—	—	100×40
	LSTM (2-layer)	—	128	128
	FC1	256 units	—	256
	FC2	128 units	—	128
	Output	10 units	—	10
CNN–LSTM				
	Input	—	—	100×40
	Conv1 (fuse)	$1 \times 2,\ 32$	1×2	$100 \times 20 \times 32$
	Conv2 (temp)	$4 \times 1,\ 32$	1×1	$100 \times 20 \times 32$
	Conv3 (fuse)	$1 \times 2,\ 32$	1×2	$100 \times 10 \times 32$
	Conv4 (temp)	$4 \times 1,\ 32$	1×1	$100 \times 10 \times 32$
	Conv5 (agg)	$1 \times 10,\ 32$	1×1	$100 \times 1 \times 32$
	Inception	—	—	$100 \times 1 \times 192$
	LSTM (1-layer)	—	64	64
	FC1	256 units	—	256
	FC2	128 units	—	128
	Output	10 units	—	10

[1] Two successive 4×1 temporal convolutions, stride 1×1, padding preserves length.
[2] Three parallel branches on a $100 \times 1 \times 32$ tensor: (i) 1×1 conv $\rightarrow$ 3×1 conv (64 filters each), (ii) 1×1 conv $\rightarrow$ 5×1 conv (64 filters each), (iii) 3×1 max-pool $\rightarrow$ 1×1 conv (64 filters). Outputs concatenated along channels to yield $100 \times 1 \times 192$.

Table 3.1: Summary of network architectures and layer parameters.

Because we evaluate three different cryptocurrencies, with two feature sets each (LOB and OFI), and fit five rolling windows per coin feature combination for each of the three architectures (CNN, LSTM, CNN–LSTM), we train in total

$$3\text{ (coins)} \times 2\text{ (features)} \times 3\text{ (architectures)} \times 5\text{ (windows)} = 90$$

distinct models. We fit five windows to ensure that each model is tested on five non-overlapping out-of-sample periods. All models are implemented in PyTorch and trained on a single NVIDIA GeForce RTX 4070 GPU, a process described appendix A. Finally, table 3.2 summarises the inputs and hyper-parameters used for each model.

Model	Input	L	N_p	η	B	E	ES
CNN	LOB	12	1.02×10^7	10^{-3}	64	50	Yes
CNN	OFI	12	1.02×10^7	10^{-3}	64	50	Yes
LSTM	LOB	4	2.86×10^5	10^{-3}	64	50	Yes
LSTM	OFI	4	2.71×10^5	10^{-3}	64	50	Yes
CNN-LSTM	LOB	13	4.37×10^5	10^{-3}	64	50	Yes
CNN-LSTM	OFI	13	4.21×10^5	10^{-3}	64	50	Yes

Table 3.2: Summary of inputs and hyper-parameters used in this study. L: number of layers; N_p: number of parameters; η: learning rate; B: batch size; E: epochs; ES: early stopping.

3.3 Data Collection & Preprocessing

High-frequency limit order book data are obtained from ByBit capturing every update as it occurred. This collection includes full order book snapshots and subsequent incremental updates that modify individual price levels. Each record comprises the message topic the update type the event timestamp the computer timestamp lists of bid price and volume pairs lists of ask price and volume pairs and the corresponding sequence and update identifiers. Such a schema provides a millisecond level view of market activity and preserves the precise ordering of events necessary for microstructure analysis.

The dataset covers the period from January to March 2024 and encompasses three cryptocurrency pairs ADAUSDT, BTCUSDT and ETHUSDT. These instruments are chosen to span a broad spectrum of liquidity regimes as reflected in their differing tick sizes average daily traded volumes and typical depth profiles. Over the three month interval this yields approximately 55 million updates for ADA, 108 million updates for BTC and 100 million updates for ETH.

During preprocessing raw snapshots and incremental updates are merged into a unified chronological series and strictly ordered by timestamp to maintain causality.

Nested lists of bid and ask entries are unfolded into distinct price and volume fields for the first ten levels on each side of the book thereby yielding 40 price and volume variables at each time step. Any records exhibiting incomplete level data or missing values are discarded to ensure a consistent feature matrix. The detailed structure of the processed dataset is described in appendix B.

For each timestamp a mid-price feature is computed as the arithmetic mean of the best bid and best ask, as described in section 2.2. Thereafter order flow imbalance features are extracted as defined in equations (2.4) and (2.5). To mitigate the impact of extreme observations each feature series is winsorized at the 0.1% level. The final output consists of a time aligned tabular dataset containing synchronized price volume and order flow imbalance features ready for input to downstream modelling.

4

Results

This section presents our empirical findings on the predictability of short-horizon mid-price log-returns using deep learning models trained on limit order book data and its order flow imbalance (OFI) summary (2.5). We evaluate forecast performance in terms of explained variance (3.4), directional accuracy, statistical differences between feature sets, and computational efficiency. For each metric, we compare convolutional (CNN), recurrent (LSTM), and hybrid CNN-LSTM models, and assess the trade-off between using the full high dimensional book state inputs and the more compact OFI feature set.

We begin by quantifying out-of-sample predictability and how it varies with forecast horizon and choice of input features. Next, we test the statistical significance of observed performance differences between these feature inputs. We then examine each model's ability to correctly forecast the direction of price moves. Finally, we analyze the computational cost of training.

4.1 Predictability over Short Horizons

Figure 4.1 highlights the inability of a classic linear autoregressive model to extract any meaningful signal from raw order book states. Across all three coins and every forecast horizon, the ARX's out-of-sample R^2 (3.4) hovers essentially at zero, showing that without nonlinear feature extraction or memory, stationary price and volume levels contain no exploitable predictability over these short time horizons.

By contrast, deep neural networks are able to extract meaningful predictive signals from microstructure inputs, resulting in substantial forecasting gains. At the first horizon, $h_1 = 0.2\,\Delta t$, the hybrid CNN-LSTM, which combines convolutional feature extraction with sequential memory, explains about 5% of the variance in out-of-sample mid-price log-returns when trained on the full 40-dimensional book-state. The simpler LSTM follows closely, and even the purely convolutional CNN captures just below 4%. This illustrates how local depth patterns, via convolution, and temporal dependencies, via recurrence, work together to uncover predictive patterns in the limit order book.

Reducing inputs to the 10-dimensional order flow imbalance series (2.5) yields only a modest performance drop, with all three architectures maintaining over 4% ex-

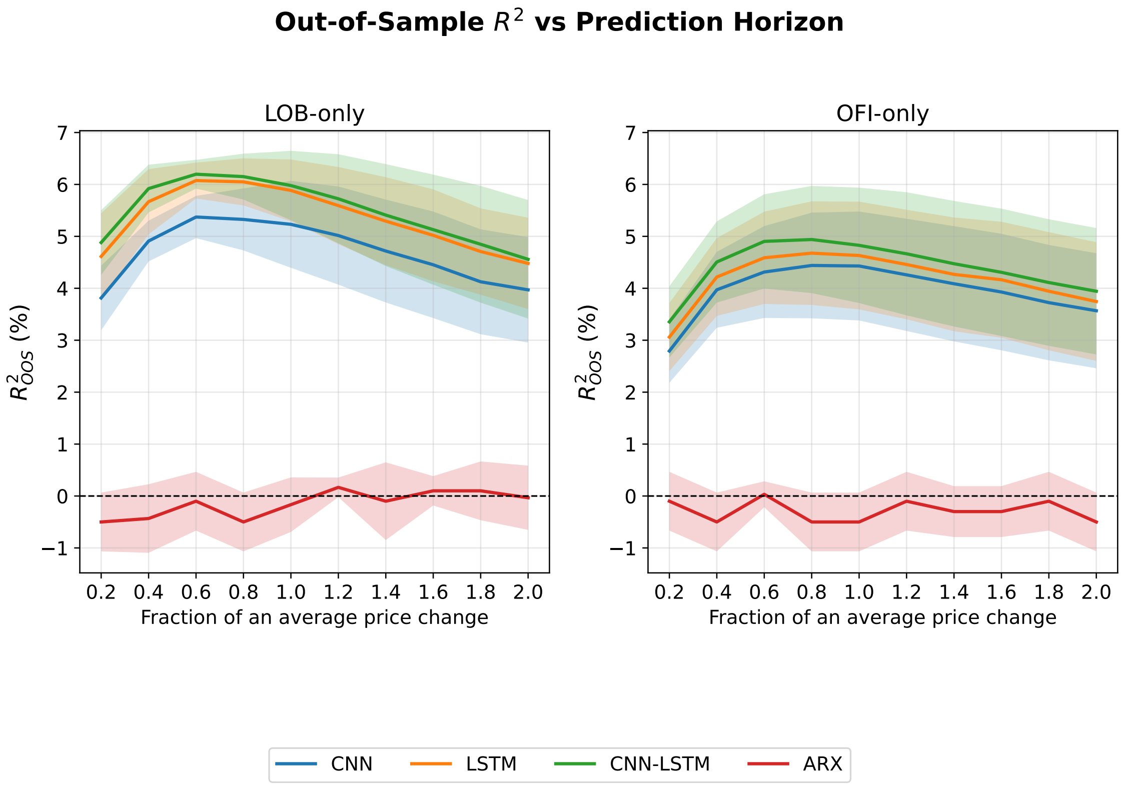

Figure 4.1: R^2_{OOS} of different model-input pairs as a function of prediction horizon, expressed in fractions of an average mid-price change. Left panel models are trained on the full 40-dimensional order book state. Right panel models are trained on the 10-dimensional OFI series. Curves correspond to CNN (blue), LSTM (orange), CNN-LSTM (green), and ARX (red). Shaded bands denote one standard deviation of R^2_{OOS} computed over all test samples (three coins $\times$ 5 rolling windows).

plained variance for forecast horizons h_2 and greater. This demonstrates that OFI preserves much of the underlying signal while omitting detailed patterns that arise from the full structure of the order book, which accounts for the gap in peak R^2_{OOS} between raw book and OFI inputs.

Examining the left panel in more detail, we see a pronounced peak in model performance around $h_3 = 0.6\,\Delta t$. Starting from $0.2\,\Delta t$, all three neural nets steadily improve as the horizon lengthens, suggesting that very short price movements carry weak but detectable structure that becomes more pronounced up to roughly half of an average price change. Beyond that point, predictive power slowly declines. By $h_{10} = 2.0\,\Delta t$, the CNN-LSTM has fallen back to below 5%, the LSTM similarly, and the CNN to 4%. This slow decay reflects the inherently fleeting nature of microstructure signals: as the forecast horizon extends further beyond the most recent order book updates, the available depth and flow patterns become less informative, and the models finite lookback window of $W = 100$ events may limit their ability to capture longer range dependencies.

Turning to the right panel, each curve shifts downward by approximately $1 - 1.5$ percentage points at its maximum when using the OFI inputs. Moreover, the ± 1 SD around each curve widen a bit compared to the LOB case. This increase in dispersion indicates that while OFI captures the core imbalance signal, its explanatory power fluctuates more across different test windows. This suggests that certain details or interactions present across multiple levels of the order book are not fully captured by the OFI summary.

Together, these observations show that the full book state representation provides both stronger peak performance and greater stability, yielding about one extra percentage point explained variance at the optimal horizon and tighter performance bands. At the same time, the OFI transform remains an effective condensed summary, preserving much of the signal with a quarter the amount of features.

4.2 Feature Input Performance

To assess whether the additional predictive power of the full book over the OFI summary holds consistently across all three network architectures, we compute the average uplift in out-of-sample R^2 between LOB and OFI inputs for each combination of coin and out-of-sample window. For each block, defined as one coin and one rolling window, we average the uplift across the three network architectures $m \in \{\text{CNN, LSTM, CNN-LSTM}\}$, resulting in 15 block means per forecast horizon.

For each forecast horizon h_k, the block mean is

$$\bar{\delta}_{b,k} \;=\; \frac{1}{3}\sum_{m}\Big(R^2_{\text{OOS,LOB},m}(b,k) - R^2_{\text{OOS,OFI},m}(b,k)\Big)\,,$$

where $b = 1,\ldots,15$ indexes all 3 coins and 5 windows.

To quantify statistical uncertainty and test whether the observed uplift is systematic, we use a nonparametric block bootstrap. Because the results from the three model architectures are not independent when evaluated on the same coin and window, we treat each coin and window combination as a single block and average the uplift across architectures within each block. We then repeatedly sample, with replacement, from the set of 15 block means and recalculate the mean uplift for each bootstrap sample. The empirical distribution of these means provides an estimate of the sampling variability, from which we construct 95% confidence intervals using the 2.5th and 97.5th percentiles. The bootstrap distribution is also used to compute a two-sided p-value for the null hypothesis that the mean uplift is zero. This procedure allows us to assess statistical significance while properly accounting for model dependence within each block.

Table 4.1 reports, for each forecast horizon, the observed mean uplift, the 95% block-bootstrap confidence interval, and the corresponding bootstrap p-value.

Horizon k	**Mean ΔR^2**	**95 % CI**	**p−value**
1	0.0137	[0.0134, 0.0139]	< 0.001
2	0.0127	[0.0081, 0.0167]	< 0.001
3	0.0128	[0.0066, 0.0185]	< 0.001
4	0.0116	[0.0036, 0.0185]	0.0028
5	0.0107	[0.0023, 0.0183]	0.0134
6	0.0098	[0.0012, 0.0178]	0.0198
7	0.0086	[-0.0003, 0.0168]	0.0530
8	0.0073	[-0.0018, 0.0157]	0.0886
9	0.0063	[-0.0018, 0.0144]	0.1226
10	0.0058	[-0.0033, 0.0141]	0.1860

Table 4.1: Block-bootstrap estimates of the mean uplift in out-of-sample R^2 when using full book-state inputs versus OFI inputs. Each block corresponds to one coin and one rolling window, giving 15 blocks total. Each bootstrap replicate samples these blocks with replacement for $B = 10{,}000$ bootstrap replicates.

At the three shortest horizons, the uplift from using full book state features is highly significant, where the 95% confidence intervals lie entirely above zero and the bootstrap p-values are less than 0.01. This statistically significant advantage persists through horizons h_1 to h_6 with all $p < 0.05$, before fading as the intervals widen and p-values increase. Beyond h_6, the observed uplift becomes small and is not statistically significant at the 5% level, indicating that any remaining advantage from the full LOB inputs over OFI could plausibly arise by chance at horizons $k > 6$.

This confirms that the book state representation delivers an advantage at the shortest time scales, while its incremental benefits diminish as the forecast horizon extends.

4.3 Directional Accuracy

To complement our analysis of out-of-sample R^2, we also evaluate each model's sign accuracy. This measures how often the model correctly predicts the direction of non-zero log-returns. That is, whether both the predicted and true log-returns are positive or both are negative.Returns that are exactly zero are ignored in this calculation, as their direction is not meaningful.

Let y_i denote the true mid-price log-return for test sample i, and let $\hat{y}_i$ denote the model's predicted log-return for the same sample. We define the set of relevant test indices as

$$\mathcal{I} = \{i : y_i \neq 0\}.$$

The sign accuracy at horizon k is then given by

$$\mathrm{SignAcc}_k = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \mathbb{1}\Big(\mathrm{sign}(\hat{y}_i) = \mathrm{sign}(y_i)\Big),$$

where $\mathbb{1}(\cdot)$ is the indicator function, which equals 1 if its argument is true and 0 otherwise.

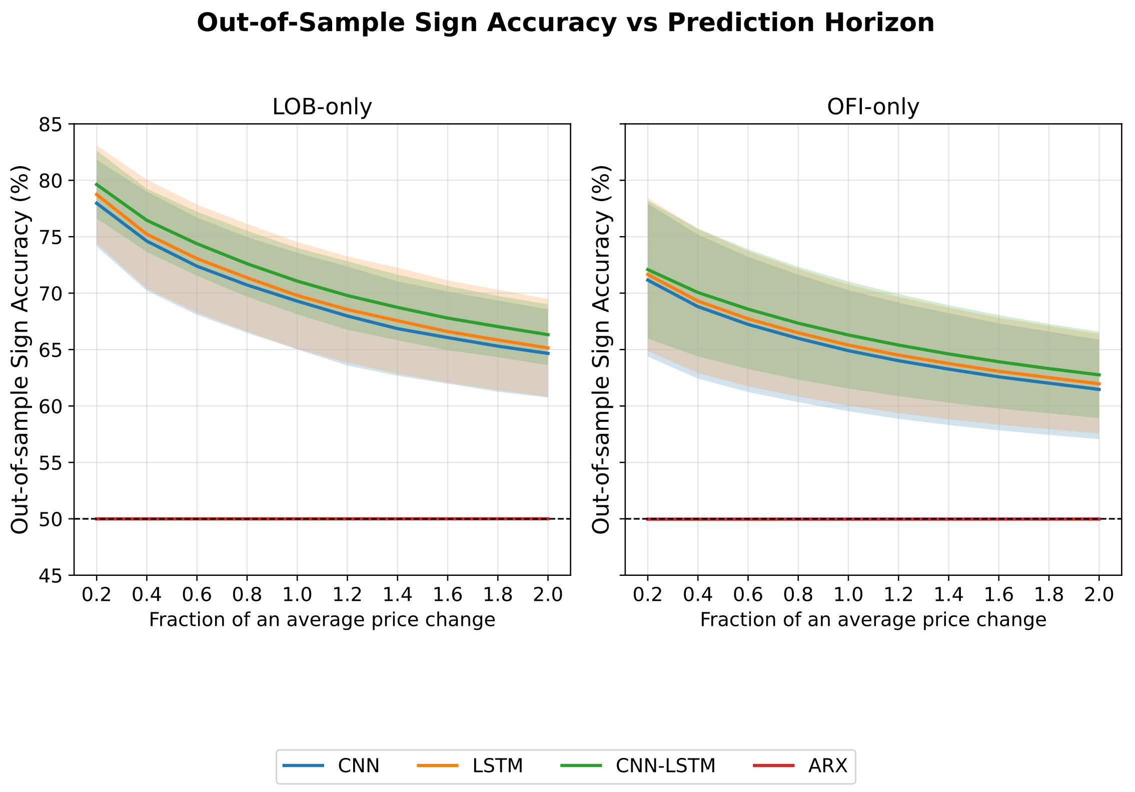

Figure 4.2: Out-of-sample sign accuracy of deep models vs. forecast horizon, aggregated over ADA, BTC, and ETH. The left panel shows the full 40-dimensional LOB inputs and the right panel shows the 10-dimensional OFI inputs. Curves show mean ±1 SD across five rolling windows and the ARX baseline (red) sits on the 50% random guessing line.

Figure 4.2 shows the mean sign accuracy for the CNN (blue), LSTM (orange), and CNN-LSTM (green) models, aggregated across ADA, BTC, and ETH. Using the full 40-dimensional limit order book inputs, the hybrid CNN-LSTM model achieves the highest peak sign accuracy, reaching around 80% at the shortest forecast horizon h_1. In comparison, the LSTM and CNN models peak at approximately 77%. As the forecast horizon increases, sign accuracy declines smoothly for all models, settling at about $65 - 66\%$ by h_{10}. In contrast, the ARX model (red) remains flat at 50% across all horizons and input types, indicating that this linear benchmark is unable to extract any directional information beyond random guessing.

When the models use the more compact 10-dimensional OFI summary features, peak sign accuracy drops by roughly $6 - 7$ percentage points for CNN-LSTM, with comparable decreases observed for LSTM and CNN. There is also slightly greater variability across different time windows in this setting. These results closely track the patterns seen in our R^2_{OOS} analysis, reinforcing that the CNN-LSTM hybrid provides the strongest directional predictions overall. Moreover, while the OFI summary preserves much of the models' predictive power, some accuracy is lost compared to the full limit order book representation.

4.4 Computational Efficiency

We conclude by assessing the computational cost of training our models on different input representations, as summarized in table 4.2. The table reports the mean training time per epoch for the CNN-LSTM model across ADA, BTC, and ETH on a NVIDIA RTX 4070 GPU (3.2.5).

When using the full 40-dimensional LOB inputs, training times are asset-dependent, where ADA completes an epoch in approximately 53 minutes, while BTC and ETH require 75-76 minutes, reflecting higher update frequencies and data volumes. Switching to the 10-dimensional OFI summary (2.5) reduces per epoch training times by 20-25%, to 43 minutes for ADA and 60 minutes for BTC and ETH. Notably, this efficiency gain comes at the cost of only about a 0.5 percentage point decrease in peak R^2_{OOS} (see figure 4.1).

These results highlight a trade off in that richer LOB features improve peak predictive accuracy but increase training time, while the OFI inputs offer a substantial gain in computational efficiency with only a modest reduction in forecast performance.

Symbol	Features	Time per Epoch
ADA/USDT	OFI-only	43 min 03 s
ADA/USDT	LOB-only	52 min 57 s
BTC/USDT	OFI-only	59 min 51 s
BTC/USDT	LOB-only	1 h 15 min 37 s
ETH/USDT	OFI-only	59 min 19 s
ETH/USDT	LOB-only	1 h 16 min 21 s

Table 4.2: CNN-LSTM training time per epoch on a NVIDIA RTX 4070.

5 Discussion & Conclusions

This thesis has investigated the effectiveness of deep learning models for predicting short-term mid-price log-returns in cryptocurrency markets, comparing their performance against a classical linear baseline, and examining how the use of stationary feature transformations influences predictive power. Through out-of-sample experiments across several coins, time periods, and both raw and engineered input features, we are able to draw several clear conclusions.

First, we find a strong difference between deep learning models and classical linear models. The ARX model (2.8), which is commonly used in finance for its simplicity and transparency, does not show any meaningful predictive power at high-frequency. For all assets and for both the raw limit order book and the order flow imbalance features, the ARX model gives out-of-sample R^2 (3.4) values that are basically zero. This is not just a small underperformance. Instead, it suggests that linear models do not have what is needed to capture useful information from electronic order books at this scale. The constantly changing nature of order flow, combined with sudden changes in market conditions, means that linear relationships do not last long enough for these models to exploit. Even when more features and lags are added, the ARX model does not improve, and does not do better than simply predicting the average. This suggests that any predictability that does exist is either used up almost immediately by other market participants or is hidden in complex, nonlinear patterns.

Deep learning models, on the other hand, are able to find small but clear signals in the same data. Both convolutional networks (2.3.2) and recurrent models (2.3.2), and especially the hybrid CNN-LSTM model (2.3.2), manage to achieve positive out-of-sample R^2 across all the forecast horizons and coins that were tested. These improvements, although modest in size, show up again and again in different test periods and across different market conditions. The hybrid CNN-LSTM is the most effective overall. Its structure allows it to learn from both the local details in each order book snapshot (using convolution) and from the patterns over time in order flow (using recurrence). CNN models are good at picking up spatial patterns in the book, such as clusters of orders or sudden changes in liquidity, while LSTM models are better at tracking how these patterns develop over time. When combined, these two approaches give the model a better chance of picking up on the small regularities in the data.

Another result is that deep learning models do not only have higher R^2, they also do a better job of getting the direction of price changes right. This is measured by sign accuracy, which shows how often the model correctly predicts whether the price will go up or down. All deep learning models achieve sign accuracy well above chance, especially at the shortest horizons. The CNN-LSTM model stands out in this respect, reaching the highest sign accuracy and again demonstrating the benefit of combining spatial and temporal learning. Even small improvements in correctly predicting direction are meaningful in high-frequency trading, where profits are based on accumulating many small statistical edges.

A part of this study was to look at how feature engineering changes the results, especially when using a stationary transformation like the order flow imbalance (OFI) (2.5) instead of the full limit order book. When we compare models trained on the full 40-dimensional book state to those trained on the 10-dimensional OFI summary, we see a clear trade off. The OFI summary, while being simpler and lower dimensional, keeps most of the useful information for predicting very short-term returns. Models using OFI lose only a small amount of explained variance and sign accuracy compared to those using the full book, especially at the shortest horizons. This shows that the main predictive signal in these markets is often contained in the net flow of buy and sell orders, and that deep learning models can work well even when the input features are engineered summaries. However, there is still a small gap in performance, suggesting that some subtle patterns in the full book are lost when switching to OFI. The CNN-LSTM model continues to be the strongest even with the transformed features, showing that the advantages of deep learning hold up regardless of the input representation.

It is also important to note the practical side of these findings. Training deep models on high dimensional book data is demanding in terms of computation and memory, especially for assets with more trading activity. Switching to OFI features not only simplifies the model but also makes training faster, reducing the time per training epoch by about a quarter, with only a slight drop in performance. This is an important trade off for real trading systems, where speed and efficiency matter as much as raw predictive accuracy.

Looking more broadly, the results show that predicting high-frequency returns in cryptocurrency markets is a difficult task with limited scope for improvement. The failure of linear models, even with many features, shows that these markets are complex and change quickly. Deep learning models can find some weak signals that simpler models miss, but even the best results are only modest improvements. This suggests that while these markets are not perfectly efficient, any predictable patterns are hard to find and require advanced methods to exploit. At the same time, the fact that deep models are able to get even a small edge, where simpler models fail, shows the value of sophisticated pattern recognition in modern algorithmic trading.

For practioners, these findings give some guidance. Choosing the right model and input features should balance predictive performance with the costs and practicalities of building, maintaining, and running these models in real time. Deep learning models offer some advantage, but this comes with higher complexity and computational requirements. At the same time, much of the predictive signal may be preserved using engineered summaries like OFI, which are easier to interpret and train on.

Conclusively, deep learning models are effective at extracting weak but useful predictive signals from high-frequency cryptocurrency order book data, and they clearly outperform classical linear approaches in this setting. Transforming the inputs to stationary, low dimensional summaries like OFI keeps most of this predictive edge, though a small amount is lost. These results show that modern machine learning tools can find subtle regularities in market microstructure, but also highlight the challenges posed by market efficiency and the practical limits of high-frequency prediction.

5.1 Limitations & Future Work

One important limitation is the choice of dataset. The analysis is restricted to a three month window and includes only three of the most liquid cryptocurrency pairs. This focus allows for high data quality and controls for extreme changes in market structure, but it means that the findings may not apply to less liquid markets, different exchanges, or to periods marked by higher volatility and structural breaks. Expanding the analysis to cover a wider range of assets, longer historical periods, and more varied market conditions would help to determine how robust the results are and whether deep learning methods can generalize to other settings.

Another limitation concerns the information available in the dataset. The models in this study use only the limit order book states and do not include historical trade data. Trades, especially large or aggressive ones, may carry information about order flow and short-term direction that is not fully reflected in the order book alone. Incorporating detailed trade data, such as trade sizes and directions, could add further predictive power to the models and reveal patterns not captured by order book features alone.

The selection of input features and model architectures is also a limitation. While this study compares the full limit order book and a summary imbalance signal, there are many other ways to engineer features that might capture useful information, such as features based on queue dynamics, or liquidity measures at various depths. Further research could explore more targeted feature engineering to extract specific microstructure signals or to better represent latent supply and demand imbalances.

The range of tested models is limited to convolutional, recurrent, and hybrid CNN-LSTM architectures with standard design choices and a fixed set of hyperparameters. Recent developments in attention mechanisms, transformer architectures, and model ensembles, may be better suited to capturing long range dependencies and nonlinear interactions present in high-frequency market data [43]. Exploring deeper or alternative network architectures, experimenting with attention-based models, or using ensemble techniques could provide additional gains in predictive accuracy.

In addition, while this study measures predictive performance using out-of-sample

R^2 and sign accuracy, it does not connect these results directly to realized trading performance. In practice, a profitable trading strategy must account for transaction costs and market impact, which can erode statistical edges. Incorporating predictive models into a more realistic trading and backtesting framework, and evaluating them on economic performance measures such as profit and loss or Sharpe ratio, would be a valuable next step.

Finally, the study uses fixed hyperparameter settings and a single training regime. Market conditions change over time, and model parameters that work well in one period may not be optimal later. Further work could look at adaptive methods for model selection and tuning, as well as techniques for online learning and continual model updating in response to changing market regimes.

Bibliography

[1] Hayden Adams, Noah Zinsmeister, and Dan Robinson. Uniswap v2 core. Technical whitepaper, Uniswap, March 2020.

[2] Yacine Aït-Sahalia, Jianqing Fan, Lirong Xue, and Yifeng Zhou. How and when are high-frequency stock returns predictable? Technical Report 30366, National Bureau of Economic Research, August 2022.

[3] Jakob Albers, Mihai Cucuringu, Sam Howison, and Alexander Y. Shestopaloff. The good, the bad, and latency: Exploratory trading on bybit and binance, November 2024. Available at SSRN: https://ssrn.com/abstract=4677989.

[4] Irene Aldridge. *High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems.* Wiley Trading Series. John Wiley & Sons, Hoboken, New Jersey, 2 edition, 2013.

[5] Anthony Alexander. Latency arbitrage in cryptocurrency markets: Analyzing execution speeds & liquidity dynamics, February 2025. Available at SSRN: https://ssrn.com/abstract=5143158.

[6] J. Almeida and T. C. Gonçalves. Cryptocurrency market microstructure: a systematic literature review. *Annals of Operations Research*, 332:1035–1068, 2024. Received: 06 August 2022; Accepted: 25 September 2023; Published: 27 October 2023; Issue Date: January 2024.

[7] Emmanuel Bacry, Iacopo Mastromatteo, and Jean-François Muzy. Hawkes processes in finance. *Market Microstructure and Liquidity*, 1(1), 2015.

[8] Andrea Barbon and Angelo Ranaldo. On the quality of cryptocurrency markets: Centralized versus decentralized exchanges. Swiss finance institute research paper no. 22-38, University of St. Gallen, School of Finance, March 2024. Forthcoming in University of St. Gallen, School of Finance Research Paper Series.

[9] Ekkehart Boehmer, Charles M. Jones, Xiaoyan Zhang, and Xinran Zhang. Tracking retail investor activity. *The Journal of Finance*, 76(5):2249–2305, 2021.

[10] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986.

[11] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[12] Jean-Philippe Bouchaud, Julius Bonart, Jonathan Donier, and Martin Gould. *Trades, Quotes and Prices: Financial Markets Under the Microscope*. Cambridge University Press, New York, 2018. Includes bibliographical references and index.

[13] George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley, 5 edition, 2015.

[14] Qing Cao, Karyl B. Leggio, and Marc J. Schniederjans. A comparison between Fama and French's model and artificial neural networks in predicting the Chinese stock market. *Computers & Operations Research*, 32(10):2499–2512, 2005. Applications of Neural Networks.

[15] Agostino Capponi and Ruizhe Jia. Liquidity provision on blockchain-based decentralized exchanges, January 2021. Forthcoming in *Review of Financial Studies*; Available at SSRN: https://ssrn.com/abstract=3805095.

[16] Álvaro Cartea, Sebastian Jaimungal, and José Penalva. *Algorithmic and High-Frequency Trading*. Cambridge University Press, Cambridge, United Kingdom, 2015. Includes bibliographical references and index.

[17] Inc. Coinbase Global. Shareholder letter: Q1 2024. Exhibit 99.1 to Form 8-K, SEC Accession No. 0001679788-24-000087, May 2024.

[18] Rama Cont, Sasha Stoikov, and Rishi Talreja. A stochastic model for order book dynamics, September 2008. Available at SSRN: https://ssrn.com/abstract=1273160.

[19] Rama Cont, Sergey Stoikov, and Rishi Talreja. A stochastic model for order book dynamics. *Operations Research*, 58(3):549–563, 2010.

[20] David Easley, Maureen O'Hara, Songshan Yang, and Zhibai Zhang. Microstructure and market dynamics in crypto markets, April 2024. Available at SSRN: https://ssrn.com/abstract=4814346.

[21] Robert F. Engle. Autoregressive conditional heteroskedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982.

[22] Robert F. Engle and John R. Russell. Autoregressive conditional duration: A new model for irregularly spaced transaction data. *Econometrica*, 66(5):1127–1162, 1998.

[23] Ethereum Foundation. Blocks. https://ethereum.org/en/developers/docs/blocks/, 2025. Accessed: 2025-05-14.

[24] Etherscan. Ethereum average block time chart. https://etherscan.io/chart/blocktime, 2025. Accessed: 2025-05-14.

[25] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence*

and Statistics, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, May 2010. PMLR.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[27] 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.

[28] Klaus Grobys, Josephine Dufitinema, Niranjan Sapkota, and James W. Kolari. What's the expected loss when Bitcoin is under cyberattack? A fractal process analysis. *Journal of International Financial Markets, Institutions and Money*, 77:101534, 2022.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[30] Terrence Hendershott and Ryan Riordan. Algorithmic trading and the market for liquidity. *Journal of Financial and Quantitative Analysis*, 48(4):1001–1024, 2013.

[31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[32] Serkan Imisiker and Bedri Kamil Onur Tas. Wash trades as a stock market manipulation tool. *Journal of Behavioral and Experimental Finance*, 20:92–98, 2018. First posted to SSRN on March 7, 2016; Available at SSRN: https://ssrn.com/abstract=2476874.

[33] Invesco Asia Pacific. Bitcoin and Portfolio Choice: An Assessment of Bitcoin in Multi-Asset Portfolios. Technical report, Invesco Asia Pacific, September 2024.

[34] Heike Joebges, Hansjörg Herr, and Christian Kellermann. Crypto assets as a threat to financial market stability. *Eurasian Economic Review*, 2025.

[35] Kaiko Research Team. How is crypto liquidity fragmentation impacting markets?, August 2024.

[36] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[37] Shimon Kogan, Igor Makarov, Marina Niessner, and Antoinette Schoar. Are cryptos different? Evidence from retail trading. *Journal of Financial Economics*, 159:103897, 2024.

[38] Petter N. Kolm, Jeremy Turiel, and Nicholas Westray. Deep order flow imbalance: Extracting alpha at multiple horizons from the limit order book. *Mathematical Finance*, 33(4):1044–1081, 2023.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105, 2012.

[40] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[41] Jason Liu and Apostolos Serletis. Volatility in the cryptocurrency market. *Open Economies Review*, 30:779–811, August 2019.

[42] Igor Makarov and Antoinette Schoar. Trading and arbitrage in cryptocurrency markets. *Journal of Financial Economics*, 135(2):293–319, 2020.

[43] Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA; London, England, 2022.

[44] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, Haifa, Israel, 2010.

[45] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[46] John Nickolls. Scalable parallel programming with cuda: Introduction. In *Proceedings of the 20th IEEE Hot Chips Symposium (HCS)*, pages 1–9. IEEE, 2008.

[47] Fabian Schär. Decentralized finance: On blockchain- and smart contract-based financial markets, March 2020. Available at SSRN: https://ssrn.com/abstract=3571335.

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

[49] Lukas Sparer and Alexander Neulinger. Protecting human investors with blockchain-based circuit breakers. In *Proceedings of the 6th International Conference on Blockchain Computing and Applications (BCCA)*, pages 592–596, Dubai, United Arab Emirates, November 2024. IEEE.

[50] 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.

[51] Lu Zhang and Lei Hua. Major issues in high-frequency financial data analysis: A survey of solutions, December 2024. Available at SSRN: https://ssrn.com/abstract=4834362.

[52] 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.

[53] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *42nd IEEE Symposium on Security and Privacy (SP)*, pages 428–445, San Francisco, CA, USA, May 2021. IEEE.

A

GPU Acceleration

Modern graphics-processing units (GPUs) accelerate neural network training by executing many identical arithmetic operations in parallel [39]. While a conventional CPU offers a small number of powerful cores that operate mostly sequentially, a GPU deploys thousands of simpler cores under a single-instruction-multiple-thread (SIMT) scheme: each core runs the same instruction on different fragments of data [46]. This design is well suited to deep learning, where both forward and backward passes are dominated by large, regular matrix computations.

During the forward pass, the m input vectors in a mini-batch are concatenated into a matrix

$$\mathbf{X} \in \mathbb{R}^{F\times m}.$$

For a dense layer with weight matrix $\mathbf{W} \in \mathbb{R}^{d\times F}$ and bias vector $\mathbf{b} \in \mathbb{R}^{d}$, the affine transformation produces

$$A = \mathbf{W}\mathbf{X} + \mathbf{b}\mathbf{1}_{1\times m},$$

where each column of $A \in \mathbb{R}^{d\times m}$ contains the linear combination of inputs and bias for one sample in the batch. The activation function ϕ is then applied elementwise

$$\phi(A)_{ij} = \phi\Big(A_{ij}\Big) \quad \forall i, j,$$

with each GPU thread handling one scalar independently, further exploiting parallelism [39].

Backpropagation (2.14) computes the gradient of the loss with respect to each parameter by reversing the forward computation. By the chain rule, this amounts to further matrix multiplications and elementwise products that mirror the forward pass. All such computations are handled by highly optimized routines in vendor libraries such as cuBLAS and cuDNN, with intermediate tensors remaining in GPU memory and avoiding slow transfers to and from the CPU [39].

Parameter updates are also performed directly on the GPU. For example, the Adam optimizer maintains two auxiliary tensors m_t and v_t for each parameter vector θ.

Given the stochastic gradient g_t, the update equations are

$$\begin{aligned} m_t &= \beta_1\, m_{t-1} + (1-\beta_1)\, g_t, \\ v_t &= \beta_2\, v_{t-1} + (1-\beta_2)\, g_t^{\odot 2}, \\ \theta_{t+1} &= \theta_t \;-\; \eta_t\, \frac{m_t/(1-\beta_1^t)}{\sqrt{v_t/(1-\beta_2^t)} \;+\; \varepsilon}, \end{aligned}$$

where $\beta_1, \beta_2 \in (0,1)$ control the decay of the first and second-moment estimates, η_t is the learning rate, and ε prevents division by zero. All these updates are performed in parallel across parameters, with efficient use of GPU memory.

Because every stage of the forward pass, backward pass, and parameter update leverages the GPU's parallelism, training on a single commodity GPU can reduce wall-clock time per epoch by one to two orders of magnitude compared to a multi-core CPU, even though the underlying optimization algorithm is unchanged [39].

B

Structure of Processed Data

All model inputs are constructed from a processed tabular dataset in which each row represents a single limit order book update, that is, a market event or message, and each column encodes a specific feature or target value. The full dataset includes both meta data,such as instrument identifier and event timestamp, engineered features, and all possible target columns, but only the relevant subset is supplied as input to each model.

For each book update, the dataset records the following.

- ***Meta-data columns:*** Instrument identifier and event timestamp, retained for reference and evaluation but not used as model inputs.
- ***Raw limit order book features:*** For each of the first ten levels on both bid and ask sides, price and volume are stored as individual columns, interleaved by level and side. This becomes: bid price, bid volume, ask price, ask volume, repeated for $i = 1, \dots, 10$.
- ***Engineered features:*** Additional columns such as mid-price, bid-ask spread, bid/ask order flow, and order flow imbalance (OFI) (2.5) at each level.
- ***Target columns:*** Ten columns containing the buffered log-return for each forecast horizon.

Depending on the experiment, either the raw limit order book features or the OFI features are selected as the model input. The organization of these input matrices is illustrated in table B.1.

b_1	$v_1^{(b)}$	a_1	$v_1^{(a)}$	$\cdots$	b_{10}	$v_{10}^{(b)}$	a_{10}	$v_{10}^{(a)}$
–	–	–	–	…	–	–	–	–
–	–	–	–	…	–	–	–	–

Table B.1: Structure of the input matrix for the raw limit order book features. Each row is a market event, and columns are grouped as price and volume for each bid b, and ask a level $i = 1, \dots, 10$ (see section 2.2).

For experiments using the engineered OFI feature set, only the columns representing order flow imbalance at each level are used as input, as shown in table B.2.

OFI_1	OFI_2	OFI_3	OFI_4	OFI_5	OFI_6	OFI_7	OFI_8	OFI_9	OFI_{10}
–	–	–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–	–	–

Table B.2: Structure of the input matrix for order flow imbalance (OFI) features. Each row is a market event wher columns are the OFI at each depth level $i = 1, \ldots, 10$ as defined in (2.5).

The feature columns for the raw book input are specifically arranged to interleave price and volume by level and side. For each price level, the bid price is immediately followed by bid volume, then the corresponding ask price and ask volume. This design helps the model, especially for convolutional neural networks, to scan for local patterns and interactions at each level of the book and across the bid and ask sides. For models trained on OFI features, only the ten columns OFI_1 through OFI_{10} are included, which summarize net order flow at each level.

This structure is chosen to expose local book dynamics and interactions between bids and asks directly to the network, maximizing the likelihood that the model will learn microstructural patterns relevant for return prediction. During training, only the relevant subset of features, raw LOB or OFI, is selected as input. All meta data, engineered features, and targets remain in the dataset for bookkeeping, post processing, and evaluation, but are excluded from the model input.