International Journal of Forecasting 40 (2024) 1587–1621

Contents lists available at ScienceDirect

International Journal of Forecasting

journal homepage: www.elsevier.com/locate/ijforecast

The short-term predictability of returns in order book markets: A deep learning perspective

Lorenzo Lucchese [a,*], Mikko S. Pakkanen [a,b], Almut E.D. Veraart [a]

[a] Department of Mathematics, Imperial College London, 180 Queen's Gate, London SW7 2AZ, United Kingdom
[b] Department of Statistics and Actuarial Science, University of Waterloo, 200 University Avenue, West, Waterloo N2L 3G1, Canada

ARTICLE INFO

Article history:
Dataset link: www.lobsterdata.com, www.github.com/lorenzolucchese/deepOBs, 10.5281/zenodo.10665431

Keywords:
Price forecasting
Order book
High-frequency trading
Deep learning
Neural networks
Comparative studies
Model selection
Model confidence sets
Financial markets

ABSTRACT

This paper uses deep learning techniques to conduct a systematic large-scale analysis of order book-driven predictability in high-frequency returns. First, we introduce a new and robust representation of the order book, the volume representation. Next, we conduct an extensive empirical experiment to address various questions regarding predictability. We investigate if and how far ahead there is predictability, the importance of a robust data representation, the advantages of multi-horizon modeling, and the presence of universal trading patterns. We use model confidence sets, which provide a formalized statistical inference framework well suited to answer these questions. Our findings show that at high frequencies, predictability in mid-price returns is not just present but ubiquitous. The performance of the deep learning models is strongly dependent on the choice of order book representation, and in this respect, the volume representation appears to have multiple practical advantages.

© 2024 The Authors. Published by Elsevier B.V. on behalf of International Institute of Forecasters. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

1.1. Financial markets and exchanges: the rise of High-Frequency traders

A financial market is an ensemble of market agents willing to buy or sell a certain financial security, such as a stock, bond, or derivative. Today, most trades occur on electronic exchanges, virtual places that bring buyers and sellers together, facilitating the occurrence of transactions. At the time of writing, the two largest U.S. equity exchanges by market capitalization are the NYSE, a hybrid (floor and electronic) auction market accounting for 20% of the U.S. equities market transactions, and the Nasdaq, a fully electronic dealer market executing about 16% of such trades (source: Cboe Exchange, Inc.). Both markets allow traders to access live order book information, i.e., the collection of all standing orders for a given security. Theoretically, this allows for symmetric information across traders, who should all have access to the same data regarding market depth, liquidity, and price discovery dynamics.

In practice, traders have access to different technologies, receiving market data and submitting orders at different latencies. Almost two decades ago, a new market participant emerged in the quest to exploit the advantages gained by faster access to markets. These market players are today known as High-Frequency Traders (HFTs) and, over the years, have rapidly grown to represent a significant share of the market (Abergel, Lehalle, & Rosenbaum, 2014). Their role has since been the object of a fierce debate: their critics claim HFTs engage in predatory – and sometimes illegal – behavior, while their supporters believe HFTs benefit the market by providing liquidity, reducing spreads, and helping price discovery dynamics. In this paper, we will not delve into questions regarding the legitimacy of HFT practices, but we will

* Corresponding author.
E-mail address: lorenzo.lucchese17@imperial.ac.uk (L. Lucchese).

https://doi.org/10.1016/j.ijforecast.2024.02.001
0169-2070/© 2024 The Authors. Published by Elsevier B.V. on behalf of International Institute of Forecasters. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

instead aim to independently analyze the value of immediate access to order book information. Over the past few decades, HFT companies have engaged in a fierce race to zero latency, making vast economic efforts to reduce their latency by just a few microseconds. What we aim to explore in this research is one of the possible reasons why such a race happened in the first place. Specifically, we will be analyzing the predictive value of order book data, i.e., to what extent can a trader with immediate access to the order book predict the market's future direction?

It is also important to note that, over the past couple of decades, the trading process has become increasingly complex due to market fragmentation, availability of new technologies such as smart order routers (SORs), and – sometimes controversial – regulation, e.g., Reg-NMS (US Securities and Exchange Commission, 2005) and MiFID (European Parliament and Council, 2004). While practices that exploit arbitrage between competing trading venues exist, in this research, we will assume the trader has access to a single electronic order book-based exchange, namely the Nasdaq.

1.2. Order book predictability: asking the right questions

As discussed in the opening paragraph, we would like to explore whether, contrary to low-frequency returns, ultra-high-frequency returns tend to display predictability. Empirical studies (Sirignano & Cont, 2019) have shown that price formation dynamics, i.e., next mid-price moves, are predictable. This paper will try to understand whether such predictability persists at longer horizons. Intuitively, predictability in high-frequency returns may be understood to arise simply from the way an order book market is structured, i.e., the side of the order book with less liquidity is more likely to erode faster, resulting in a price increase/decrease, or as the product of recurring trading patterns in response to liquidity information. The approaches considered in the literature for forecasting high-frequency returns from order book data can be roughly divided into two categories: relatively simple models built on carefully handcrafted features (Aït-Sahalia, Fan, Xue, & Zhou, 2022) and more sophisticated architectures applied directly to raw order book data (Kolm, Turiel, & Westray, 2021; Zhang & Zohren, 2021; Zhang, Zohren, & Roberts, 2019). In this research, we will focus on the latter class of models, leveraging the ability of deep learning techniques to learn complex dependence structures. We will consider a specific class of deep learning models, introduced by Zhang et al. (2019), designed to extract features from order book data. Empirical evidence (Bengio, Courville, & Vincent, 2013) suggests that, although deep learning models can extract complex features, the way data is represented may significantly impact model performance. Hence, we will explore how model performance varies when changing how the order book data is arranged. Equipped with this class of models, we will aim to investigate the following questions, which naturally arise from our preceding discussion:

1. Do high-frequency returns display order book-driven predictability? If so, how far ahead can we predict?
2. Which order book representations perform best?
3. Can we use a single model across multiple horizons?
4. Can we use a single model across multiple stocks?

We aim to answer all these questions in a formalized statistical inference framework based on model confidence sets (Hansen, Lunde, & Nason, 2011).

1.3. Related work and contributions

It is important to note that mathematical modeling of order book dynamics is a very broad and active area of research, ranging from Hawkes process models for order and trade events in continuous time (Bacry & Muzy, 2014; Large, 2007) to discrete-time synthetic data-driven order book generation (Byrd, Hybinette, & Balch, 2020; Coletta, Moulin, Vyetrenko, & Balch, 2022). The work presented in this paper can be seen as contributing to two parallel research streams in the literature for order book-driven mid-price predictions. On the one hand, we expand on the deep learning ideas discussed in Kolm et al. (2021), Zhang and Zohren (2021), Zhang et al. (2019) by introducing new data representations and carrying out a disciplined comparison between the specifications. On the other hand, we explore a set of questions related to short-term price predictability in a similar spirit to Aït-Sahalia et al. (2022) but under a different class of models.

In the broader context of the questions addressed in this paper, related works are those of Sirignano and Cont (2019), exploring the universality of order book dynamics, and Wu, Mahfouz, Magazzeni, and Veloso (2021), advocating for robust representations of order books. We base our experimental procedure on model confidence sets, introduced by Hansen et al. (2011). We believe this formalized statistical inference framework perfectly suits our aim of addressing questions that require comparisons between multiple models and benchmarks.

The two main contributions of this paper are summarized as follows. First, we introduce a deep learning model for mid-price forecasting based on a more robust representation of the order book, which we will refer to as deepVOL. This representation allows us to easily adapt the model to the setting where more granular L3 data[1] is available. Second, we provide new empirical results addressing essential questions regarding short-term price predictability in a disciplined experimental framework.

2. The space of models under consideration

2.1. The order book

At a given point in time, an order book contains all the (visible) buy and sell orders placed for a given security on a specific exchange. The lowest ask price, resp. the highest bid price, is known as the first ask order book level, resp. the first bid order book level. Subsequent levels are defined accordingly. An example of a 10-level

[1] See Section 2.1 for the definitions of L1, L2, and L3 order book data.

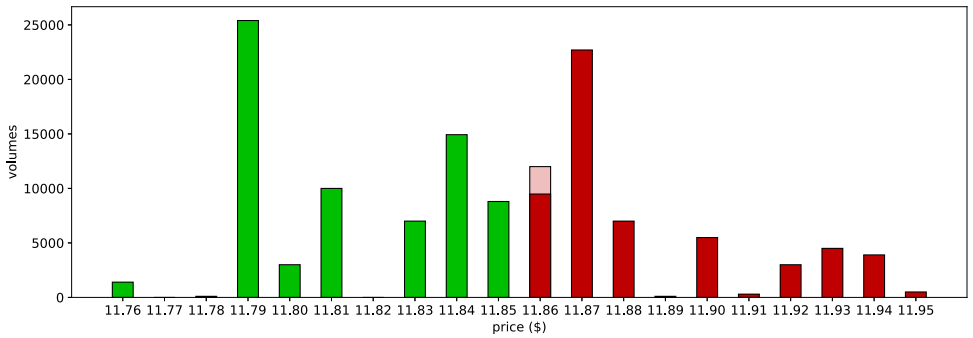


Fig. 1. A sample snapshot of an order book. Ask (resp. bid) volumes are denoted by red (resp. green) bars. The lighter red shaded area represents the change in the order book shape when some liquidity at the best ask price is removed. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

order book snapshot is displayed in Fig. 1. Orders can be submitted on an evenly spaced discrete set of prices in electronic exchanges, known as ticks. The smallest price increment is known as the tick size; for most U.S. traded stocks, it corresponds to $0.01. Not all tick prices may have standing orders; therefore, the first ten-level ask/bid prices may not coincide with the first ten ask/bid tick prices.

There are three main types of actions that traders can request on an exchange:

- a limit order: an order to buy a given quantity of the security at a given price;
- a market order: an order to buy a given quantity of the security at the best available price;
- a cancellation or partial deletion: an order to fully or partially delete a standing limit order.

While market orders are always immediately executed once posted, passive limit orders, i.e., orders that do not cross the spread, will sit in the order book until they are matched. A limit order entering the market at a tick price where other limit orders are already present will be added at the end of the standing queue. Trades occur each time a market or aggressive limit order is posted; the requested volume matches the standing limit orders according to price-time priority. We will not delve into the detailed characteristics of all the different order types, but it suffices to point out that the three actions described above represent the fundamental drivers of the evolution of order books. For example, in Fig. 1, the change in order book shape may be due to either a (partial) deletion of a first-level buy limit order or the execution of a sell order (either a market order or an aggressive limit order).

Electronic exchanges operate continuously during trading hours, so order books are sometimes called continuous books—exchanges such as the Nasdaq time stamp each event at nanosecond precision. The trading activity level may vary significantly between different stocks, and the time elapsed between consecutive events may differ by orders of magnitude. For this reason, we define an alternative time clock: a discrete order book clock, which increments by one each time an action is executed on the order book. Throughout our discussion, we will explore questions of predictability with respect to this order book clock, which is the same as the one considered in Ntakaris, Magris, Kanniainen, Gabbouj, and Iosifidis (2018). Some authors, for example, Aït-Sahalia et al. (2022), consider alternative order book-driven time clocks, such as transaction and volume clocks.

We understand that models based on order book-specific clocks might be challenging to use in practical trading applications. However, we believe order book-based clocks provide a more natural measure of time for exploring predictability and are more suitable for comparing results across stocks than physical time: a 100 ms time horizon has a significantly different meaning for stocks with different levels of trading activity.

Another important observation is that not all market participants might access order book data at the same level of granularity. The Nasdaq Quotation Dissemination Service makes the following distinctions:

- L1 data: the best bid and ask prices and corresponding volumes;
- L2 data: all available bid and ask prices and corresponding volumes;
- L3 data: all available bid and ask prices and corresponding volumes split among the orders in the queue.

We will compare models' performance when different data granularity levels are available. Fig. 2 visually compares L1, L2, and L3 data.

We introduce the following notation: at time $t \in \mathbb{Z}$ we denote by

- $p^{(l)}_{x,t}$, $v^{(l)}_{x,t}$ for $l = 1, 2, \ldots$ the lth level ask/bid price and volume, for $x \in \{a, b\}$;
- $m_t = \dfrac{p^{(1)}_{b,t} + p^{(1)}_{a,t}}{2}$ the mid-price;
- $\pi^{(j)}_{x,t}, s^{(j)}_{x,t}$ for $j = 1, 2, \ldots$ the jth ask/bid tick price from the mid and corresponding volume for $x \in \{a, b\}$, i.e., this is defined recursively as

$$\pi^{(1)}_{x,t} = \begin{cases} m_t & \text{if } m_t \in \vartheta\mathbb{N}, \\ m_t \pm \frac{\vartheta}{2} & \text{if } m_t \notin \vartheta\mathbb{N} \end{cases}$$

$$\pi^{(j)}_{x,t} = \pi^{(j-1)}_{x,t} \pm \vartheta \quad \text{for } j = 2, 3, \ldots,$$

where ϑ is the tick size, $\vartheta\mathbb{N} = \{k\vartheta : k \in \mathbb{N}\}$ is the set of possible tick prices and $\pm$ depends on $x \in \{a, b\}$.

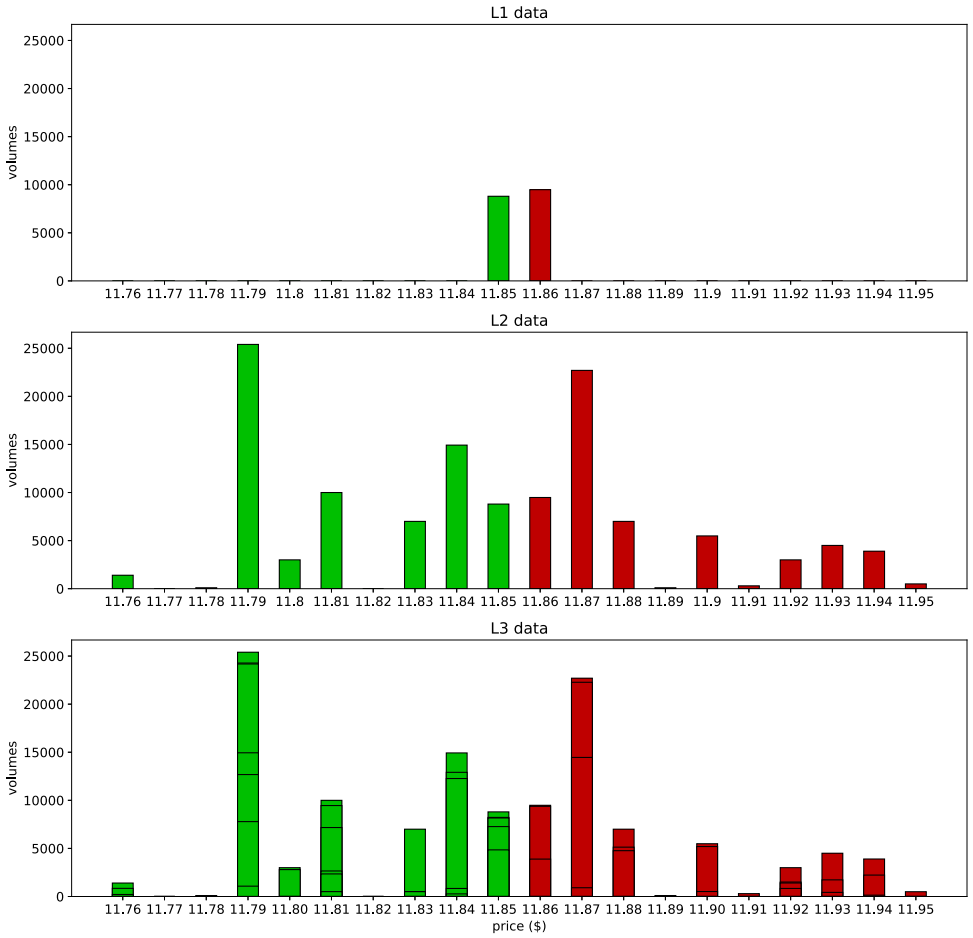


Fig. 2. L1, L2 and L3 representations of the order book in Fig. 1.

- $q_{x,t}^{(i,k)}$ for $k = 1, 2, \ldots$ the queue corresponding to volume $s_{x,t}^{(i)}$ and ordered by time priority.

2.2. Predictability of returns

We will explore models that aim to identify short-term predictability in returns. We first introduce the familiar regression framework for return predictions. We then rephrase the task in terms of a classification problem, extending the definition of predictability to this setting.

We will be exploring predictability arising from past order book data, and thus, we define the information σ-algebra $\mathcal{F}_t$ to be

$$\mathcal{F}_t = \sigma(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}),$$

where $\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}$ are order book derived features at times $t, \ldots, t-T+1$ for some look-back window of length T.

Predictions in the regression framework. Let us first consider the regression setting. At time t we denote by $r_{t,t+h} \in \mathbb{R}$ the h-step ahead mid-price return, as defined in 3.2.2. Given order book information at time t, $\mathcal{F}_t$, there exists a measurable function g such that

$$\mathbb{E}[r_{t,t+h} \mid \mathcal{F}_t] = g(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}),$$

or, equivalently,

$$r_{t,t+h} = g(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}) + \epsilon_t,$$

for some mean-zero noise variable ϵ_t orthogonal to the space of $\mathcal{F}_t$-measurable random variables. A prediction is defined to be any $\mathcal{F}_t$-measurable random variable and the best prediction is the $\mathcal{F}_t$-measurable random variable $r_{t,t+h}^{(t)}$ which minimizes the expected cost

$$\mathbb{E}[C(r_{t,t+h}^{(t)}, r_{t,t+h})],$$

for some appropriate cost function $C : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. In the case of quadratic cost $C(r_1, r_2) = (r_1 - r_2)^2$, we have

$$r_{t,t+h}^{(t)} = \mathbb{E}[r_{t,t+h} \mid \mathcal{F}_t] = g(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}).$$

Different choices of the cost function are possible; for example, when C is absolute cost, i.e., $C(r_1, r_2) = |r_1 - r_2|$, the best prediction is given by the conditional median of $r_{t,t+h}$ given $\mathcal{F}_t$. Given a parametric family of models $\{g_\theta(\cdot)\}_{\theta \in \Theta}$ and an observed training data set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}, r_{t,t+h})\}_{t \in \mathcal{I}_{\text{train}}}$ one can first learn a function $g_{\hat{\theta}}$ approximating g and then produce the return predictions

$$\hat{r}_{t,t+h}^{(t)} = g_{\hat{\theta}}(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}),$$

for test data points $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_t, \dots, \mathbf{x}_{t-T+1}, r_{t,t+h})\}_{t\in\mathcal{I}_{\text{test}}}$. Assuming returns to be stationary, we say that there is order book-driven predictability if the learned predictions outperform an *unpredictive benchmark* prediction on the testing set with respect to the chosen cost function $C(\cdot,\cdot)$.

Predictions in the classification framework. In this paper, we will discretize the space of returns by grouping mid-price movements as downward, no-change, and upward.[2] We hence introduce the discretized return random variable

$$c_{t,t+h} = \begin{cases} \downarrow & \text{if } r_{t,t+h} \in (-\infty, -\gamma), \\ = & \text{if } r_{t,t+h} \in [-\gamma, +\gamma], \\ \uparrow & \text{if } r_{t,t+h} \in (+\gamma, +\infty), \end{cases}$$

for an appropriate choice of $\gamma > 0$, cf. Section 3.2.2. Instead of modeling only the expected conditional return, in the classification setting, one aims to approximate the whole conditional distribution, i.e., find measurable functions $p_\downarrow, p_=, p_\uparrow$ such that

$$\mathbb{P}(c_{t,t+h} = *|\mathcal{F}_t) = p_*(\mathbf{x}_t, \dots, \mathbf{x}_{t-T+1}),$$

for $* \in \{\downarrow, =, \uparrow\}$. The discretized return prediction for an unobserved sample is then given by the minimizer of the expected misclassification cost:

$$\hat{c}_{t,t+h} = \underset{*\in\{\downarrow,=,\uparrow\}}{\operatorname{argmin}} \sum_{\star\in\{\downarrow,=,\uparrow\}} c_{*|\star} p_{\star,\hat{\theta}}(\mathbf{x}_t, \dots, \mathbf{x}_{t-T+1}),$$

where the parametric conditional probabilities $p_{\downarrow,\hat{\theta}}, p_{=,\hat{\theta}}, p_{\uparrow,\hat{\theta}}$ are learnt from a training data set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_t, \dots, \mathbf{x}_{t-T+1}, c_{t,t+h})\}_{t\in\mathcal{I}_{\text{train}}}$ and $c_{*|\star}$ is the misclassification cost of a $*$ observation classified as a $\star$. Assuming the return process is stationary, we say there is order book-driven predictability if the learned predictions outperform an *unpredictive benchmark* prediction on a test set regarding total misclassification cost.

Specifying an appropriate cost function is task-dependent. For example, a trader using the predictions as trading signals might be more interested in the correctness of up and down predictions than no-change ones. On the other hand, a market maker might prioritize the correctness of no-change predictions when using these to decide whether to tighten quoted spreads. In both cases, the consequences of different types of errors are asymmetric and heavily impact the choice of the cost matrix $C = \{c_{*|\star}\}_{*,\star\in\{\downarrow,=,\uparrow\}}$. In the classification framework, an alternative approach is to compare the predicted conditional distributions, $\{p_*(\mathbf{x}_t, \dots, \mathbf{x}_{t-T+1})\}_{*\in\{\downarrow,=,\uparrow\}}$ to the realized outcomes directly. This approach does not require specifying a cost matrix C nor a return prediction. The predicted conditional distribution (the output of our model) is directly compared with the observed data via a suitable "distance" on the space of probability measures $\mathcal{P}(\{\downarrow, =, \uparrow\})$, where realized returns are encoded as Dirac measures. A natural choice of such "distance" is given by the categorical cross-entropy, as this can be interpreted (under appropriate assumptions) as the negative log-likelihood of the test set:

$$\begin{aligned} &-\log \mathbb{P}(\mathcal{D}_{\text{test}} \mid \theta) \\ &\quad\propto -\frac{1}{|\mathcal{I}_{\text{test}}|} \sum_{t\in\mathcal{I}_{\text{test}}} \sum_{*\in\{\downarrow,=,\uparrow\}} \mathbb{1}_{\{c_{t,t+h}=*\}} \log p_{*,\hat{\theta}}(\mathbf{x}_t, \dots, \mathbf{x}_{t-T+1}), \end{aligned}$$

where the model parameters $\hat{\theta}$ are learnt from the training set $\mathcal{D}_{\text{train}}$. As this paper does not target a specific trading strategy but is more concerned with general predictability questions, we will evaluate our models based on the categorical cross-entropy loss. In the following, we will thus say that there is order book-driven predictability if the learned conditional distributions outperform an *unpredictive benchmark* distribution on a test set relative to categorical cross-entropy loss. As discussed in Appendix A.1, the natural choice for the loss used in training will also be categorical cross-entropy.

Note that in both the regression and classification framework, we compare the learned predictions with those obtained from an *unpredictive benchmark* model to determine whether there is predictability by using a score/cost function. Simply comparing point estimates of the scores does not provide a sound statistical argument answering the question of whether there is predictability: the difference in score may simply be due to statistical variability. This is where the model confidence set (Hansen et al., 2011) procedure comes in. As discussed in Section 4, this provides a statistical testing framework to determine whether the learned models statistically outperform the *unpredictive benchmark*, i.e., whether there is predictability.

Remark 2.1. The definition of predictability is intrinsically tied to that of *unpredictive benchmark*. Different choices for *unpredictive benchmark* models are possible; for example, in the regression framework, a natural choice is given by (a version of) the efficient market hypothesis (EMH) (Fama, 1970). In this case, the unpredictive hypothesis assumes the conditional expected return to be 0. Under the classification framework, the simple EMH does not translate into a clear-cut model for the conditional distribution of returns. In this setting, we consider a natural unpredictive hypothesis a slightly stronger version of the EMH: under the *unpredictive benchmark*, returns are assumed to be IID and independent of any information up to time t. Therefore, the empirical distribution of the training set will give the *unpredictive benchmark* conditional distribution. As a side remark, we believe it is important to note the EMH was initially proposed in a very different market environment and at significantly higher latencies than the ones considered in this paper. Nevertheless, we believe the EMH provides a natural unpredictive benchmark hypothesis when testing for predictability.

As mentioned in Section 1.2, there are two main approaches considered in the literature for exploring the short-term predictability of returns in order book

[2] We believe that, for high-frequency returns, a classification framework is more natural: mid-prices move in (half)-tick increments, and hence the corresponding returns live on a discrete lattice (see Section 3.2.2). Moreover, grouping returns in three classes (downward, no-change, and upward) significantly reduces the effects of idiosyncratic noise while preserving the ability to investigate questions regarding predictability.

markets: the first considers carefully handcrafted features $\mathbf{x}_t, \ldots, \mathbf{x}_{t-T}$ and relatively simple specifications for the prediction functions g or $(p_\downarrow, p_=, p_\uparrow)$, e.g., linear specifications or decision trees, the second – which we will explore in this paper – uses raw order book features $\mathbf{x}_t, \ldots, \mathbf{x}_{t-T}$ as inputs to more complex prediction functions, e.g., deep neural network architectures. We assume the reader is familiar with deep learning techniques and thus give a brief overview of the relevant concepts in Appendix A.1; more detailed expositions can be found in Goodfellow, Bengio, and Courville (2016).

2.3. Deep learning models for short-term return predictions in order book markets

In the previous section, we set up the learning framework for the return prediction task. We now discuss how the neural networks covered in Appendix A.1 can be combined to model price formation mechanisms and predict h-step ahead high-frequency returns. We will consider a specific class of deep learning models based on the deepLOB architecture (Zhang et al., 2019).

2.3.1. deepLOB, (Zhang et al., 2019)

This network acts on raw order book input—a CNN module and an inception module feed into an LSTM layer, which produces the final classification output. The CNN and inception module aim to extract short-term spatio-temporal features in the data, while the LSTM module works on longer-term dependencies.

The deepLOB architecture is summarized in Fig. 3 and is made up of the following components:

- **Input**. The first L level raw order book information - price and volume - with a look-back window of length T is used as input. The input at time t is thus a $(T \times 4L)$ array given by

$$\{\mathbf{x}_{t-\tau}\}_{\tau=0}^{T-1} = \left\{ \left(p_{a,t-\tau}^{(l)}, v_{a,t-\tau}^{(l)}, p_{b,t-\tau}^{(l)}, v_{b,t-\tau}^{(l)} \right)_{l=1}^{L} \right\}_{\tau=0}^{T-1} \in \mathbb{R}^{T\times 4L}.$$

  A feature-wise rolling window z-score standardization is applied to the input, i.e., $v_{a,t-\tau}^{(1)}$ is standardized using the mean and standard deviation of the first level ask volumes over the previous five days.
- **CNN module**. Convolutions are applied to the data in both the spatial and temporal dimensions. The spatial convolutions aim to aggregate information across order book levels, and the temporal convolutions can be understood as smoothing operations. The CNN module is summarized in Fig. 4.
- **Inception module**. This module up-samples the convoluted data by applying various temporal convolutions with different filter lengths (time-window). Each temporal convolution can be interpreted as a (weighted) moving average. It is similar in spirit to the computation of technical indicators, but the frequencies at which this is applied are substantially different.
- **LSTM**. The Long Short-Term Memory layer takes the multidimensional time series produced by the inception module and feeds it through a recurrent network structure to extract longer-term dependencies among the data. The last hidden state of the LSTM is passed through a dense layer with a softmax activation function to produce the h-step ahead return prediction $\downarrow, =$ or $\uparrow$.

The exact details of the deepLOB architecture can be found in Table 13.

2.3.2. deepOF, (Kolm et al., 2021)

In the deepLOB model, order book states, which are non-stationary, are mapped to a stationary quantity, returns. While in theory, this shouldn't be a problem (due to the universality property of deep neural networks), Kolm et al. (2021) argue that using some form of stationary input might improve model performance by facilitating the training procedure.

A stationary order book quantity is order flow. First-level order flow describes the net flow of orders at the best bid and ask. This was introduced in Cont, Kukanov, and Stoikov (2013) to explore the price impact of order book events. This single quantity parsimoniously models the instantaneous effect of order book events on prices. The first-level bid and ask order flows corresponding to the order book event occurring at time t (in order book time) are defined by:

$$bOF_t^{(1)} = \begin{cases} v_{b,t}^{(1)} & \text{if } p_{b,t}^{(1)} > p_{b,t-1}^{(1)}, \\ v_{b,t}^{(1)} - v_{b,t-1}^{(1)} & \text{if } p_{b,t}^{(1)} = p_{b,t-1}^{(1)}, \\ -v_{b,t-1}^{(1)} & \text{if } p_{b,t}^{(1)} < p_{b,t-1}^{(1)}, \end{cases}$$

$$aOF_t^{(1)} = \begin{cases} -v_{a,t-1}^{(1)} & \text{if } p_{a,t}^{(1)} > p_{a,t-1}^{(1)}, \\ v_{a,t}^{(1)} - v_{a,t-1}^{(1)} & \text{if } p_{a,t}^{(1)} = p_{a,t-1}^{(1)}, \\ v_{b,t}^{(1)} & \text{if } p_{a,t}^{(1)} < p_{a,t-1}^{(1)}. \end{cases}$$

The difference between the two is known as order flow imbalance:

$$OFI_t^{(1)} = bOF_t^{(1)} - aOF_t^{(1)}.$$

Bid order flow corresponds to the net change in volume at the best bid level; the three cases in the definition can be understood as:

- a new bid order being placed at a higher price than the current best bid;
- the order volume at the best bid price increasing or decreasing;
- the entire volume at the best bid price being consumed, thus decreasing the best bid price.

As discussed in Section 2.1, volumes can increase or decrease due to orders being placed, executed, or canceled. A similar interpretation holds for ask order flow.

In Xu, Gould, and Howison (2019), the authors explore flows of volumes at deeper levels in the order book by introducing multi-level order flow. The definitions are exactly as above, with superscript (1) replaced by general (l). We note that in both Cont et al. (2013) and Xu et al. (2019), the authors investigate the explanatory power of (multi-level) order flow imbalance for price changes, i.e.,

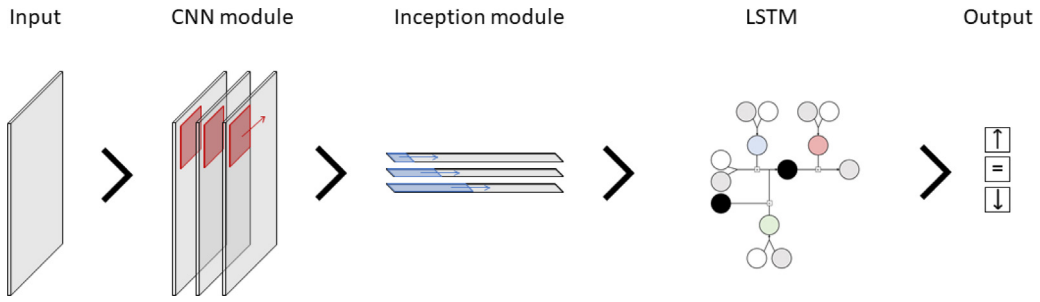


Fig. 3. Core modules of network architectures.

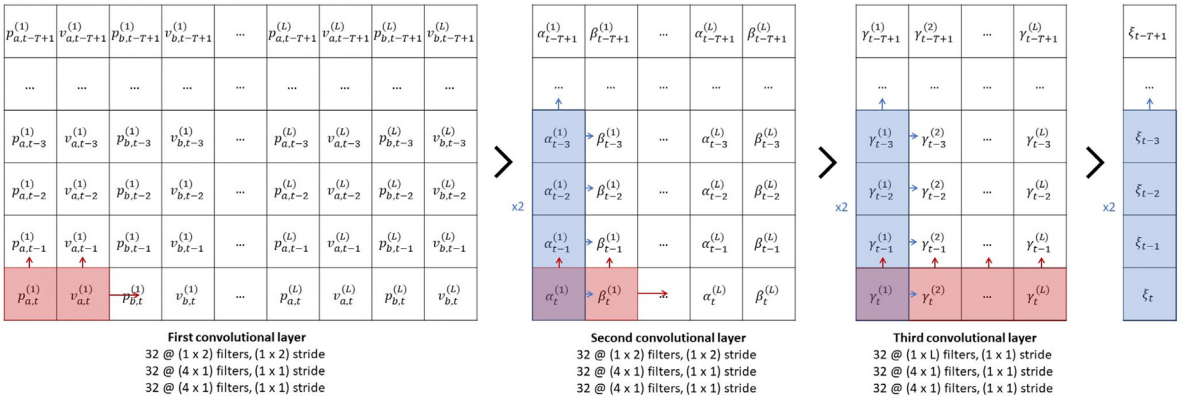


Fig. 4. CNN module deepLOB.

the relationship between contemporaneous order flow imbalance and price changes.

In our setting, as in Kolm et al. (2021), we are instead interested in exploring the predictive power of order flow, i.e., the relationship between past order flow and future price changes. We will hence refer to deepOF as the deepLOB architecture with stationary order flow input, i.e., a $(T \times 2L)$ array given by

$$\{\mathbf{x}_{t-\tau}\}_{\tau=0}^{T-1} = \left\{ \left(aOF_{t-\tau}^{(l)}, bOF_{t-\tau}^{(l)} \right)_{l=1}^{L} \right\}_{\tau=0}^{T-1} \in \mathbb{R}^{T \times 2L}.$$

The order flow input enters the CNN module in Fig. 4 at the second convolutional layer. The rest of the architecture is the same as deepLOB, as detailed in Table 13. We note that the first convolutions applied to the order flow input, i.e., the second convolutional layer in Fig. 4, aggregate information across bid and ask order flows, essentially computing a weighted order flow imbalance.

Remark 2.2. The original deepOF specification in Kolm et al. (2021) was structured as a (multi-horizon) regression task. In their setting, the last layer of the deep neural network maps to $\mathbb{R}$ instead of $\mathcal{P}(\{\downarrow, =, \uparrow\})$. Moreover, as discussed in Section 2.5, the way multi-horizon predictions are produced in the original work does not rely on the encoder–decoder structure we use for our multi-horizon models. Another slight difference with the experiments in Kolm et al. (2021) is in the standardization procedure: instead of standardizing each feature by its training mean and standard deviation, we use a rolling window approach.

2.4. deepVOL

2.4.1. The need for a robust representation

As discussed in the previous section, the main difference between deepLOB and deepOF is how the data is fed into the model. While deepLOB uses raw order book data as input, deepOF uses a derived quantity and order flow. In general, the success of deep learning tasks depends on how the data is represented. The task of predicting returns in order book markets is no exception. One should adopt a robust data representation to achieve the best possible results. Wu et al. (2021) identify five main desiderata for a robust representation of order book data:

- Region of interest: the entire order book may contain a wide range of prices. The data representation should select a region of interest based on a complexity-performance trade-off.
- Efficiency: the data representation should avoid excessive dimensionality.
- Validity: the data representation should have a simple definition of valid manipulations.
- Smoothness: the data representation should be robust to small perturbations.
- Compatibility: the data representation should be compatible with the deep learning architecture.

We note the order book representations used in deepLOB and deepOF do not conform to these desiderata. Order book states organized by 'level' do not have a simple validity (price and volume information are intrinsically tangled and would lose their significance if treated separately in a black box algorithm), are not robust to small perturbation (small orders added at empty ticks completely change the order book feature vector), and are incompatible with deepLOB's CNN module (the spatial structure is not homogeneous as there is no fixed interval between levels). It turns out that while the 'level' representation of the order book may be easily understandable by humans, it is less so for statistical models. In addition to not satisfying the desiderata, this representation does not respect the following fundamental but implicit assumption of deep learning models: signals at the same input entry should come from the same source. In the 'level' representation, as new order book events happen, the same signal (i.e., a posted order) may move between levels.

Therefore, we introduce volume features, which robustly represent order book data. Fixing a window of size $W > 0$ we define:

$$\mathbf{x}_t = \left(s^{(W)}_{b,t}, \dots, s^{(1)}_{b,t}, s^{(1)}_{a,t}, \dots, s^{(W)}_{a,t}\right) \in \mathbb{R}^{2W},$$

where $s^{(j)}_{x,t}$ for $x \in \{a, b\}$ are the bid/ask volumes corresponding to the jth price from the mid $\pi^{(j)}_{x,t}$, as defined in Section 2.1. We note that the volume representation indeed satisfies the five desiderata: a region of interest is identified (via the window $W > 0$), it is efficient (for the same dimension of input it may convey more or less information than the 'level' representation, depending on how sparse the orders are placed in the order book), it has a simple validity (all entries are in the same units), it is robust to small perturbations (new orders at empty levels minimally affect the feature) and is compatible with the CNN architecture (the spatial structure of the volumes is homogeneous). An intuitive visualization of this representation as a one-dimensional gray-scale strip is given in Fig. 6; when including a time dimension, this naturally becomes a two-dimensional gray-scale image. The main drawback of the volume representation is that it is sparse when orders are placed far apart in the order book, and a larger window $W > 0$ may be required. Our definition of volume features is similar to the mid-price-centered moving window representation of Wu et al. (2021), with the latter living in $\mathbb{R}^{2W+1}$ instead of $\mathbb{R}^{2W}$ and using $\pm$ signs to distinguish between bid and ask volumes. The need for a new, more robust order book representation was reached independently.

We must adapt the CNN module to use the deepLOB architecture with this new data representation. In particular, we fold the volume input into a three-dimensional array

$$\left\{s^{(j)}_{x,t-\tau}\right\}_{\tau\in\mathcal{T},\, j\in\mathcal{W},\, x\in\{a,b\}} \in \mathbb{R}^{T\times W\times 2},$$

where $\mathcal{T} = \{0, \dots, T-1\}$ and $\mathcal{W} = \{1, \dots, W\}$, and feed this into a three-dimensional convolutional layer with a $(1 \times 2 \times 2)$ filter and $(1 \times 1 \times 1)$ stride. This layer aims to extract imbalances in the order book by comparing volumes on the two sides of the mid-price. Fig. 7 depicts the CNN module with the appropriate changes. The rest of the deepVOL architecture is the same as deepLOB, with one slight difference in how the data is normalized. Thinking of the volume representation as a gray-scale image, a natural choice of normalization is

$$s^{(j)}_{x,t-\tau} \mapsto \frac{s^{(j)}_{x,t-\tau}}{\max\limits_{\tau'\in\mathcal{T},\, j'\in\mathcal{W},\, x'\in\{a,b\}} s^{(j')}_{x',t-\tau'}},$$

for $\tau \in \mathcal{T}$, $j \in \mathcal{W}$ and $x \in \{a, b\}$, instead of the rolling window standardization applied to deepLOB and deepOF features.

Remark 2.3. One could define a volume flow quantity based on the distance from the middle; in the same way, order flow describes the flow of orders based on the level. Following the same motivation for considering order flow in deepOF presented in Section 2.3.2, one could consider using a volume flow quantity as input to the deep learning architectures with the desired robust representation properties.

2.4.2. L3 volume features

All models considered, deepLOB, deepOF, and deepVOL, use L2 data as input. If one has access to more granular data, i.e., L3 data breaking down each volume queue into single orders, one might be interested in leveraging this information to obtain higher predictive performance. We thus define a natural extension of the volume representation considered in the previous section.

Let us denote by $(q^{(j,1)}_{x,t}, q^{(j,2)}_{x,t}, \dots) \in \mathbb{R}^{\mathbb{N}}$ for $x \in \{a, b\}$ the queue at the jth bid/ask price from the mid $\pi^{(j)}_{x,t}$ as introduced in Section 2.1. Here, $q^{(j,k)}_{x,t}$ denotes the volume of the kth order in the queue sorted by time priority and is set to zero if there is no such order. The aggregated volume at $\pi^{(j)}_{x,t}$ is given by:

$$s^{(j)}_{x,t} = \sum_{k\geq 1} q^{(j,k)}_{x,t}.$$

A natural extension of the volume representation would, therefore, be to consider

$$\left\{\left(q^{(j,1)}_{x,t-\tau}, q^{(j,2)}_{x,t-\tau}, \dots\right)\right\}_{\tau\in\mathcal{T},\, j\in\mathcal{W},\, x\in\{a,b\}} \in \mathbb{R}^{T\times W\times 2\times\mathbb{N}}.$$

Unfortunately, this infinite-dimensional array cannot be directly fed into machine learning models. We, therefore, cut off the queue at a given depth level. To avoid discarding precious information, we aggregate all orders past the maximum depth level at the end of the queue. For a given depth level $D > 0$, we thus consider the L3 volume feature

$$\left\{\left(q^{(j,1)}_{x,t-\tau}, \dots, q^{(j,D-1)}_{x,t-\tau}, \sum_{k\geq D} q^{(j,k)}_{x,t-\tau}\right)\right\}_{\tau\in\mathcal{T},\, j\in\mathcal{W},\, x\in\{a,b\}} \in \mathbb{R}^{T\times W\times 2\times D}.$$

Remark 2.4. A similar approach to the one used for cutting off the queue might be applied to aggregate volumes sitting deeper in the order book when deriving L2 representations. This would give a better idea of the total liquidity in the order book.

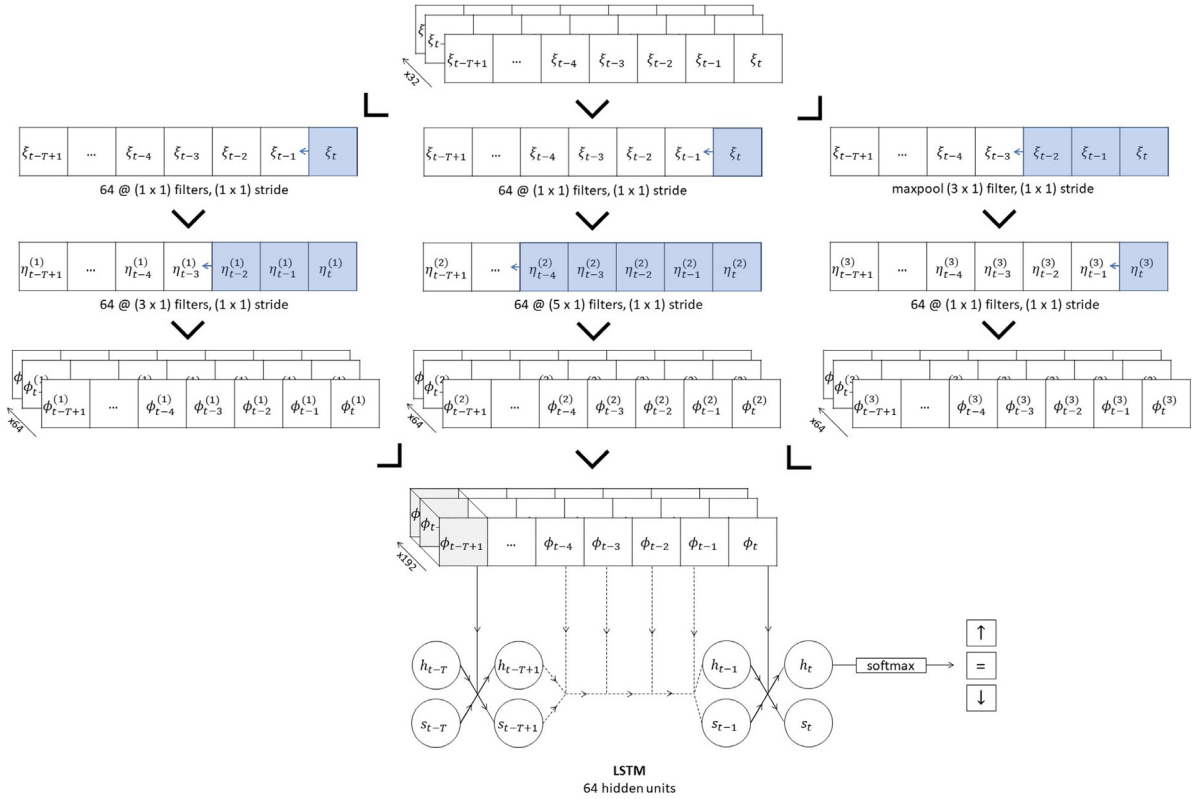


Fig. 5. Inception module and LSTM layer.

1400	0	100	25400	3000	10000	0	7000	14930	8800	12000	22700	7000	100	5490	300	3000	4500	3900	500
1400	0	100	25400	3000	10000	0	7000	14930	8800	9484	22700	7000	100	5490	300	3000	4500	3900	500

Fig. 6. Gray-scale visualization of the volume representation of the order book in Fig. 1.

We add an initial convolutional layer to extract relevant information from the queue, which maps each queue to a weighted sum of the order sizes. The weighted aggregated volumes are then fed into the deepVOL architecture as above. Fig. 8 summarizes the resulting CNN module.

The full architectures for deepVOL and deepVOL L3 are detailed in Table 14.

2.5. Multi-horizon models

So far we considered single horizon models, i.e., order book input x_t is mapped to a three class distribution $(p_\downarrow, p_=, p_\uparrow)$ corresponding to the discretized return $c_{t,t+h}$ at fixed horizon h. In the following, we consider a generalization of the architectures considered thus far to the setting where the forecasting horizon is a vector $\mathbf{h} = (h_1, \ldots, h_K)$. In this case, the modeling task consists of predicting the distributions of the discretized returns $c_{t,t+\mathbf{h}} = (c_{t,t+h_1}, \ldots, c_{t,t+h_K})$.

The simplest way of adjusting the current models to the multi-horizon framework would be to replace the last softmax layer in Fig. 5 with K parallel dense softmax layers. The last hidden state of the LSTM module would hence be mapped to an array of size $3 \times K$, corresponding to K distributions over three classes. A similar architecture – though in the regression framework – is considered in Kolm et al. (2021). While this approach is perfectly valid, it does not use the task's sequential nature, potentially neglecting an important structural feature of the data.

In this paper, we will leverage architectures inspired by machine translation, which are naturally suited for sequential forecasting. Specifically, we will consider encoder–decoder models: an encoder maps the input data to a latent summary state (also known as context vector), and then a decoder rolls forward predictions sequentially. In this context let $z_{t-T+1}, \ldots, z_t$ denote the last T hidden states of an encoder at time t, then a decoder rolls forward

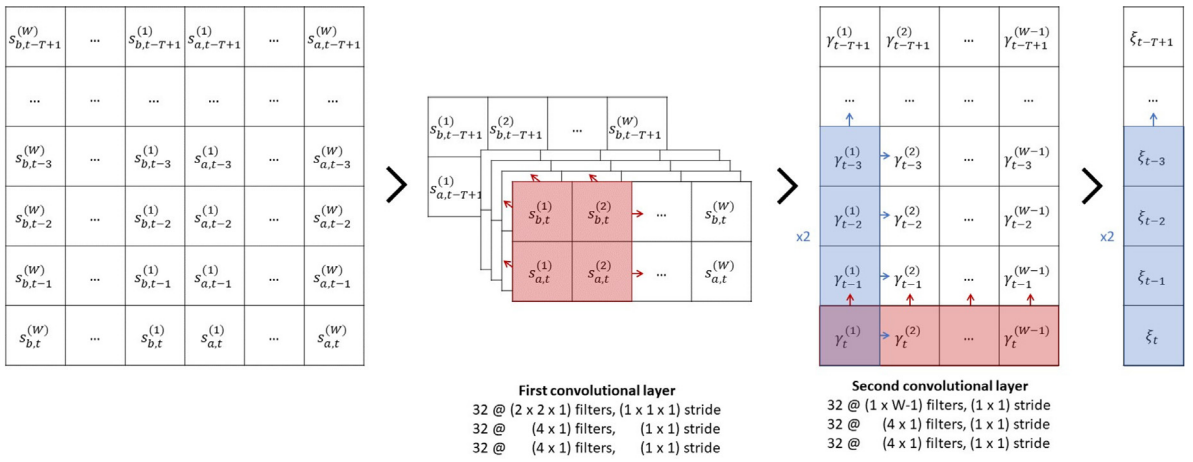


Fig. 7. CNN module deepVOL.

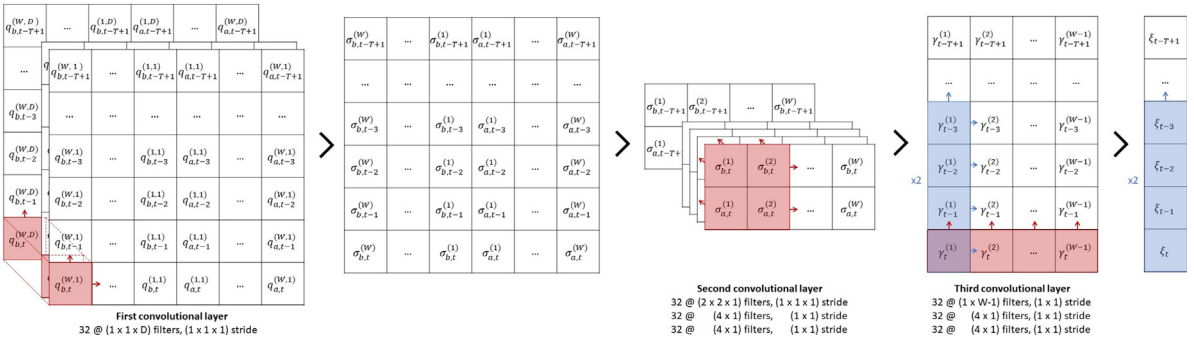


Fig. 8. CNN module deepVOL L3. Abusing notation slightly we denote $\sum_{k\geq D} q^{(j,k)}_{x,t-\tau}$ by $q^{(j,D)}_{x,t-\tau}$.

predictions by:

$$z'_0 = z_t,\ p_0 = \hat{p}_0,$$
$$c_k = h(z_{t-T+1}, \ldots, z_t),$$
$$z'_k = f([z'_{k-1}, c_k], p_{k-1}),\ \ p_k = g(z'_{k-1}, c_k),$$

for $k = 1, \ldots, K$. Here $h(\cdot)$ is a function acting on the hidden states of the encoder to extract the context vector c_k (this may depend on other inputs as well, such as previous hidden states of the decoder), $f(r, p)$ is a recurrent layer with recurrent input r and exogenous input p, $g(\cdot)$ is an output layer depending on both the decoder hidden state and the context. The general mechanism of such encoder–decoder architecture is visualized in Fig. 9.

Our experiments will consider a sequence-to-sequence decoder (Cho et al., 2014), the simplest example of such architecture. In this setting, we set $c_k \equiv z_t$, implicitly assuming that at every forecasting horizon, the last hidden state of the encoder summarizes all the relevant information required to make the prediction. More complex architectures exist; for example, Luong, Pham, and Manning (2015) introduce an attention-based decoder that uses a weighted combination of all the hidden states of the encoder as a context vector. In this setting, different weights are used at different forecasting horizons, selectively accessing hidden states of the encoder during decoding. While attention-based networks have been successfully applied for high-frequency mid-price predictions, cf. Zhang and Zohren (2021) and Tran, Iosifidis, Kanniainen, and Gabbouj (2019), in this work, we wish to exemplify the potential of multi-horizon models and thus restrict ourselves to simple seq2seq decoders only.

In the experiments, we set $f(\cdot)$ as an LSTM and $g(\cdot)$ as a dense layer with softmax activation. Moreover p_0 is initialized at $\hat{p}_0 = (0, 1, 0)$.

Using multi-horizon forecasting was first proposed for deepLOB in Zhang and Zohren (2021): the LSTM module of the deepLOB architecture discussed in Section 2.3.1 acts as an encoder, mapping order book states to the LSTM final latent vector $z_t = [h_t, s_t]$. A seq2seq decoder then rolls forward the prediction, producing distributional forecasts at horizons $\mathbf{h} = (h_1, \ldots, h_K)$. The LSTM module with seq2seq decoder is illustrated in Fig. 10. Nothing stops us from applying the same multi-horizon structure to the output of deepOF and deepVOL convolutional modules.

Full details of the multi-horizon architectures can be found in Tables 13 and 14.

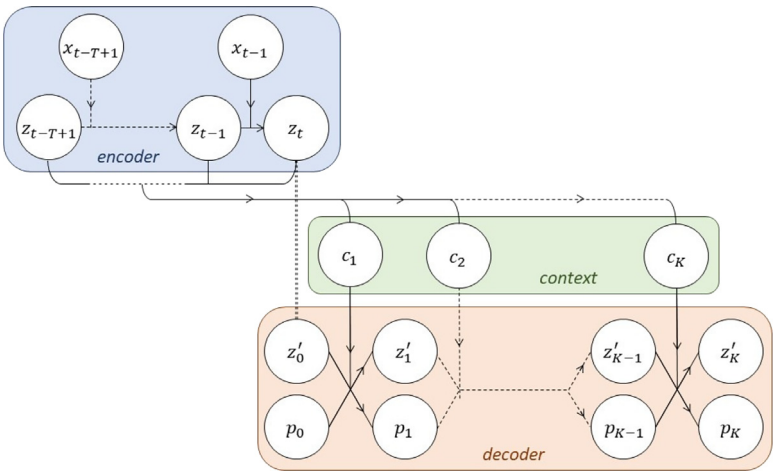


Fig. 9. A general encoder–decoder architecture.

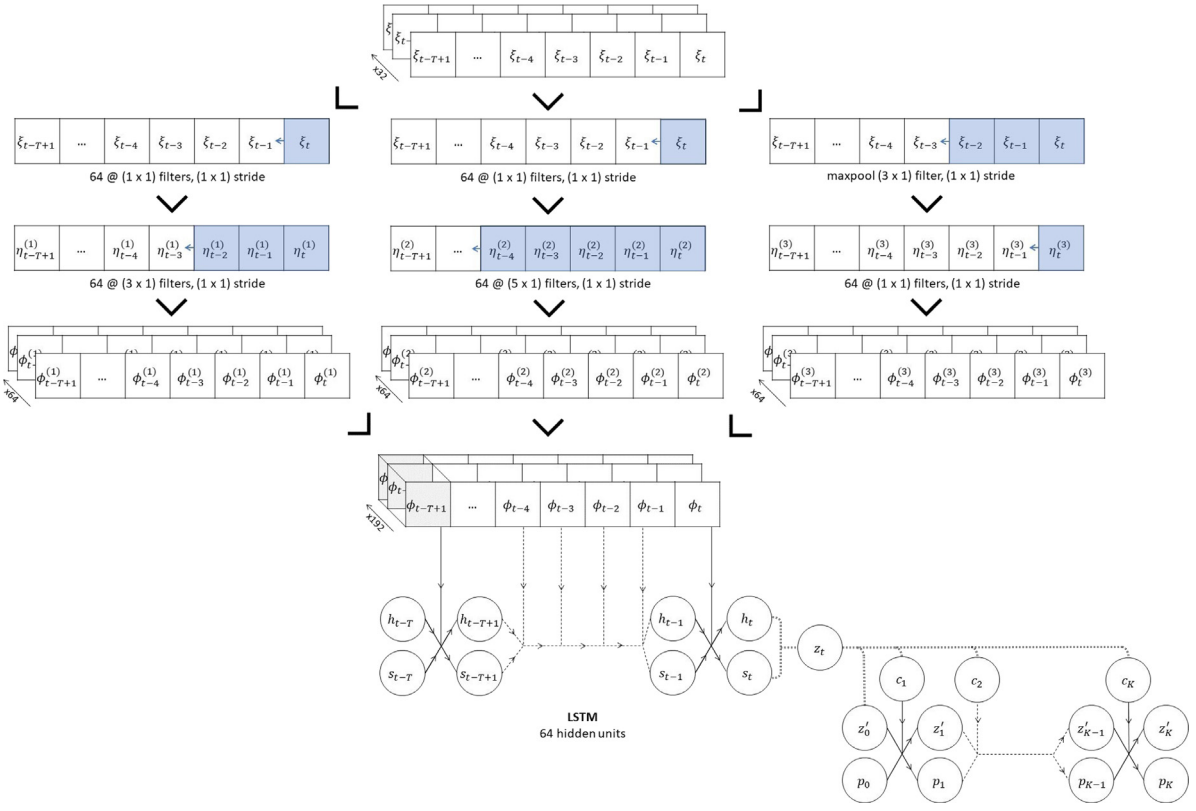


Fig. 10. Inception module and LSTM layer with seq2seq decoder for multi-horizon forecasting.

Remark 2.5. We adopt the same multi-horizon framework as in Zhang and Zohren (2021), where the response $c_{t,t+h_k}$ at horizon $h_k \in \{h_1, \ldots, h_K\}$ corresponds to the return from time t to time $t + h_k$. Alternatively, one might wish to consider as multi-horizon responses the subsequent incremental returns $c_{t+h_{k-1},t+h_k}$, i.e., the returns between time $t + h_{k-1}$ and time $t + h_k$. By choosing evenly spaced h_k's, one would obtain more consistent responses across prediction time steps $\{1, \ldots, K\}$ and possibly improve the performance of the multi-horizon models.

3. Data set

This section introduces the data set used in the experiments presented in Section 4. First, we briefly describe how LOBSTER order book data is related to the Nasdaq ITCH feed. Next, we detail how we process LOBSTER data

Table 1
Selected stocks' characteristics, daily averages.

Ticker	Updates (000)	Trades (000)	Price Changes (000)	Price (USD)	Spread (bps)	Volume (USD MM)
LILAK	49.89	1.81	3.77	18.09	15.92	2.63
QRTEA	121.32	5.23	2.79	13.54	9.75	9.53
XRAY	83.07	5.41	7.11	52.32	4.35	21.67
CHTR	80.97	6.75	19.21	400.68	6.22	111.71
PCAR	131.38	6.85	13.03	70.63	3.82	34.43
EXC	298.32	7.58	7.21	47.35	2.45	37.55
AAL	398.13	13.15	10.99	30.65	3.88	50.29
WBA	328.5	13.2	15.18	57.81	2.54	75.19
ATVI	423.01	17.97	20.25	49.87	2.81	91.82
AAPL	1137.36	64.19	127.86	215.8	0.99	1156.3

Table 2
Example message from a LOBSTER message file. This example corresponds to the order book dynamics shown in Fig. 1.

Event ID	Timestamp (sec)	Event Type	Order ID	Size (shares)	Price	Direction
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1312	34714.133632201	deletion	206833312	2516	$11.86	sell
⋮	⋮	⋮	⋮	⋮	⋮	⋮

to obtain features and responses for the experiments. We also provide some descriptive statistics of the data.

We consider the same universe of stocks and trading period as in Kolm et al. (2021): through LOBSTER (Huang & Polak, 2011), we access one year of open (9:30 EST) to close (16:00 EST) trading data for 115 Nasdaq tickers from January 2, 2019, to January 31, 2020. We select a subset of ten stocks to preserve a sufficiently varied set of liquidity characteristics to produce results in feasible computational time. Table 1 summarizes the ten tickers and their liquidity characteristics. For full details on how the ten stocks were selected, see Appendix A.8.

3.1. LOBSTER data, (Huang & Polak, 2011)

Section 2.1 described the basic mechanisms governing electronic exchanges such as the Nasdaq. Every day, trading activity on the Nasdaq alone results in hundreds of thousands of order book updates for each stock, cf. Table 1. To keep track of all events occurring on the exchange and communicate them efficiently to (subscribed) market participants, the Nasdaq uses the TotalView-ITCH protocol. For efficiency, instead of streaming the entire state of the order book after each update, only information on the event changing the order book is sent out to market participants. An example of a (decoded) ITCH message is reported in Table 2; note these are timestamped at nanosecond precision. It is up to each market participant to store the current state of the order book and update it each time a new message arrives.

The service LOBSTER provides to academic researchers is to reconstruct the historical order book from Nasdaq's Historical TotalView-ITCH data. For each selected ticker and date, LOBSTER returns a message and order book files subsampled at a given granularity. In practical terms, a 10-level data granularity yields the set of messages corresponding to order book updates in the first ten levels along with the corresponding reconstructed order book 'chopped at ten levels.' The evolution of the order book determined by the message in Table 2 is reported in Table 3. More information on the order book reconstruction algorithm used by LOBSTER can be found in Huang and Polak (2011) and on the website www.lobsterdata.com.

3.2. Processed data

From the historic LOBSTER order book data, we build the features $\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}$ and the target responses $c_{t,t+h}$. In the experiments in Section 4, we apply the learning framework of Section 2.2 with the deep learning models introduced in Section 2.3 to these feature-response pairs, using the model confidence set (MCS) procedure for model comparison. As previously discussed in Section 2.1, we will measure time t (and prediction horizons h) using an order book-driven clock, ticking every time an event occurs on the order book. This clock corresponds to the Event ID in Tables 2 and 3. Note we access LOBSTER data up to level ten, thus our order book clock is conditional on the updating event being in the first ten levels. By construction, the data contains all information on new limit orders, market orders, and cancellations restricted to the first ten levels. We apply some minor pre-processing steps to the LOBSTER data, summarized in Appendix A.3.

3.2.1. Features: order book, order flow and volume

We start by discussing how to derive the features $\mathbf{x}_t$ at each time point t from the raw LOBSTER data, i.e., from Tables 2 and 3. While building L1/L2 order books, order flow, and volume features is quite simple, reconstructing L3 volume data requires more work.

The raw order book input $\mathbf{x}_t$ used in deepLOB (Zhang et al., 2019) and described in Section 2.3.1 simply corresponds to the LOBSTER data in Table 3, i.e., at $t = 1312$

Table 3
Example order book update from a LOBSTER order book file. This example corresponds to the order book dynamics shown in Fig. 1.

Event ID	Timestamp (sec)	$p_a^{(1)}$	$v_a^{(1)}$	$p_b^{(1)}$	$v_b^{(1)}$	$\cdots$	$p_a^{(10)}$	$v_a^{(10)}$	$p_b^{(10)}$	$v_b^{(10)}$
⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮
1311	34713.685155243	\$11.86	12 000	\$11.85	8800	$\cdots$	\$11.95	500	\$11.73	5500
1312	34714.133632201	\$11.86	9484	\$11.85	8800	$\cdots$	\$11.95	500	\$11.73	5500
⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮

the 10-level order book feature is given by

$$\mathbf{x}_t = \left(p_{a,t}^{(1)}, v_{a,t}^{(1)}, p_{b,t}^{(1)}, v_{b,t}^{(1)}, \ldots, p_{b,t}^{(10)}, v_{b,t}^{(10)}\right) = \left(11.86, 9484, 11.85, 8800, \ldots, 11.73, 5500\right).$$

The features are standardized using a 5-day rolling window.

To compute the order flow input $\mathbf{x}_t$ used in deepOF (Kolm et al., 2021) we apply the equations given in Section 2.3.2 to the LOBSTER data in Table 3, i.e., at $t = 1312$ the 10-level order flow feature is given by

$$\mathbf{x}_t = \left(aOF_t^{(1)}, bOF_t^{(1)}, \ldots, bOF_t^{(10)}\right) = \left(-2516, 0, \ldots, 0\right).$$

Again, the features are standardized using a 5-day rolling window.

To construct the volume input $\mathbf{x}_t$ at L2 granularity for the deepVOL model described in Section 2.4, we select only the volume information from Table 3 adding in zeros corresponding to empty price ticks, i.e., at $t = 1312$ the volume feature with window size $W = 10$ is given by

$$\mathbf{x}_t = \left(s_{b,t}^{(10)}, \ldots, s_{b,t}^{(1)}, s_{a,t}^{(1)}, \ldots, s_{a,t}^{(10)}\right) = \left(1400, \ldots, 8800, 9484, \ldots, 500\right).$$

Note that, in this example, $s_{b,t}^{(10)} = 1400 \neq 5500 = v_{b,t}^{(10)}$ since some bid price ticks are empty, cf. Fig. 1. As discussed in Section 2.4, volume features are normalized using max-scaling over the whole input array $(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1})$.

Finally, to construct the L3 volume features, one needs to work with the message file in Table 2 to keep track of the queues. For example, the volume queue at the first ask price, i.e., $\pi_{a,t}^{(1)} = \$11.86$, given at time $t = 1311$ by

$$\left(q_{a,t}^{(1,1)}, q_{a,t}^{(1,2)}, q_{a,t}^{(1,3)}, q_{a,t}^{(1,4)}, q_{a,t}^{(1,5)}\right) = \left(2516, 2000, 1484, 4500, 1500\right),$$

is updated at time $t = 1312$ to

$$\left(q_{a,t}^{(1,1)}, q_{a,t}^{(1,2)}, q_{a,t}^{(1,3)}, q_{a,t}^{(1,4)}\right) = \left(2000, 1484, 4500, 1500\right),$$

by the deletion in Table 2. All other queues are left unchanged. For more details on the complexities of reconstructing volume features from LOBSTER data, see Appendix A.3.1. A deep dive into the distributions of the processed order book, order flow, and volume features is carried out in Appendix A.4.1.

3.2.2. Responses: categorical mid-price returns

In this paper, we are interested in answering questions regarding the predictability of market returns. Inevitably, how returns, i.e., the target responses, are defined profoundly affects this analysis. Here, in line with the related literature, we treat the mid-price as the "true" price and define returns relative to it, but it is important to note that, by definition, this is not a tradable price. We define the return at horizon h as

$$r_{t,t+h} = \frac{\overline{m}_{t+h}^{(k)} - m_t}{m_t}, \quad \text{where } \overline{m}_{t+h}^{(k)} = \frac{1}{2k+1}\sum_{i=-k}^{k} m_{t+h+i},$$

and m_t denotes the mid-price at time t and k is a fixed smoothing window. The mid-price m_t is computed from Table 3 by $\frac{1}{2}\left(p_{b,t}^{(1)} + p_{a,t}^{(1)}\right)$, i.e., at times $t = 1311$ and $t = 1312$ we have $m_t = \$11.855$. This definition is subject to two possible interpretations. Treating the smoothed mid-price as a de-noised estimate of the true (latent) price h steps ahead, we can understand the return as the percentage change of the true (latent) price relative to the current mid. Alternatively, the return can be understood as the average return one would experience by entering a position at the current mid and exiting it roughly h steps ahead (assuming mid-mid trading). For all the horizons $h \in \{10, 20, 30, 50, 100, 200, 300, 500, 1000\}$ which are considered in Section 4 we fix $k = 5$. In Appendix A.6, we discuss other methods for defining mid-price returns and their shortcomings.

Remark 3.1. When defining the returns, we assume immediate access to the order book. In practice though, hardware and software constraints lead to non-zero time lags when receiving messages and sending orders to the exchange. While, in our setting, such latencies have a negligible impact on the definition of returns, we discuss how their presence could be more precisely accounted for in Appendix A.7.

All our experiments are carried out in a classification framework where the discretized returns are defined as

$$c_{t,t+h} = \begin{cases} \downarrow & \text{if } r_{t,t+h} \in (-\infty, -\gamma), \\ = & \text{if } r_{t,t+h} \in [-\gamma, +\gamma], \\ \uparrow & \text{if } r_{t,t+h} \in (+\gamma, +\infty), \end{cases}$$

for some $\gamma > 0$. To make the three classes roughly symmetric and as balanced[3] as possible, we empirically choose γ from the training set $\mathcal{D}_{\text{train}}$ by

$$\hat{\gamma}_h = \frac{|\hat{Q}_h(0.33)| + \hat{Q}_h(0.66)}{2},$$

[3] Note that, especially for short horizons h, many of the returns $r_{t,t+h}$ can be exactly zero, leading to $\hat{Q}_h(0.33) = \hat{Q}_h(0.66) = 0$. When this happens the three classes are unbalanced.

where $\hat{Q}_h$ is the empirical quantile function of the training set returns $\{r_{t,t+h}\}_{t\in\mathcal{I}_{\text{train}}}$. As discussed in Section 4, we will be splitting our data $\mathcal{D}$ in disjoint windows $\mathcal{D}_w = \mathcal{D}_{w,\text{train}} \cup \mathcal{D}_{w,\text{val}} \cup \mathcal{D}_{w,\text{test}}$ for $w = 1, \ldots, W$. Therefore, the choice of γ will be window w and horizon h specific. Descriptive statistics of the target return labels for the first window $w = 1$ are reported in Appendix A.4.2.

We note that since stocks trade on a discrete grid of prices determined by the tick size[4] ϑ, also the mid-price m_t evolves on a discrete price (with steps of size $\vartheta/2$). One could thus define the dollar return from t to $t+h$ by the number of (half) ticks of the mid-price moves, i.e.,

$$R_{t,t+h} = m_{t+h} - m_t \in \{\ldots, -\vartheta, -\vartheta/2, 0, \vartheta/2, \vartheta, \ldots\},$$

This return is, by definition, discretized, and thus, one could directly apply classification models (grouping large negative and positive returns to obtain a finite number of classes). A similar approach is used in Sirignano and Cont (2019) when predicting the next change in mid-price. In our work, we consider the estimate for the "true" mid-price to be

$$\overline{m}^{(k)}_{t+h} = \frac{1}{2k+1}\sum_{i=-k}^{k} m_{t+h+i},$$

which lives on a much finer grid than m_{t+h}: over the, possibly quite long, time horizon h multiple changes to the mid-price might occur. In this case, the smallest change has little meaning and so we group the dollar returns

$$R_{t,t+h} = \overline{m}^{(k)}_{t+h} - m_t \in \left\{\ldots, -\frac{\vartheta}{4k+2}, 0, \frac{\vartheta}{4k+2}, \ldots\right\},$$

into larger classes given by

$$(-\infty, -m_t\gamma),\ [-m_t\gamma, +m_t\gamma],\ (+m_t\gamma, +\infty).$$

4. Experiments

In this section, we explore the four questions introduced in Section 1.2 and attempt to answer them via the statistical framework provided by model confidence sets (MCS). This inference procedure allows to compare a set of competing models $\mathcal{M}_0$ based on observed loss time series $\{L_{i,w}\}_{w=1}^{W}$ where $L_{i,w}$ denotes the loss of model $i \in \mathcal{M}_0$ at time $w \in W$. To apply the model confidence set procedure to the data described in Section 3, we divide the 55 weeks from January 14, 2019, to January 31, 2020 into $W = 11$ five-week periods,[5] as represented in Fig. 11. Each window of data $\mathcal{D}_w$ is divided into a training-validation set $\mathcal{D}_{w,\text{train}} \cup \mathcal{D}_{w,\text{val}}$, the first four weeks, and a test set $\mathcal{D}_{w,\text{test}}$, the fifth week. The joint training-validation set is then further split into training and validation sets by randomly selecting five days out of the four weeks for validation. First, the training dataset $\mathcal{D}_{w,\text{train}}$ is used to choose the γ threshold for defining the return labels, as detailed in Section 3.2.2 (note that the choice of γ is specific to the choice of window, horizon, and ticker). Then, the joint training-validation dataset $\mathcal{D}_{w,\text{train}} \cup \mathcal{D}_{w,\text{val}}$ is used to train the models. For the *unpredictive benchmark* model, this simply means determining the empirical distribution of returns in $\mathcal{D}_{w,\text{train}} \cup \mathcal{D}_{w,\text{val}}$, cf. Appendix A.4.2. For the deep neural network architectures, this amounts to finding the optimal parameters that minimize the training weighted cross-entropy loss. We use Adam optimization with validation-based early stopping as described in Appendix A.1. Once the model has been trained we compute the out-of-sample losses on $\mathcal{D}_{w,\text{test}}$, i.e., for period $w \in \{1, \ldots, 11\}$ and model $i \in \mathcal{M}_0$

$$\begin{aligned} L_{i,w} &= \text{cce}(\hat{\mathbf{p}}^i_{w,\text{test}}, \mathbf{c}_{w,\text{test}}) \\ &= -\frac{1}{|\mathcal{I}_{w,\text{test}}|}\sum_{t\in\mathcal{I}_{w,\text{test}}}\sum_{*\in\{\downarrow,=,\uparrow\}} \mathbb{1}_{\{c_{t,w,\text{test}}=*\}}\log\hat{p}^i_{*,t}, \end{aligned}$$

is the categorical cross-entropy loss corresponding to the estimated probabilities $\hat{\mathbf{p}}^i_{w,\text{test}}$ when compared with the observations $\mathbf{c}_{w,\text{test}}$. $\hat{\mathbf{p}}^i_{w,\text{test}}$ are the class probabilities produced by model i on the testing set $\mathcal{D}_{w,\text{test}}$ after being trained on the training-validation set $\mathcal{D}_{w,\text{train}} \cup \mathcal{D}_{w,\text{val}}$. The time series of test losses $\{L_{i,w}\}_{w=1}^{11}$ are then "fed through" the MCS procedure described in Appendix A.2 to obtain the set of MCS p-values $\{p_i^{\text{MCS}}\}_{i\in\mathcal{M}_0}$. The intuitive interpretation of these p-values is: if model $i \in \mathcal{M}_0$ has an MCS p-value lower than a prescribed confidence level, then it is deemed statistically inferior to other models in $\mathcal{M}_0$ at that confidence level. This naturally justifies the following definition of order book-driven predictability.

Definition 4.1. For $\alpha \in (0, 1)$, we say that there is order book-driven predictability at confidence level $1-\alpha$ if $p^{\text{MCS}}_{\text{benchmark}} < \alpha$.

In other words, if the *unpredictive benchmark* is deemed to be statistically inferior to some of the other models in $\mathcal{M}_0$, at least one order book-driven model does better than the *unpredictive benchmark*, i.e., such model is predictive.

One of the strengths of applying the model confidence set procedure in this context is that it allows us to fully account for parameter estimation error, while classic methods for comparing deep learning models do not. To make this statement precise, let us make explicit the dependence on the test loss

$$L_{i,w} = \text{cce}(\mathbf{p}^i(\mathbf{X}_{w,\text{test}}; \hat{\theta}_i(\mathcal{D}_{w,\text{train}} \cup \mathcal{D}_{w,\text{val}}, \epsilon_{i,w})), \mathbf{c}_{w,\text{test}}),$$

on the training-validation data $\mathcal{D}_{w,\text{train}} \cup \mathcal{D}_{w,\text{val}}$, the fitting procedure initialization seed $\epsilon_{i,w}$ and the testing data $\mathcal{D}_{w,\text{test}} = \{\mathbf{X}_{w,\text{test}}, \mathbf{c}_{w,\text{test}}\}$. In the related literature, deep learning models are often compared by fitting the model

[4] $\vartheta = \$0.01$ for many US listed stocks and all the stocks considered in this paper.

[5] Data from the week starting on January 7, 2019, is used to start the rolling window standardization required for deepLOB and deepOF architectures

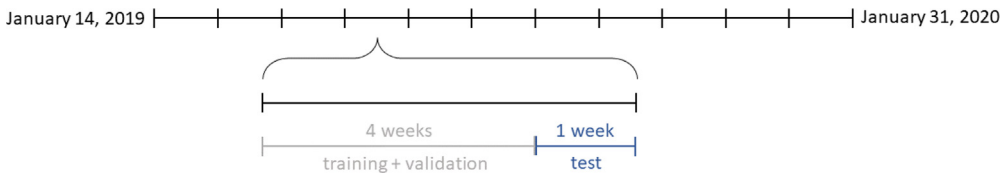


Fig. 11. Data set windowing for experiments.

to the same data $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}}$ starting with different seeds and reporting the out-of-sample average loss and standard error, This accounts only for the uncertainty in ϵ and not in the sampled training and testing data sets (cf. Remark A.1). Our approach instead naturally considers overall parameter estimation uncertainty, comprised of both statistical and optimization errors, by comparing model losses as functions of the random processes $\mathcal{D}_{w,\text{train}} \cup \mathcal{D}_{w,\text{val}}$, $\epsilon_{w,i}$ and $\mathcal{D}_{w,\text{test}}$. This provides a more robust comparison of model performance, i.e., specification error.

Remark 4.1. Not all training procedures may converge to the optimal combination of parameters. This is a characteristic of any model learned via numerical optimization methods. We further note that no hyperparameter tuning is carried out for any models in our experimental setup. When using these models in a production setting, one may obtain better results by selecting parameters using cross-validation on the training-validation set. Parameters which one may investigate tuning include:

- Architecture hyperparameters:
  - number of filters in each convolutional layer (we fix 32 channels);
  - number of weighted averages and lengths of averaging windows;
  - number of LSTM hidden nodes (we fix 64 hidden nodes);
  - decoder type for multi-horizon models (we use seq2seq).
- Feature hyper-parameters:
  - number of levels/size of window (we fix $L = 10$ and $W = 10$);
  - queue depth in L3 volume features (we fix queue depth $D = 10$);
  - look-back window length (we fix $T = 100$).
- Training/Optimization hyper-parameters:
  - Adam learning rate (we fix $\eta = 0.01$);
  - Adam parameters (we fix $\epsilon = 1$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$);
  - batch size (we fix batch size 256);
  - number of epochs (we fix 50 epochs);
  - early stopping patience (we fix ten epochs validation patience);
  - training set downscale factor[6] (we subset the training dataset by a factor of 10).

Here, we are not interested in obtaining the best fit for a specific model but in comparing different models/benchmarks on a level playing field. We thus leave questions related to hyper-parameter tuning for future work.

All code is developed in Python with the `tensorflow` library and the `keras` API, with some layers requiring custom `tensorflow` methods. Due to the computationally intensive nature of the experiments, specialized infrastructure was required to store the data (5TB) and train the models (GPUs). All computations were carried out on Imperial College's High-Performance Computing cluster (Imperial College Research Computing Service, 2022), which provides access to several RTX6000 GPUs.

The results discussed hereafter are specific to the experimental setup under consideration, i.e., they are specific to the selected stocks, time period, and models. Different experimental setups may lead to different results.

4.1. Do high-frequency returns display predictability? If so, how far ahead can we predict?

To understand whether high-frequency returns display predictability, we consider the following set of models:

$$\mathcal{M}_0 = \{\text{benchmark, deepLOB(L1), deepOF(L1), deepLOB(L2), deepOF(L2), deepVOL(L2), deepVOL(L3)}\},$$

where L1 models only use the first level of the order book, while L2 and L3 models use the first ten levels. For each return horizon $h \in \{10, 20, 30, 50, 100, 200, 300, 500, 1000\}$ and TICKER $\in$ {LILAK, QRTEA, XRAY, CHTR, PCAR, EXC, AAL, WBA, ATVI, AAPL} we determine the MCS p-value of each model in $\mathcal{M}_0$. The MCS p-value for the *unpredictive benchmark* model is reported in Table 4. Lighter

[6] In all our experiments, we subset the training dataset by a factor of 10, i.e., subsequent feature-response observations in the dataset are obtained by sliding forward by ten order book events. This serves multiple purposes: on the one hand, it significantly reduces the computational effort required to train the models. On the other hand, it helps to prevent the risk of overfitting to specific patterns arising in the training dataset. Down-sampling the dataset might also have adverse effects; for example, reducing the available training data of an illiquid stock by a factor of 10 might result in insufficient information to train these data-intensive models robustly.

Table 4
MCS p-values of the *unpredictive benchmark* model for the ten tickers and nine horizons under consideration. When the p-value is low, at least one of the order book-driven models statistically outperforms the *unpredictive benchmark*, i.e., there is order book-driven predictability according to Definition 4.1.

	$h = 10$	$h = 20$	$h = 30$	$h = 50$	$h = 100$	$h = 200$	$h = 300$	$h = 500$	$h = 1000$
LILAK	0.00	0.00	0.42	1.00	1.00	1.00	1.00	1.00	1.00
QRTEA	1.00	0.01	0.01	0.00	0.00	0.00	0.08	0.77	0.54
XRAY	0.00	0.00	0.00	0.01	0.10	0.23	0.48	1.00	0.93
CHTR	0.00	0.00	0.04	0.00	0.42	1.00	1.00	1.00	0.97
PCAR	0.00	0.00	0.00	0.00	1.00	1.00	0.60	1.00	1.00
EXC	0.26	0.00	0.00	0.00	0.00	0.00	0.00	0.51	1.00
AAL	0.11	0.00	0.00	0.00	0.00	0.71	1.00	1.00	1.00
WBA	0.00	0.00	0.00	0.00	0.02	0.27	0.13	0.01	1.00
ATVI	0.00	0.00	0.00	0.00	0.00	0.78	1.00	1.00	1.00
AAPL	0.00	0.00	0.00	0.23	1.00	1.00	1.00	1.00	1.00

Table 5
How often each model is in the α-MCS when predictability is identified at the corresponding level α.

	α = 0.05	α = 0.01
benchmark	0%	0%
deepLOB(L1)	7%	11%
deepOF(L1)	23%	22%
deepLOB(L2)	5%	11%
deepOF(L2)	88%	89%
deepVOL(L2)	65%	84%
deepVOL(L3)	77%	86%

green shading corresponds to $\alpha = 0.05$ while darker green is $\alpha = 0.01$.

The results reported in Table 4 show that predictability is systematically present at high frequencies. For most of the stocks under consideration, we were able to identify predictability up to 50 order book events ahead at the 99% confidence level.

Except for LILAK, which is the most illiquid stock, we observe a substantial correlation between the persistence in predictability and the average Updates to Price Changes ratio, cf. Table 1. Recall that in our setting, the horizon h is measured in order book updates; this can be interpreted as it being easier for the deep learning models to predict returns resulting from fewer price changes. One might thus expect to obtain a more consistent maximum predictable horizon across stocks when using a price change-driven clock to measure time.

4.2. Which order book representations perform the best?

Having discussed the extent to which the class of models under consideration can identify predictability, it is natural to ask which models perform best. This corresponds to determining the specifications consistently placed in the set of superior models in the MCS framework. We restrict our attention to the horizons and stocks at which predictability is identified. For each model in consideration, we count the number of times it is identified as a superior model. The results are reported in Table 5.

We can observe how order book representations influence model performance from the results in Table 5 (at the 99% confidence level). When considering deep learning models for short-term return prediction, having access to L2 data provides a significant advantage over L1 data. We see that models with L1 data are rarely placed in the set of superior models when predictability is identified. When going from L2 to L3 data instead, the increased granularity does not seem to provide a clear advantage: deepVOL(L2) and deepVOL(L3) display similar performance. Our experiment suggests that L3 data might be excessively granular when predicting high-frequency returns from order books.

The choice of features representing the order book is also crucial when leveraging deep learning methods for return prediction. Using order flow or volume representations provides a significant performance improvement: the basic deepLOB model ends up being included in the set of best models only 10% of the time and is outperformed even by the model with only first-level (L1) order flow. Volume- and order flow-based models (with L2/L3 data granularity) display comparable performance, placed in the set of superior models in 85%–90% of the predictable horizons.

4.3. Can we use a single model across multiple horizons?

This section explores whether using a seq2seq decoder to produce multi-horizon predictions is beneficial. Such models have the clear advantages of having a single set of weights (the network size is only slightly bigger than single horizon networks) and outputting multiple predictions in similar run times. This significantly reduces the memory required to store the models and the time needed to train them. But how do they perform when compared to their single-horizon counterparts?

To answer this question, we run the same experiment as in Sections 4.1 and 4.2 but enlarge the set of models with the seq2seq specifications. We focus only on prediction horizons $h \in \{10, 20, 30, 50\}$. The results are reported in Tables 6 and 7.

From Table 7, we note that, for each input type, the seq2seq specifications outperform their single horizon counterparts. We suggest this behavior might be due to the increased availability of information in a multi-horizon setting. For a given input-target pair, when targets are multi-horizon, more information is available on the "order book regime" the inputs should be mapped to. Multi-horizon models can thus learn a more granular map from the input variables to the latent space of "order

Table 6
MCS p-values of the *unpredictive benchmark* model for the ten tickers and nine horizons under consideration when seq2seq models are also considered. When the p-value is low, at least one of the order book-driven models statistically outperforms the *unpredictive benchmark*, i.e., there is order book-driven predictability according to Definition 4.1.

	$h = 10$	$h = 20$	$h = 30$	$h = 50$
LILAK	0.00	0.00	0.31	0.55
QRTEA	1.00	0.10	0.06	0.00
XRAY	0.00	0.00	0.00	0.00
CHTR	0.00	0.00	0.01	0.01
PCAR	0.00	0.00	0.00	0.00
EXC	0.23	0.00	0.00	0.00
AAL	0.08	0.00	0.00	0.00
WBA	0.00	0.00	0.00	0.00
ATVI	0.01	0.00	0.00	0.00
AAPL	0.00	0.00	0.00	0.07

Table 7
How often each model is in the α-MCS when predictability is identified at the corresponding level α.

	$\alpha = 0.05$	$\alpha = 0.01$
benchmark	0%	0%
deepLOB(L1)	6%	10%
deepOF(L1)	13%	16%
deepLOB(L2)	6%	13%
deepOF(L2)	63%	71%
deepVOL(L2)	63%	74%
deepVOL(L3)	75%	84%
deepLOB(L1, seq2seq)	13%	26%
deepOF(L1, seq2seq)	16%	16%
deepLOB(L2, seq2seq)	16%	39%
deepOF(L2, seq2seq)	91%	97%
deepVOL(L2, seq2seq)	78%	84%
deepVOL(L3, seq2seq)	81%	87%

book regimes", which might be beneficial for producing predictions.

Remark 4.2. Running the model confidence set procedure again with a larger set of models leads to a counter-intuitive situation where fewer prediction horizons are identified. One would expect that adding models could only increase the number of horizons at which predictability is identified. While this is true in the limit as the number of horizons increases to infinity, it does not hold in the setting where we can access a finite set of observations. With this observation in mind, we note that results in Table 6 are consistent with those in Table 4.

4.4. Can we use a single model across multiple stocks?

In a similar spirit to Sirignano and Cont (2019), we investigate questions regarding the universality of order book dynamics. Intuitively, at a microstructural level, securities traded by market participants with similar characteristics may be subject to the same trading patterns independently of the underlying stock's properties. In Sirignano and Cont (2019), the authors observe that price formation dynamics driven by past order book information, i.e., next mid-price moves, display common patterns across different stocks. In this paper, we explore whether similar results hold over longer horizons.

We run the same experiment as in Section 4.1 but train the models on multiple stocks simultaneously. Specifically, we split the set of 10 stocks under consideration into two; a first "in-sample" set of stocks given by {QRTEA, CHTR, EXC, WBA, AAPL}, and a second "out-of-sample" set {LILAK, XRAY, PCAR, AAL, ATVI}. For each window w, we use all the training-validation data for the "in-sample" stocks to train (and validate) the models. We then evaluate the trained models on the test data of both the "in-sample" and the "out-of-sample" stocks. The results are reported in Table 8.

Focusing on "in-sample" stocks, we note that using universal models leads to results that are partially inconsistent with those in Table 4. A possible interpretation is that universal models may pick up different order book dynamics from those identified by stock-specific models. In this sense, relatively illiquid stocks with overall "standard" trading behavior might benefit from universal models thanks to the greater availability of data. We suggest this might be the case for QRTEA and EXC at horizon $h = 10$. When, instead, the ticker is mainly subject to stock-specific trading patterns, universal models have a hard time detecting predictability – we believe this might be the case for CHTR and AAPL. These results are intrinsically tied to the observations of Remark 4.3.

Remarkably, universal models can identify predictability for "out-of-sample" stocks. This provides evidence of the presence of common trading patterns in the order books of different tickers. For AAL and ATVI, universal models can consistently outperform the *unpredictive benchmark* predictions without ever learning from the stock's past order book data.

Since we believe universal and stock-specific models may be picking up different trading patterns, a natural question is whether the conclusions from Section 4.2 carry over to the universal setting. The results in Table 9 suggest the superior power of L2 over L1 data is retained for universal models. In this case, though, the increased granularity of L3 data appears beneficial. Moreover, the volume representation outperforms both order book and order flow inputs when considering universal models. These results thus suggest that some predictive universal patterns can be extracted only from the most granular data. This contrasts with stock-specific models, which achieve good predictive performance simply based on order flow information (which is, by construction, also contained in volume-based features).

Remark 4.3. As discussed in Section 3.2.2, the choice of γ used to define the return classes is stock-specific. This means that up/down labels for different securities may correspond to different numbers of mid-price changes. This aligns with our choice of order book-driven clock, which is also stock-specific. It is important to note that the choices of γ and clock t might not entirely account for structural trading differences of stocks, making it harder to identify universal trading patterns. This contrasts with Sirignano and Cont (2019), where the next

Table 8
MCS p-values of the *unpredictive benchmark* model for the ten tickers and nine horizons under consideration, universal models only. When the p-value is low, at least one of the order book-driven models statistically outperforms the *unpredictive benchmark*, i.e., there is order book-driven predictability according to Definition 4.1. $(*)$ are "in-sample" stocks.

	$h = 10$	$h = 20$	$h = 30$	$h = 50$	$h = 100$	$h = 200$	$h = 300$	$h = 500$	$h = 1000$
LILAK	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
QRTEA $(*)$	0.00	0.00	0.01	0.06	0.00	0.13	0.59	0.34	1.00
XRAY	0.03	1.00	1.00	1.00	1.00	1.00	0.62	0.81	1.00
CHTR $(*)$	1.00	1.00	1.00	1.00	1.00	1.00	0.94	1.00	1.00
PCAR	0.08	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
EXC $(*)$	0.00	0.00	0.00	0.00	0.00	0.39	0.69	1.00	1.00
AAL	0.00	0.00	0.00	0.00	0.00	0.58	0.60	1.00	1.00
WBA $(*)$	0.00	0.00	0.00	0.00	0.02	0.58	0.36	0.14	1.00
ATVI	0.00	0.00	0.00	0.00	0.60	0.74	0.88	1.00	1.00
AAPL $(*)$	0.07	1.00	1.00	1.00	1.00	0.59	1.00	0.65	1.00

Table 9
How often each model is in the α-MCS when predictability is identified at the corresponding level α.

	$\alpha = 0.05$	$\alpha = 0.01$
benchmark	0%	0%
deepLOB(L1, universal)	0%	0%
deepOF(L1, universal)	0%	0%
deepLOB(L2, universal)	8%	24%
deepOF(L2, universal)	42%	62%
deepVOL(L2, universal)	71%	81%
deepVOL(L3, universal)	92%	100%

mid-price move has a straightforward universal structural interpretation. There are a couple more differences with Sirignano and Cont (2019) we want to point out:

- First, in Sirignano and Cont (2019), the authors consider a much bigger set of stocks, comprised of 500 "in-sample" stocks used for training and 500 "out-of-sample" stocks, making their model more general and thus more universal.
- Second, in that paper, the authors also investigate questions regarding the stationarity of price formation dynamics. This entails using a single long training window: the greater data availability leads to more stable results. To employ the model confidence set procedure and allow the models to capture patterns specific to different economic regimes, we did not adopt this approach in our work.
- The LSTM model in that paper differs from the deepLOB/deepOF/deepVOL architectures. In Sirignano and Cont (2019), the LSTM specification is an online model, i.e., inputs at time t represent the order book state at time t only and are fed through an LSTM-based architecture updating the stored hidden state and producing output predictions. In our models, instead, there is no storage of hidden units between one prediction and the next, i.e., at each time step t, we input the order book history between $t - T + 1$ and t (in the form of raw order books, order flow or volumes) and, after appropriate convolutional feature extraction, apply an LSTM to the processed sequential data to produce the prediction. When studying dependence on order book history T, Sirignano and Cont (2019) refer to the cut-off horizon used in the backpropagation through time computation of gradients during the training procedure.

5. Conclusions and outlook

This paper explored empirical questions regarding the predictability of mid-price returns driven by order book data in centralized exchanges. The predictability in price formation dynamics, already considered in the sense of the next mid-price change in Sirignano and Cont (2019), was found to persist up to 50–300 order book updates ahead, horizons over which multiple mid-price changes may occur. Such predictable horizons might vary from a few milliseconds to nearly half a second, depending on the stock under consideration. These results contrast with low-frequency returns, where predictability is harder to identify but easier to trade once discovered. In fact, predictability is not always exploitable in the high-frequency context due to technological limitations and market microstructure issues. Therefore, a related and more challenging question is to understand whether the predictability identified in this paper is tradable.

The experiments were carried out using specific deep learning architectures. Other than identifying predictable horizons, we aimed to answer questions related to the models under consideration. In particular, we found strong empirical evidence for using L2 data over L1 data. But, from the results presented in this paper, using even more granular order book information (L3) seems to benefit only universal models. The experiments also highlight the importance of carefully choosing a data representation: models based on the basic order book level representation are considerably outperformed by those with order flow or volume inputs. Finally, we found empirical evidence for the presence of universal trading patterns in order book dynamics. Our preliminary results suggest that deep learning architectures may pick up different predictable order book patterns when trained on a pool of stocks instead of a single one. The volume representation has some considerable theoretical advantages; in particular,

it is robust to small perturbations, which might explain its superior predictive ability when considering universal models.

Many further theoretical questions regarding predictability, which we believe to be of research interest, remain unanswered. First, it would be natural to explore whether predictability in returns can be entirely explained by order book structure or if recurring trading patterns play a relevant role. Next, one could compare the predictive performance of the deep learning architectures discussed in this paper to that of the models based on carefully engineered features considered in Aït-Sahalia et al. (2022). Finally, only Nasdaq-traded stocks were considered in this paper. This is a centralized electronic dealer market on which relatively big companies are listed. It would be interesting to explore whether similar results are obtained for securities traded on exchanges with different market structures. With appropriate experimental setups, we believe this paper's model confidence set procedure provides a solid framework to tackle all these questions.

On the practical side, there are some relevant issues one should consider. First, it is essential to note that in this paper, we focus on mid-to-mid returns, which are not tradable in practice. When considering specific trading applications, one should thus define the return labels appropriately, for example, when working with ask-to-bid returns. Second, in this paper, we use an order book-driven clock. In practice, this means the prediction horizon is intrinsically stochastic, which might be a problem in execution. Another critical aspect that would need to be accounted for in practical high-frequency trading is total execution speed. This consists of order book information latency, model prediction run time, and order submission speed. Zhang et al. (2019) and Kolm et al. (2021) argue that their models (which are also considered in this paper) are sufficiently fast to be used by traders with good connections: see Appendix A.7 for a discussion on the effects of infrastructure latency.

As with any trading application, one must also account for his trade's impact on the order book. Moreover, if many traders exploit the same inefficiencies, this might erode all predictive power above the lowest tradable latencies. Overall, we believe that while the predictability identified in this paper might not be directly tradable through a standalone strategy, it might still help some market players, such as market makers, gauge the direction of the market and adjust their quotes accordingly.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data and code availability

The data used in this paper can be accessed from www.lobsterdata.com, a subscription-based order book dataset for academic research. The code used to carry out the experiments is available on GitHub at www.github.com/lorenzolucchese/deepOBs, the results presented in this paper correspond to Release v1.0. For reproducibility purposes we have deposited the trained weights and relative performance metrics on Zenodo with DOI 10.5281/zenodo.10665431.

Acknowledgments

This research has been supported by the EPSRC, United Kingdom Centre for Doctoral Training in Mathematics of Random Systems: Analysis, Modelling and Simulation (EP/S023925/1). We would like to thank Thomas Oliver (InferStat) for helpful discussions on the subject.

Appendix

A.1. Deep learning

A.1.1. Deep neural network architectures

When approximating a function[7] f from a set of models $\{f_\theta\}_{\theta\in\Theta}$, such as in the return prediction task introduced in Section 2.2, a quite common approach is to consider a parametric family of deep neural networks. Theoretically motivated by the universality of such models, this approach does not rely on the functional form of f being correctly specified.

The main idea behind deep neural networks is to lift the input into a higher dimensional space to extract relevant latent features before projecting onto output space. Mathematically a neural network consists of a composition of functions, known as layers, $f_l : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$ for $l = 1, \dots, L$ where d_0 is the dimensionality of input space and d_L is the dimensionality of the output space, for example $d_L = 3$ in our three-class classification task. A neural network is considered deep if the number of layers L is large.

In the simplest case the layers f_l's are "activated" affine transformations, i.e., for input $z_{l-1} \in \mathbb{R}^{d_{l-1}}$

$$f_l(z_{l-1}) = \sigma_l(W_l z_{l-1} + b_l),$$

where $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and $b_l \in \mathbb{R}^{d_l}$ are learnable parameters, also known as weights, and σ_l is a non-linear activation function applied element-wise. Neural networks of the form $f = f_L \circ \cdots \circ f_1$ such that each f_l is of this form have been shown to be universal for the class of continuous functions (Cybenko, 1989; Hornik, Stinchcombe, & White, 1990), i.e., any continuous function (on a bounded domain) can be approximated arbitrarily well by a large enough network in the supremum norm.[8] Related works have shown the representational benefits of depth. There are functions that deep networks can construct with polynomially many parameters, which instead require exponentially many parameters when considering shallow networks (Telgarsky, 2015). More complex

[7] This notation covers both regression $f(\cdot) = g(\cdot)$ and classification $f(\cdot) = (p_\downarrow(\cdot), p_=(\cdot), p_\uparrow(\cdot))$ tasks.

[8] It follows directly that the class of neural networks is dense in $L^p(\mathcal{X})$ space, for bounded $\mathcal{X} \subseteq \mathbb{R}^d$.

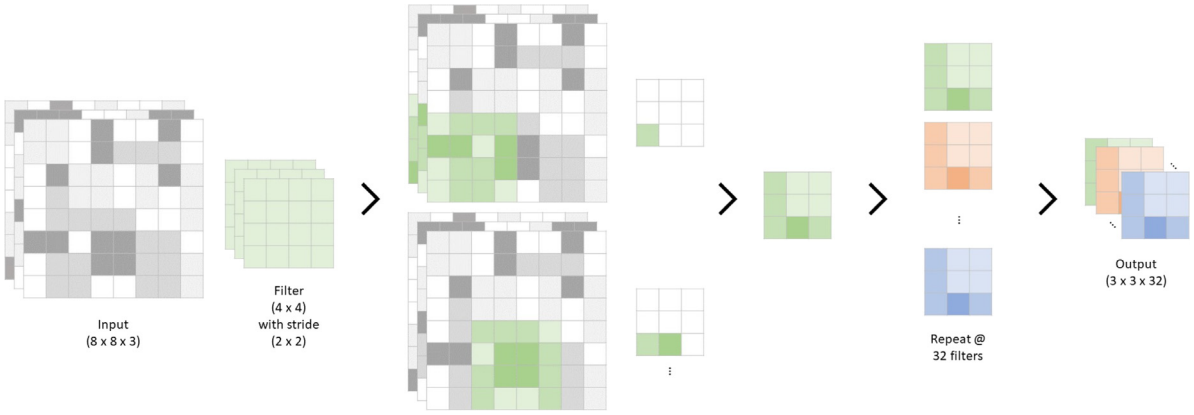


Fig. 12. Convolutional layer, input $z_{l-1} \in \mathbb{R}^{8\times 8\times 3}$, filter size $(n_l \times m_l) = (4 \times 4)$, stride $(s_l \times t_l) = (2 \times 2)$ and number of filters $k_l = 32$.

and domain-specific forms of layers f_l exist; we briefly introduce convolutional and recurrent/LSTM layers.

Convolutional layers. Convolutional layers are a natural choice when the input is an image, i.e.,

$$z_{l-1} \in \mathbb{R}^{d_{l-1}} = \mathbb{R}^{h_{l-1}\times w_{l-1}\times c_{l-1}},$$

where h_{l-1} is the height in pixels, w_{l-1} is the width in pixels and c_{l-1} is the number of channels, for example $c_{l-1} = 3$ if the image is in RGB. A convolutional layer is a specific case of a generic dense layer with parameter restrictions to account for adjacencies in the input's structure. It consists of k_l weight kernels, also known as filters, which are convoluted with the input image to produce the output. Mathematically a 2-dimensional convolution with k_l $(n_l \times m_l)$ filters and stride $(s_l \times t_l)$ is described by

$$[f_l(z_{l-1})]_{i,j,k} = \sum_{n=1}^{n_l}\sum_{m=1}^{m_l}\sum_{c=1}^{c_{l-1}}\left\{[W_l^{(k)}]_{n,m,c}[z_{l-1}]_{s_l(i-1)+n,t_l(j-1)+m,c} + b_l^{(k)}\right\},$$

where $W_l^{(k)} \in \mathbb{R}^{n_l\times m_l\times c_{l-1}}$, for $k = 1,\dots,k_l$, are the k_l filters, $b_l^{(k)} \in \mathbb{R}$ are bias terms and $[\cdot]_{i,j,k}$ denotes the i,j,k-th entry of a three dimensional tensor. The output then lives in $\mathbb{R}^{h_l\times w_l\times k_l}$, where $h_l = (h_{l-1} - n_l)/s_l + 1$ and $w_l = (w_{l-1} - m_l)/t_l + 1$, assuming that $s_l \mid (h_{l-1} - n_l)$ and $t_l \mid (w_{l-1} - m_l)$. While this equation might look daunting, convolutional layers are often easily understood via a graphical depiction, as in Fig. 12. Many empirical studies (He, Zhang, Ren, & Sun, 2016; Krizhevsky, Sutskever, & Hinton, 2012) have investigated the ability of convolutional layers to extract relevant features from input with grid-like topologies. Section 2.3 discusses how this ability might be leveraged to extract relevant features from order books.

Recurrent/LSTM layers. When the input has a built-in temporal structure, recurrent layers can naturally account for this. This type of layer retains information over time, discovering temporal dependencies in the data. Recurrent layers may be used with streaming data to obtain online predictions or applied to a whole time series, yielding a single output. We focus on the latter case, i.e., assuming the input $z_{l-1} \in \mathbb{R}^{d_{l-1}}$ has a temporal structure of the form

$$z_{l-1} = \left(z_{l-1}^{(1)},\dots,z_{l-1}^{(T_{l-1})}\right)^{\mathrm{T}} \in \mathbb{R}^{T_{l-1}\times n_{l-1}},$$

the recurrent layer $z_l = f_l(z_{l-1})$ is given by

$$h_{l-1}^{(0)} \in \mathbb{R}^{m_{l-1}},$$
$$h_{l-1}^{(t)} = \phi_l\left(h_{l-1}^{(t-1)}, z_{l-1}^{(t)}\right) \text{ for } t = 1,\dots,T_{l-1}, \quad z_l = h_{l-1}^{(T_{l-1})},$$

where $\phi_l : \mathbb{R}^{m_{l-1}} \times \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{m_{l-1}}$ is a parameterized recurrent function and the $h_{l-1}^{(t)}$'s are known as hidden states. The most widely used type of recurrent layer is Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997). While other types of recurrent layers may suffer from vanishing or exploding gradients when training, the specific structure of LSTMs largely prevents such problems (Hochreiter, Bengio, Frasconi, & Schmidhuber, 2001). In an LSTM layer, each hidden state h is augmented with a memory state s, and hence the recurrence becomes

$$\left(h_{l-1}^{(t)}, s_{l-1}^{(t)}\right) = \phi_l\left(h_{l-1}^{(t-1)}, s_{l-1}^{(t-1)}, z_{l-1}^{(t)}\right).$$

The equations governing the LSTM recurrence are given by

$$h_{l-1}^{(t)} = o_{l-1}^{(t)} \odot \tanh(s_{l-1}^{(t)}),$$
$$s_{l-1}^{(t)} = g_{l-1}^{(t)} \odot \sigma\left(b_l + U_l z_{l-1}^{(t)} + W_l h_{l-1}^{(t-1)}\right) + f_{l-1}^{(t)} \odot s_{l-1}^{(t-1)},$$

where the input, forget, and output gates depend on the context, i.e., on $h_{l-1}^{(t-1)}, z_{l-1}^{(t)}$, via:

$$g_{l-1}^{(t)} = \sigma\left(b_l^g + U_l^g z_{l-1}^{(t)} + W_l^g h_{l-1}^{(t-1)}\right),$$
$$f_{l-1}^{(t)} = \sigma\left(b_l^f + U_l^f z_{l-1}^{(t)} + W_l^f h_{l-1}^{(t-1)}\right),$$
$$o_{l-1}^{(t)} = \sigma\left(b_l^o + U_l^o z_{l-1}^{(t)} + W_l^o h_{l-1}^{(t-1)}\right),$$

for sigmoid activation function σ, element-wise product $\odot$ and parameters $b_l, b_l^g, b_l^f, b_l^o \in \mathbb{R}^{m_{l-1}}$, $U_l, U_l^g, U_l^f, U_l^o \in \mathbb{R}^{m_{l-1}\times n_{l-1}}$ and $W_l, W_l^g, W_l^f, W_l^o \in \mathbb{R}^{m_{l-1}\times m_{l-1}}$. The network diagram of an LSTM layer is given in Fig. 13. LSTMs summarize relevant context information in the memory cell $s_{l-1}^{(t)}$, performing exceptionally well on tasks with a

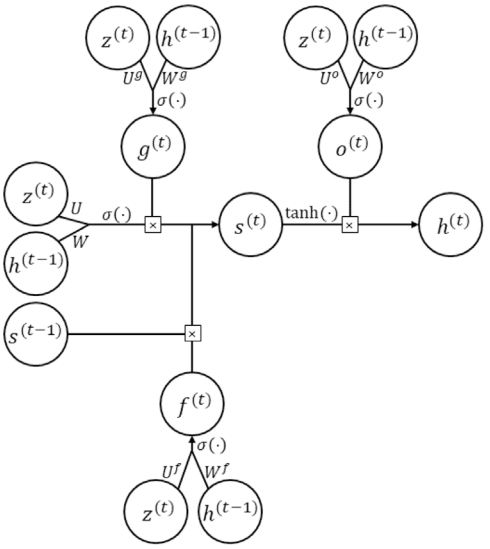


Fig. 13. LSTM layer network diagram.

strong temporal dependency (Melis, Dyer, & Blunsom, 2017).

A.1.2. Training deep neural networks

As discussed in Section 2.2, once we have specified a parametric model for approximating the prediction function $f \approx f_\theta$ we wish to learn the best approximation to f from a training set of observed input–output data $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i\in\mathcal{I}_{\text{train}}}$. To do so, we aim to find the parameter combination θ which yields predictions closest to the observations:

$$\hat{\theta} = \underset{\theta\in\Theta}{\operatorname{argmin}}\, L(\theta \mid \mathcal{D}_{\text{train}}) = \underset{\theta\in\Theta}{\operatorname{argmin}} \left\{ \frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{i\in\mathcal{I}_{\text{train}}} \mathcal{L}\big(f_\theta(x_i), y_i\big) \right\},$$

where $\mathcal{L}(\cdot,\cdot)$ is a loss function quantifying the distance between the prediction and the observed value, and θ contains all network parameters (weights matrices, bias terms, CNN filters...). The form of $\mathcal{L}$ may be chosen using probabilistic arguments or according to some other task-specific criteria.

When the model f_θ is a deep neural network, the loss landscape $L(\theta) = L(\theta \mid \mathcal{D}_{\text{train}})$ may be very complex, and finding a "good" minimizer is often a difficult task. The most widely used approach is to use (some variant of) gradient descent in which the model parameters are iteratively updated by

$$\theta^{(n+1)} = \theta^{(n)} - \eta \nabla_\theta L\big(\theta^{(n)}\big),$$

until a convergence criterion is met. Here $\eta > 0$ is a fixed learning rate, and the weights $\theta^{(0)}$ are chosen according to an appropriate initialization rule. When considering deep neural network architectures, the number of parameters is often very large, and therefore, second-order methods are often intractable or computationally infeasible. Many variants of the first-order gradient descent algorithm exist, though. For example, the stochastic gradient descent algorithm updates parameter values based on estimates of the gradient $\nabla_\theta L(\theta^{(n)})$ computed from a random subset (known as a batch) of the training data set. More advanced first-order optimizers consider momentum and/or use adaptive learning rates. In our experiments, we use the Adam optimizer (Kingma & Ba, 2015), which updates the parameters based on adaptive estimates of first and second-order moments:

$$m^{(n+1)} = \frac{1}{1-\beta_1^n}\Big\{\beta_1 m^{(n)} + (1-\beta_1)\nabla_\theta L\big(\theta^{(n)}\big)\Big\},$$

$$v^{(n+1)} = \frac{1}{1-\beta_2^n}\Big\{\beta_2 v^{(n)} + (1-\beta_2)\big[\nabla_\theta L\big(\theta^{(n)}\big)\big]^2\Big\},$$

$$\theta^{(n+1)} = \theta^{(n)} - \eta \frac{m^{(n+1)}}{\sqrt{v^{(n+1)}} + \epsilon},$$

for parameters $\eta, \beta_1, \beta_2, \epsilon > 0$.

Remark A.1. Note that, contrary to other model specifications, such as linear models, the fitting procedure of deep neural networks is not closed form. This adds a further layer of uncertainty. Any model is subject to the following two sources of uncertainty: one due to model specification (i.e., $f \approx f_\theta$) and one due to parameter estimation uncertainty (i.e., statistical error arising from estimating θ from a finite sample). Models without closed forms for $\hat{\theta}$ from observed data are further subject to numerical optimization error in the parameter estimation phase. The optimization procedure may sometimes not converge to a (global) minimum. With deep neural networks, we thus trade off smaller model uncertainty (at least theoretically due to the universality property) for larger parameter estimation error.

When the task is a classification problem f_θ is usually chosen so that it maps to $\{p \in [0,1]^C : \|p\|_1 = 1\}$, where C is the number of classes in the set $\mathcal{C}$. This allows a probabilistic interpretation of the deep neural network outputs, i.e.,

$$f_\theta(x) = \{p_{c,\theta}(x)\}_{c\in\mathcal{C}}, \text{ where } p_{c,\theta}(x) = \mathbb{P}(y = c \mid x, \theta).$$

In this case, a natural choice of the loss function to use during training is the cross-entropy loss, which can be understood by maximum likelihood arguments: assuming the responses are independent given the features and the distribution of the features is independent of the parameters θ the negative log-likelihood is

$$L(\theta \mid \mathcal{D}_{\text{train}}) = -\log \mathbb{P}(\mathcal{D}_{\text{train}} \mid \theta) \propto -\frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{i\in\mathcal{I}_{\text{train}}} \sum_{c\in\mathcal{C}} \mathbb{1}_{\{y_i=c\}} \log p_{c,\theta}(x_i).$$

When training samples for the classes are unbalanced, say class $\bar{c} \in \mathcal{C}$ has many more samples than all other classes, the optimization algorithm tends to get stuck in the trivial minimum given by the Dirac measure at $\bar{c} \in \mathcal{C}$. One way to mitigate this effect is to re-weight the categorical cross entropy loss such that the network is less incentivized to move towards the trivial minimum. With appropriate weighting, $\bar{c} \in \mathcal{C}$ observations influence the

direction of the gradient less, thus reducing the strength of the attraction towards the trivial minimum. Mathematically, one uses the following weighted categorical cross-entropy loss

$$L(\theta \mid \mathcal{D}_{\text{train}}) = -\frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{i \in \mathcal{I}_{\text{train}}} \sum_{c \in \mathcal{C}} w_c \mathbb{1}_{\{y_i = c\}} \log p_{c,\theta}(x_i),$$

where the weights w_c are chosen to be inversely proportional to the class's training set density. For example, in our experiments, we set:

$$w_c = w_c(\mathcal{D}_{\text{train}}) = \frac{|\mathcal{I}_{\text{train}}|}{\#\{c_i \in \mathcal{D}_{\text{train}} : c_i = c\}}.$$

A.2. Model confidence sets

The model confidence set procedure introduced by Hansen et al. (2011) provides a formalized statistical inference framework to compare competing models. It does not assume that any of the models is the true data-generating process but simply aims to identify a set of models that will contain the best model with a given level of confidence, known as model confidence set, MCS. Therefore, a model confidence set is analogous to a confidence interval for a parameter. To identify the best models, one must compare the model outputs (either as estimated class probabilities or as forecasts) by selecting an appropriate loss/score function.

Let us denote by $\mathcal{M}_0$ the set of models under consideration and by $\{L_{i,w}\}_{w=1}^{W}$ for $i \in \mathcal{M}_0$ the time series of losses. In the MCS framework for models $i, j \in \mathcal{M}_0$ and window $w \in \{1, \dots, W\}$ we define:

$$d_{ij,w} = L_{i,w} - L_{j,w}.$$

Under a stationarity assumption on the $(d_{ij,w})_{w \geq 1}$, we define the set of best models among those in $\mathcal{M}_0$ as

$$\mathcal{M}^* = \{i \in \mathcal{M}_0 : \mathbb{E}[d_{ij}] \leq 0 \ \forall j \in \mathcal{M}_0\}.$$

This is the set we want to identify by using the MCS procedure. The MCS algorithm is then defined as follows.

Definition A.1. Let $\delta_{\mathcal{M}}$ and $e_{\mathcal{M}}$ be an equivalence test and an elimination rule.

Step 0. Initially set $\mathcal{M} = \mathcal{M}_0$.
Step 1. Test $H_{0,\mathcal{M}} : \mathbb{E}[d_{ij}] = 0, \ \forall i, j \in \mathcal{M}$ using $\delta_{\mathcal{M}}$ at level α.
Step 2. If $H_{0,\mathcal{M}}$ is accepted, set $\hat{\mathcal{M}}^*_{1-\alpha} = \mathcal{M}$; otherwise, use $e_{\mathcal{M}}$ to eliminate an object from $\mathcal{M}$ and repeat the procedure from Step 1.

Under appropriate assumptions on the equivalence test and elimination rule,[9] the model confidence set $\hat{\mathcal{M}}^*_{1-\alpha}$ has the following asymptotic properties:

(i) $\liminf_{W \to \infty} \mathbb{P}(\mathcal{M}^* \subset \hat{\mathcal{M}}^*_{1-\alpha}) \geq 1 - \alpha$,
(ii) $\lim_{W \to \infty} \mathbb{P}(i \in \hat{\mathcal{M}}^*_{1-\alpha}) = 0$ for $i \notin \mathcal{M}^*$,

where $W \to \infty$ denotes the number of sample periods. If, furthermore, the equivalence test and elimination rule satisfy a coherency condition,[10] we have the following finite sample property:

(iii) $\mathbb{P}(\mathcal{M}^* \subset \hat{\mathcal{M}}^*_{1-\alpha}) \geq 1 - \alpha$.

In Hansen et al. (2011), the authors give multiple practical examples of equivalence tests and elimination rules. We focus on implementing the MCS procedure with tests constructed from t-statistics and bootstrap estimators. This choice of equivalence test and elimination rule has two practical advantages: it does not require estimating a variance–covariance matrix for the time series $\{(d_{ij,w})_{i,j \in \mathcal{M}_0}, w \geq 1\}$ which might be difficult when $|\mathcal{M}_0| \approx W$ and it satisfies the coherency condition required for the finite sample property (iii) of the MCS procedure. Let us assume the following holds:

Assumption A.1. For some $r > 2$ and $\gamma > 0$, it holds that that $\{(d_{ij,w})_{i,j \in \mathcal{M}_0}, w \geq 1\}$ is strictly stationary, α-mixing of order $-r/(r-2)$ and $\mathrm{Var}(d_{ij,w}) > 0$, $\mathbb{E}|d_{ij,w}|^{r+\gamma} < \infty$ for all $i, j \in \mathcal{M}_0$.

For a set of models $\mathcal{M} \subset \mathcal{M}_0$ with time series of relative performances $\{(d_{ij,w})_{i,j \in \mathcal{M}}, w \geq 1\}$ define the following quantities $\forall i, j \in \mathcal{M}$:

$$\bar{d}_{ij} = \frac{1}{W} \sum_{w=1}^{W} d_{ij,w}, \quad \bar{d}_{i\cdot} = \frac{1}{|\mathcal{M}|} \sum_{j \in \mathcal{M}} \bar{d}_{ij},$$

respectively measuring the sample relative performance of the models i and j and the sample relative performance of model i and all other models in $\mathcal{M}$. We can then introduce the following t-statistics $\forall i \in \mathcal{M}$:

$$t_{i\cdot} = \frac{\bar{d}_{i\cdot}}{\sqrt{\widehat{\mathrm{Var}}(\bar{d}_{i\cdot})}},$$

where $\widehat{\mathrm{Var}}(\bar{d}_{i\cdot})$ are appropriate estimators of the variances of the $\bar{d}_{i\cdot}$'s. The associated test statistic is $T_{\max,\mathcal{M}} = \max_{i \in \mathcal{M}} t_{i\cdot}$. The equivalence test and elimination rule are given by:

($\delta_{\mathcal{M}}$) Reject $H_{0,\mathcal{M}}$ if $T_{\max,\mathcal{M}} > F_{\varrho}^{-1}(1-\alpha)$ where F_{ϱ} is the limiting distribution of $T_{\max,\mathcal{M}}$ as $W \to \infty$.
($e_{\mathcal{M}}$) Eliminate $\mathrm{argmax}_{i \in \mathcal{M}} t_{i\cdot}$.

The asymptotic distribution of the test statistic $T_{\max,\mathcal{M}}$ is nonstandard because it depends on a (usually unknown) nuisance parameter ϱ under the null. The value of ϱ is not strictly necessary since we can consistently estimate the distribution of $T_{\max,\mathcal{M}}$ using a bootstrapping procedure. The equivalence test compares the observed value $T_{\max,\mathcal{M}}$ with the bootstrap empirical quantile. The resulting procedure preserves the asymptotic[11] properties (i) and (ii) and the finite sample property (iii) under Assumption A.1. For details of the bootstrapping procedure, see the Appendix of Hansen et al. (2011).

[9] We require the equivalence test $\delta_{\mathcal{M}}$ to have asymptotic level α and asymptotic power 1, while the elimination rule $e_{\mathcal{M}}$ needs to satisfy $\lim_{W \to 0} \mathbb{P}(e_{\mathcal{M}} \in \mathcal{M}^* \mid H_{1,\mathcal{M}}) = 0$.

[10] The required coherency condition is $\mathbb{P}(\delta_{\mathcal{M}} = 1, e_M \in \mathcal{M}^*) \leq \alpha$.

[11] As both the sample size W and the number of bootstrap samples B increase.

Remark A.2. The test based on the t-statistic discussed above relies on the fact that the null hypothesis $H_{0,\mathcal{M}}$ can be equivalently rewritten as

$$H_{0,\mathcal{M}} : \mathbb{E}[d_{i\cdot}] = 0, \ \forall i \in \mathcal{M}, \quad \text{where} \quad d_{i\cdot} = \frac{1}{|\mathcal{M}|} \sum_{j \in \mathcal{M}} d_{ij}.$$

This paper presents results using MCS p-values, which are defined as follows.

Definition A.2. Let $(\delta_{\mathcal{M}}, e_{\mathcal{M}})$ be the equivalence test and elimination rule associated with an MCS procedure as defined in Definition A.1. Denote by $M = |\mathcal{M}_0|$ the total number of competing models in $\mathcal{M}_0$. The elimination rule defines a decreasing sequence of random sets $\mathcal{M}_0 \supset \ldots \supset \mathcal{M}_{M-1}$ by successively eliminating models $e_1, \ldots, e_M$. Let p_m denote the p-value associated with the null hypothesis $H_{0,\mathcal{M}_m}$ under the equivalence test $\delta_{\mathcal{M}_m}$ with the convention that $p_{M-1} \equiv 1$. Then the MCS p-value of e_m, the m-th model eliminated, is defined as $p_i^{\mathrm{MCS}} = \max_{m' < m} p_{m'}$.

Note that the definition of MCS p-value is such that for $i \in \mathcal{M}_0$ one has $p_i^{\mathrm{MCS}} \geq \alpha \iff i \in \hat{\mathcal{M}}^*_{1-\alpha}$.

Remark A.3. The MCS procedure is quite flexible. It allows us to compare models between each other and to compare them to an *unpredictive benchmark* model (by including the *unpredictive benchmark* in $\mathcal{M}_0$). When only two models are considered, i.e., $|\mathcal{M}_0| = 2$, the MCS procedure reduces to testing the null hypothesis $H_0 : \mathbb{E}[d_{i_1 i_2}] = 0$.

The model confidence set procedure relies on the assumption that the loss differences time series $\{(d_{ij,w})_{i,j \in \mathcal{M}_0}, \ w \geq 1\}$ is stationary and α-mixing. This means that even when the loss process $\{(L_{i,w})_{i \in \mathcal{M}_0}, \ w \geq 1\}$ is non-stationary, e.g., model performance is tied to regime changes, the model confidence set procedure might still be applicable. In Section 4, we assume the time series $\{(d_{ij,w})_{i,j \in \mathcal{M}_0}, \ w \geq 1\}$ to be stationary based on intuitive arguments, but we discuss some statistical checks which might be carried out to provide further evidence in this context. First of all, it is worth noting that general tests for stationarity are not available: most tests require assuming a parametric form on the time series. The simplest example would be to impose a linear structure on the time series, i.e., assume the difference time series $\{\boldsymbol{d}_w, \ w \geq 1\} = \{(d_{ij,w})_{i,j \in \mathcal{M}_0}, \ w \geq 1\}$ follows a Vector Auto-Regressive process of order 1:

$$\boldsymbol{d}_w = \Gamma \boldsymbol{d}_{w-1} + \boldsymbol{\eta}_w,$$

where $\boldsymbol{\eta}_w \sim N(0, \Sigma_\eta)$ are serially uncorrelated. In this setting, one can consider various tests to determine whether the process is stationary. For example, one may apply multivariate homogeneous Dickey–Fuller tests (Harvey & Bates, 2003) to test the null of a unit root against the alternative of stationarity or a multivariate KPSS test to test (short-memory) stationarity as the null and the presence of a unit root as the alternative (Nyblom & Harvey, 2000).

A.3. Data processing

From LOBSTER (Huang & Polak, 2011), we obtain the order book and message files introduced in Section 3.1. These files are jointly processed to obtain the desired order book features, i.e., raw order books, order flow and volumes, and responses as described in Section 3.2. Before extracting the feature-response pairs, the data is treated in the following way:

1. order book states with crossed quotes are removed;
2. states occurring at the same time stamp (at nanosecond precision) are collapsed onto the last state, e.g., an aggressive order executing against multiple resting limit orders;
3. the first and last 10 min of market activity are dropped;
4. the smoothed returns at the requested horizons – as defined in Section 3.2.2 – are computed and matched to the corresponding features.

The data provided by LOBSTER is easily accessible and does not exhibit significant anomalies. The only data inconsistencies we found were on 2019-11-05 for the WBA ticker after the trading halt occurring between 13:32:50.233001415 and 13:37:50.2335639.

A.3.1. Volume data processing

LOBSTER provides level-based data that can be easily manipulated to derive order book and order flow features (we access the first $L = 10$ levels). Deriving the volume features from LOBSTER data instead is a non-trivial task. If we set a window $W > L$ for the volume representation, some levels past $L = 10$ might have prices that fall in the volume window, i.e., $p_{x,t}^{(L+1)} \in \{\pi_{x,t}^{(1)}, \ldots, \pi_{x,t}^{(W)}\}$, and we won't have the required information. To avoid this issue in all our experiments, we set $W = L = 10$. Reconstruction of L3 queue data has a further complication: every time a quoting price leaves the 10-level range, all orders submitted/canceled at this price are not recorded. Therefore, there is no way of reconstructing what happens at such a price level outside of the 10-level range. To minimize the effect of this shortcoming, we adopt the following systematic approach: each time a quoting price (re-)enters the 10-level range, we initialize the corresponding queue with a single order of size given by the total liquidity available at that price level (even if it may be made up of multiple independent orders).

These issues could be mitigated by accessing data deeper in the order book from LOBSTER, say $L = 50$ or 100, while keeping a relatively small window $W \ll L$.

A.4. Data set descriptive statistics: ATVI deep dive

This section reports some descriptive statistics of the processed data used for the experiments. We focus on one representative stock, Activision Blizzard Inc. (ATVI).

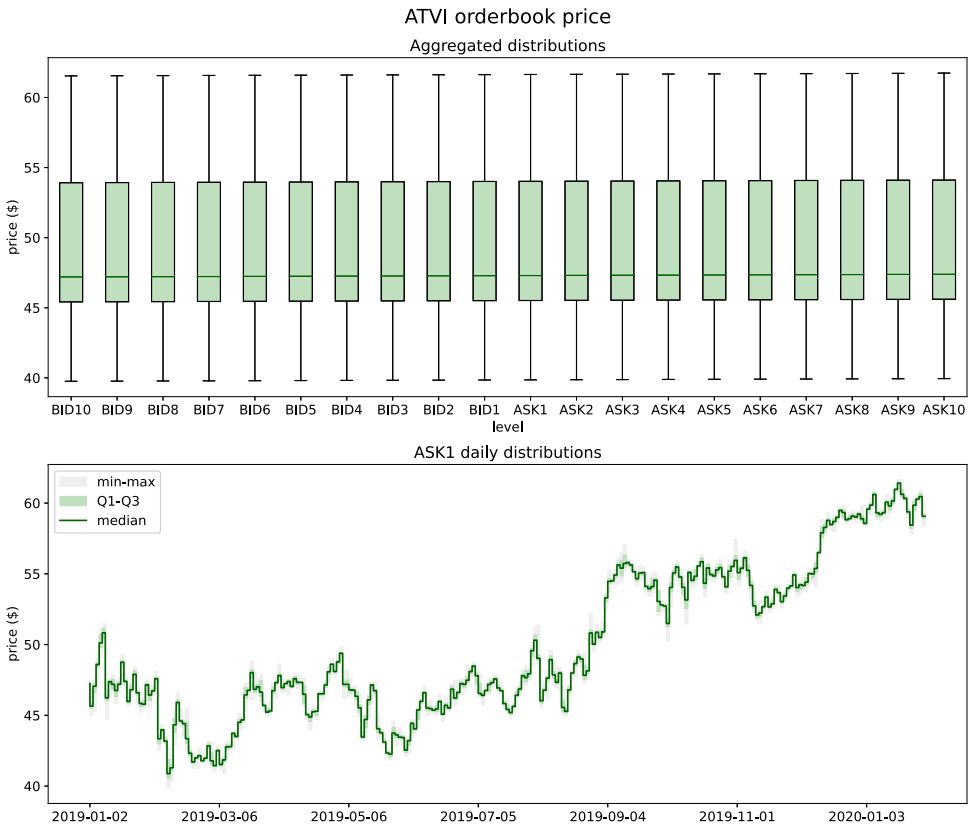


Fig. 14. ATVI order book prices during the period from January 2nd, 2019 to Januray 31st, 2020.

First, we explore the distribution of the order book, order flow, and volume features. For each level/price tick, we report the aggregated distributional boxplot. Moreover, to understand the temporal behavior of the data, we fix one specific level/ price tick and look at how the distribution varies daily. Finally, we analyze the distribution of the target categorical returns. To do so, we fix the first window in the experiment procedure, i.e., $w = 1$ in Section 4, and plot the unconditional train, validation, and test distributions at each prediction horizon $h \in \{10, 20, 30, 50, 100, 200, 300, 500, 1000\}$.

A.4.1. Features

In Figs. 14–18, we plot the aggregate and temporal distributions of the features $\mathbf{x}_t$ described in Section 3.2.1. We note the following:

- As expected, order book prices display significant non-stationarity, cf. Fig. 14.
- The difference between order book volumes $v^{(l)}_{t,x}$, Fig. 15, and volumes $s^{(w)}_{t,x}$, Fig. 17, is that in the former we measure the distance from the mid in levels, i.e., the volume is strictly positive, while in the latter we use all, i.e., also possibly empty, price ticks.
- In Fig. 16, we note that order flow is zero most of the time. By definition, we have non-zero flow only when an event affects exactly that level or the whole order book shifts.
- To provide some insight on the distribution of L3 volume features, we plot queue depth statistics in Fig. 18. We note that most queue depths are less than nine orders long; this justifies the queue depth cut-off of length ten chosen in our experiments.
- Finally, looking at Fig. 15, Figs. 17 and 18, one can note the difference between first level/price tick dynamics and those of the rest of the order book: active price discovery leads to lower volumes and shorter queues.

A.4.2. Responses

In Fig. 19, we plot the unconditional empirical distributions of the target responses defined in Section 3.2.2 for the first window $w = 1$. Note that the threshold $\hat{\gamma}_h$ is only based on the training data set. As detailed in Section 4, the train and validation sets are randomly partitioned from a four-week period, i.e., $\mathcal{D}_{w,\text{train}} \cup \mathcal{D}_{w,\text{val}}$, and thus are likely to display similar volatility regimes. On the other hand, the test set $\mathcal{D}_{w,\text{test}}$ corresponds to the following week, cf. Fig. 11, and thus may exhibit a different volatility profile. By joining the train and validation distribution, we obtain the *unpredictive benchmark* described in Remark 2.1 and used to define predictability in the experiments in Section 4.

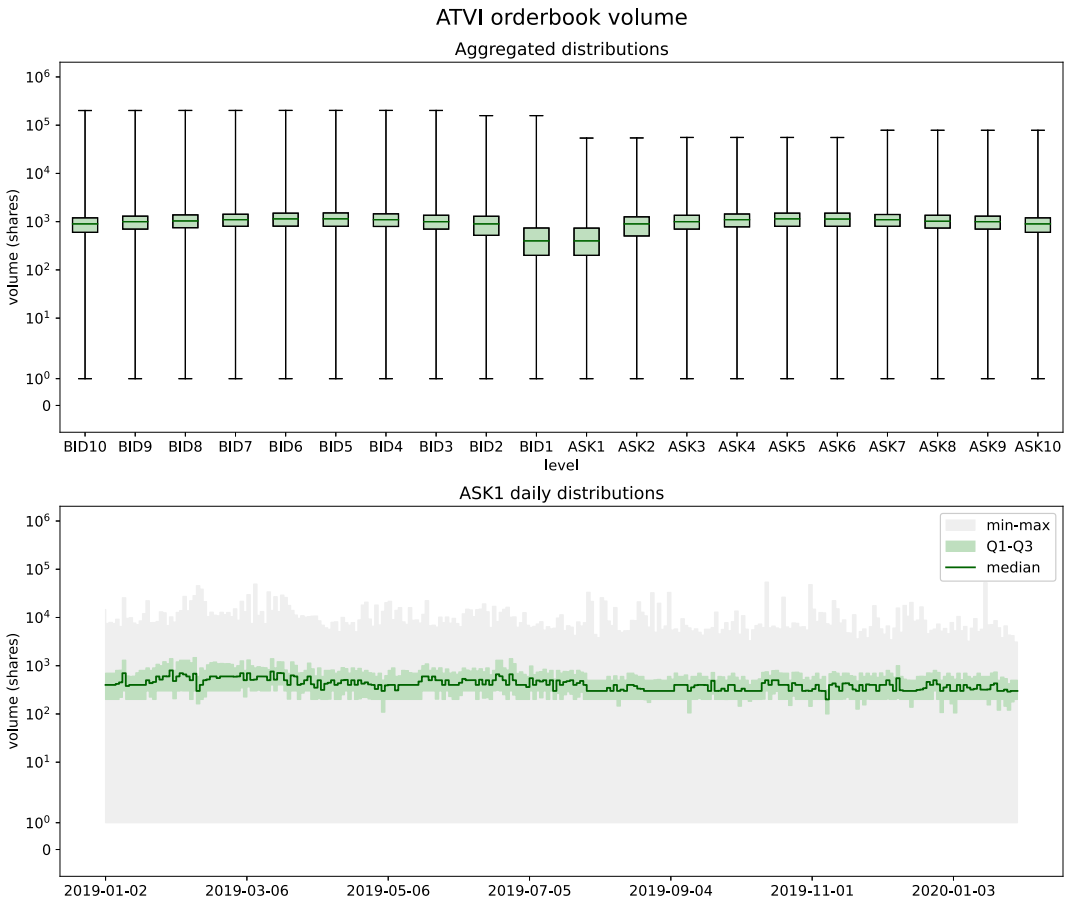


Fig. 15. ATVI order book volumes during the period from January 2nd, 2019 to Januray 31st, 2020.

In Fig. 20, we explore the dependence structure of the return labels by plotting.[12] the previous h-step return $c_{t-h,t}$ against the next h-step return $c_{t,t+h}$. We note that for all horizons $h \in \{10, 20, 30, 50, 100, 200, 300, 500, 1000\}$, a contingency chi-square test rejects the null hypothesis of independence. The fact that return labels display some form of persistency has a two-fold interpretation. On the one hand, it provides evidence towards the existence of predictability in the data, i.e., against the i.i.d. Efficient Market Hypothesis on which the *unpredictive benchmark* is based, cf. Remark 2.1. On the other, it suggests that a simple one-step ahead Markovian model of returns may display predictive power, i.e., a model which predicts the next price move $c_{t,t+h}$ conditional on the last return $c_{t-h,t}$ only. In this respect, we note the return label $c_{t-h,t}$ can be computed from the features $(\mathbf{x}_{t-T+1}, \dots, \mathbf{x}_t)$ for T sufficiently large. Given the universal approximation property of neural networks, any such simple model can thus be considered 'nested' in the deep learning models treated in this paper. In Appendix A.5, we explore how well a specific model based on $c_{t-h,t}$ only performs when compared to the more complex deep learning models considered in this paper.

A.5. Simpler predictive models: an empirical auto-regressive specification

This section explores how a simple model performs compared to the deep learning architectures considered in this paper. We base the model on the simple empirical observation that the target return labels display time-dependence, cf. Appendix A.4.2 and Fig. 20. We consider a one-step ahead.[13] auto-regressive (AR) model, i.e., we aim to estimate $p_*(\star) = \mathbb{P}(c_{t,t+h} = * \mid c_{t-h,t} = \star)$ for $*, \star \in \{\downarrow, =, \uparrow\}$. We use a non-parametric approach, setting $\hat{p}_*(\star)$ in period $w \in \{1, \dots, 11\}$ to be the empirical distribution of $c_{t,t+h} = * \in \{\downarrow, =, \uparrow\}$ given the previous return label was $c_{t-h,t} = \star \in \{\downarrow, =, \uparrow\}$ in the training and validation set $\mathcal{D}_{w,\text{train}} \cup \mathcal{D}_{w,\text{val}}$, i.e., the row normalised matrices displayed in Fig. 20. Formally, this is the maximum likelihood estimator of the conditional probabilities $\mathbb{P}(c_{t,t+h} = * \mid c_{t-h,t} = \star)$ given the data set $\mathcal{D}_{w,\text{train}} \cup \mathcal{D}_{w,\text{val}}$. We repeat the experiments of Sections 4.1

[12] To be more precise, we plot $c_{t-h-k,t}$ against $c_{t,t+h}$ to avoid any kind of overlap between the labels, cf. Section 3.2.2 We drop the k for clarity of exposition and notational convenience.

[13] To avoid any look-ahead bias, we condition on $c_{t-h-k,t}$ where k is the smoothing factor introduced in Section 3.2.2 We write $c_{t-h,t}$ for clarity of exposition.

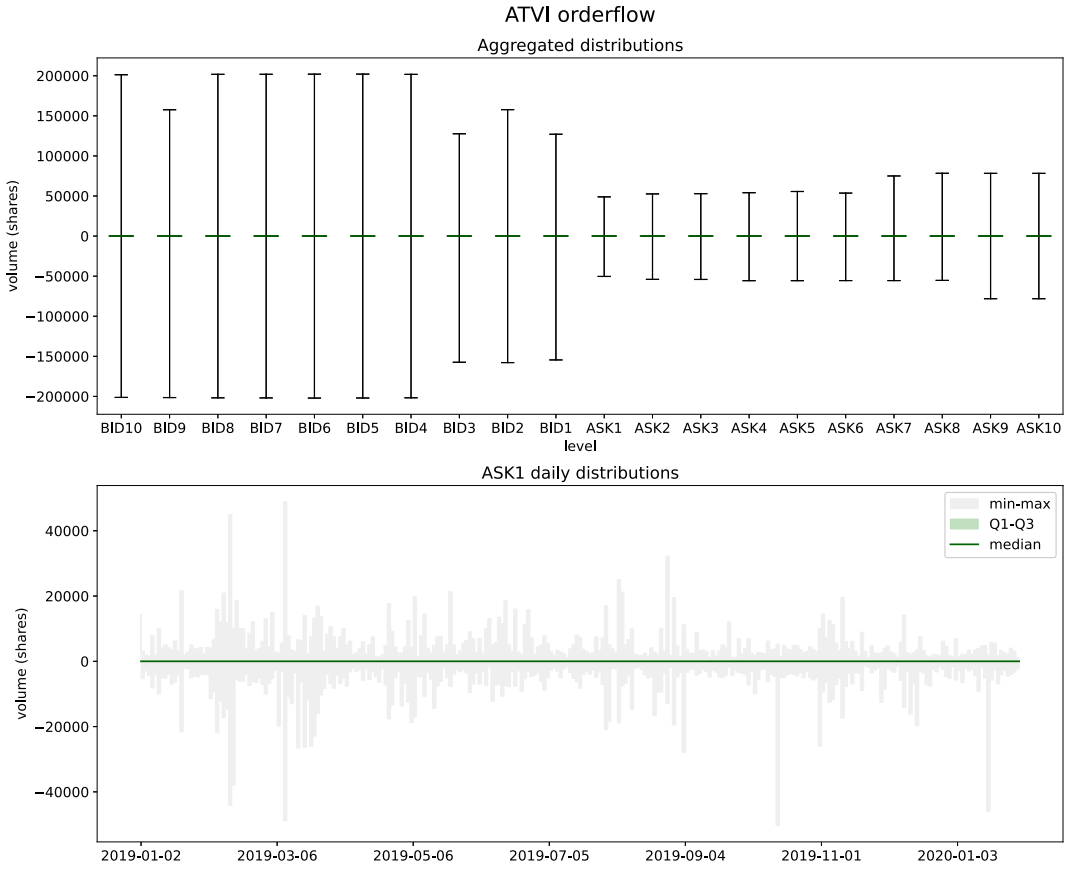


Fig. 16. ATVI order flow during the period from January 2nd, 2019 to Januray 31st, 2020.

Table 10
MCS *p*-values of the *unpredictive benchmark* model for the ten tickers and nine horizons under consideration. When the *p*-value is low, at least one of the order book-driven models statistically outperforms the *unpredictive benchmark*, i.e., there is order book-driven predictability according to Definition 4.1.

	$h = 10$	$h = 20$	$h = 30$	$h = 50$	$h = 100$	$h = 200$	$h = 300$	$h = 500$	$h = 1000$
LILAK	0.00	0.00	0.38	0.40	0.04	0.08	0.00	0.01	1.00
QRTEA	0.00	0.01	0.03	0.00	0.00	0.00	0.06	0.85	0.26
XRAY	0.00	0.00	0.00	0.01	0.09	0.35	0.35	0.73	0.85
CHTR	0.00	0.00	0.03	0.00	0.00	0.00	0.06	0.14	0.56
PCAR	0.00	0.00	0.00	0.00	0.86	0.42	0.31	0.17	1.00
EXC	0.12	0.00	0.00	0.00	0.00	0.00	0.00	0.46	0.38
AAL	0.04	0.00	0.00	0.00	0.00	0.65	0.70	0.10	0.38
WBA	0.00	0.00	0.00	0.00	0.01	0.23	0.04	0.00	0.18
ATVI	0.00	0.00	0.00	0.00	0.01	0.61	0.34	0.12	0.03
AAPL	0.00	0.00	0.00	0.16	0.08	0.03	0.05	0.06	0.09

and 4.2 with this empirical AR model in the set $\mathcal{M}_0$. The updated results are reported in Tables 10 and 11 (we only report results at the 99% confidence level, i.e., $\alpha = 0.01$).

The results reported in Table 10 are consistent with those in Table 4. The main differences are due to some additional predictable horizons identified by including the empirical auto-regressive model. This mainly occurs for stocks with low liquidity (LILAK, QRTEA, and CHTR), where the low data regime makes it more challenging for deep learning models to train effectively.

In Table 11, we note the greater expressivity of deep learning models provides a significant advantage compared to the simple empirical auto-regressive model. While the simple AR model does display predictive power,

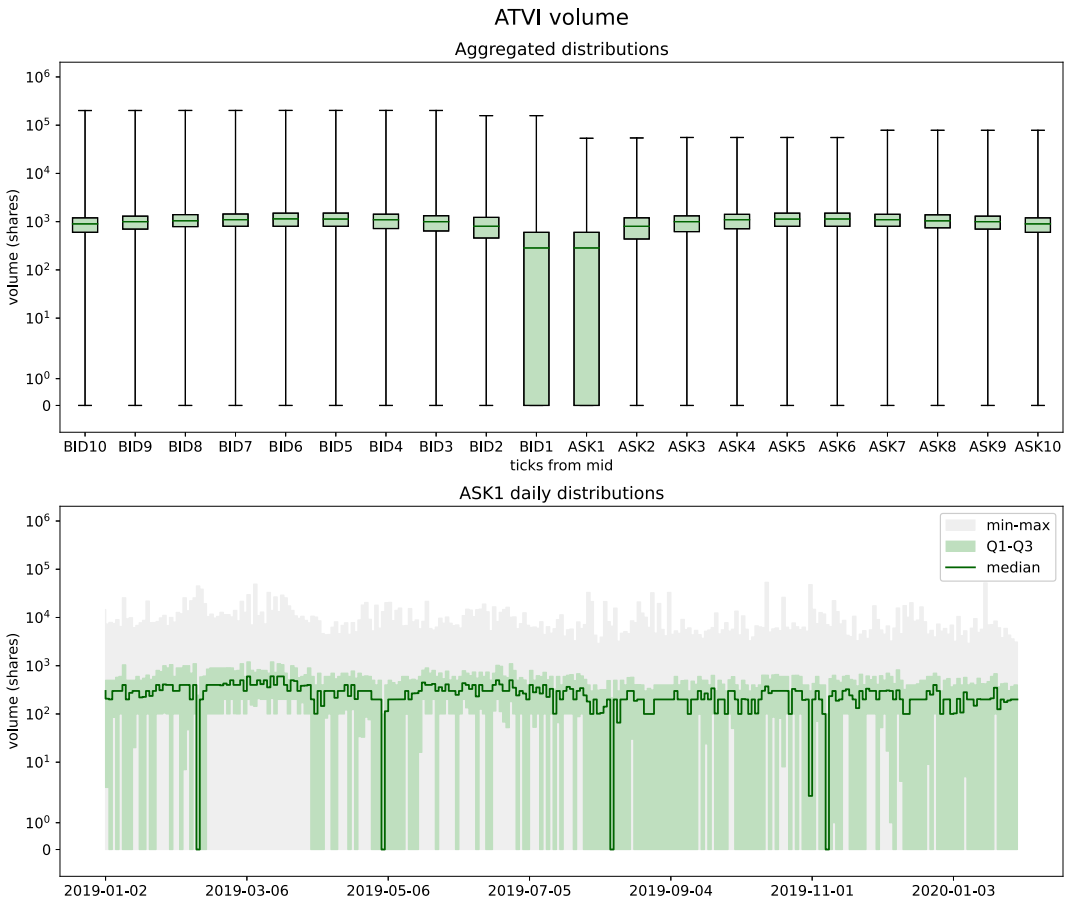


Fig. 17. ATVI volumes during the period from January 2nd, 2019 to Januray 31st, 2020.

Table 11
How often model is in the α-MCS when predictability is identified at the level $\alpha = 0.01$.

	α = 0.01
benchmark	0%
empirical AR model	28%
deepOF(L1)	12%
deepOF(L1)	26%
deepLOB(L2)	12%
deepOF(L2)	81%
deepVOL(L2)	77%
deepVOL(L3)	79%

it is often outperformed by the order book-driven deep learning models deepOF(L2) and deepVOL(L2/L3). In Table 12, we dive deeper into this analysis by reporting for each ticker-horizon combination, whether the simple AR model or deepVOL(L3) is in the MCS of the best models.

While our results suggest deepOF and deepVOL architectures significantly outperform the simple AR model, a more in-depth study comparing these specifications to the models in Aït-Sahalia et al. (2022) is required to shed some light on the question of whether the expressivity of deep learning techniques is the key to producing good predictive models or if careful feature engineering can be as effective. This analysis is particularly relevant in applications where prediction speed plays a fundamental role; see Appendix A.7.

A.6. Return definition

The way returns are defined is often an overlooked topic that inevitably significantly affects model evaluation. Here, we state two alternative definitions to that given in Section 3.2.2 taken from the literature and discuss their properties. In all cases, the mid-price is treated as the "true" price, and returns are defined relative to it. Recall this is not a tradable price.

- In Ntakaris et al. (2018) the authors define the return at horizon h as

$$r_{t,t+h} = \frac{\overline{m}_{t,t+h} - m_t}{m_t}, \quad \text{where } \overline{m}_{t,t+h} = \frac{1}{h}\sum_{i=1}^{h} m_{t+i},$$

and m_t denotes the mid-price at time t. This definition of return compares the smoothed mid-price over the next h time steps to the current mid-price. In this case, the return can be the average mid-to-mid return over the next h time steps. As the horizon h becomes longer, so does the window over

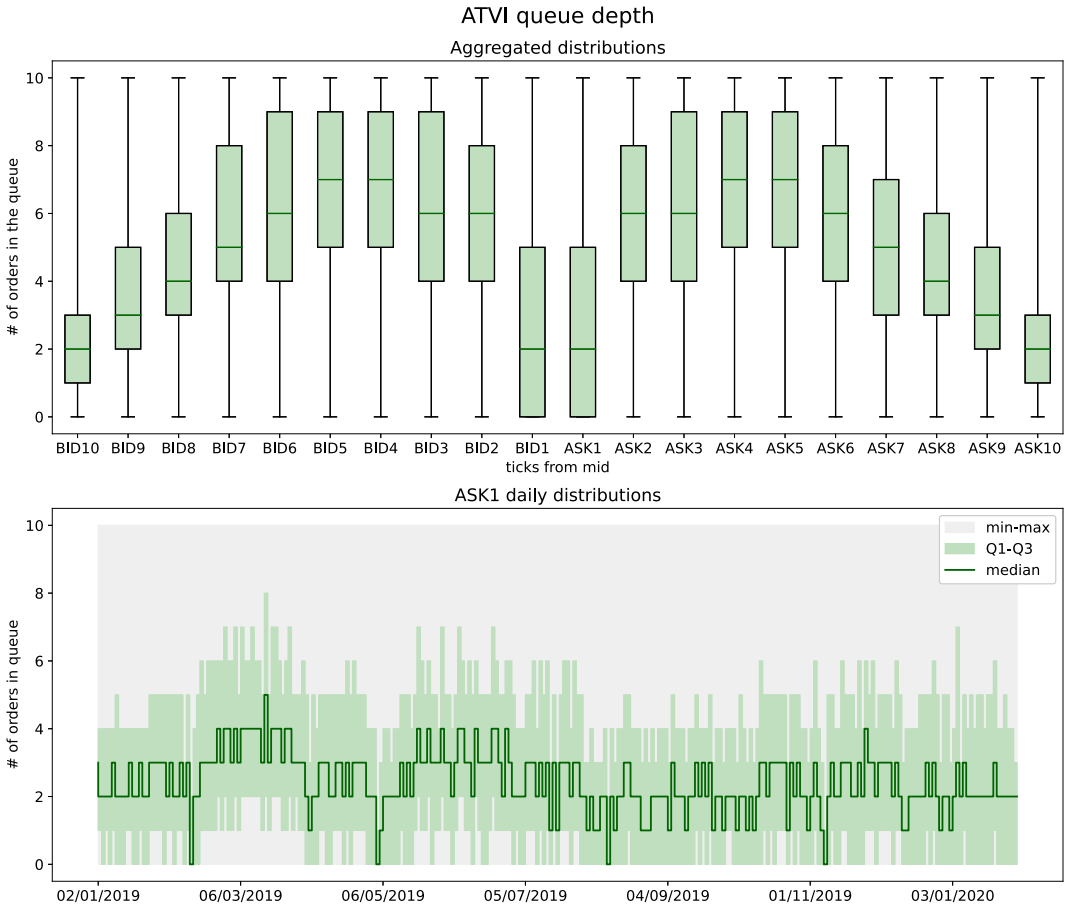


Fig. 18. ATVI queue depths during the period from January 2nd, 2019 to Januray 31st, 2020.

Table 12
Comparison between emprical AR model and deepVOL(L3) MCS results, $\alpha = 0.01$. Shading indicates whether the empirical AR model and/or deepVOL(L3) is in the set of best models. White: neither, light blue: empirical AR model only, blue: deepVOL(L3), dark blue: both. Grey shading indicates the ticker-horizon combination was not deemed predictable at the 99% confidence level.

	$h = 10$	$h = 20$	$h = 30$	$h = 50$	$h = 100$	$h = 200$	$h = 300$	$h = 500$	$h = 1000$
LILAK									
QRTEA									
XRAY									
CHTR									
PCAR									
EXC									
AAL									
WBA									
ATVI									
AAPL									

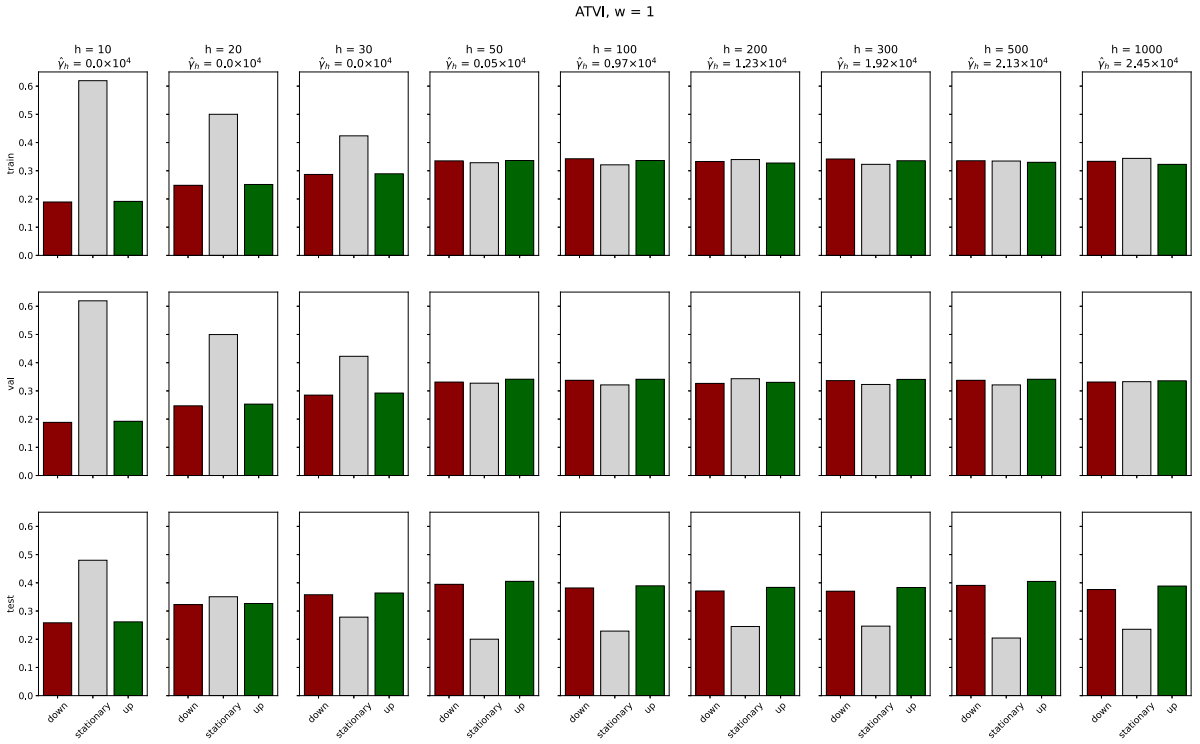


Fig. 19. ATVI return labels during the first window of the experiment, i.e., from January 14th, 2019, to February 15th, 2019.

which we smooth the returns. This has a couple of practical disadvantages in terms of the predictability questions we wish to explore. First, predictability over $h = 100$ time steps might be due to changes in the mid-price over the first, say, $h = 50$ time steps. Thus, it becomes difficult to discuss questions of persistency in predictability. Second, the definition produces a correlation between returns at different horizons, r_{t+h_1} and r_{t+h_2}, inducing a hidden bias in multi-horizon models.

- In Zhang et al. (2019) and Zhang and Zohren (2021) the authors define the return at horizon h by

$$r_{t,t+h} = \frac{\overline{m}_{t,t+h} - \overline{m}_{t-h,t}}{\overline{m}_{t-h,t}}, \text{ where}$$
$$\overline{m}_{t,t+h} = \frac{1}{h}\sum_{i=1}^{h} m_{t+i}, \ \overline{m}_{t-h,t} = \frac{1}{h}\sum_{i=0}^{h-1} m_{t-i},$$

and m_t denotes the mid-price at time t. This definition has similar drawbacks to the previous one. But, additionally, we believe it may suffer from look-ahead bias. For example, if the mid-price has been going up in the past h steps (information that is included in our covariates), then it is more likely the return $r_{t,t+h}$ will be positive.

The definition of returns used in this paper, cf. Section 3.2.2, does not suffer from the issues identified for these two return specifications.

Remark A.4. The definition of returns intrinsically depends on the clock used to measure time. In Kolm et al. (2021), the authors use a physical time clock with stock-specific horizon h. In their setting, the authors use simple, un-smoothed mid-price returns,

$$r_{t,t+h} = \frac{m_{t+h} - m_t}{m_t}.$$

With our choice of order book-driven clock, even assuming mid-mid trading, there is no way of placing a trade exactly h order book events ahead, i.e., the prediction horizon is random. Smoothing the exit price can thus be understood as averaging out the uncertainty in the execution time. Moreover, in our work, we aim to investigate questions regarding structural market predictability. In our setting, smoothing mid-prices leads to better estimates of the true (latent) prices by weakening idiosyncratic noise effects. As pointed out in the conclusions in Section 5 though, it may be more appropriate to consider a physical time clock and un-smoothed returns in practical trading applications.

A.7. Infrastructure latency

In this paper, we explored the predictive value of order book data by assuming instantaneous access to such information. Due to technological constraints, market participants experience varying degrees of delay when engaging with the order book. Let us write $\lambda^{n,\text{view}}$ and $\lambda^{n,\text{act}}$ for the latencies that market participant n experiences when

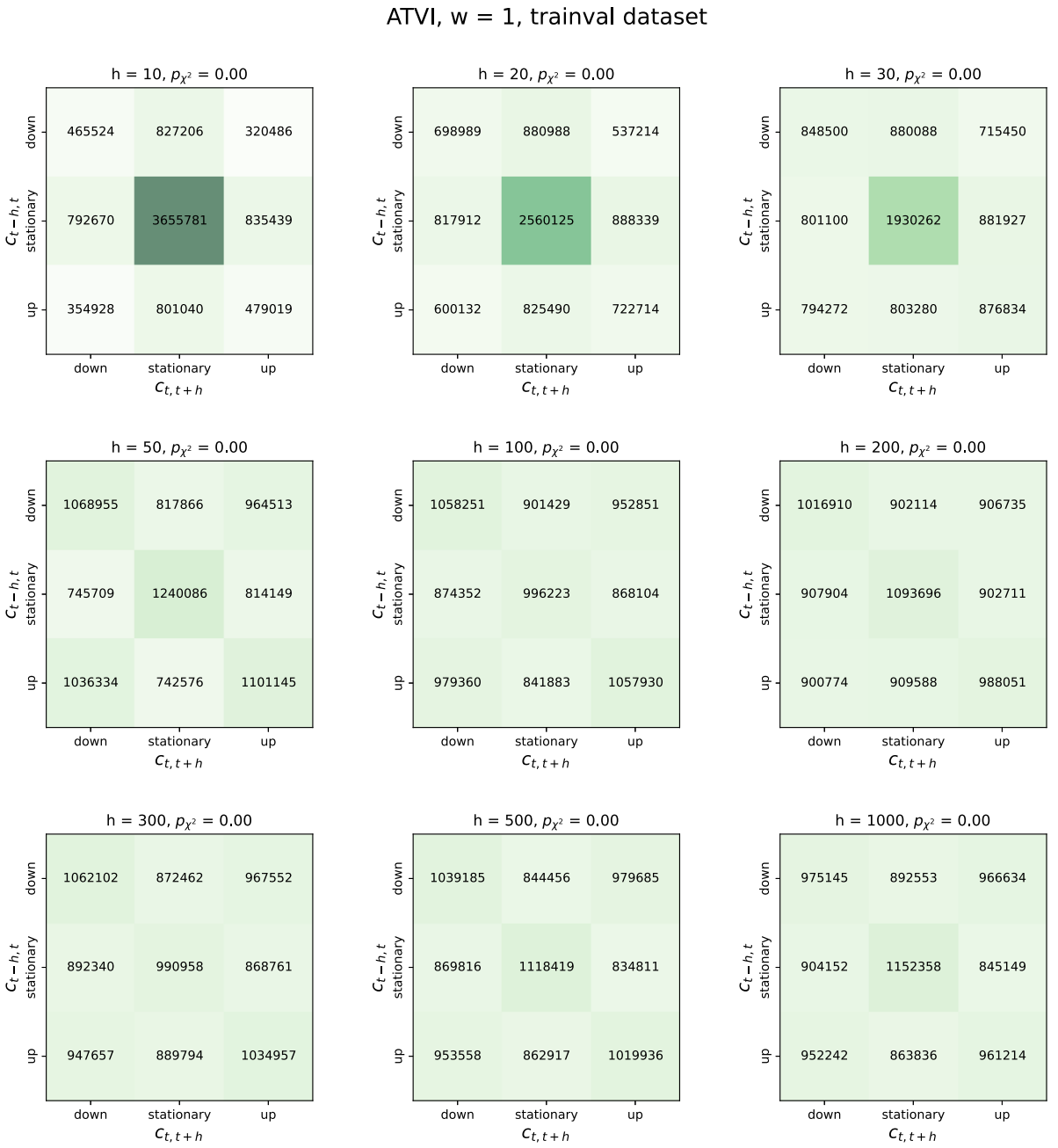


Fig. 20. ATVI return labels dependence in the joint training and validation data set during the first window of the experiment, i.e., from January 14th, 2019, to February 15th, 2019.

viewing and acting on the order book, respectively. For $x \in \{\text{view}, \text{act}\}$ we can decompose

$$\lambda^{n,x} = \lambda_0^x + \lambda_+^{n,x},$$

where λ_0^{act} is the time it takes the financial entity running the market, e.g., the Nasdaq, to execute an action on the order book once received and λ_0^{view} is the time it takes the market entity to send out the information regarding an order book update. Market participants make large efforts to reduce their idiosyncratic latencies $\lambda_+^{n,x}$ for $x \in \{\text{view}, \text{act}\}$ by optimizing software and hardware, one prototypical example being co-location. Note that $\lambda_+^{n,\text{act}}$ also includes the time needed to take a trading decision which, in our setting, requires a forward pass through a pre-trained predictive model, e.g., deepLOB/deepOF/deepVOL. Fig. 21 represents how market-wide and idiosyncratic latencies affect market participants.

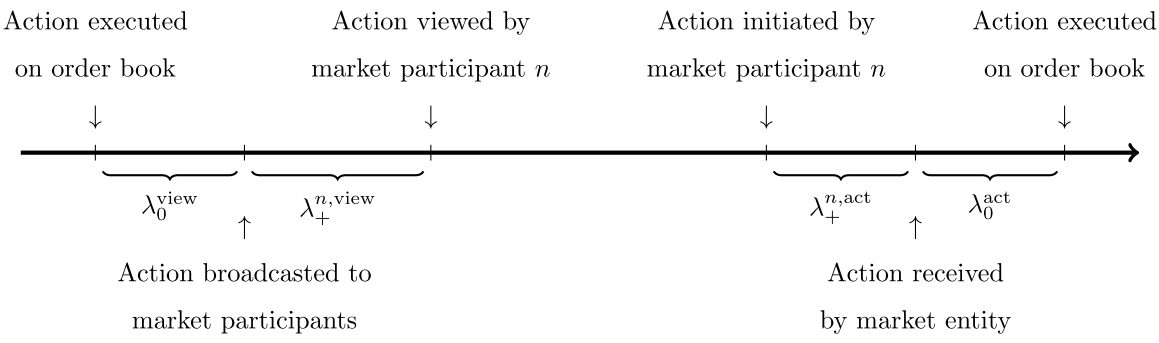


Fig. 21. Market-wide and idiosyncratic latencies.

When considering practical trading applications, one should thus take into account the effect these latencies may have on the definition of returns, cf. Section 3.2.2 and Appendix A.6. For example, denoting by $t = 1, 2, \ldots$ the order book-driven clock considered in this paper, one may want to predict the return between physical times $\tau_t + \lambda^{n,\text{tot}}$ and $\tau_{t+h} + \lambda^{n,\text{tot}}$ where

$$\lambda^{n,\text{tot}} = \lambda^{n,\text{view}} + \lambda^{n,\text{act}} = \lambda_0^{\text{tot}} + \lambda_+^{n,\text{tot}},$$

is the total round trip latency time to view and act once the order book clock ticks, and $\tau : \mathbb{N} \to [0, \infty)$ is the increasing random function mapping the order book clock to physical time. An interesting analysis in this setting would be to analyze how predictability varies as a function of $\lambda_+^{n,\text{tot}} = \lambda_+^{n,\text{view}} + \lambda_+^{n,\text{act}}$.

To quantify the magnitude of these time lags and better understand their impact, we briefly summarize some infrastructure tests published by market participants.

- As of 2023, the Nasdaq reports the door-to-door speed of its matching engine, i.e., $\lambda_0^{\text{tot}} = \lambda_0^{\text{act}} + \lambda_0^{\text{view}}$, to be 'sub-40 μs, with the fastest production implementation at 14 μs' (Nasdaq, 2023). Studies from 2015 report the door-to-door latencies of Brazilian and European futures exchanges in the order of a few hundred microseconds (Kirilenko & Lamacie, 2015; Menkveld & Zoican, 2017).
- Leading industry producers of co-located network adapters and feed handlers report the total speed for processing an order book message and sending a trade action, i.e., tick-to-trade latency $\lambda_+^{n,\text{tot}} = \lambda_+^{n,\text{view}} + \lambda_+^{n,\text{act}}$, to be of the order of a couple of microseconds (μs) (CSPi, 2023; Enyx, 2020). This is achieved via the use of specific hardware known as FPGA cards. These figures assume relatively simple trading decision applications; when working with more sophisticated predictive models, such as the ones considered in this paper, one should account for a couple of hundred microseconds in tick-to-trade latency, see Table III in Zhang et al. (2019) for unoptimized forward-pass run times.

The total latency $\lambda^{k,\text{tot}}$ is thus less than a fraction of a millisecond (ms), while the average time elapsed between consecutive price changes in our data ranges from 0.18 to 8.38 s depending on the liquidity of the stock, cf. Table 1. The effect of infrastructural latencies on the definition of returns, cf. Section 3.2.2, is thus negligible.

A.8. Stock selection

Ideally, we would like to work with the same set of stocks as in Kolm et al. (2021). Due to computational limitations, we could conduct experiments only on a subset of ten tickers from the total 115 NASDAQ stocks considered in the original paper. We selected ten stocks with diverse liquidity characteristics, hopefully providing a sufficiently representative sub-sample of the whole set.

We use the stock characteristics provided in Kolm et al. (2021), Table 6, to choose the sub-sample. For each liquidity characteristic – Updates, Trades, Price Changes, Spread – we compute a sub-score based on the characteristic's rank. For example, the stock with the most updates is assigned an update score of 1, while the stock with the least number of updates is assigned an update score of 0. The characteristic-specific scores are then averaged to obtain a general "liquidity score" for each stock. The ten chosen stocks correspond to the ten evenly spaced quantiles of the "liquidity score". The liquidity characteristics of the ten selected stocks are reported in Table 1.

A.9. Model details

In the experiments, we use the model architectures reported in Tables 13 and 14. To speed up the training procedure, we use batch normalization (Ioffe & Szegedy, 2015) (with momentum = 0.6) after every convolutional layer and inception block. To prevent overfitting, we use a dropout layer (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) (with noise = 0.2) positioned after the inception module.

All the models are implemented with float32 precision policy: float32 is used as a computation and variable data type at every layer. Loss and gradient computations are also carried out with float32 precision.

Table 13
DeepLOB and DeepOF network description.
(a) Single horizon, (b) Multi-horizon.

Model architectures							
Model		DeepLOB			DeepOF		
		Architecture	Description	Output size	Architecture	Description	Output Size
Input Layer			Raw order book states, $\left\{\left(p^{(l)}_{a,t-\tau}, v^{(l)}_{a,t-\tau}, p^{(l)}_{b,t-\tau}, v^{(l)}_{b,t-\tau}\right)_{l=1}^{L}\right\}_{\tau=0}^{T-1}$	$(T, 4L, 1)$		Multi-level order flow, $\left\{\left(aOF^{(l)}_{t-\tau}, bOF^{(l)}_{t-\tau}\right)_{l=1}^{L}\right\}_{\tau=0}^{T-1}$	$(T, 2L, 1)$
First Convolutional layer	Spatial convolution	32 (1×2) filters, (1×2) stride	Convolve price and volume information	$(T, 2L, 32)$			
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 2L, 32)$			
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 2L, 32)$			
Second Convolutional layer	Spatial convolution	32 (1×2) filters, (1×2) stride	Convolve bid and ask price-volume information	$(T, L, 32)$	32 (1×2) filters, (1×2) stride	Convolve order flow information	$(T, L, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, L, 32)$	32 (4×1) filters, padding	Convolve through four time-steps	$(T, L, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, L, 32)$	32 (4×1) filters, padding	Convolve through four time-steps	$(T, L, 32)$
Third Convolutional layer	Spatial convolution	32 $(1 \times L)$ filters, (1×1) stride	Convolve information through the whole book	$(T, 1, 32)$	32 $(1 \times L)$ filters, (1×1) stride	Convolve information through the whole book	$(T, 1, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$
Inception module (parallel blocks)	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$
	Temporal convolution	64 (3×1) filters, padding	Convolve through three time-steps, "MA(3)"	$(T, 1, 64)$	64 (3×1) filters, padding	Convolve through three time-steps, "MA(3)"	$(T, 1, 64)$
	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$
	Temporal convolution	64 (5×1) filters, padding	Convolve through five time-steps, "MA(5)"	$(T, 1, 64)$	64 (5×1) filters, padding	Convolve through five time-steps, "MA(5)"	$(T, 1, 64)$
	Temporal max pooling	(3×1) pool size, (1×1) stride & padding	Rolling maximum, three time-steps	$(T, 1, 32)$	(3×1) pool size, (1×1) stride & padding	Rolling maximum, three time-steps	$(T, 1, 32)$
	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	64 (4×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$
Concatenate and reshape			Concatenate and reshape the outputs of the inception module to obtain an $\mathbb{R}^{192}$-valued time series of length T	$(T, 192)$		Concatenate and reshape the outputs of the inception module to obtain an $\mathbb{R}^{192}$-valued time series of length T	$(T, 192)$
LSTM module	LSTM	64 hidden units	Capture longer term dependencies	(64)	64 hidden units	Capture longer term dependencies	(64)
(a) Output layer	Dense layer	Softmax activation	Produce classification output	(3)	Softmax activation	Produce classification output	(3)
(b) Decoder module	LSTM decoder (seq2seq/attention)	64 hidden units, softmax activation	Produce sequential classification output	(5, 3)	64 hidden units, softmax activation	Produce sequential classification output	(5, 3)

Table 14
DeepVOL and DeepVOL L3 network description.
(a) Single horizon, (b) Multi-horizon.

Model architectures							
Model		DeepVOL			DeepVOL L3		
		Architecture	Description	Output Size	Architecture	Description	Output size
Input Layer			Volume representation, $\left\{s^{(j)}_{x,t-\tau}\right\}_{\tau\in\{0,\dots,T-1\},\, j\in\{1,\dots,W\},\, x\in\{a,b\}}$	$(T, W, 2, 1)$		L3 volume representation, $\left\{q^{(j,k)}_{x,t-\tau}\right\}_{\tau\in\{0,\dots,T-1\},\, j\in\{1,\dots,W\},\, x\in\{a,b\},\, k\in\{1,\dots,D\}}$	$(T, W, 2, D, 1)$
First Convolutional layer	Spatial convolution				32 $(1 \times 1 \times$ D) filters, $(1 \times 1 \times 1)$ stride	At each price, convolve the queue volumes	$(T, W, 2, 32)$
Second Convolutional layer	Spatial convolution	32 $(1 \times 2 \times 2)$ filters, $(1 \times 1 \times 1)$ stride	Convolve volumes on opposite sides of mid	$(T, W-1, 32)$	32 $(1 \times 2 \times 2)$ filters, $(1 \times 1 \times 1)$ stride	Convolve volumes on opposite sides of mid	$(T, W-1, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, W-1, 32)$	32 (4×1) filters, padding	Convolve through four time-steps	$(T, W-1, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, W-1, 32)$	32 (4×1) filters, padding	Convolve through four time-steps	$(T, W-1, 32)$
Third Convolutional layer	Spatial convolution	32 $(1 \times W-1)$ filters, (1×1) stride	Convolve information through the whole book	$(T, 1, 32)$	32 $(1 \times W-1)$ filters, (1×1) stride	Convolve information through the whole book	$(T, 1, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$
Inception module (parallel blocks)	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$
	Temporal convolution	64 (3×1) filters, padding	Convolve through three time-steps, "MA(3)"	$(T, 1, 64)$	64 (3×1) filters, padding	Convolve through three time-steps, "MA(3)"	$(T, 1, 64)$
	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$
	Temporal convolution	64 (5×1) filters, padding	Convolve through five time-steps, "MA(5)"	$(T, 1, 64)$	64 (5×1) filters, padding	Convolve through five time-steps, "MA(5)"	$(T, 1, 64)$
	Temporal max pooling	(3×1) pool size, (1×1) stride & padding	Rolling maximum, three time-steps	$(T, 1, 32)$	(3×1) pool size, (1×1) stride & padding	Rolling maximum, three time-steps	$(T, 1, 32)$
	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	64 (4×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$
Concatenate and reshape			Concatenate and reshape the outputs of the inception module to obtain an $\mathbb{R}^{192}$-valued time series of length T	$(T, 192)$		Concatenate and reshape the outputs of the inception module to obtain an $\mathbb{R}^{192}$-valued time series of length T	$(T, 192)$
LSTM module	LSTM	64 hidden units	Capture longer term dependencies	(64)	64 hidden units	Capture longer term dependencies	(64)
(a) Output layer	Dense layer	Softmax activation	Produce classification output	(3)	Softmax activation	Produce classification output	(3)
(b) Decoder module	LSTM decoder (seq2seq/attention)	64 hidden units, softmax activation	Produce sequential classification output	(5, 3)	64 hidden units, softmax activation	Produce sequential classification output	(5, 3)

References

Abergel, F., Lehalle, C.-A., & Rosenbaum, M. (2014). Understanding the stakes of high-frequency trading. *The Journal of Trading*, *9*, 49–73. http://dx.doi.org/10.3905/jot.2014.9.4.049, URL: https://jot.pm-research.com/content/9/4/49, arXiv:https://jot.pm-research.com/content/9/4/49.full.pdf.

Aït-Sahalia, Y., Fan, J., Xue, L., & Zhou, Y. (2022). *How and when are high-frequency stock returns predictable? NBER working papers 30366*, National Bureau of Economic Research, Inc., URL: https://ideas.repec.org/p/nbr/nberwo/30366.html.

Bacry, E., & Muzy, J.-F. (2014). Hawkes model for price and trades high-frequency dynamics. *Quantitative Finance*, *14*, 1147–1166. http://dx.doi.org/10.1080/14697688.2014.897000, URL: http://www.tandfonline.com/doi/abs/10.1080/14697688.2014.897000.

Bengio, Y., Courville, A. C., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, *35*, 1798–1828. http://dx.doi.org/10.1109/TPAMI.2013.50, URL: https://ieeexplore.ieee.org/document/6472238.

Byrd, D., Hybinette, M., & Balch, T. H. (2020). ABIDES: Towards high-fidelity multi-agent market simulation. In *Proceedings of the 2020 ACM SIGSIM conference on principles of advanced discrete simulation* (pp. 11–22). Miami FL Spain: ACM, http://dx.doi.org/10.1145/3384441.3395986, URL: https://dl.acm.org/doi/10.1145/3384441.3395986.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., et al. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing* (pp. 1724–1734). Association for Computational Linguistics., http://dx.doi.org/10.3115/v1/D14-1179, URL: https://aclanthology.org/D14-1179.

Coletta, A., Moulin, A., Vyetrenko, S., & Balch, T. (2022). Learning to simulate realistic limit order book markets from data as a world agent. In *Proceedings of the third ACM international conference on AI in finance* (pp. 428–436). New York NY USA: ACM, http://dx.doi.org/10.1145/3533271.3561753, URL: https://dl.acm.org/doi/10.1145/3533271.3561753.

Cont, R., Kukanov, A., & Stoikov, S. (2013). The price impact of order book events. *Journal of Financial Econometrics*, *12*, 47–88. http://dx.doi.org/10.1093/jjfinec/nbt003, arXiv:https://academic.oup.com/jfec/article-pdf/12/1/47/2439285/nbt003.pdf.

CSPi (2023). Tick-to-trade latency. URL: http://www.cspi.com/wp-content/uploads/2016/06/Tick-to-Trade-Latency_FINAL-2.pdf. (Accessed 30 August 2023).

Cybenko, G. V. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*, 303–314.

Enyx (2020). NASDAQ totalview latency tests. URL: www.enyx.com/performance-reports/nxfeed/NASDAQ-Totalview/index.html. (Accessed 30 August 2023).

European Parliament and Council (2004). Directive 2004/39/EC on markets in financial instruments amending Council Directives 85/611/EEC and 93/6/EEC and Directive 2000/12/EC of the European Parliament and of the Council and repealing Council Directive 93/22/EEC.

Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, *25*, 383–417, URL: http://www.jstor.org/stable/2325486.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press, http://www.deeplearningbook.org.

Hansen, P. R., Lunde, A., & Nason, J. M. (2011). The model confidence set. *Econometrica*, *79*, 453–497. http://dx.doi.org/10.3982/ECTA5771, URL: https://onlinelibrary.wiley.com/doi/abs/10.3982/ECTA5771, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.3982/ECTA5771.

Harvey, A., & Bates, D. (2003). *Multivariate unit root tests and testing for convergence*: *Cambridge working papers in economics 0301*, Faculty of Economics, University of Cambridge, URL: https://ideas.repec.org/p/cam/camdae/0301.html.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE conference on computer vision and pattern recognition* (pp. 770–778). Los Alamitos, CA, USA: IEEE Computer Society, http://dx.doi.org/10.1109/CVPR.2016.90, URL: https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.90.

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: The difficulty of learning long-term dependencies.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780. http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Hornik, K., Stinchcombe, M., & White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, *3*, 551–560. http://dx.doi.org/10.1016/0893-6080(90)90005-6, URL: https://www.sciencedirect.com/science/article/pii/0893608090900056.

Huang, R., & Polak, T. (2011). LOBSTER: Limit order book reconstruction system. *Information Systems & Economics eJournal*.

Imperial College Research Computing Service (2022). High-performance computing cluster. http://dx.doi.org/10.14469/hpc/2232.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of machine learning research*: *vol. 37*, *Proceedings of the 32nd international conference on machine learning* (pp. 448–456). Lille, France: PMLR, URL: https://proceedings.mlr.press/v37/ioffe15.html.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. CoRR, abs/1412.6980.

Kirilenko, A. A., & Lamacie, G. (2015). Latency and asset prices. http://dx.doi.org/10.2139/ssrn.2546567, URL: https://papers.ssrn.com/abstract=2546567.

Kolm, P. N., Turiel, J. D., & Westray, N. (2021). Deep order flow imbalance: Extracting alpha at multiple horizons from the limit order book. *Econometric Modeling: Capital Markets - Portfolio Theory eJournal*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Eds.), *Advances in neural information processing systems, vol. 25*. Curran Associates, Inc., URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Large, J. (2007). Measuring the resiliency of an electronic limit order book. *Journal of Financial Markets*, *10*, 1–25. http://dx.doi.org/10.1016/j.finmar.2006.09.001, URL: https://www.sciencedirect.com/science/article/pii/S1386418106000528.

Luong, T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 1412–1421). Lisbon, Portugal: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/D15-1166, URL: https://aclanthology.org/D15-1166.

Melis, G., Dyer, C., & Blunsom, P. (2017). On the state of the art of evaluation in neural language models. CoRR, abs/1707.05589. URL: http://arxiv.org/abs/1707.05589.

Menkveld, A. J., & Zoican, M. A. (2017). Need for speed? Exchange latency and liquidity. *The Review of Financial Studies*, *30*, 1188–1228. http://dx.doi.org/10.1093/rfs/hhx006, URL: https://academic.oup.com/rfs/article/30/4/1188/2966376.

Nasdaq (2023). Exchange trading technology. URL: www.nasdaq.com/solutions/nasdaq-trading-technology/exchange-matching. (Accessed 30 August 2023).

Ntakaris, A., Magris, M., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2018). Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. *Journal of Forecasting*, *37*, 852–866. http://dx.doi.org/10.1002/for.2543, URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/for.2543, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/for.2543.

Nyblom, J., & Harvey, A. (2000). Tests of common stochastic trends. *Econometric Theory*, *16*, 176–199, URL: http://www.jstor.org/stable/3533193.

Sirignano, J., & Cont, R. (2019). Universal features of price formation in financial markets: Perspectives from deep learning. *Quantitative Finance*, *19*, 1449–1459. http://dx.doi.org/10.1080/14697688.2019.1622295.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958, URL: http://jmlr.org/papers/v15/srivastava14a.html.

Telgarsky, M. (2015). Representation benefits of deep feedforward networks. arXiv e-prints arXiv:1509.08101.

Tran, D. T., Iosifidis, A., Kanniainen, J., & Gabbouj, M. (2019). Temporal attention-augmented bilinear network for financial time-series data analysis. *IEEE Transactions on Neural Networks and Learning Systems*, *30*, 1407–1418. http://dx.doi.org/10.1109/TNNLS.2018.2869225, URL: https://ieeexplore.ieee.org/document/8476227/.

US Securities and Exchange Commission (2005). Regulation NMS - rules and amendments to joint industry plans.

Wu, Y., Mahfouz, M., Magazzeni, D., & Veloso, M. (2021). *Towards robust representation of limit orders books for deep learning models*: *Papers 2110.05479*, arXiv.org., URL: https://ideas.repec.org/p/arx/papers/2110.05479.html.

Xu, K., Gould, M. D., & Howison, S. D. (2019). Multi-level order-flow imbalance in a limit order book. *Market Microstructure and Liquidity*, *4*.

Zhang, Z., & Zohren, S. (2021). *Multi-horizon forecasting for limit order books: Novel deep learning approaches and hardware acceleration using intelligent processing units*: *Papers 2105.10430*, arXiv.org., URL: https://ideas.repec.org/p/arx/papers/2105.10430.html.

Zhang, Z., Zohren, S., & Roberts, S. (2019). DeepLOB: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, *67*, 3001–3012. http://dx.doi.org/10.1109/TSP.2019.2907260.