LOBIN: In-Network Machine Learning for Limit Order Books

Xinpeng Hong, Changgang Zheng, Stefan Zohren, and Noa Zilberman
Department of Engineering Science, University of Oxford, Oxford, United Kingdom
{xinpeng.hong, changgang.zheng, stefan.zohren, noa.zilberman}@eng.ox.ac.uk

***Abstract*—Machine learning is driving the evolution of algorithmic trading, but the demands for fast execution speed remain. Although both aim to increase profitability, embedding more powerful machine learning approaches and lowering trading latencies are hard to achieve simultaneously. Offloading machine learning inference to programmable network devices, also referred to as in-network machine learning, provides a delicate balance between the two ends of this trade-off. In this paper, we present LOBIN, providing machine learning based market prediction using high-frequency market data feeds. LOBIN builds limit order books and conducts inference within programmable switches. Compared with server-based solutions, LOBIN predicts future stock price movements with lower latency, higher throughput, and a minor impact on machine learning performance.**

***Index Terms*—In-network computing, machine learning, programmable switches, P4, microstructure market data, limit order books, time series prediction.**

I. Introduction

Algorithmic trading has been growing over the past few decades as financial firms automate processes traditionally done by human traders. It uses computer algorithms to automatically execute orders under preset trading instructions [1]. As an essential form of algorithmic trading, high-frequency trading (HFT) is characterized by placing larger numbers of orders within a minimum time and being able to react quickly under changing market conditions [2]. The latency of the HFT market participants has been measured in microseconds [3].

The rise of artificial intelligence further drives the growth of high-frequency algorithmic trading, with machine learning (ML) approaches becoming widespread in the field of HFT [4], [5]. However, the increasing complexity of ML models used also challenges existing trading systems, creating a demand for latency reduction throughout the trading process.

A common HFT problem is predicting future price movements from market microstructure signals, which has been proven to be feasible and profitable [4]. There are a number of previous works focusing on market forecasting utilizing electronic limit order books (LOBs) combined with ML models [6]–[9]. For a particular stock, a real-time LOB is constructed from unmatched limit orders that are predetermined with specific prices. It contains a wealth of information that can be used as ML features [10].

This work was partly funded by VMware and we acknowledge support from Intel. Stefan Zohren thanks the Oxford-Man Institute of Quantitative Finance for financial support. We also thank Zihao Zhang for the valuable discussions on this work.

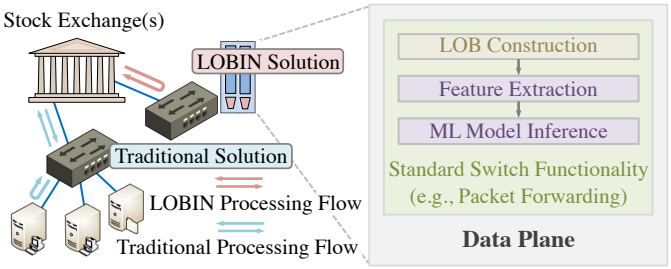


Fig. 1. General working scenario of LOBIN.

In-network computing offloads applications to run into programmable network devices [11]. In-network ML, as a specific type of in-network computing, deploys pre-trained ML models within network devices and conducts inference there for lower latency, higher throughput, and more efficient power utilization [12], [13]. By design, in-network ML provides a practical solution for reducing the latency of time-sensitive financial applications, such as trading scenarios.

As nearly no previous works focus on time-sensitive applications of in-network ML, this paper provides an in-depth study on the application of in-network ML to market prediction using LOBs. We design and implement LOBIN, an in-network prototype for future price movement prediction by maintaining a LOB based on market-by-order (MBO) data feeds. LOBIN stands for "Limit Order Books In Network". Figure 1 shows a general working scenario of LOBIN compared with traditional solutions. Deployed on a Tofino [14] hardware switch, Tofino 2 (emulation environment), and a BMv2 [15] software switch model, LOBIN demonstrates the feasibility of the solution. According to the evaluation results, LOBIN can achieve a minimal loss of prediction performance compared to server-based implementation while lowering latency to the microsecond level.

The main contributions of this paper are as follows:

- We study the application of in-network ML to market prediction from micro-structure data and provide a proof of concept demonstrating its feasibility. This is the first of its kind to practically explore time-sensitive applications of in-network ML.
- We design and implement a prototype that builds and updates LOBs in the programmable data plane based on high-frequency market data feeds. We integrate the workflow of building and updating LOBs with ML-related processes, deploying it on both hardware and software programmable network devices.

- We evaluate the prototype within a local networked-system testbed in terms of both ML prediction performance and networking performance. We compare different ML models, different stock sections, and different solutions including the benchmark running on servers. The prototype presents the benefit of latency reduction and the capability of maintaining ML-based functionalities.

II. Background and Motivation

In this section, we provide background information about basic concepts and related works from three different domains. We also motivate the need for exploring the cross-cutting area positioned at the intersection field of ML, programmable networking, and financial modeling.

A. ML for Market Prediction

Centering on buying and selling assets in the marketplace, financial trading is an area where the application of ML approaches has become mainstream. Since financial time series data are inherently non-stationary and nonlinear, containing high noise [16], the applicability of traditional statistical methods is often constrained when dealing with this type of data. Aiming for profitability, ML models have been proven to have the capability to help with almost every point in the trading process, including forecasting price movement, generating trading signals, optimizing order execution, and making trading decisions [5].

Among all aforementioned tasks, predictions on stock market price movement remain a big challenge because future fluctuations are influenced by countless internal and external factors, such as economic factors, investor sentiment, company performance, and industry performance [17]. To perform better in financial forecasting, researchers have applied and assessed different ML approaches based on historical data to accomplish more accurate predictions [18]–[20].

ML model complexity is continuously growing, using massive financial data more efficiently, laying an increasingly high burden on traditional processor-based platforms and leading to a decrease in computational speed. However, in the scope of HFT, intense competition among traders requires lightning-speed real-time trade execution. Low latency is significantly crucial for generating profits because many of them are often based on short-lived opportunities such as cross-market arbitrage and breaking news [5]. In today's competition, a small fraction of trading firms with the fastest HFT systems continues to amass a large share of trading revenues [21]. Therefore, accelerating ML-based market prediction remains a research challenge.

B. Network Devices for Trading Acceleration

To achieve lower latency, researchers focus not only on the optimization of trading algorithms and strategies but also on system-level solutions based on software design and hardware implementation. To date, different network devices have been used for trading acceleration, including application-specific integrated circuits (ASICs), graphic processing units (GPUs), or field-programmable gate arrays (FPGAs) [22]. Some studies used FPGAs for accelerating market data feed processing or trading applications, e.g. [23], [24] while others also utilized customized network interface cards (NICs) and optimized software for lower latency [25]. Significant progress was driven by industry in developing hardware-based acceleration which can support financial services applications, e.g. [26], [27]. However, none of the previous works has attempted to accelerate ML-based market prediction *for making trading decisions within the network itself*. Given the innovative benefits that in-network computing and in-network ML can provide in terms of latency reduction, they may become a potential solution for this gap. If proven to be beneficial, a new generation of trading systems may be derived in the future. Since ASICs can offer superior performance such as fast processing speed compared to their counterparts [28], we choose to use switching ASICs for the lowest end-to-end latency.

C. In-Network ML

As the root of in-network computing, network programmability has facilitated network evolution. The emergence of software-defined networking (SDN) enabled networks to be intelligently controlled through software applications [29]. A specialized language, Programming Protocol Independent Packet Processors (P4), was developed for configuring how network devices process and forward packets within the data plane [30]. Protocol-Independent Switching Architecture (PISA) serves as a common data plane programming model in P4 based on a programmable match-action pipeline [31].

Some P4 targets, such as BMv2 and P4Pi, use standard CPU to run packet forwarding programs [32], while some others are based on hardware including FPGAs, switches, and NICs. P4 programs run on these target devices for packet forwarding and computing, offering programmability within the data plane. This provides a seedbed for offloading server applications to programmable network devices, which drove the emergence of in-network computing.

In-network computing offloads applications in part, or in full, to the data plane. It takes advantage of lower overhead in space, energy, and cost domain, as well as higher process efficiency of network devices [11]. In-network computing has been applied in several areas, including caching [33], DNS [11], and distributed systems [34]. Some studies focused on benefiting ML applications by offloading some parts of ML functions into the data plane such as feature extraction and weight aggregation [35], [36]. To date, a number of ML models have been implemented in the field of in-network ML, making its application to different fields possible [37], [38]. However, the range of in-network ML use cases is still limited and needs further exploration and extension. In fact, most relevant works focused on fields such as security and anomaly detection [39], [40], and the domain of financial trading is barely explored. Our previous work considered market prediction, but only using stateless messages (MBO) [38]. Our poster Linnet [41], proposed building LOBs, but only considered software switches, significantly different from resource-

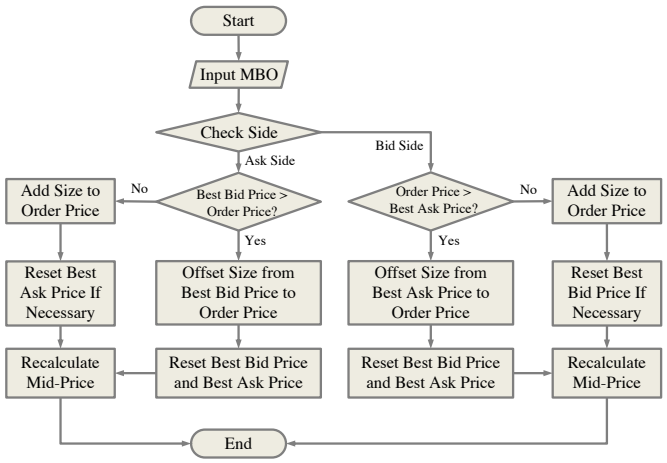


Fig. 2. Workflow of updating a LOB with MBO feeds.

constrained switch-ASIC. The work presented in this paper fills these gaps.

III. Overview of LOBIN

This section provides an overview of our solution, LOBIN, with its key ideas and technical design explained.

A. Limit Order Books

Market by order (MBO) data is an order-based data feed that provides the details of each trade instruction for a certain stock [42]. It contains an order's timestamp, unique identifier, action (whether to add a new order, cancel an existing order, or update the price or quantity for the existing order), side (whether to buy or sell the given security), price, and quantity. Implicitly derived from MBO data, LOBs present a collection of unmatched limit orders which is waiting to be executed at pre-specified or better price levels [43]. A limit order used to buy an asset at or below a pre-specified price is also called a bid order. In contrast, an ask order (or an offer) is to sell an asset at or above a given price level [8]. In a LOB, the two types of limit orders reside in the bid side and ask side correspondingly. The midpoint of the highest bid price and the lowest ask price is called the mid-price.

In most of today's securities markets, bid orders and ask orders are executed following the principle of price/time priority [44], [45]. This principle demonstrates how limit orders are prioritized for execution. More specifically, orders are operated firstly based on the best price, and if the specified prices of multiple orders are the same, the priority is given to the earliest one, breaking ties.

When a new order is to be executed in the matching engine, the information it provides is used to determine how to update the LOB. Figure 2 shows the workflow of updating the LOB with MBO data feeds. Taking an ask order as an example to illustrate the process, its limit price is compared to the best bid price. If the price of the new order is higher, the new order will be unmatched and reside at its price level of the LOB. If the price of the new order is equal to or lower than the bid, meaning that it crosses the bid-ask spread, it will be matched with the unexecuted order(s) at the best bid price level. When there is still some quantity of the new order remaining after matching, the remaining quantity will be matched with the unexecuted order(s) at the lower price level(s), as long as their price is bigger or equal to the price of the limit order. If all the unexecuted order(s) whose price level(s) is (are) higher than the price of the new order has (have) been matched, the remaining quantity of the new order will reside at its price level as an offer. The best bid price and the best ask price within the LOB are updated if needed – those changes result in the volatility of the mid-price. Updating the LOB with a bid order is similar.

Figure 3 illustrates how MBO data update a LOB using four examples. The first column presents how a new ask order with the price P_4 updates a LOB. Since P_4 is on the ask side, the quantity of the order settles with the existing volume at P_4. Similarly, the second column shows that the size of a new bid order with the price P_1 resides at P_1 of the LOB. The third column demonstrates the situation when a new bid order with a price higher than the best offer is used to update the LOB. In the example, the price of the bid order is P_5 while the existing best offer is P_4. The new order is matched with the existing orders at P_4 first, and then with the ones at P_5 if the unexecuted orders at P_4 are not sufficient for matching. After execution, the best offer becomes P_5, and thus the mid-price changes from the average of P_3 and P_4 to P_4. Then in the last column, the LOB needs to be updated with a new ask order at the price P_3. Since it equals to current best bid of the LOB, the unexecuted orders that can be matched with it are only the ones at P_3. If they are not enough to match the whole quantity of the new order, the remaining volume of it then resides at P_3 and P_3 becomes the best offer. The mid-price then changes to the average of P_2 and P_3 as the figure shows.

B. LOBIN's Design

LOBIN accelerates stock price movement prediction through the construction and updating of LOBs in the programmable data plane. For a given individual stock, when a new MBO message is received, LOBIN updates the LOB. It then extracts features, obtaining information on both price and quantity from different levels on the bid and ask sides of the LOB. The mid-price at each point in time is also used to create labels, representing the direction of price changes.

Figure 4 shows the system architecture of LOBIN, including components on the server, control plane, and data plane. In the first step, running on a server, historical market data feeds are used to train an ML model, using LOB-based features. The trained model is mapped to the data plane, and table entries are generated. The mapping includes generating a P4 program, which contains both the mapped ML inference model and LOB-related logic: LOB construction and update, and feature extraction. The generated P4 program is loaded to the programmable data plane, while table entries are loaded through the control plane. This design enables obtaining prediction results directly within the data plane, based on the trained ML model.

Fig. 3. An illustration of how MBO data updates a LOB.

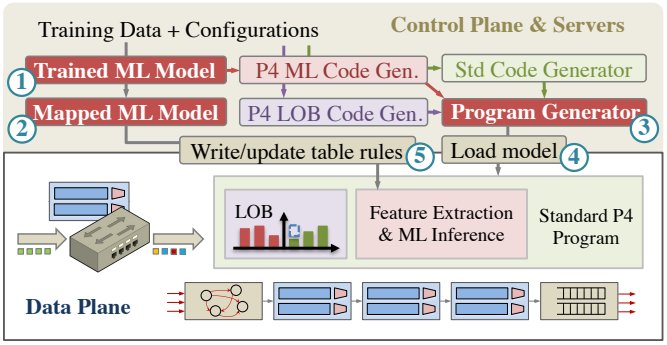


Fig. 4. System design of LOBIN.

IV. Implementation

We implement LOBIN using P4 on BMv2 and Intel Tofino switch-ASIC. Both implementations are based on the same concept: the LOB is initialized with an equal volume at each price level, before transactions are executed. For each incoming MBO, the LOB is updated. After each message, the mid-price is computed based on the highest bid and the lowest offer at that timestamp, and LOB status features are extracted and used for ML inference.

Programmable data planes are limited by different constraints, such as limited amount of memory, limited number of stages, and limited operation. This section explains using pseudocode the update process of a LOB within a data plane, and the solutions used by LOBIN to overcome the key challenges to enabling the logic within switch hardware.

A. Behavioral Implementation

BMv2 is an open-source behavioral software switch maintained by the P4 workgroups that serves as a target. BMv2 can be used in different development environments, such as P4Pi [32] and eBPF. It is often used as a reference switch for functionality evaluation. As BMv2 is not as constrained as switch-ASICs, it is used as the first P4 data plane target for the general LOBIN process shown in Figure 2. Algorithm 1 presents the pseudocode of the updating process of a LOB with a bid order. The algorithm of updating a LOB with an ask order is similar, and omitted for brevity.

The algorithm uses three registers (R_a, R_b, and R_l) to store the lowest ask price, the highest bid price, and current state of the LOB (the current quantity at each price level), respectively. These variables are stateful, and need to be maintained and updated over time. Other variables in Algorithm 1 are user-defined metadata, used as intermediates to update the LOB state, as well as the lowest ask price and the highest bid price if necessary.

B. Tofino Implementation

Intel Tofino is a P4-based programmable switch ASIC, which utilizes the Tofino Native Architecture (TNA), an architecture that is very similar to Portable Switch Architecture (PSA) used in BMv2. Tofino offers a Tbps bit rate, with sub-microsecond latency [46], which can potentially be deployed at the network edge or access and provide LOB services. However, similar to other hardware devices, it has strong resource constraints such as the number of stages or the amount of memory [47]. While BMv2 is suitable for P4 prototypes, using Tofino indicates the feasibility of using commodity off-the-shelf devices in a real-world environment.

The constraints of Tofino's ASIC design, mean that the general solution previously described needs to be adapted to the hardware. To overcome the limitations of the platform, the process of updating the LOB with MBO data should be modified. The main implementation challenges are:

- **Lack of loops:** LOBIN's workflow requires loops for several purposes. For instance, to locate maximum bid, minimum offer, or when executing a new bid order which needs to be served by multiple offer levels in the LOB, due to its quantity. While loop unrolling is one solution, its cost is resources and stage consumption is high.
- **Cost of comparisons:** LOB construction and maintenance require a lot of price and order volume comparisons. For example, for an incoming offer, its price needs to be compared with the best bid and the best offer. If an offer is matched with a bid, their quantities need to be compared, and any remaining unmatched quantity needs to be handled. On hardware, each comparison consumes a processing stage and reduces scalability.

Algorithm 1 Update a LOB with a new bid order (General)

- R_l: An array tracking LOB states (Register Array).
- R_a: Position of lowest ask price (Register).
- R_b: Position of highest bid price (Register).
- I: Position of price levels in LOB.
- P_a: Position of lowest ask price (Metadata).
- P_b: Position of highest bid price (Metadata).
- P_o: Price level position of a new order.
- S_o: Size (unexecuted) of a new order.

function UPDATEMINASK(I)
  while $R_l[I] == 0$ **do**
    $I \leftarrow I + 1$
  $P_a \leftarrow I$ ▷ Update the lowest ask price.

function MAIN(P_o, S_o)
  $P_a, P_b \leftarrow R_a, R_b$ ▷ Read values from registers.
  if $P_o < P_a$ **then** ▷ If order resides in LOB.
    $R_l[P_o] \leftarrow R_l[P_o] + S_o$ ▷ Update LOB.
    if $P_o > P_b$ **then**
      $P_b \leftarrow P_o$ ▷ Update the highest bid price.
  else ▷ Order needs to be matched.
    for $I = P_a, \ldots, P_o$ **do** ▷ Iteration for matching.
      if $R_l[I] >= S_o$ **then** ▷ LOB order size at the I_{th} price level is sufficient to match.
        $R_l[I] \leftarrow R_l[I] - S_o$ ▷ Update LOB.
        UPDATEMINASK(I)
        break
      else ▷ LOB order size at the I_{th} price level is insufficient to match.
        $S_o \leftarrow S_o - R_l[I]$ ▷ New unexecuted size.
        $R_l[I] \leftarrow 0$ ▷ Update LOB.
        if $I == P_o$ **then**
          $R_l[I] \leftarrow S_o$ ▷ Unmatched size of the new order resides in LOB.
          $P_b \leftarrow I$ ▷ Update the highest bid.
          UPDATEMINASK($I + 1$)
  $R_a, R_b \leftarrow P_a, P_b$ ▷ Write values back to registers.

- **Registers access:** LOB must maintain state over MBO messages and be quickly updated, which requires the use of registers. However, a register can be accessed only once in the pipeline, and it is not allowed to read a register at the beginning of the pipeline and then write an updated value back at the end of the pipeline. To overcome this challenge, recirculation can be used, with one pass for reading and the second for writing.

In addition to the challenges above, the sequential call to $if - else$ conditions, as used in Algorithm 1, can easily exceed the maximum number of pipeline stages. Consequently, LOBIN's processing flow is modified from a hierarchical structure into a flatter, parallel design.

The key idea in the solution is using regular logic to update the quantity at a single price level in the LOB, and operating

Algorithm 2 Update a LOB with a new bid order on Tofino

- F: Recirculation flag (1 if a recirculated message).
- P_h: Highest price level position in LOB (Constant).
- S_I: Order size at the I^{th} price level in LOB.
- a, b, c: Temporary variables for comparisons.
- Other variable definitions are as in Algorithm 1.

function OPERATION(I)
  $S_I \leftarrow R_l[I]$ ▷ Read order size at the I^{th} price level.
  $a \leftarrow I - P_a$ ▷ Distance from lowest ask price.
  $b \leftarrow P_o - I$ ▷ Distance from new order's price.
  $c \leftarrow S_o - S_I$ ▷ Gap between new order's unexecuted size and LOB size at the I^{th} price level.
  if $a \geq 0$ **and** $b > 0$ **and** $c \geq 0$ **then** ▷ Order size at the I^{th} price level is insufficient to match new order.
    $S_I \leftarrow 0$ ▷ Update LOB size.
    $S_o \leftarrow c$ ▷ Update unexecuted order size.
  else if $a \geq 0$ **and** $b == 0$ **and** $c > 0$ **then** ▷ The unmatched size of the new order resides in LOB.
    $S_I \leftarrow c$ ▷ Update LOB size.
    $P_b \leftarrow I$ ▷ Update the highest bid price.
    $P_a \leftarrow I + 1$ ▷ Update the lowest ask price.
  else if $a \geq 0$ **and** $b == 0$ **and** $c == 0$ **then** ▷ Order size at the I^{th} price level is just enough to match.
    $S_I \leftarrow 0$ ▷ Update LOB size.
    $P_a \leftarrow I + 1$ ▷ Update the lowest ask price.
  else if $a \geq 0$ **and** $b \geq 0$ **and** $c < 0$ **then** ▷ Order size at the I^{th} price level is more than sufficient to match.
    $S_I \leftarrow S_I - S_o$ ▷ Update LOB size.
    $P_a \leftarrow I$ ▷ Update the lowest ask price.

function MAIN(P_o, S_o)
  if $F == 0$ **then**
    $P_a \leftarrow R_a$ ▷ Read the current lowest ask price from register.
    $P_b \leftarrow R_b$ ▷ Read the current highest bid price from register.
    if $P_o \leq P_b$ **then** ▷ If order resides in LOB.
      $R_l[P_o] \leftarrow R_l[P_o] + S_o$ ▷ Update LOB.
    else if $P_b < P_o < P_a$ **then** ▷ If order resides in LOB and the highest bid price needs updating.
      $R_l[P_o] \leftarrow R_l[P_o] + S_o$ ▷ Update LOB.
      $P_b \leftarrow P_o$ ▷ Update the highest bid price.
    else if $P_a \leq P_o$ **then** ▷ Order matches.
      for $I = 0, \ldots, P_h$ **do** ▷ Iteration for operation.
        OPERATION(I)
    $F \leftarrow 1$ ▷ Update recirculation flag.
  else if $F == 1$ **then**
    $R_a \leftarrow P_a$ ▷ Write back the updated lowest ask price to register.
    $R_b \leftarrow P_b$ ▷ Write back the updated highest bid price to register.
    for $I = 0, \ldots, P_h$ **do**
      $R_l[I] \leftarrow S_I$ ▷ Write back updated LOB size.

through all price levels using the same logic, regardless of the current order's price. The algorithm compares each price level with the price of the new bid order and the current minimum offer. It also compares the quantity of each price level with the order size to be matched. Based on the comparisons' results, it decides if and how to operate on the current price level, as well as if to update the best bid and the best offer.

Algorithm 2 shows the pseudocode of updating a LOB with a bid order on Tofino. Updating an ask order follows a similar process. While line 22 of Algorithm 1 and line 36 and 42 of Algorithm 2 use *for* commands, these are only for illustration purposes, and loop unrolling is applied in practice.

The P4 implementation of LOBIN integrates the LOB code with ML inference code generated using Planter [38]. LOB generation requires 622 lines of P4 code for BMv2 and 680 lines for Tofino. Automatically generated inference code varies across ML models, but for example, a decision tree (DT) model requires 390 lines for BMv2 and 165 lines for Tofino. Architecture-related code requires less than 100 lines for both targets. The implementation also supports Tofino 2, without changes to either LOB or Planter code.

V. Evaluation

The evaluation of LOBIN covers both aspects of ML performance and system performance, in comparison with server-based benchmarks.

A. Experimental Setup

An APS-Networks BF6064T-X Intel Tofino switch is used for evaluation. The 64×100G ports switch runs SDE 9.4.0. Software development, including support for Tofino 2, uses SDE 9.9.0 running on a server. Server-based experiments, including traffic generation, are conducted with two ASUS ESC4000A-E10 servers, equipped with AMD EPYC 7302P CPU and 256GB DDR4 RAM, using Ubuntu 20.04 LTS. Both servers are connected to the switch using NVIDIA ConnectX-5 100G NICs and direct attach cables.

The dataset used in this study is NASDAQ's Historical TotalView-ITCH sample data feeds [48]. MBO messages for a number of NASDAQ stocks are extracted for the date 2019-03-27. In the data source, there are no available data feeds spanning two or more consecutive days, so we use data from a single date. An open-source tool [49] is used for reconstructing MBO messages of one specific stock. The Planter [38] framework is used for in-network ML deployment.

B. Prediction Performance

Model prediction performance is tested for several different ML algorithms that are commonly used for forecasting future stock price movement, including k-means (KM), k-nearest neighbors (KNN), decision trees (DTs), random forests (RFs), and extreme gradient boosting (XGB). A smoothing labeling method [7] is used to draw more consistent signals from highly-stochastic financial data feeds. Synthetic minority over-sampling technique (SMOTE) is applied to address the class-imbalance problem [50].

Three stocks are picked for the evaluation, representing companies in the NASDAQ Composite index with the largest market capitalization in three different sectors: Paypal (PYPL) - financials, PepsiCo (PEP) - consumer staples, and Alphabet (GOOGL) - communication services. Add-order messages of these three stocks are used.

ML model training uses up to ten price level volumes of a LOB and the mid-price as input features for server-based and BMv2-based prediction. For Tofino-based prediction, six price levels volumes and the mid-price are used, due to resource constraints on the switch. Scikit-learn (sklearn) library [51] is used for training and to run the prediction on the server. The server-based benchmark is trained with unlimited-size models, while the switch-based models are of limited size. The labels are used to predict future mid-price movement (up, down, and stationary) over the next 100 ticks.

To explore performance gains with resources' scalability, the prediction performance of Tofino 2 is emulated. Tofino 2 is less resource constrained than Tofino. As it has more stages, it is possible to use larger LOBs, extract more features, and utilize ML models that consume more stages.

Table I shows the experimental results in terms of accuracy (ACC) and F1-score (F1), the most common evaluation metrics measuring ML model prediction performance.

PYPL (Financials)								
	Tofino		Tofino 2		BMv2		Server (Sklearn)	
Model	ACC	F1	ACC	F1	ACC	F1	ACC	F1
KM	31.15	26.55	45.25	31.15	60.97	25.25	65.00	37.59
KNN	49.32	22.02	52.47	28.83	68.44	27.09	70.25	53.55
DT	51.13	50.29	65.40	57.67	73.98	57.77	73.98	57.77
RF	54.51	50.54	62.36	56.38	74.10	57.84	74.43	58.25
XGB	55.42	55.85	62.36	56.28	73.76	58.54	74.55	59.32
PEP (Consumer Staples)								
	Tofino		Tofino 2		BMv2		Server (Sklearn)	
Model	ACC	F1	ACC	F1	ACC	F1	ACC	F1
KM	33.70	22.66	43.43	22.60	39.52	34.28	45.30	39.59
KNN	40.50	31.67	65.15	26.31	52.60	38.70	65.92	49.71
DT	56.88	52.08	63.45	54.94	64.09	62.39	66.98	66.62
RF	58.26	53.81	67.13	59.55	64.35	62.89	67.43	67.19
XGB	58.71	58.69	67.31	60.83	67.74	66.48	67.77	70.12
GOOGL (Communication Services)								
	Tofino		Tofino 2		BMv2		Server (Sklearn)	
Model	ACC	F1	ACC	F1	ACC	F1	ACC	F1
KM	55.95	23.92	58.98	24.73	62.36	27.06	63.46	52.94
KNN	55.59	23.82	60.97	25.25	64.45	26.13	69.30	61.17
DT	52.60	52.61	54.59	54.18	64.98	60.46	65.44	60.87
RF	53.06	53.50	54.79	54.33	65.33	56.80	65.44	65.33
XGB	58.86	56.13	63.16	63.24	66.64	63.45	77.50	68.09

TABLE I
EXPERIMENTAL RESULTS (%) OF ML PREDICTION PERFORMANCE.

The benchmark ML performance results, running on the server, achieve performance comparable to previous (non-network) studies that used similar features from LOBs for the same prediction task [8]. Their test results, using different neural networks, achieved F1-score ranging from 40.84% to 76.58% with the prediction horizon being also 100. In comparison, the results of our ML benchmarks are reasonable. KM and KNN do not perform as well for some stocks because of the low signal-to-noise ratio of their market data feeds.

As the results in Table I show, the average accuracy loss of LOBIN when running on Tofino is 16.47%, with an average loss of 15.60% for F1-score compared with server benchmarks. The LOBIN solution running on BMv2 has an average accuracy loss of 3.30% with an average F1-score loss of 9.53%. Among all the ML models tested, tree-based ensemble methods are the ones that perform the best in terms of ML prediction with low accuracy and F1-score loss.

Tofino 2 has an average accuracy loss of 8.40% and an average F1-score loss of 12.79% compared with the baseline, an improvement relative to Tofino as two additional features from a larger LOB can be supported. Consequently, LOBIN can achieve better performance on Tofino 2 hardware.

To further explore LOBIN's ML performance on different targets, we compute the average accuracy and F1-score ratios relative to a server baseline. As shown in Figure 5, the accuracy and F1-scores of Tofino for five models can respectively achieve 75.63% and 71.83% of the ML benchmarks' accuracy and F1-scores. The accuracy and F1-score ratios for Tofino 2 are higher, being 87.87% and 76.61%, correspondingly. For BMv2, the two ratios reach 94.95% and 82.37%. As LOBIN uses fewer features and limited ML model sizes, while providing latency benefits, its performance is promising.

C. Relative latency and throughput

The latency of Tofino is under NDA, therefore we report our measurements of pipeline and framework latencies relative to Tofino's reference switch (*switch.p4*). LOBIN's relative pipeline latency is computed based on data reported by SDE and includes one recirculation per MBO message. LOBIN's framework latency is measured between two servers, with a switch in the middle. The relative framework latency is the ratio of the measured LOBIN latency to the measurement of simple forwarding through the switch. Precision Time Protocol (PTP) using ptp4l toolkit is used for the measurement.

As shown in Figure 6 (a), all of LOBIN's models have a similar or lower latency than the reference *switch.p4*. This illustrates that even under resource constraints and with packet recirculation, LOBIN still achieves comparable latency to simple packet switching. While on framework level, the measured latency of LOBIN is higher than simple forwarding, as Figure 6 (b) LOBIN provides more than 10% improvement compared with the NASDAQ order-matching server benchmark [52] and achieves microsecond-level latency.

In a throughput test, packets are sent using DPDK 20.11.1 and PktGen 21.03.0. Two of the pipelines (32 ports) are connected in a snake configuration, one port to the next in the pipe, with recirculation through unused pipelines. LOBIN achieves full 1.6Tbps ($16 \times 100Gbps$) on each of the pipelines, and 3.2Tbps in total.

VI. Discussion

LOBIN makes a contribution to latency reduction of short-term price predictions from LBO data *By Design*. Even in comparison with inference solutions using SmartNICs or FPGA, LOBIN eliminates the latency of getting to the host, with

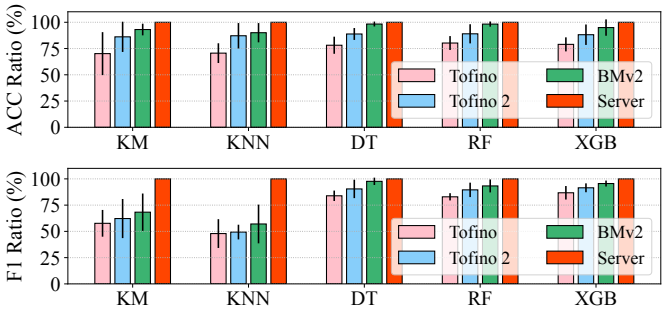
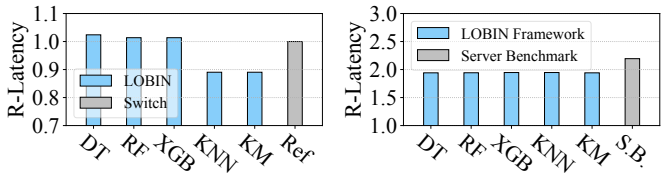


Fig. 5. The average ratio (%) of accuracy and F1-score, Tofino, Tofino 2, and BMv2 relative to the benchmark, for different models.



(a) Pipeline R-Latency (b) Framework R-Latency

Fig. 6. (a) The pipeline relative latency (R-Latency) on Tofino for different models, measured for standalone ML and standalone $switch.p4$. (b) The framework R-Latency on Tofino for different models. (Stock: PYPL).

minimal overhead beyond that of standard switch forwarding. While LOBIN is implemented on Tofino, using low-latency programmable switches can reduce latency further.

Prediction performance can be further improved using hybrid ML deployment, as proposed in [53]. This means that a price-movement prediction in the switch will be labeled only if the confidence level of the prediction is high. Otherwise, the MBO would be sent to a server for prediction by a larger model. Such a deployment can reduce latency for most transactions, without compromising on prediction performance.

While LOBIN is proven to be feasible on hardware targets, canceling and updating orders are left for future work. LOBIN's hardware constraints also don't allow an unbounded increase in LOB size. This work may inspire more industry-driven improvement of current programmable switches to support such extensions.

VII. Conclusion

This paper presented a time-sensitive application of in-network ML in financial trading. LOBIN, a prototype for constructing and updating limit order books within the data plane, was designed and deployed on programmable network devices. The evaluation shows that LOBIN can maintain high prediction accuracy compared with a benchmark while achieving ultra-low latency. The automated implementation of LOBIN makes it easy to explore new stocks, different datasets, and other ML models. It is the first step toward true in-network applications in financial trading.

References

[1] T. Hendershott, C. M. Jones, and A. J. Menkveld, "Does algorithmic trading improve liquidity?" *The Journal of finance*, vol. 66, no. 1, pp. 1–33, 2011.

[2] P. Gomber and M. Haferkorn, "High frequency trading," in *Encyclopedia of Information Science and Technology, Third Edition*. IGI Global, 2015, pp. 1–9.

[3] M. Zook and M. H. Grote, "The microgeographies of global finance: High-frequency trading and the construction of information inequality," *Environment and Planning A: Economy and Space*, vol. 49, no. 1, pp. 121–140, 2017.

[4] M. Kearns and Y. Nevmyvaka, "Machine learning for market microstructure and high frequency trading," *High Frequency Trading: New Realities for Traders, Markets, and Regulators*, 2013.

[5] B. Huang *et al.*, "Automated trading systems statistical and machine learning methods and hardware implementation: a survey," *Enterprise Information Systems*, vol. 13, no. 1, pp. 132–144, 2019.

[6] A. N. Kercheval and Y. Zhang, "Modelling high-frequency limit order book dynamics with support vector machines," *Quantitative Finance*, vol. 15, no. 8, pp. 1315–1329, 2015.

[7] A. Ntakaris *et al.*, "Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods," *Journal of Forecasting*, vol. 37, no. 8, pp. 852–866, 2018.

[8] Z. Zhang, S. Zohren, and S. Roberts, "DeepLOB: Deep convolutional neural networks for limit order books," *IEEE Transactions on Signal Processing*, vol. 67, no. 11, pp. 3001–3012, 2019.

[9] A. Tsantekidis *et al.*, "Using deep learning for price prediction by exploiting stationary limit order book features," *Applied Soft Computing*, vol. 93, p. 106401, 2020.

[10] S. Baruch, "Who benefits from an open limit-order book?" *The Journal of Business*, vol. 78, no. 4, pp. 1267–1306, 2005.

[11] Y. Tokusashi *et al.*, "The case for in-network computing on demand," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019.

[12] D. Sanvito, G. Siracusano, and R. Bifulco, "Can the network be the AI accelerator?" in *NetCompute*, 2018, pp. 20–25.

[13] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *HotNets*, 2019, pp. 25–33.

[14] "Barefoot Tofino," https://www.barefootnetworks.com/products/brief-tofino/.

[15] "The reference P4 software switch," https://github.com/p4lang/behavioral-model.

[16] C. Cheng *et al.*, "Time series forecasting for nonlinear and non-stationary processes: a review and comparative study," *Iie Transactions*, vol. 47, no. 10, pp. 1053–1071, 2015.

[17] M. Obthong *et al.*, "A survey on machine learning for stock price prediction: algorithms and techniques," *International Conference on Finance, Economics, Management and IT Business*, 2020.

[18] M. Ballings *et al.*, "Evaluating multiple classifiers for stock price direction prediction," *Expert systems with Applications*, vol. 42, no. 20, pp. 7046–7056, 2015.

[19] J. Patel *et al.*, "Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques," *Expert systems with applications*, vol. 42, no. 1, pp. 259–268, 2015.

[20] S. Basak *et al.*, "Predicting the direction of stock market prices using tree-based classifiers," *The North American Journal of Economics and Finance*, vol. 47, pp. 552–567, 2019.

[21] M. Baron *et al.*, "Risk and return in high-frequency trading," *Journal of Financial and Quantitative Analysis*, vol. 54, no. 3, 2019.

[22] E. Nurvitadhi *et al.*, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," in *FPL*. IEEE, 2016, pp. 1–4.

[23] G. W. Morris, D. B. Thomas, and W. Luk, "Fpga accelerated low-latency market data feed processing," in *2009 17th IEEE Symposium on High Performance Interconnects*. IEEE, 2009, pp. 83–89.

[24] Q. Tang *et al.*, "A scalable architecture for low-latency market-data processing on fpga," in *2016 IEEE Symposium on Computers and Communication (ISCC)*. IEEE, 2016, pp. 597–603.

[25] H. Subramoni *et al.*, "Streaming, low-latency communication in on-line trading systems," in *IPDPSW*. IEEE, 2010, pp. 1–8.

[26] V. Santosh and S. Singh, "Announcing DPU-based acceleration for NSX," *VMWare*, 2022. [Online]. Available: https://blogs.vmware.com/networkvirtualization/2022/08/announcing-dpu-based-acceleration-for-nsx.html/

[27] AMD, "AMD announces new Alveo X3 series for electronic trading," *HPC Wire*, 2022. [Online]. Available: https://www.hpcwire.com/off-the-wire/amd-announces-new-alveo-x3-series-for-electronic-trading/

[28] J. W. Lockwood *et al.*, "A low-latency library in FPGA hardware for high-frequency trading (HFT)," in *2012 IEEE 20th annual symposium on high-performance interconnects*. IEEE, 2012, pp. 9–16.

[29] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "SDN security: A survey," in *2013 IEEE SDN For Future Networks and Services (SDN4FNS)*. IEEE, 2013, pp. 1–7.

[30] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[31] N. McKeown, *PISA: Protocol Independent Switch Architecture*, 2015, P4 Workshop.

[32] S. Laki *et al.*, "P4Pi: P4 on Raspberry Pi for networking education," *ACM SIGCOMM Computer Communication Review*, vol. 51, no. 3, pp. 17–21, 2021.

[33] X. Jin *et al.*, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 121–136.

[34] H. T. Dang *et al.*, "P4xos: Consensus as a network service," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1726–1738, 2020.

[35] A. Sapio *et al.*, "Scaling distributed machine learning with in-network aggregation," *arXiv preprint arXiv:1903.06701*, 2019.

[36] C. Lao *et al.*, "ATP: In-network aggregation for multi-tenant learning," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 741–761. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/lao

[37] C. Zheng and N. Zilberman, "Planter: seeding trees within switches," in *Proceedings of the SIGCOMM'21 Poster and Demo Sessions*, 2021, pp. 12–14.

[38] C. Zheng *et al.*, "Automating in-network machine learning," *arXiv preprint arXiv:2205.08824*, 2022.

[39] J.-H. Lee and K. Singh, "Switchtree: in-network computing and traffic analyses with random forests," *Neural Computing and Applications*, pp. 1–12, 2020.

[40] M. Zang *et al.*, "P4pir: in-network analysis for smart iot gateways," in *Proceedings of the SIGCOMM'22 Poster and Demo Sessions*, 2022, pp. 46–48.

[41] X. Hong *et al.*, "Linnet: limit order books within switches," in *Proceedings of the SIGCOMM'22 Poster and Demo Sessions*, 2022, pp. 37–39.

[42] Z. Zhang, B. Lim, and S. Zohren, "Deep learning for market by order data," *Applied Mathematical Finance*, vol. 28, no. 1, pp. 79–95, 2021.

[43] M. D. Gould *et al.*, "Limit order books," *Quantitative Finance*, vol. 13, no. 11, pp. 1709–1742, 2013.

[44] T. Preis *et al.*, "Multi-agent-based order book model of financial markets," *EPL (Europhysics Letters)*, vol. 75, no. 3, p. 510, 2006.

[45] C. A. Parlour and D. J. Seppi, "Limit order markets: A survey," *Handbook of financial intermediation and banking*, vol. 5, pp. 63–95, 2008.

[46] APS Networks, "BF6064X-T Advanced Programmable Switch," https://www.aps-networks.com/wp-content/uploads/2021/07/210712_APS_BF6064X-T_V04.pdf[Online, accessed Feb-2023].

[47] F. Hauser *et al.*, "A survey on data plane programming with P4: Fundamentals, advances, and applied research," *Journal of Network and Computer Applications*, p. 103561, 2022.

[48] "NASDAQ OMX PSX TotalView-ITCH 5.0," 2014. [Online]. Available: http://www.nasdaqtrader.com/content/technicalsupport/specifications/dataproducts/PSXTVITCHSpecification_5.0.pdf

[49] M. Bernasconi-De-Luca, L. Fusco, and O. Dragić, "martinobdl/itch: Itch50converter," 2021. [Online]. Available: https://zenodo.org/record/5209267

[50] N. V. Chawla *et al.*, "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[51] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[52] J. Bonart and M. D. Gould, "Latency and liquidity provision in a limit order book," *Quantitative Finance*, vol. 17, no. 10, pp. 1601–1616, 2017.

[53] C. Zheng *et al.*, "IIsy: Practical in-network classification," *arXiv preprint arXiv:2205.08243*, 2022.