In-Network Machine Learning for Real-Time Transaction Fraud Detection

Xinpeng Hong[a,*], **Changgang Zheng**[a] **and Noa Zilberman**[a]

[a]Department of Engineering Science, University of Oxford, Oxford, United Kingdom

Abstract. Machine learning (ML) has become a mainstream approach in the fight against transaction fraud for its intelligence. For financial institutions and businesses, low-latency detection of fraudulent transactions in real-time is highly important as it enables rapid identification and prevention. Concurrently mitigating fraudulent transactions by using ML while also reducing latency remains a challenging endeavor, for which performing inference within programmable network devices offers a potential solution. In this paper, we introduce MIND, conducting ML-based fraud detection within programmable devices. MIND is prototyped on both software and hardware network devices, including BMv2, Intel Tofino, and NVIDIA BlueField-2 DPU, and is evaluated with three publicly available transaction datasets. Experimental results demonstrate that MIND detects transaction fraud in real-time, with a throughput of 6.4 terabits per second and microsecond-scale latency. Compared with server-based solutions, MIND can process over $\times 800$ more transactions per second, along with a latency reduction of over $\times 1300$ per transaction. At the same time, MIND attains 99.94% of server-based benchmarks' accuracy and 93.66% of their F1-score, exhibiting only marginal degradation in classification performance. Therefore, MIND offers substantial savings in the number of servers, leading to reduced costs and energy consumption, while providing a better customer experience.

1 Introduction

Fraudulent activities are widespread in financial transactions nowadays. These refer to any intentional deception or misrepresentation by an individual or group with the aim of obtaining unauthorized benefits [38]. Transaction fraud can have significant and wide-ranging harmful effects on individuals, businesses, and society as a whole. According to a study by Juniper Research, merchants' cumulative financial losses from online transaction fraud are predicted to surpass $343 billion between 2023 and 2027 [30]. Therefore, fraud detection plays a crucial role in preventing and reducing the possibility of potential harm. Fraud Detection Systems (FDS) are designed to identify and flag suspicious fraudulent behaviors, allowing for prompt intervention and prevention of further damage [2]. However, the ever-evolving nature of fraudulent activities, coupled with fraudsters' ability to adapt to fraud detection measures, presents a considerable challenge to traditional FDS with statistical and rule-based analysis components [46].

To better secure electronic commerce systems, most FDS utilize machine learning (ML) algorithms to recognize fraudulent patterns and detect them in real-time transaction data streams [6]. ML has significantly improved detection accuracy and reduced false positives thanks to its ability to analyze complex datasets and learn from anomalies in historical transactions [4]. Despite the promising results demonstrated by previous ML-based works, accurate and prompt detection remains a formidable task. Significant data imbalance and considerable variability of fraudulent transactions both contribute to the complexity [14, 33]. To reach optimal performance, there is an ongoing development of increasingly advanced models. However, embedding more sophisticated ML models into FDS conflicts with the aim of lowering detection time, which is essential to prevent fraudulent transactions from being processed and completed. Latency directly impacts the ability of FDS to swiftly identify and respond to suspicious activities. Most real-time fraud detection services, both in research and industry, currently operate at the millisecond-level latency [10, 8, 16, 17]. However, the escalating velocity at which fraudulent activities can occur requires FDS to identify suspicious transactions not within seconds or even milliseconds, but microseconds [19].

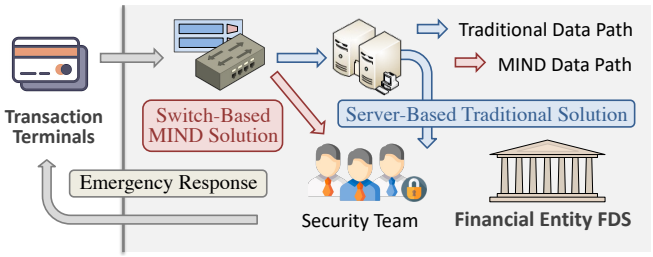


Figure 1. General working scenario of MIND.

Programmable network devices can process data streams in parallel with other network functions. They have been shown to reduce latency compared to servers, particularly when used for real-time inference tasks (also referred to as in-network ML) [40, 50] by deploying pre-trained ML models into the data plane, which focuses on the actual forwarding process of data packets. However, no previous works studied its application to transaction fraud detection. This paper presents a first-time research work in this area, specifically in the context of using in-network classifiers to distinguish between fraudulent and genuine transactions. We introduce MIND, an in-network prototype that detects incoming fraudulent transactions. Figure 1 shows a general deployment scenario of MIND compared with traditional server-based FDS solutions of a financial entity. Typically, a transaction starts at a transaction terminal, passing through one or more network switches, before arriving at a server-based FDS. When a fraudulent transaction is detected, it is flagged to the se-

* Corresponding Author. Email: xinpeng.hong@eng.ox.ac.uk.

curity team and an emergency response is triggered. In MIND, the fraudulent transaction is detected within the switch and immediately flagged, eliminating the server processing and delay component.

MIND is implemented on a software switch (BMv2) and a hardware switch (Intel Tofino), demonstrating its feasibility. Evaluation results indicate that MIND outperforms a server-based benchmark by processing over $\times 800$ more transactions per second and achieving microsecond-level latency with a significant reduction of over $\times 1300$ per transaction. At the same time, MIND attains a comparable detection performance with server-based benchmarks across evaluated models and datasets. Consequently, MIND has the potential to eliminate a large number of servers, and thereby reduce cost and energy consumption while enhancing the customer experience.

The main contributions of this paper are as follows:

- We study the practical application of in-network ML to financial fraud detection using transaction data and present a proof of concept to demonstrate the feasibility of this approach. To the best of our knowledge, this study is the first of its kind to investigate the use of in-network ML for low-latency transaction fraud detection.
- We design and implement a prototype using transaction data for feature engineering and ML classification in a programmable data plane. Integrating the use-case-related workflow with ML processes, we deploy the prototype on a software switch and a commodity hardware switch.
- We evaluate the prototype within a local testbed, considering a wide range of ML and networking performance metrics across different commonly used in-network ML models. Given the absence of prior switch-based fraud-detection research, server-based benchmarks are used for comparative analysis. The results indicate that MIND offers terabit-scale throughput and microsecond-level latency while maintaining high detection performance.

2 Related Work

ML-Based Transaction Fraud Detection. Financial institutions and businesses face the impracticality of manually inspecting each transaction due to the sheer volume of transactions, and users anticipate instantaneous responses. Rule-based expert systems have been used for decades, drawing on established industry practices to detect known fraudulent behaviors [24]. However, the emergence of online transaction fraud with complex patterns, which differs significantly from its traditional counterpart, has rendered traditional FDS ineffective in detecting and intercepting such activities [54].

In contrast, ML algorithms can effectively process large datasets and adjust themselves based on new information, enabling them to capture financial phenomena and adapt to evolving patterns of fraudulent activities. Therefore, different ML models have been employed to detect transaction fraud, such as support vector machines (SVMs) [43], naive Bayes (NB) [5], decision trees (DTs) [39], random forests (RFs) [51, 32], and extreme gradient boosting (XGB) [37]. Previous works have also conducted comparative analysis of multiple models applied to this field [6, 59, 4, 33, 31].

The recent surge in transaction data volume and the growing complexity of fraudulent activities have led to the need for more intricate ML models, posing challenges for real-time data processing. While some prior studies have focused on low-latency financial fraud detection [10, 9, 8], none of them attempted to employ programmable network devices for achieving lower latency during data processing. The most relevant and latest study presents an XGBoost accelerator based on FPGA for speeding up inference and applying it in transaction fraud detection [16]. All of the CPU, GPU, and FPGA-based designs are constrained to achieving latency at the millisecond level. However, in the prevention of fraudulent activities, the difference between milliseconds and microseconds can sometimes signify the contrast between incurring substantial financial losses or avoiding them entirely [19].

In-Network ML. The ever-growing complexity of modern computer networks has made traditional network management practices insufficient, leading to the necessity for network programmability [34]. Programmable network devices serve as a flexible and scalable infrastructure, empowering network administrators to automate network management and respond promptly to dynamic demands.

In-network computing refers to the practice of executing computations and processing data within the network infrastructure itself instead of solely depending on computing resources on end hosts [48]. The network infrastructure involves a wide range of programmable network devices such as switches, FPGAs, and NICs. The utilization of in-network computing capitalizes on the benefits of reduced overhead in terms of space, energy, and cost, as well as the superior process efficiency of network devices [47, 48]. It has been applied to various applications including network functions, caching, coordination, and distributed systems (e.g., [22, 23, 35, 15]).

In-network ML represents a special utilization scenario of in-network computing, which refers to the offloading of ML functions or inference into the data plane of programmable network devices [41, 25]. Previous works on in-network ML have explored and applied a variety of models, thereby enabling their application across different domains [55]. While in-network ML has been identified as a viable solution for latency reduction [20, 21, 57], the scope of its use cases remains constrained and its application in finance has barely been explored yet, necessitating this work.

3 Design of Proposed Solution

As a replacement for traditional server-based solutions, MIND is designed to be deployed within the FDS of financial entities. MIND enables real-time transaction fraud detection through feature engineering and ML inference within the programmable data plane. More specifically, upon receiving a fresh transaction record, MIND selects, manipulates, and transforms raw data into ML features all within the data plane. The features are then used for inference, classifying the transaction as fraudulent or not. Figure 2 presents the system architecture design of MIND, described next. The control plane focuses on managing and controlling the network traffic, while the data plane is responsible for the actual forwarding of network packets.

3.1 Offline Training and Mapping

A MIND deployment starts by using transaction data feeds as inputs for ML training on a server, using both raw and hand-crafted data fields as features (as in Figure 2-I). The resulting trained model (Figure 2-II) is mapped into a data plane compliant model (Figure 2-III), using match-action table constructs. A P4 [7] program is generated from this process based on the target platform architecture (Figure 2-IV). P4 is a specialized language utilized for configuring the packet processing and forwarding mechanisms of a data plane. The generated P4 program consists of three main parts: architecture-related code, feature-engineering-related code, and inference-model-related code. The architecture-related code is where standard network switching functionality is located and where the other two components are integrated. The code pertaining to feature engineering

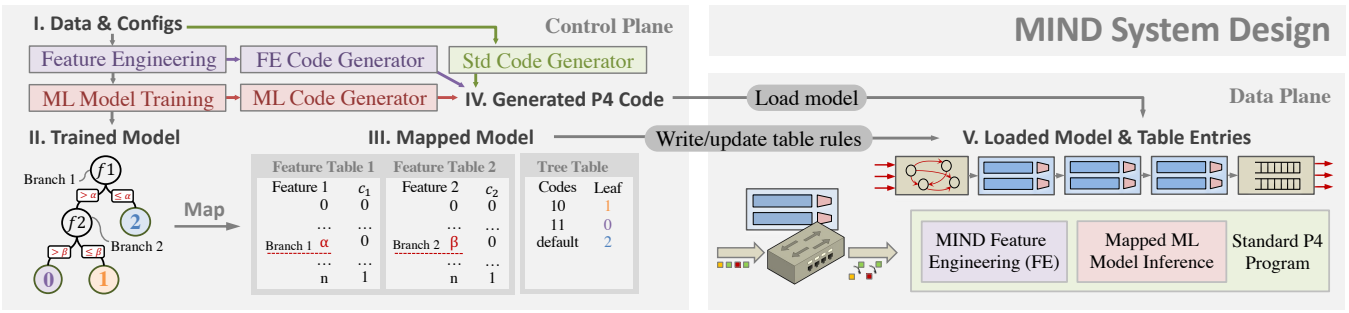


Figure 2. System design of MIND with components and processes on the control plane and data plane.

varies across datasets, while the code related to ML inference is generated based on up-to-date mapping methodologies [57]. After compilation, the P4 code is loaded onto the data plane target, while table entries are loaded through the control plane (Figure 2-V). This allows for real-time data processing and detection directly within the data plane, based on the pre-trained ML model.

In general, the mapping process converts ML models trained to run on a CPU into a network device pipeline. Unlike CPUs, where instructions move through the pipeline, network devices move data through the pipeline while instructions are stored in tables. An illustration of the model inference mapping process to the data plane is provided in Figure 2-II & III, with a DT model serving as an example. This process entails decomposing the model into a series of straightforward operations that can be executed by the processing pipeline. A tree-like structure with nodes and branches is obtained from model training as shown in Figure 2-II. It exemplifies a partitioning of feature space based on two features $f1$ and $f2$. When a data sample traverses the tree during inference, the sequence of branches it follows forms a feature split path. This path involves comparing the input feature values against threshold values and subsequently proceeding down the appropriate branch of the tree. To map such model structure to the data plane pipeline, match-action table rules are generated by translating the feature split path into feature tables and a tree table, as shown in Figure 2-III. The feature tables utilize input feature values and feature splits as match conditions and are associated with encoded values as actions that direct the data to the next tree node and branch, while the tree table records code pairs as match conditions to yield the labeling decisions.

3.2 Real-Time Data Processing

Despite variations in transaction data feeds, they consistently contain a diverse range of transaction information, such as credit card purchases, online transfers, cash withdrawals, and other monetary exchanges. The data fields may include transaction timestamps, transaction amounts, merchant details, customer identities, location information, device used for the transaction, and other attributes.

As raw data fields may not be directly suitable for effective model training, feature engineering is essential for ML-based fraud detection as it transforms variables from raw transaction data into informative features, capturing relevant patterns and relationships specific to fraudulent activities. By crafting engineered features, ML models can better discern between legitimate and fraudulent transactions, leading to improved detection performance. Additionally, feature engineering helps mitigate the curse of dimensionality [49], allowing models to handle large volumes of transaction data efficiently and enhancing detection performance.

Real-time data processing in MIND consists of two key components in addition to network forwarding: feature engineering and ML inference. Due to variations in raw data fields among different transaction feeds, the process of feature engineering varies depending on the specific new features desired and the selection of raw features to be utilized in their generation. Further elaboration and illustrative instances of this process are provided in Section 4. Once the features have been processed and transformed through engineering techniques, they are used as inputs to the ML models to detect fraudulent transactions.

It is crucial to consider the limitations imposed by the programmable data plane within switches when fitting the process of feature engineering. The complexity of this process must be constrained, as programmable switches are typically not equipped to handle intricate mathematical operations, such as trigonometric functions, logarithmic functions, and exponential functions, nor operations like matrix multiplication or calculus. Section 4 explains how the limitations are overcome to provide a practical implementation.

4 Implementation

We implement MIND using P4 on both Intel Tofino switch-ASIC and BMv2. ML models are pre-trained on a server using Python and the scikit-learn (sklearn) library [36]. This section presents the P4 targets, primary implementation obstacles with proposed solutions, and implementation details.

4.1 P4 Targets

BMv2 is a widely used open-source behavioral software switch developed and maintained by the P4 workgroups. It is often used as a reference switch for functionality evaluation and offers greater flexibility compared to switch-ASICs as it is less resource-constrained than hardware targets.

The second target is Intel Tofino, a programmable switch-ASIC with multi-Tbps (terabits per second) data rate and sub-microsecond latency [3]. Tofino can potentially be deployed at network edge or financial-entity access, offering transaction fraud detection services. However, like most programmable switch-ASICs, Tofino faces resource constraints such as limited stages and memory [12]. While BMv2 serves as a suitable choice for P4 prototyping, the utilization of Tofino demonstrates the viability of employing off-the-shelf platforms in real-world scenarios.

4.2 Key Challenges and Solutions

Due to the inherent hardware constraints, the process of feature engineering needs to be tailored to the device. The key challenges and

solutions involved in MIND's implementation are presented below.

Lack of Data Types: P4 is designed for packet processing and forwarding tasks, focusing on the manipulation of fixed-size fields within packet headers, commonly represented as integers. Therefore, P4-capable programmable data planes do not have native support for floating-point numbers. However, in transaction feeds, transaction amounts are usually floating-point numbers. To facilitate alignment with the P4 data plane, those amounts are converted to integers. Since amounts often significantly vary between transactions, the conversion does not necessarily incur a loss of information or affect ML performances. While the conversion may discard the decimal precision, the remaining integer values retain the relative magnitude and order of the original amount.

Lack of Mathematical Operations: P4 supports only a restricted subset of fundamental arithmetic and logical operations. These operations comprise fundamental mathematical computations such as addition, subtraction, logical operations (such as Xor), and bit shift. Additionally, P4 accommodates conditional constructs like $if-else$ conditions, providing basic decision-making capabilities. However, other commonplace operations like multiplication, division, and variable comparisons are not supported. As these operations are indispensable in feature engineering for ML, different strategies are used to perform them indirectly. For example, to compare two variables within an $if-else$ condition, an implicit comparison is needed. First, the difference between two variables is calculated, and then this difference is compared with 0. To enable operations like multiplication, division, and other complex calculations, using match-action table lookups is a viable solution. By executing table lookups using specific keys or matching conditions, complex feature engineering can be implemented. Furthermore, instead of storing each set of {operands, result} separately, lookup tables can efficiently group multiple entries with shared attributes using scaling mechanisms (e.g., ternary matching).

Limitations on Value Range of a Feature: Up-to-date in-network ML mapping approaches mainly use match-action table lookups to enable ML algorithms within the data plane [57]. When a processed feature has an exceptionally wide value range, challenges arise. Specifically, the range of values correlates to the number of entries required in the match-action tables, and may exceed the resource constraints of the hardware in terms of memory. Transaction amounts can significantly vary across transactions, with fraudulent transactions often characterized by exceptionally large amounts. Consequently, the amount field must be processed and mapped to the data plane. Our approach is to partition transaction amounts into two distinct components. These two components can be utilized as ML features, substituting the original single feature and reducing effectively the total table entries. The first part encapsulates the least significant value: thousands and below, while the second stores the significant bits. The second part is a more important feature because it indicates the magnitude of the transaction amount.

4.3 Data Plane Implementation

MIND's P4 implementation incorporates feature engineering code integrated with ML inference code generated using Planter, a modular framework for mapping trained ML models to programmable network devices [57]. The implementation ensures compatibility with Tofino 2, requiring no modifications to existing code. Tofino 2 is less resource-constrained than Tofino, enabling the execution of more intricate feature engineering, extraction of more features, and supporting ML models that require a higher number of stages. To cover diverse transaction feeds, feature engineering is performed separately on three representative datasets [28, 42, 44] characterized by different raw fields, which have been widely employed in previous works [26, 29] and are also utilized for our evaluation.

The first transaction feed originates from a dataset [28] generated by the PaySim mobile money simulator based on real transactions obtained from one-month financial logs of a mobile money service deployed in Africa. Each entry includes information on a time step, transaction amount, unique identifiers of the customer and recipient, initial balances of both customer and recipient before the transaction, and new balances of both after the transaction. Driven by the possibility of zero balances as a distinguishing factor between fraudulent and legitimate transactions, two new features (E_{orig} and E_{dest}) can be created, capturing errors in the originating and destination accounts for each transaction. The parameter A_{trans} represents the transaction amount. The variables I_{orig} and N_{orig} signify the initial balance of the originating account and the new balance following the transaction, respectively. Similarly, I_{dest} and N_{dest} represent the corresponding values for the destination account. The computation process is demonstrated in Equation 1 and Equation 2.

$$E_{orig} = N_{orig} + A_{trans} - I_{orig} \tag{1}$$

$$E_{dest} = I_{dest} + A_{trans} - N_{dest} \tag{2}$$

Generated by a data generation tool [18], the second feed [42] contains credit card transactions, including the transaction date and time, amount, the category associated with the merchant, and customer information, such as their credit card number, name, gender, date of birth, address, city, state, occupation, and geographic coordinates. New features can also be derived from these raw fields. For example, the population of the customer's city (P_{city}) can be transformed into an additional feature (C_{city}) that categorizes the city based on its level of population density, as shown in Equation 3.

$$C_{city} = \begin{cases} 1, & 0 < P_{city} < 10^4 \\ 2, & 10^4 \leq P_{city} < 5 * 10^4 \\ 3, & P_{city} \geq 5 * 10^4 \end{cases} \tag{3}$$

The third type of feeds [44] consists of transactions conducted by users across multiple terminals during the period spanning from January 2023 to June 2023. Each transaction within it comprises distinct identifiers denoting the transaction, customer, and source terminal, as well as the precise date and time of occurrence. Additionally, it includes the transaction amount, as well as the duration in seconds and days that transpired prior to the transaction. The emphasis of feature engineering lies in the creation of new time-related features (D_{week}, $F_{weekday}$, and $F_{workinghour}$). D_{week} takes on values ranging from 0 to 6, each corresponding to a specific day of the week. Its calculation is predicated on Zeller's Congruence, an algorithmic method designed to ascertain the specific day of the week corresponding to a given date [53]. Since all transactions in the dataset occur in the year 2023, the algorithm can be expressed as Equation 4, where D and M denote the day and month of the transaction date, respectively. Two additional features ($F_{weekday}$ and $F_{workinghour}$) are shown in Equation 5 and Equation 6. They are generated to indicate whether the transaction took place on a weekday and during working hours, respectively. The parameter H denotes the hour of the transaction, spanning from 0 to 23.

$$D_{week} = \begin{cases} (D + \left\lfloor \frac{13(M+13)}{5} \right\rfloor - 8) \mod 7, & M \in \{1, 2\} \\ (D + \left\lfloor \frac{13(M+1)}{5} \right\rfloor - 7) \mod 7, & otherwise \end{cases} \tag{4}$$

$$F_{weekday} = \begin{cases} 0, & D_{week} \in \{0, 1\} \\ 1, & otherwise \end{cases} \quad (5)$$

$$F_{workinghour} = \begin{cases} 1, & 6 \leq H \leq 19 \\ 0, & otherwise \end{cases} \quad (6)$$

The solutions explained in Section 4.2 are applied in the implementation of MIND. For instance, modulo operations depicted in Equation 4 are enabled within the P4 data plane using match-action table lookups. These three datasets serve as mere illustrations, as there is an opportunity for further exploration in feature engineering and the discovery of new features based on raw transaction datasets.

5 Evaluation

In this section, we use openly available financial datasets to evaluate MIND, testing its ML detection performance, latency, and throughput. We also compare MIND running on commodity programmable switches, Data Processing Units (DPUs), and server-based benchmarks using different ML approaches. The results show the advantages of MIND in improving throughput, lowering latency, and maintaining detection performance across datasets and hardware targets.

Dataset Description: This study uses three publicly-available financial fraud detection datasets for evaluation, also mentioned in Section 4. The first dataset [28] contains around 6.36 million entries and about 0.13% of them are fraudulent transactions. The second dataset [42] consists of 1.85 million credit card transactions, with around 0.52% marked as fraudulent. The third dataset [44] has a collection of 1.75 million transactions, exhibiting a class imbalance with 13.45% of the transactions being flagged as fraudulent.

Experiment Setup: The evaluation uses an APS-Networks BF6064T-X Intel Tofino switch featuring 64×100G ports running SDE 9.4.0. To conduct server-based experiments, including traffic generation, two ASUS ESC4000A-E10 servers are used, each furnished with an AMD EPYC 7302P CPU, 256GB DDR4 RAM, and running Ubuntu 20.04 LTS. The switch is connected to both servers via NVIDIA ConnectX-5 100G NICs and direct attach cables. The NVIDIA BlueField-2 DPU runs Ubuntu 22.04 LTS, and is equipped with eight ARM Cortex-A72 cores, where BMv2 runs.

5.1 ML Performance

In this binary classification, fraudulent transactions are labeled as positive instances and others as negative. However, there is often a marked class imbalance with the negative class substantially outnumbering the positive, leading to models that may disproportionately favor the majority class and reduce effectiveness in detecting fraud. To counteract this, the synthetic minority over-sampling technique (SMOTE) is carried out on the datasets [11]. This approach generates synthetic samples from the minority class, thereby enabling models to learn from more balanced datasets.

In addition to common metrics for ML performance evaluation, including precision, sensitivity, specificity, accuracy, and F1-score (F1), two more informative metrics for imbalanced datasets are used, namely Matthew's correlation coefficient (MCC) [13] and balanced classification rate (BCR) [45]. The possible range of MCC is between -1 and 1, where 1 represents a perfect classification, 0 represents a random classification, and -1 represents a completely wrong classification. For all other six metrics, the possible value ranges from 0 to 1, where 1 denotes the best score and 0 indicates the worst.

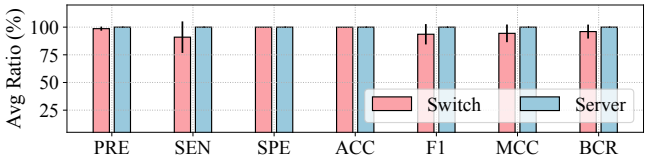


Figure 3. The average (avg) ratio of precision (PRE), sensitivity (SEN), specificity (SPE), accuracy (ACC), F1, MCC, and BCR, switch-based solutions relative to server-based benchmarks, across models and datasets.

ML performances on all three datasets are presented in Table 1, covering the five models that are commonly used in this task and also supported by up-to-date in-network ML mapping approaches. All the values presented in the table have been rounded to two decimal places, so the representation of 100% does not denote an exact 100%, but rather a value within the range of 99.995...% to 99.999...%.

While both BMv2 and Tofino exhibit the same outcomes, as indicated in the "Switch" columns of Table 1, they both experience a reduction in evaluation metrics when compared to the server-based benchmark results, as reflected in the "Server" columns. The accuracy and F1 loss average at 0.06% and 5.20% respectively across all evaluated datasets and models, highlighting the overall effectiveness of MIND in classification. More specifically, loss of precision, sensitivity, and specificity are 0.85%, 7.19%, and 0.02%, respectively. In terms of MCC and BCR, an average discrepancy of 4.63% and 3.60% is showcased respectively, illustrating MIND's ability to handle imbalanced datasets.

The average ratios of all metrics relative to a server benchmark are also computed. As shown in Figure 3, the average precision and sensitivity of MIND can respectively achieve 98.63% and 90.94% of the ML benchmarks' precision and sensitivity. The F1 ratio averaged at 93.57%, indicating a moderate balance between precision and sensitivity. The MCC and BCR ratios reach 94.39% and 95.99%, respectively. The specificity and accuracy ratios are even higher, being 99.98% and 99.94% correspondingly. In conclusion, even with constrained model sizes and sometimes fewer features, MIND demonstrates promising performance across models and datasets.

To further explore MIND's ML performance with different models, we calculate the metrics ratios of each model relative to a server benchmark, thus facilitating a comprehensive analysis. As shown in Figure 4, SVM and RF have relatively lower ratios for most metrics, compared with the other three in-network ML algorithms. The observed performance can be attributed to several factors beyond the inherent loss resulting from their mapping approaches. In the case of SVM, server-based benchmarks benefit from the ability to employ a broader range of features and utilize the entire training set for ML model training. However, the implementation of in-network SVM necessitates a more restricted feature set and training on a subset of the data to accommodate the inherent limitations of the switch. Regarding RF, it encounters challenges primarily due to the constraint of mapping, wherein the feasibility of mapping a model is restricted by its size. This leads to a substantial discrepancy in model sizes between server-based benchmarks and switch-based solutions. Considering the second dataset to illustrate, the server-based benchmark employs an ensemble of 19 DTs with a maximum depth of 18 for each tree. Such model size proves impractical for a switch-based implementation, compelling a reduction in the number of DTs to 5, with a maximum depth of 5 for each DT. As indicated by the notes of Table 1, in-network SVM and NB encounter suitability and scalability challenges compared to tree ensemble models, as they can exceed the stage limitation when dealing with a higher number of features.

The ML performance on the second dataset is comparatively lower

Dataset 1: PaySim Financial Fraud Detection Dataset [28]														
	Precision		Sensitivity		Specificity		Accuracy		F1		MCC		BCR	
Model	Switch	Server	Switch	Server	Switch	Server	Switch	Server	Switch	Server	Switch	Server	Switch	Server
SVM‡	86.44	88.86	47.41	81.66	99.99	99.99	99.92	99.96	61.24	85.11	63.99	85.16	73.70	90.82
NB‡	33.77	34.00	32.48	33.03	99.92	99.92	99.83	99.83	33.12	33.51	33.04	33.43	66.20	66.47
DT	96.46	98.91	99.57	99.70	100.00	100.00	99.99	100.00	97.99	98.52	98.00	98.53	99.78	99.85
RF	99.55	100.00	94.38	99.70	100.00	100.00	99.99	100.00	96.90	99.85	96.93	99.85	97.19	99.85
XGB	96.51	99.94	99.57	99.70	100.00	100.00	99.99	100.00	98.02	99.82	98.03	99.82	99.78	99.85
Dataset 2: Sparkov Credit Card Transactions Fraud Detection Dataset [42]														
	Precision		Sensitivity		Specificity		Accuracy		F1		MCC		BCR	
Model	Switch	Server	Switch	Server	Switch	Server	Switch	Server	Switch	Server	Switch	Server	Switch	Server
SVM	9.86	10.58	63.04	63.31	96.90	97.12	96.72	96.94	17.05	18.12	24.05	25.03	79.97	80.21
NB	31.67	32.11	47.22	47.85	99.45	99.46	99.17	99.19	37.92	38.43	38.28	38.81	73.34	73.66
DT	94.18	94.33	57.35	78.71	99.98	99.98	99.76	99.86	71.29	85.82	73.39	86.11	78.66	89.34
RF	94.38	95.80	48.14	76.20	99.98	99.98	99.74	99.86	63.76	84.88	67.31	85.37	74.06	88.09
XGB	94.42	94.76	64.67	80.62	99.98	99.98	99.80	99.87	76.77	87.12	78.05	87.34	82.33	90.30
Dataset 3: Fraudulent Transaction Detection Dataset [44]														
	Precision		Sensitivity		Specificity		Accuracy		F1		MCC		BCR	
Model	Switch	Server	Switch	Server	Switch	Server	Switch	Server	Switch	Server	Switch	Server	Switch	Server
SVM	97.01	98.27	96.30	96.37	99.54	99.74	99.10	99.29	96.65	97.31	96.14	96.91	97.92	98.05
NB	99.42	98.78	95.20	96.26	99.91	99.82	99.28	99.34	97.27	97.50	96.89	97.13	97.56	98.04
DT	99.99	100.00	96.41	96.42	100.00	100.00	99.52	99.52	98.17	98.18	97.92	97.92	98.21	98.21
RF	99.99	100.00	96.41	96.42	100.00	100.00	99.52	99.52	98.17	98.18	97.92	97.92	98.21	98.21
XGB	99.99	100.00	96.41	96.42	100.00	100.00	99.52	99.52	98.17	98.18	97.92	97.92	98.21	98.21

‡ In-network SVM and NB on Dataset 1 surpass Tofino's predefined 12-stage constraint as a higher number of features are used compared to the other two datasets. Consequently, their performance is emulated based on Tofino 2 which provides the capability to accommodate up to 20 stages. Models in the remaining entries can all fit within Tofino's limitations.

Table 1. ML classification performance with all three datasets (%). MIND runs limited-size models on BMv2 and Tofino (or emulated on Tofino 2). The benchmark runs on a server using Sklearn and unlimited-size models.

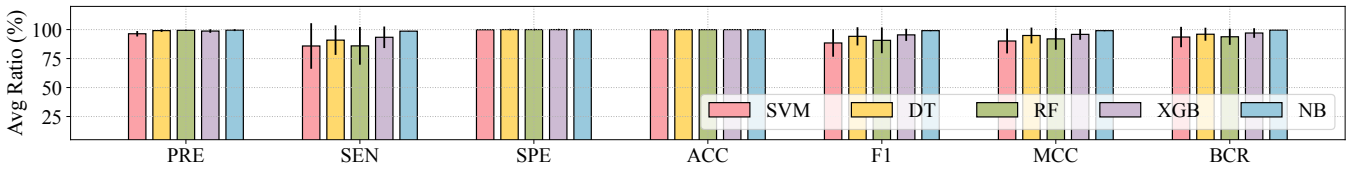


Figure 4. The average (avg) ratio (%) of different ML models in terms of all used metrics, switch-based solutions relative to server-based benchmarks.

than that of the other two datasets primarily due to the available features' limited effectiveness in the dataset. Notably, SVM and NB exhibit inferior performance when compared to tree ensemble models which possess a greater capacity for capturing intricate nonlinear relationships and interactions among features. However, even tree ensemble models require a large model size to achieve a relatively high ML performance. In our evaluation, server-based DT has a depth of 9, while RF and XGB are composed of 19 DTs with a maximum tree depth of 18 and 4 DTs with a maximum tree depth of 8, respectively.

Notably, publicly accessible server-based solutions utilizing these models demonstrate similar performance across the three datasets [27, 42, 44], thereby validating our results. The benchmark outcomes also exhibit consistency with prior studies using the same datasets [26, 29]. Additionally, while there is clear potential for future advancements in in-network deep learning (DL), these models are so far impractical to fit directly into programmable switches due to their computational demands exceeding the devices' capabilities.

5.2 Networking Performance

To assess networking performance, an extensive evaluation is conducted on the data plane, focusing on latency and throughput, providing insights into the efficiency and effectiveness of MIND. Latency measures the time a data packet travels from the source to the destination through the hardware switch, while throughput refers to

Figure 5. The pipeline R-Latency of MIND on Tofino with different in-network ML models across datasets and *switch.p4*.

the amount of data that can be transmitted over the switch within a given period.

Due to Tofino-related NDAs, we present the relative pipeline latency (R-Latency) of MIND to that of an Intel reference switch program (*switch.p4*), both determined by Tofino's compiler. The *switch.p4* program describes the data plane of an L2/L3 switch, supporting a range of features encompassing basic L2 switching and basic L3 routing [1]. As shown in Figure 5, MIND exhibits even lower latency than half of *switch.p4* across all datasets and evaluated in-network models, except for the in-network SVM and NB on dataset 1, as explained in the notes of Table 1). This presents the capacity of MIND to attain latency performance comparable to that of simple packet switching, even under resource constraints. Additionally, MIND can be deployed in parallel with other networking functions.

Moreover, to evaluate MIND's framework latency, measurements are conducted between two servers with a switch positioned in between. The framework R-Latency is calculated as the ratio be-

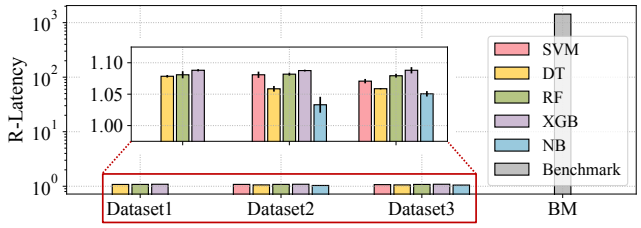


Figure 6. The framework R-Latency of MIND on Tofino with different in-network ML models across all datasets. The benchmark (BM) is a server-based industrial solution for real-time payment fraud detection.

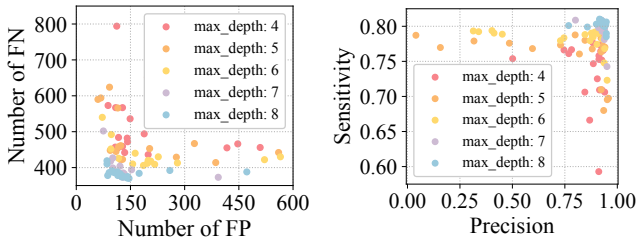
tween the measured MIND latency and the measurement obtained from simple forwarding through the switch. Precision Time Protocol (PTP) with the ptp4l toolkit is applied in the measurements. The findings depicted in Figure 6 reveal that the measured latency of MIND is on average approximately 7.21% higher than simple forwarding on the framework level. More importantly, MIND achieves a latency level at the microsecond scale, which stands in stark contrast to an AI-driven real-time fraud detection solution running on a single server that typically operates at the millisecond latency level [17]. Scaling down from millisecond to microsecond level, MIND provides around $\times 1300$ latency reduction. The similar relative relationship among models' latency presented in Figure 5 and Figure 6 mutually cross-validates the correctness of the experiments.

To evaluate maximum throughput capacity, a snake configuration is used, whereby traffic is routed in a loop from each port to the next, enabling communication across all 64 ports. Packet generation is done using DPDK 20.11.1 and PktGen 21.03.0. The results show that MIND reaches line rate on a Tofino commercial switch, achieving a throughput of 6.4Tbps across all three datasets. While server-based solutions process 3.5 million inferences per second [17], MIND reaches a packet rate of 2.9 Bpps (billion packets per second), providing $\times 800$ throughput improvement. In conclusion, while ML performance may vary with different datasets, the switch-system performance is guaranteed by design.

Beyond raw transactions, the datasets used include private customer information. Such information need not leave the financial institute's premise (with the network platform at its edge) and can be securely stored in encrypted databases on smart networking platforms with encryption/decryption cores, such as DPUs. Therefore, an evaluation of throughput and latency is conducted on an NVIDIA BlueField-2 DPU. P4C compiler is used to compile the P4 code utilizing the v1model architecture to a BMv2 running on DPU's Arm cores. On average, MIND achieves sub-millisecond latency across models, outperforming server-based solutions by $\times 4$. Additionally, our experiments explore the use of Transport Layer Security (TLS), a widely used encryption protocol designed to ensure robust communications security across networks. Kernel TLS offloading using two DPUs is deployed and tested via OpenSSL, demonstrating MIND can be practical when traffic encryption is used.

6 Discussion

In fraud detection, false negatives (FN) are instances wherein FDS fail to identify fraudulent transactions, allowing them to proceed as legitimate, leading to potential financial losses and reputation damage. In contrast to FN, false positives (FP) occur when valid transactions are mistakenly identified as fraudulent, causing customer dissatisfaction and burdening operational resources due to false alarms. An ideal model should minimize both FN and FP, but there is often an inherent trade-off between the two. For instance, implementing stringent rules to reduce FN may inadvertently elevate FP, and vice versa. This trade-off is also evident in the interplay between precision and sensitivity. We use XGB on the Sparkov simulated dataset [42] as an illustrative example of the trade-off, varying the number of trees from one to twenty while maintaining constant maximum tree depth. Five rounds of experimentation are conducted, with maximum tree depth increased from 4 to 8 for the respective rounds. The results, shown in Figure 7 (a), reveal this characteristic trade-off pattern, wherein models exhibiting elevated FN correspondingly demonstrate decreased FP, and vice versa. Figure 7 (b) shows that the precision-sensitivity trade-off follows a similar pattern: higher precision may be accompanied by lower sensitivity, and vice versa. Addressing this trade-off depends on the relative significance of FN versus FP (or sensitivity versus precision). In transaction fraud detection, reducing FN (or increasing sensitivity) is often prioritized, as the primary goal is minimizing the losses caused by fraud.



(a) Scatter plot of FP-FN (b) Scatter plot of PRE-SEN

Figure 7. (a) Scatter plot of the number of FP versus the number of FN for different parameter settings. (b) Scatter plot of the precision (PRE) rate versus sensitivity (SEN) rate for different parameter settings.

Knowledge from separate decentralized nodes can be shared, where nodes of MIND send digest messages to a controller, enabling online and federated learning, following ideas applied to IoT traffic [52]. While the deployment can be distributed [58], MIND's use case requires the ability to locally analyze and flag fraudulent transactions, meaning that a MIND instance needs to be able to act autonomously on transactions. Alternatively, a hybrid ML deployment [56] can be used to further improve classification performance by using a limited-size model on a switch and a large model on a server.

7 Conclusion

This paper presented a time-sensitive application of in-network ML for detecting transaction fraud. A prototype named MIND was developed, leveraging programmable network devices to detect fraudulent transactions within the data plane. MIND achieves high classification performance and line-rate throughput while maintaining ultra-low, microsecond-scale latency. Compared to server-based benchmarks, MIND processes over $\times 800$ more transactions per second and reduces latency by over $\times 1300$ per transaction. Moreover, the automated implementation of MIND facilitates the future processing of new datasets and other ML models. This work is a proof of concept for achieving ML-based in-network detection for transaction fraud based on real-time feeds.

Acknowledgements

This work was partly funded by VMware. We acknowledge support from Intel and NVIDIA.

References

[1] switch.p4. https://github.com/p4lang/switch/tree/master/p4src.

[2] A. Abdallah, M. A. Maarof, and A. Zainal. Fraud Detection System: A Survey. *J. Netw. Comput. Appl.*, 68:90–113, 2016.

[3] 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].

[4] J. O. Awoyemi, A. O. Adetunmbi, and S. A. Oluwadare. Credit Card Fraud Detection Using Machine Learning Techniques: A Comparative Analysis. In *ICCNI 2017*, pages 1–9. IEEE, 2017.

[5] A. C. Bahnsen, A. Stojanovic, et al. Improving Credit Card Fraud Detection with Calibrated Probabilities. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 677–685. SIAM, 2014.

[6] S. Bhattacharyya, S. Jha, et al. Data Mining for Credit Card Fraud: A Comparative Study. *Decis. Support Syst.*, 50(3):602–613, 2011.

[7] P. Bosshart, D. Daly, G. Gibb, et al. P4: Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[8] B. Branco, P. Abreu, et al. Interleaved Sequence RNNs for Fraud Detection. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3101–3109, 2020.

[9] S. Cao, X. Yang, et al. TitAnt: Online Real-time Transaction Fraud Detection in Ant Financial. *arXiv preprint arXiv:1906.07407*, 2019.

[10] F. Carcillo et al. Scarff: A Scalable Framework for Streaming Credit Card Fraud Detection with Spark. *Inf. Fusion*, 41:182–194, 2018.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, et al. SMOTE: Synthetic Minority Over-Sampling Technique. *JAIR*, 16:321–357, 2002.

[12] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, O. Rottenstreich, S. A. Monetti, and T.-Y. Wang. Fine-grained queue measurement in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 15–29, 2019.

[13] D. Chicco and G. Jurman. The Advantages of the Matthews Correlation Coefficient (MCC) over F1 Score and Accuracy in Binary Classification Evaluation. *BMC genomics*, 21(1):1–13, 2020.

[14] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi. Credit Card Fraud Detection and Concept-Drift Adaptation with Delayed Supervised Information. In *IJCNN 2015*, pages 1–8. IEEE, 2015.

[15] H. T. Dang, P. Bressana, et al. P4xos: Consensus as A Network Service. *IEEE/ACM Transactions on Networking*, 28(4):1726–1738, 2020.

[16] A. Gajjar et al. FAXID: FPGA-Accelerated XGBoost Inference for Data Centers using HLS. In *FCCM 2022*, pages 1–9. IEEE, 2022.

[17] T. Groenfeldt. IBM Puts AI On A Chip To Improve Fraud Detection In Real-Time Payments. *Forbes*, 2022. URL https://www.forbes.com/sites/tomgroenfeldt/2022/04/06/ibm-puts-ai-on-a-chip-to-improve-fraud-detection-in-real-time-payments/.

[18] B. Harris. Sparkov Data Generation. https://github.com/namebrandon/Sparkov_Data_Generation, 2022.

[19] K. Herrell. Defeating latency is at the heart of the AI, fraud and data challenges at banks. *BAI*, 2020. URL https://www.bai.org/banking-strategies/defeating-latency-is-at-the-heart-of-the-ai-challenge-at-banks/.

[20] X. Hong, C. Zheng, S. Zohren, and N. Zilberman. Linnet: Limit Order Books Within Switches. In *Proceedings of the SIGCOMM'22 Poster and Demo Sessions*, pages 37–39. 2022.

[21] X. Hong, C. Zheng, et al. LOBIN: In-Network Machine Learning for Limit Order Books. In *HPSR 2023*, pages 159–166. IEEE, 2023.

[22] X. Jin, X. Li, H. Zhang, et al. NetCache: Balancing Key-value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 121–136, 2017.

[23] X. Jin, X. Li, H. Zhang, et al. NetChain: Scale-free Sub-RTT Coordination. In *NSDI 2018*, pages 35–49, 2018.

[24] Y. Kou, C.-T. Lu, S. Sirwongwattana, and Y.-P. Huang. Survey of Fraud Detection Techniques. In *IEEE International Conference on Networking, Sensing and Control, 2004*, volume 2, pages 749–754. IEEE, 2004.

[25] C. Lao, Y. Le, K. Mahajan, et al. ATP: In-Network Aggregation for Multi-Tenant Learning. In *NSDI*, volume 21, pages 741–761, 2021.

[26] L. K. Lok, V. A. Hameed, and M. E. Rana. Hybrid Machine Learning Approach for Anomaly Detection. *Indonesian Journal of Electrical Engineering and Computer Science*, 27(2):1016, 2022.

[27] E. Lopez-Rojas. Synthetic Financial Datasets For Fraud Detection. https://www.kaggle.com/datasets/ealaxi/paysim1, 2017.

[28] E. Lopez-Rojas, A. Elmir, and S. Axelsson. PaySim: A Financial Mobile Money Simulator for Fraud Detection. In *EMSS 2016*, pages 249–255. Dime University of Genoa, 2016.

[29] A. Madhavi and T. Sivaramireddy. Real-time credit card fraud detection using spark framework. In *Machine Learning Technologies and Applications*, pages 287–298. Springer, 2021.

[30] N. Maynard. Online Payment Fraud: Market Forecasts, Emerging Threats & Segment Analysis 2022-2027. *Juniper Research*, 2022. URL https://www.juniperresearch.com/researchstore/fintech-payments/online-payment-fraud-research-report.

[31] S. Mittal and S. Tyagi. Performance Evaluation of Machine Learning Algorithms for Credit Card Fraud Detection. In *Confluence 2019*, pages 320–324. IEEE, 2019.

[32] S. Nami and M. Shajari. Cost-Sensitive Payment Card Fraud Detection Based on Dynamic Random Forest and k-Nearest Neighbors. *Expert Systems with Applications*, 110:381–392, 2018.

[33] X. Niu et al. A Comparison Study of Credit Card Fraud Detection: Supervised Versus Unsupervised. *arXiv preprint arXiv:1904.10604*, 2019.

[34] B. A. A. Nunes, M. Mendonca, et al. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications surveys & tutorials*, 16(3):1617–1634, 2014.

[35] V. Olteanu, A. Agache, A. Voinescu, and C. Raiciu. Stateless Datacenter Load-Balancing with Beamer. In *NSDI 2018*, pages 125–139, 2018.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

[37] C. V. Priscilla and D. P. Prabha. Influence of Optimizing XGBoost to Handle Class Imbalance in Credit Card Fraud Detection. In *ICSSIT 2020*, pages 1309–1315. IEEE, 2020.

[38] A. Reurink. Financial Fraud: A Literature Review. *Journal of Economic Surveys*, 32(5):1292–1325, 2018.

[39] Y. Sahin et al. A Cost-Sensitive Decision Tree Approach for Fraud Detection. *Expert Systems with Applications*, 40(15):5916–5923, 2013.

[40] D. Sanvito, G. Siracusano, and R. Bifulco. Can The Network Be The AI Accelerator? In *Proceedings of the 2018 Morning Workshop on In-Network Computing*, pages 20–25, 2018.

[41] A. Sapio, M. Canini, et al. Scaling Distributed Machine Learning with In-Network Aggregation. *arXiv preprint arXiv:1903.06701*, 2019.

[42] K. Shenoy. Credit Card Transactions Fraud Detection Dataset. https://www.kaggle.com/datasets/kartik2112/fraud-detection, 2022.

[43] G. Singh, R. Gupta, A. Rastogi, et al. A Machine Learning Approach for Detection of Fraud Based on SVM. *International Journal of Scientific Engineering and Technology*, 1(3):192–196, 2012.

[44] S. Singh. Fraudulent Transaction Detection Dataset. https://www.kaggle.com/datasets/sanskar457/fraud-transaction-detection, 2023.

[45] A. Tharwat. Classification Assessment Methods. *Applied computing and informatics*, 17(1):168–192, 2020.

[46] A. Thennakoon, C. Bhagyani, S. Premadasa, S. Mihiranga, and N. Kuruwitaarachchi. Real-time Credit Card Fraud Detection Using Machine Learning. In *Confluence 2019*, pages 488–493. IEEE, 2019.

[47] Y. Tokusashi, H. Matsutani, and N. Zilberman. LaKe: the Power of In-Network Computing. In *ReConFig 2018*, pages 1–8. IEEE, 2018.

[48] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman. The Case for In-Network Computing On Demand. In *Proceedings of the 14th EuroSys Conference 2019*, pages 1–16, 2019.

[49] M. Verleysen and D. François. The Curse of Dimensionality in Data Mining and Time Series Prediction. In *International work-conference on artificial neural networks*, pages 758–770. Springer, 2005.

[50] Z. Xiong and N. Zilberman. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *Proceedings of the 18th ACM workshop on hot topics in networks*, pages 25–33, 2019.

[51] S. Xuan, G. Liu, Z. Li, et al. Random Forest for Credit Card Fraud Detection. In *ICNSC 2018*, pages 1–6. IEEE, 2018.

[52] M. Zang, C. Zheng, T. Koziak, N. Zilberman, et al. Federated Learning-Based In-Network Traffic Analysis on IoT Edge. In *2023 IFIP Networking Conference (IFIP Networking)*, pages 1–6. IEEE, 2023.

[53] Zeller. Die Grundaufgaben der Kalenderrechnung auf neue und vereinfachte Weise gelöst, 1882.

[54] Z. Zhang, X. Zhou, X. Zhang, L. Wang, and P. Wang. A Model Based on Convolutional Neural Network for Online Transaction Fraud Detection. *Security and Communication Networks*, 2018, 2018.

[55] C. Zheng, X. Hong, D. Ding, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman. In-Network Machine Learning Using Programmable Network Devices: A Survey. *IEEE Communications Surveys & Tutorials*, 2023.

[56] C. Zheng, Z. Xiong, T. T. Bui, et al. IIsy: Hybrid In-Network Classification Using Programmable Switches. *IEEE/ACM Transactions on Networking*, 2024.

[57] C. Zheng, M. Zang, X. Hong, L. Perreault, R. Bensoussane, et al. Planter: Rapid Prototyping of In-Network Machine Learning Inference. *ACM SIGCOMM Computer Communication Review*, 2024.

[58] C. Zheng et al. DINC: Toward Distributed In-Network Computing. *Proceedings of the ACM on Networking*, 1(CoNEXT3):1–25, 2023.

[59] Z. Zojaji, R. E. Atani, A. H. Monadjemi, et al. A Survey of Credit Card Fraud Detection Techniques: Data and Technique Oriented Perspective. *arXiv preprint arXiv:1611.06439*, 2016.