Chapter 1

CRYPTOMINING TRAFFIC DETECTION VIA STATISTICAL FILTERING AND DEEP LEARNING

Jinxia Wei, Xizhang Huang, Wei Wan, Hao Fu, Mu Zhu, Yuhai Lu and Chun Long

Abstract As cryptojacking becomes increasingly prevalent, there is a growing need for efficient and accurate detection methods. However, since mining malware typically communicates with mining pools via encrypted traffic, detection models trained on traffic metadata often struggle to recognize new mining samples whose statistical characteristics differ from those of the training set. To address this issue, we propose Time-Embedding LSTM (TELSTM), an LSTM-based model that incorporates a novel timestamp embedding as supplementary temporal information. Instead of processing time intervals sequentially, our method derives patterns directly from the embedded timestamps, enabling TELSTM to capture more stable mining behavior and generalize better to unseen mining traffic. To ensure compatibility with high-speed networks, we also introduce an LSSVM-based classifier as a pre-processing stage. This filter efficiently discards irrelevant traffic, reducing the computational load on the subsequent deep-learning model and improving overall detection efficiency. Experiments on a dataset containing simulated campus traffic and real-world mining traffic show that our approach achieves a false negative rate of 2.57% and a false positive rate of 0.10%, demonstrating its potential for practical deployment.

Keywords: Cryptojacking, Encryption traffic, Deep learning, LSTM

1. Introduction

Cryptocurrency is a digital asset, and its core is blockchain technology. Cryptocurrencies are decentralized and rely on honest participants in the blockchain network to contribute computational power to ensure that blocks recording correct information growing at the fastest speed, also

known as the proof of work, to guarantee validity [1]. Participants who contribute computational power to the blockchain network are rewarded with cryptocurrency. While this model incentivizes more individuals to participate and help secure the network, it also introduces a new form of cyber threat known as "cryptojacking".

Cryptojacking refers to the act of attackers using a victim's machine computational resources without authorization to mine cryptocurrency for themselves. In recent years, cryptojacking has seen rapid growth, accounting for one-sixth of all malware incidents by the end of 2023 [2]. Cryptojacking incidents can occur on virtually any computer system connected to the Internet, posing a serious cybersecurity threat to personal computers, GPU computing clusters, and even Internet of Things (IoT) devices. Mining malware can spread through various means, including but not limited to vulnerability exploitation, brute-force attacks on weak passwords, and phishing emails. By distributing modified GPU drivers, attackers can exploit the computing resources of AI platforms for cryptocurrency mining [3]. By attacking popular websites and embedding mining scripts into web pages, attackers can also exploit users' browsers for illegal mining activities, affecting a large number of mobile devices. Websites such as YouTube, the Los Angeles Times, and even US and UK government websites have been affected by cryptojacking incidents [4]. Some researchers [5–7], based on their analyses of websites in the Alexa and Zmap top 1 million lists, have confirmed that cryptojacking is widespread in the wild. With the advancement of IoT technologies such as smart homes, the number of IoT devices has increased dramatically. The diversity of vendors, communication protocols, and hardware makes it very difficult to develop unified defense solutions, while also greatly expanding the attack surface for attackers [8]. Additionally, the need to remain constantly online and the widespread use of default passwords make IoT devices more vulnerable targets for mining malware.

Mining activities also result in significant energy consumption, leading to environmental issues. In 2020, the human health and climate damages caused by Bitcoin represented almost half of the financial value of each US dollar of Bitcoin created [9]. By 2024, the annualized energy consumption of Bitcoin mining had surpassed that of Poland, which ranks 24th in electricity consumption [10]. In order to protect public resources and the environment, governments and organizations in China, New York, and many other places are implementing bans on cryptocurrency mining [11–13]. Some anonymous cryptocurrencies are exploited by cybercriminals—for example, ransomware operators often demand payments in Monero to evade exposure. Unauthorized mining activity is also a strong indicator of system compromise. Developing effective de-

tection methods is essential for curbing illicit mining, promoting energy conservation and emission reduction, and combating cybercrime.

Many studies [14–19] have proposed various methods to detect mining activity. Among them, traffic-based detection is commonly adopted by organizations and institutions to combat cryptojacking attacks. This approach identifies mining activity by capturing the communication traffic generated during mining operations to locate infected hosts. Since there is no need to install detection software on every device, deploying a mining traffic detection system at the network gateway can protect all devices, including personal computers, mobile phones, and even IoT devices within the network.

Traffic-based mining detection faces three major challenges: 1) Mining malware often communicates with mining pools using encrypted traffic to evade deep packet inspection. As a result, highly distinctive mining features—such as protocol keywords related to pool subscription and job submission—cannot be accessed. Therefore, traditional machine learning methods that primarily rely on traffic metadata may not perform well when encountering new mining traffic whose statistical features differ from those in the training dataset. 2) Deep learning-based detection models often require a large amount of computational resources and cannot be deployed directly in networks with high traffic volumes. 3) Mining malware may employ malicious traffic manipulation techniques, such as inserting dummy packets, padding, and packet splitting [20], to distort communication patterns and evade detection.

We present three main contributions in this paper. First, we propose a novel timestamp embedding method that enhances the model's capacity to integrate information from multiple metadata fields and capture more discriminative features of mining traffic, thereby improving the recognition of real-world traffic that may differ from the training distribution. Second, we design a two-stage detection framework that successfully combines the speed of statistical models with the accuracy of deep learning models, making the application of complex deep learning approaches feasible for time-sensitive cryptomining traffic detection. Finally, we demonstrate that removing payload-free packets substantially increases the density of informative traffic packets within the detection window, which significantly bolsters the robustness of the deep learning-based detection model against malicious traffic manipulations. A minimal implementation of our method, which includes core functions from metadata extraction to alert generation, is available at: https://github.com/xizhang123/IFIP119-22nd-CryptoMining-Detection-Minimal-Demo.

2. Related Work

As the primary defense against cryptojacking, the detection of encrypted mining traffic has been studied for many years. Based on their detection strategies, existing approaches can be broadly categorized into three classes. The first category designs detection methods by analyzing communication patterns and identifying differences between mining and non-mining traffic [21–23]. The second category employs machine learning techniques, using metadata extracted from traffic packets as features and applying traditional machine learning classifiers for detection [24–26]. The third category combines the strengths of both approaches by designing deep learning models guided by the intrinsic characteristics of mining behavior, with the aim of achieving higher detection accuracy [18].

2.1 Detection based on Communication Patterns

Mining traffic exhibits distinctive communication characteristics, such as long-lived connections, periodic patterns, and stable packet sizes and TCP flag settings, which can be used to distinguish it from non-mining traffic. Vladimír Veselỳ and Martin Žádník. [26], Yebo Feng et al. [19], Xiaoyan Hu et al. [23] leverage the distribution of packet time intervals by comparing the Cumulative Distribution Functions (CDF) of observed traffic and labeled mining traffic using the KS-test. Shize Zhang et al. [21] further utilize blockchain-specific timing behavior, comparing the timestamps of downstream packets with newly generated blocks to compute a dynamic similarity score that improves robustness against false positives. Zhen Yang et al. [22] instead focus on packet size patterns, proposing a method based on Linear Diophantine Equations to capture the fixed-length structure of Stratum protocol traffic. This approach does not require capturing 100% of the traffic, making it more lightweight.

While these methods offer clear interpretability, their effectiveness often depends on simplified assumptions and manually designed rules. In diverse real-world network environments, such assumptions can lead to false alarms or missed detections. Some approaches, such as timestamp tracking [21], provide greater accuracy, but require long monitoring durations, which can hinder timely detection.

2.2 Detection based on Traditional Machine Learning

Traffic metadata contains crucial information for detecting cryptomining. The second class of research converts metadata fields, such as timestamps, packet size, payload size, and TCP flag settings, into numerical features, which are then used by traditional machine learning models. In 2022, Ege Tekiner et al. [8] used LogReg, K-Nearest Neighbors (KNN),Support Vector Machine (SVM) and Random Forest (RF) as classification models, with Tsfresh, an automatic feature extraction library, as feature extractor. Their method was able to detect cryptojacking in IoT networks efficiently.

In 2024, Haoran Sun et al. [27] proposed a two-stage mining detection method. The method uses machine learning models to finely recognize the type of cryptocurrency and performs active probing to avoid false alarms. Despite offering higher accuracy than rule-based methods, machine learning approaches can further refine their decision boundaries through retraining with new data. Their adaptability and ability to improve over time have made machine learning-based methods increasingly prevalent.

2.3 Detection based on Deep Learning

In 2022, research by Ki-Taek Lee et al. [20] demonstrated that techniques such as manipulating upstream traffic (e.g. packet padding, splitting and inserting dummy packets) from the compromised device, also called malicious traffic manipulation, could effectively bypass traditional machine learning models, posing new challenges for network-based cryptojacking detection.

Mining activities require timely communication, and disruptions that delay packets can significantly reduce profits [21]. To remain profitable, mining traffic must still retain inherent patterns, such as task distribution and result submission, which can be leveraged for detection. Deep learning models have powerful capabilities for modeling complex data. When designed based on the nature of mining activity, they can capture features that are more resistant to traffic manipulation. Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) are effective at capturing periodic patterns in sequences, which makes them suitable for mining detection.

Besides leverage CDF curves and KS-test Yebo Feng et al. [19] also proposed an LSTM-based model to distinguish whether mining activity was initiated by the user or by cryptojacking. Each input step to the model is a five-element tuple representing time intervals. As the model

processes the sequence, if it outputs "cryptojacking" at any step, the corresponding traffic is flagged. In 2025, Shaoke Xi et al. [18] leveraged the periodic nature of packet transmissions inherent to mining protocols and proposed a mining traffic detection system called MineShark. MineShark employs a CNN to process packet sizes and time intervals from both upstream and downstream traffic within a sliding time window. A key aspect of its design is the separate handling of upstream and downstream flows, along with the use of a sufficiently long detection window that ensures the inclusion of an adequate number of meaningful packets (e.g., flows with more than 50 packets in each direction). This architecture enables robust detection even under upstream traffic manipulation attacks [20], making MineShark one of the state-of-the-art approaches in cryptomining traffic detection.

Although MineShark demonstrates strong resistance to traffic manipulation, its robustness is constrained by its simple data preprocessing. According to the official preprocessing code, MineShark truncates all four feature sequences to the length of the shortest one before segmentation, an approach that assumes balanced packet counts in both directions. This can result in feature misalignment under normal asymmetric traffic conditions (e.g., one ACK acknowledging multiple data packets), impairing the model's ability to capture fine-grained interaction patterns. Moreover, the augmentation strategy of swapping inbound and outbound features implicitly imposes symmetry on fundamentally asymmetric client-server roles. Lastly, the scaling method (multiplying time intervals by 10 and dividing packet sizes by 10), while stabilizing value ranges, represents a rather simplistic treatment of metadata that may limit representational capacity. These implementation-level limitations highlight opportunities for more sophisticated feature engineering.

3. Methodology

In this chapter, we introduce the key characteristics of mining traffic and present our proposed detection method. The overall detection workflow is illustrated in Fig. 1.

Our method detects mining traffic based on TCP metadata. First, we extract metadata from all captured TCP packets. During the flow-tracking phase, packets are grouped into flows according to the IP addresses and port numbers of both endpoints, forming metadata sequences. For each TCP flow, the statistical features are calculated using standard statistical functions. These features are then fed into a fast filtering model, which determines whether a flow is suspicious and requires further analy-

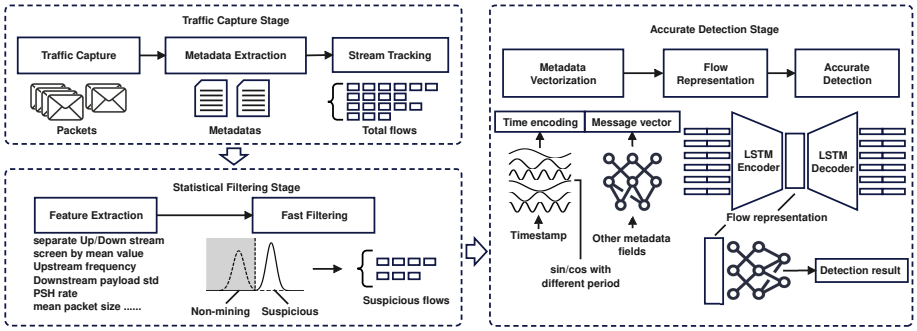

Figure 1. The detection workflow.

sis. Suspicious flows are vectorized and passed to the accurate detection model based on deep learning, which produces the final detection result.

3.1 Main Characteristics of Mining Traffic

In mining activities, miners and mining pools aim to maximize their profits. When a new block is generated, the mining pool immediately informs all miners to update their computational tasks to avoid wasting computational resources. This mechanism requires miners to maintain persistent connections with the pool, resulting in the frequent occurrence of PSH flags in downstream packets. Meanwhile, miners perform intensive computations and periodically submit results to the pool, resulting in longer yet stable interaction intervals. To ensure that results are processed promptly and not rejected due to latency, upstream packets also frequently set the PSH flag. Moreover, since mining tasks are monotonous and repetitive, communication packets tend to exhibit stable sizes. Long-lived connections, regular transmission intervals, stable packet sizes, and frequent use of the PSH flag collectively characterize the distinctive patterns of mining traffic. Fig. 2 illustrates the statistical differences between mining traffic [28, 29] and non-mining traffic [30, 31]. From the statistical plots, it is evident that mining traffic differs significantly from non-mining network traffic, providing a strong basis for detection through both statistical feature analysis and temporal sequence modeling.

3.2 Fast Filtering Model

The fast filtering model eliminates obvious non-mining traffic, reducing the load on the deep learning-based detection model and enabling scal-

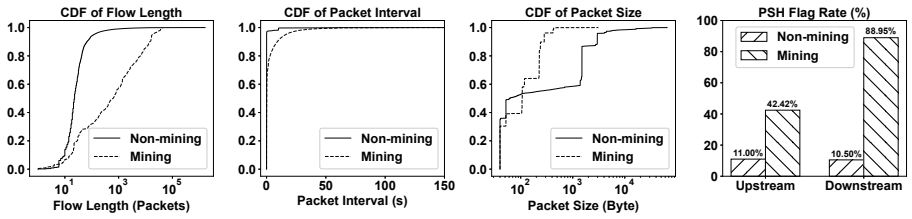

Figure 2. Statistical Differences Between Mining and Non-Mining Traffic.

ability in high-speed networks. It uses a linear regression algorithm to process statistical features and applies a threshold to distinguish mining streams from non-mining ones. In this section, we will first demonstrate the statistical feature extraction process and then show how to train and use the model.

3.2.1 Statistical Feature Extraction. The procedure of converting a TCP stream into a metadata sequence and extracting statistical features from it is illustrated in Fig. 3. Before extracting statistical features, we need to get the metadata sequence from a TCP stream. For each packet from the same TCP stream, five metadata fields should be extracted: timestamp, direction, packet size, payload size, and PSH (Push) flag setting. After getting the metadata sequence, the statistical features will be generated in three steps: packet feature extraction, payload feature extraction, and TCP flag setting feature extraction.

In the packet feature extraction and payload feature extraction step, we get the distribution features and frequency features of the packet and payload. Since the feature extraction steps for packets and payloads are identical in structure, we illustrate the process using packet feature extraction for clarity.

The distribution features describe the distribution of packets in both the time interval and packet size, while the frequency features describe the rates and patterns of packet transmission. In network communication, the client and the server play asymmetric roles, but the distribution features of all packets will not capture this property. So we need to add the upstream distribution features and downstream distribution features as supplements. For example, when we extract distribution features about packet time interval, we need not only the overall packet time interval sequence but also the upstream packet time interval sequence and downstream packet time interval sequence. In the same way, we also need sequences like upstream packet size sequence and downstream

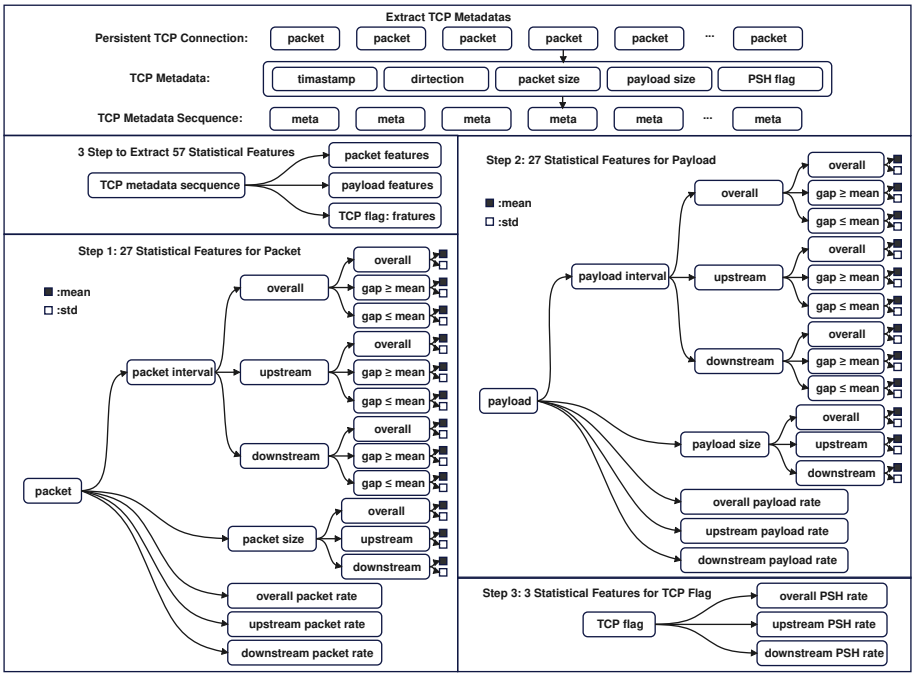

Figure 3. Extracting TCP metadata and obtaining statistical features.

packet size sequence as the supplement for the overall packet size sequence. Calculating the mean and std of the above sequences, we can get the distribution features.

Considering that the time information is more important for mining traffic detection, we need statistical features more accurate than common mean and std statistics. To accomplish this, when the mean value of the time-relative sequence has been calculated, we screen the elements based on whether they are greater equal than the mean value or less equal than the mean value, which will generate two new sequences. The new sequences retain information about the large or small values in the original sequence, thus, their mean and std statistics can provide additional insights. Since the frequency features are only meant to provide a new aspect for metadata, calculating the mean rate of the overall, upstream, and downstream sequences is sufficient.

Finally, the TCP flag setting feature extraction also needs the upstream PSH sequence and downstream PSH sequence as the supplement for the overall PSH sequence, and the flag setting rate can be used as a statistical feature. Following the above process, we will get 27 packet

features, 27 payload features, and 3 TCP flag setting features, summing up to 57 statistical features, giving a comprehensive description of TCP streams and suiting for mining detection.

3.2.2 Train and Use. To distinguish mining from non-mining traffic, we assign label 1 to statistical features extracted from mining flows and label 0 to those from non-mining flows. A Least Squares Support Vector Machine (LSSVM) with a linear kernel is trained on this $0, 1$ labeling scheme, using the bias term as the default classification threshold.

The computation procedure for the LSSVM model is shown in Equation 1. Here, the row vector $\mathbf{x}$ represents the input features, and the column vector $\mathbf{w^T}$ denotes the regression coefficients. The predicted value $\hat{y}$ is obtained by taking the weighted sum of the input features and adding the bias term. Alternatively, the bias can be treated as an additional coefficient by assigning a constant value of 1 to its corresponding feature, allowing the entire computation to be expressed purely as a dot product of two vectors.

$$\mathbf{x} \cdot \mathbf{w^T} + b = [\mathbf{x}||1] \cdot [\mathbf{w}||b]^T = \hat{y} \tag{1}$$

The goal of model fitting is to find the optimal regression coefficients that minimize the mean squared error between the predicted values and the true labels, as shown in Equation 2.

$$\arg\min_{[\mathbf{w}||b]} \sum_i (y_i - \hat{y_i})^2 \tag{2}$$

By treating the i-th component of the statistical feature vector as a random variable X_i, and the corresponding label as Y, the optimal regression coefficients can be obtained by solving Equation 3.

$$\begin{bmatrix} \overline{X_1^2} & \overline{X_1X_2} & \cdots & \overline{X_1X_n} & \overline{X_1} \\ \overline{X_2X_1} & \overline{X_2^2} & \cdots & \overline{X_2X_n} & \overline{X_2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \overline{X_nX_1} & \overline{X_nX_2} & \cdots & \overline{X_n^2} & \overline{X_n} \\ \overline{X_1} & \overline{X_2} & \cdots & \overline{X_n} & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ b \end{bmatrix} = \begin{bmatrix} \overline{X_1Y} \\ \overline{X_2Y} \\ \vdots \\ \overline{X_nY} \\ \overline{Y} \end{bmatrix} \tag{3}$$

Equation 3 can be incrementally constructed by processing the training data in batches, which avoids operating on large-scale matrices and improves memory efficiency and computational performance during training. Further extraction of polynomial features enhances the separability between mining and non-mining traffic. Since Equation 3 can be incrementally constructed, the model remains efficiently trainable even

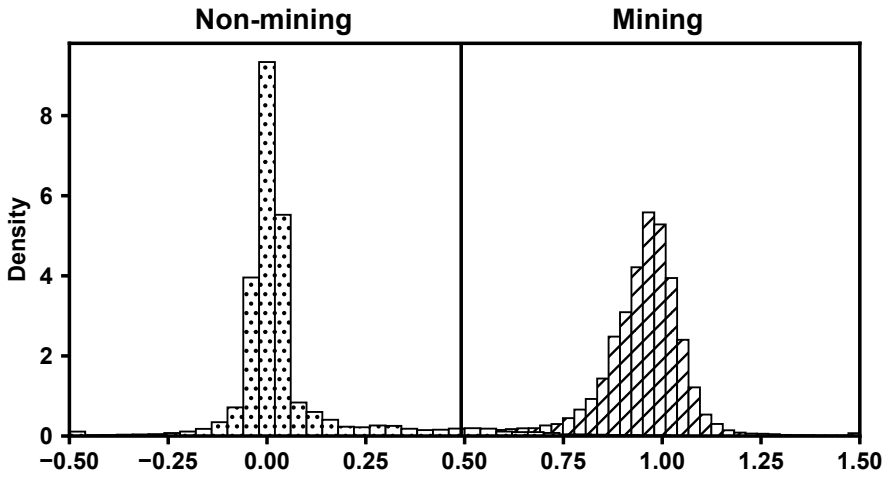

Figure 4. Separability between mining and non-mining traffic.

when the feature space expands to thousands of dimensions. We use the bias term as the default classification threshold to distinguish between mining and non-mining traffic based on the regression output. If the predicted value $\hat{y} \geq b$, the model marks the input feature as mining. The visualization of the model output is shown in Fig.4. It can be observed that mining and non-mining traffic are well separated under the default threshold. The threshold can be manually adjusted: shifting it left reduces missed detections, while shifting it right helps filter more non-mining traffic.

3.3 Accurate Detection Model

The deep learning-based Accurate Detection Model analyzes suspicious traffic filtered by the Fast Filtering Model to reduce false alarms and produce final results. It uses a multi-layer LSTM to extract temporal features from vectorized metadata, followed by a fully connected network for classification. We first describe metadata normalization and vectorization, then detail temporal feature extraction, and finally present the training and inference procedures.

3.3.1 Metadata Numerization and Vectorization. As shown in Table 1, each TCP metadata record contains multiple fields, among which four fields need to be processed: timestamp, IP length, payload length, and TCP flags. Direction information is not stored in a

Table 1. A TCP Metadata Sample.

Flow Name	Timestamp	IP Length	Payload Length	TCP Flags
IP1:Port1-IP2:Port2	10s.06us	821	781	011000

separate field. Instead, all fields except the timestamp are multiplied by –1 to indicate downstream packets.

The timestamp is recorded in Unix time. IP length covers the full IP datagram including the header, while the payload length refers only to the TCP payload. TCP flags indicate the status of six standard flags: URG, ACK, PSH, RST, SYN, and FIN. During processing, timestamps are handled separately. Each timestamp is converted to relative time by subtracting the first timestamp in the same stream. These relative times are passed to Algorithm 1 to generate the time-embedding.

Algorithm 1: Timestamp Embedding

Data: *Timestamp:* T, *Maximum and minimum period:* P_{min},P_{max}, *Half of time-embedding dimension:* N
Result: *time-embedding:* $V \in R^{2N}$
initialize $gap \leftarrow (P_{max}/P_{min})^{(1/N)}$
initialize $V \leftarrow zeros(2N)$
for $i = 1$ *to* N **do**
  $\alpha \leftarrow 2\pi/(P_{min} \cdot gap^i)$
  $V[i] \leftarrow sin(\alpha \cdot T)$
  $V[i+N] \leftarrow cos(\alpha \cdot T)$
end
return V

The IP length and the payload size are normalized using Equation 4. TCP flags are converted to six floating-point values. These, along with the normalized IP and payload lengths, are concatenated and fed into a fully connected neural network to generate the message-vector. Finally, the message-vector is scaled by a factor determined according to the training loss and then combined with the time-embedding to form the input at each time step of the model.

$$result = \begin{cases} v/1500, & v < 1500 \\ 1 + \ln(v/1500), & v \geq 1500 \end{cases} \tag{4}$$

3.3.2 Temporal Feature Extraction. After metadata numerization and vectorization, a multi-layer LSTM is used to process the sequence of vectors and extract temporal features. LSTM is a type of

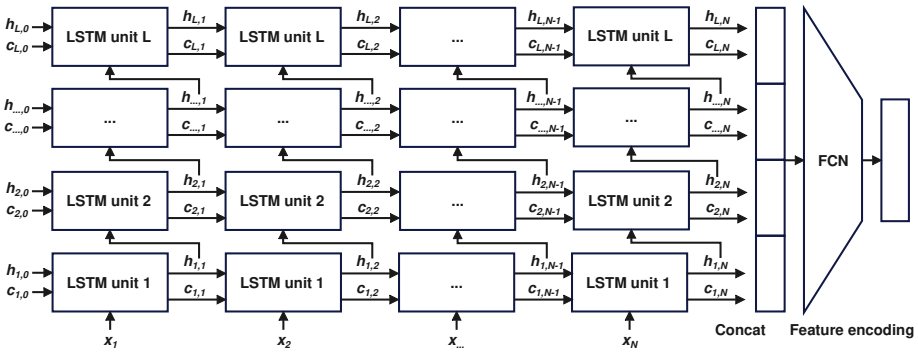

Figure 5. The temporal feature extraction process.

recurrent neural network with memory cells in its basic units, which can retain important information over multiple steps and is adept at modeling long sequences. The LSTM unit receives the current input x_t, the previous memory cell c_{t-1}, and the hidden state h_{t-1}, and then generates a new memory cell c_t and hidden state h_t. The detailed calculation process is as follows:

$$\begin{aligned} \mathbf{i} &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\ \mathbf{o} &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\ \mathbf{f} &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\ \mathbf{g} &= \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \\ \mathbf{c}_t &= \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \mathbf{g} \\ \mathbf{h}_t &= \mathbf{o} \odot \tanh(\mathbf{c}_t) \end{aligned} \tag{5}$$

The variables $\mathbf{i} \in R^d, \mathbf{o} \in R^d, \mathbf{f} \in R^d$ and $\mathbf{g} \in R^d$ represent the input gate, the output gate, the forget gate, and the update gate, respectively. Their values are determined by the learnable parameters $\mathbf{W}, \mathbf{U} \in R^{d \times d}, \mathbf{b} \in R^d$, as well as the current input $\mathbf{x_t} \in R^d$ and the previous hidden state $\mathbf{h_{t-1}} \in R^d$. The initial values of the memory cell and the hidden state, denoted by the vectors $\mathbf{c_0}$ and $\mathbf{h_0}$, respectively, are set to zeros. The sigmoid function σ is used to ensure that the values of the four control gates fall within the range $(0, 1)$. The forget gate determines which memories should be retained, the input gate and update gate decide which information needs to be added, and the output gate determines which information should be output at the current step or passed as the hidden state to the next step. The calculation process of an L layer LSTM handling a sequence of length N to extract temporal features is illustrated in Fig. 5.

The encoder processes the elements in the sequence one by one, finally generating L hidden states and L memory cells. Both hidden states and memory cells will be combined, then passed through a fully connected neural network to obtain the feature encoding.

3.3.3 Train and Use. The trainable components of the accurate detection model include the fully connected neural network that generates the message-vector, the multi-layer LSTM autoencoder, and the fully connected neural network that classifies temporal features to produce the final result. We train the model using both reconstruction loss and classification loss. The reconstruction loss, defined as the mean squared error (MSE) between input and reconstructed sequences, ensures that temporal features capture key metadata information, preventing overfitting and improving generalization. The classification loss is binary cross-entropy (BCE), guiding an accurate distinction between mining and non-mining traffic. Following Equation 6, the weight of the reconstruction loss gradually decreases following a cosine schedule, while the weight of the classification loss increases accordingly.

$$\begin{aligned} weight_{rec} &= (\cos(step \cdot \pi/(step_{total} - 1)) + 1)/2 \\ weight_{cls} &= 1 - weight_{rec} \end{aligned} \tag{6}$$

The timestamp encoder is fixed. We set the dimension of the time-embedding to 64 to cover periods exponentially ranging from 1/60 to 360 times the mean non-mining packet interval in the training set, effectively capturing packet sending rates.

4. Experiment

To verify the effectiveness of the proposed method, we conducted experiments using several public traffic datasets. In this chapter, we first present the baseline models used for comparison. Then, we introduce the public datasets and detail the data processing procedures. Finally, we present the evaluation results and the corresponding analysis, demonstrating the advantages of our proposed approach.

4.1 Baseline Models

To demonstrate the advantage of the LSSVM model in detecting mining traffic, we selected three traditional machine learning classifiers, namely KNN, RF, SVM, and included the KS-Test method proposed in CJ-Sniffer [19] as baselines. For the TELSTM model, we chose the CNN model from MineShark [18] and the LSTM detector used in CJ-Sniffer for comparison. Traditional machine learning classifiers were implemented

Table 2. Description of Training Sets

Dataset	Type	PCAP Files	Size
DI-cryptomining-detection-master [28]	Mined	7	211 KB
CJ-Sniffer-Dataset [29]	Mined	64	155 MB
CIC-IDS-2017 [30]	Non-mined	1	10.0 GB
ISCX-VPN-NonVPN-2016 [31]	Non-mined	140	25.7 GB

using the sklearn library, with the following hyperparameters: the number of neighbors for KNN was set to 3; the number of decision trees in RF was set to 30, with a maximum depth of 5; the SVM used an RBF kernel with parameters C = 0.1 and gamma = "scale". Slightly different from the original setting in CJ-Sniffer, we set the confidence level of the KS-Test to 0.01 instead of 0.1, as it performed better in our dataset. For MineShark, we transformed our data into the same format as its training data and ran the training program provided by the authors. The length of the metadata sequence in our experiments was 25, which yields only 24 time intervals. To accommodate this, we tested the CJ-Sniffer LSTM model using two different settings: one processing four time intervals per step and the other processing six.

4.2 Datasets and Usage

The training data were selected from four public traffic datasets, comprising two mining traffic datasets and two non-mining traffic datasets. To ensure that the model learns meaningful mining features rather than relying on distribution differences between datasets, we used another two datasets as an independent test set.

4.2.1 Training Sets. The details of four public traffic datasets used for model training are presented in Table 2. The DI-cryptomining-detection-master dataset, proposed by Veselý and Žádník in 2018, contains 13 PCAP files used to compare connection behavior between a probing tool and real mining software, with 7 files capturing real mining traffic. The CJ-Sniffer-Dataset, released by Yebo Feng et al. in 2022, includes 64 PCAP files of high-quality mining traffic under various scenarios, such as CPU/GPU mining, personal and server-based mining, and cryptojacking. The CIC-IDS-2017 dataset, provided by the Canadian Institute for Cybersecurity, comprises 5 PCAP files featuring a wide range of network threats and realistic benign traffic. One of the files contains entirely benign traffic generated using the B-Profile system [32], which simulates user behavior based on common protocols such

Table 3. Description of Testing Sets

Dataset	Type	PCAP Files	Size
Datacon2022-cryptomining-data [33]	Mix	1	256 MB
Auto-capture-cryptomining-data	Mined	12	4.75 MB

as HTTP, HTTPS, FTP, SSH, and email. The ISCX-VPN-NonVPN-2016 dataset, also released by the Canadian Institute for Cybersecurity, includes VPN and non-VPN traffic from seven application categories: web browsing, email, chat, streaming, file transfer, VoIP and P2P. This dataset was used to provide non-mining samples and enhance the robustness of our model.

4.2.2 Testing Sets. The two parts of the independent testing set are summarized in Table 3. The Datacon2022-cryptomining-data, provided by the DataCon Community, includes a PCAP file and the IP addresses of nine known mining pools. The dataset contains simulated campus traffic generated using Python's Selenium to randomly browse websites, along with mining traffic produced by running XMRig and NBMiner.

The Auto-capture-cryptomining-data supplements our test data set with newly captured, well-controlled Monero mining traffic. Using XMRig, we mined on three public pools and a custom-deployed fake pool. To enhance diversity, we treated variations in pool configurations, such as difficulty adjustments and TLS toggling, as distinct scenarios, each mined for 10 minutes. Traffic was captured under 12 hardware settings (CPU cores: 2, 4, 8, 12; memory: 4GB, 8GB, 16GB), enabling robust evaluation under various conditions.

4.2.3 Data Usage. We used a two-step process, consisting of metadata extraction and flow tracking, to generate experimental data from the original PCAP files. In the metadata extraction stage, we iterate through all PCAP files in the dataset, extract metadata for each TCP packet, and store it in a CSV file. Subsequently, in the flow tracking stage, the metadata entries are grouped by IP addresses and port numbers to generate metadata sequences. The direction of each packet (upstream or downstream), determined based on port ranges, IP address class, and communication order, was then attached to the metadata item.

We split the flows into fixed-length segments of 25 packets, which is suitable for machine learning methods. Since the MineShark CNN model

Table 4. Sample Numbers in Training and Testing Sets

Configuration	Condition	Training Set		Testing Set	
		Mining	Normal	Mining	Normal
25-packets	Original	36434	641176	545	11215
	Perturbed	63442	641176	1072	11215
	Perturbed + Filtered	22102	307937	238	7546
MineShark	Original	14122	170712	158	2486
	Perturbed	15000	170712	170	2486

requires both the upstream and downstream sequences longer than 50, we specially processed the data according to the authors' method for it. To maximize data utilization, two data preparation strategies are employed. For models capable of incremental training (e.g., deep learning), samples are dynamically truncated from the original long flows during training. In contrast, fixed-length samples are pregenerated for models that require processing the entire dataset at once (e.g., KNN), and are used exclusively for calculating the evaluation metrics (TP, FP, TN, FN). The KS-Test method, which is based on the CDF derived directly from the original mining traffic, does not require training.

To further evaluate the robustness of our method against malicious traffic manipulation, we introduced noise into the testing set following the approach described by Ki-Taek Lee et al. [20]. The perturbations included: (1) inserting a dummy upstream ACK packet every four seconds; (2) padding upstream packets with four bytes; (3) splitting large packets to ensure a maximum size of 500 bytes. The resulting dataset is referred to as the perturbation data. The sample numbers in training and testing set for calculating the evaluation metrics are shown in Table 4.

4.3 Evaluation and Analysis

This section presents a comprehensive assessment of our detection approach, covering three main aspects: first, a comparison between the LSSVM and traditional machine learning baselines; second, a performance evaluation of the TELSTM model against CJ-Sniffer LSTM and MineShark CNN models; and finally, an analysis of the overall efficacy and robustness of the cascade framework.

4.3.1 Performance Comparison of LSSVM with Baselines.

Table 5 compares LSSVM model with several traditional machine

Table 5. LSSVM Model with Machine Learning Baselines

Model	Training FPR	Training FNR	Testing FPR	Testing FNR
LSSVM: dim=1	4.1×10^{-2}	1.1×10^{-3}	1.1×10^{-1}	$\mathbf{3.1 \times 10^{-2}}$
LSSVM: dim=2	3.6×10^{-3}	3.3×10^{-4}	2.9×10^{-2}	5.5×10^{-2}
KNN	**0.0**	**0.0**	$\mathbf{8.9 \times 10^{-5}}$	3.2×10^{-1}
RF	8.2×10^{-4}	2.7×10^{-4}	3.6×10^{-4}	1.7×10^{-1}
SVM: RBF	2.2×10^{-3}	6.3×10^{-4}	8.9×10^{-4}	2.5×10^{-1}
KS-Test	1.8×10^{-1}	4.0×10^{-2}	2.1×10^{-1}	2.3×10^{-1}

learning models in both the training and testing datasets. The term "dim" indicates the degree of polynomial features added after extraction of statistical features. Compared to LSSVM models, machine learning models such as KNN, RF, and SVM perform well on the training set but show higher false negative rates (FNR) on the testing set. Using these models as fast filters may cause some mining traffic to be missed. The KS-Test model does not require statistical features from the metadata, which makes it easy to implement but suffers from low accuracy. LSSVM is the most ideal filter because of its achieving the lowest FNR in test dataset.

Using second-degree polynomial features significantly improves the separability between mining and non-mining traffic. This enhancement leads to reductions in both the false positive rate (FPR) and false negative rate (FNR) on the training set. Although the FNR increases slightly on the test set, the resulting lower false alarm rate is more suitable for traffic filtering applications.

4.3.2 Comparison of TELSTM with State-of-the-Art Models. In the TELSTM model, the timestamp embedding component is initialized using the average non-mining packet interval (0.186s), with a time-embedding dimension of 64. Metadata is projected through a three-layer fully connected network (input: 9, hidden: 128, output: 64) with ReLU activations, and a scale factor is applied prior to concatenation with the time embedding. We conducted a grid search to investigate the sensitivity of TELSTM to these hyperparameters. For the period range of the time embedding module ($P_{\min}$, $P_{\max}$), the loss heatmap in Fig. 6 (left) shows stable performance over 500 training steps, with only a narrow loss variation of 0.002; our chosen parameters (1/60, 360) fall within this effective region. Similarly, tuning the scale factor revealed that a value of 1024 yields the minimal classification loss after 3000 steps, while larger values bring negligible improvement. With this setup, TELSTM model achieves optimal performance within 1000 steps under the Adam

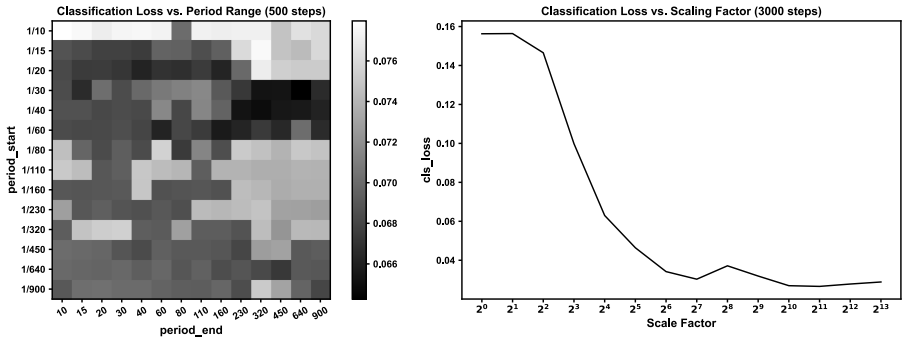

Figure 6. Hyperparameter Sensitivity Exploration.

Table 6. Performance Comparison of TELSTM with Baselines

	Training		**Testing**	
Model	**FPR**	**FNR**	**FPR**	**FNR**
TELSTM	1.1×10^{-2}	9.1×10^{-4}	2.0×10^{-3}	$\mathbf{2.4 \times 10^{-2}}$
CJ-Sniffer LSTM:4	7.4×10^{-2}	$\mathbf{6.9 \times 10^{-4}}$	4.3×10^{-3}	1.4×10^{-1}
CJ-Sniffer LSTM:6	6.9×10^{-2}	4.1×10^{-3}	4.0×10^{-3}	1.9×10^{-1}
MineShark CNN	$\mathbf{5.6 \times 10^{-4}}$	8.2×10^{-3}	$\mathbf{8.0 \times 10^{-4}}$	5.6×10^{-1}

optimizer (lr = 1e−5, betas=$(0.9, 0.98)$, ϵ = 1e−9, weight decay=0.1). In contrast, the CJ-Sniffer LSTM baseline requires a higher learning rate (3e−5) and about 6000 steps to converge, while the CNN model in MineShark is trained using the default configuration provided by its authors.

As shown in Table 6, TELSTM outperforms the MineShark CNN model and the CJ-Sniffer LSTM classifier in two different settings. This improvement is due to TELSTM's incorporation of timestamp embedding and its full utilization of metadata information, which leads to more effective mining traffic detection.

4.3.3 Ablation Experiments. To verify the validity of each component in TELSTM, we designed three ablation experiments by removing reconstruction loss, timestamp embedding, and message vector, respectively. The results are shown in Table 7. Since removing certain components may significantly alter the training dynamics, we adjusted the maximum number of training steps for each setting based on the observed loss curves. In the experiments, the model trained without

Table 7. Results of Ablation Experiments on TELSTM

	Training		**Testing**	
Configuration	**FPR**	**FNR**	**FPR**	**FNR**
Full Model	$\mathbf{1.1 \times 10^{-2}}$	9.1×10^{-4}	$\mathbf{2.0 \times 10^{-3}}$	2.4×10^{-2}
Disuse $loss_{rec}$	1.4×10^{-2}	1.0×10^{-3}	3.4×10^{-3}	5.1×10^{-2}
Timestamp only	7.2×10^{-2}	$\mathbf{2.7 \times 10^{-5}}$	8.4×10^{-3}	$\mathbf{3.7 \times 10^{-3}}$
Other Metadata only	1.6×10^{-2}	1.4×10^{-3}	4.2×10^{-3}	4.0×10^{-2}

Table 8. Cascade Detection Results

Stage	TP	FP	TN	FN	Negative	FPR	FNR
Fast Filtering	544	687	10528	1	11215	6.1×10^{-2}	1.8×10^{-3}
Accurate Detection	531	11	676	13	687	1.6×10^{-2}	2.4×10^{-2}
Final Result	531	11	11204	14	11215	9.8×10^{-4}	2.6×10^{-2}

the reconstruction loss and the model without the timestamp embedding were both trained for 1000 steps. However, the model without the message-vector required 3000 steps to reach stable performance.
Training the model without reconstruction loss or without timestamp embedding leads to a certain degree of accuracy degradation. The model that uses only timestamp embedding achieves an extremely low false negative rate (FNR), but due to the lack of supplementary information from other metadata, its false positive rate (FPR) is relatively higher. Therefore, reconstruction loss, timestamp embedding, and the use of additional metadata all contribute to improving the model's ability to detect mining traffic, with timestamp information being particularly crucial.

4.3.4 Performance and Robustness Evaluation of the Cascade Framework. The detection results of cascading LSSVM and TELSTM are shown in Table 8. In the fast filtering stage, we left-shift the default threshold by 0.3 to ensure that the proportion of missed mining traffic remains extremely low, while still efficiently filtering out most non-mining traffic. In the accurate detection stage, the number of false alarms is further reduced. This approach effectively detects mining traffic while minimizing the number of false positives.

Table 9 presents the impact of malicious traffic manipulation on the LSSVM, TELSTM, and MineShark CNN models. Since the MineShark model trained solely on our data did not perform well, we supplemented the evaluation with the MineShark CNN model loaded with pre-trained weights provided by the original authors. When dummy packet insertion,

Table 9. Malicious Traffic Manipulation Test Results

Model	Data Condition	Training		Testing	
		FPR	**FNR**	**FPR**	**FNR**
LSSVM	Original	2.4×10^{-2}	5.5×10^{-5}	6.1×10^{-2}	1.8×10^{-3}
	Perturbed	2.4×10^{-2}	3.1×10^{-2}	6.1×10^{-2}	$\mathbf{6.4 \times 10^{-2}}$
	Filtered	1.6×10^{-2}	2.4×10^{-1}	4.4×10^{-2}	6.3×10^{-1}
TELSTM	Original	1.1×10^{-2}	9.1×10^{-4}	2.0×10^{-3}	2.4×10^{-2}
	Perturbed	1.1×10^{-2}	1.2×10^{-1}	2.0×10^{-3}	3.8×10^{-1}
	Filtered	1.3×10^{-2}	4.1×10^{-4}	9.3×10^{-4}	$\mathbf{7.1 \times 10^{-3}}$
MineShark	Original	7.7×10^{-4}	1.4×10^{-2}	8.0×10^{-4}	5.4×10^{-1}
	Perturbed	7.7×10^{-4}	1.4×10^{-1}	8.0×10^{-4}	6.0×10^{-1}
MineShark*	Original	2.5×10^{-2}	4.4×10^{-3}	1.6×10^{-3}	4.4×10^{-2}
	Perturbed	2.5×10^{-2}	2.1×10^{-1}	1.6×10^{-3}	1.5×10^{-1}

Note: * denotes using author-provided pre-trained weights.

packet padding, and large packet splitting perturbations were applied to the upstream traffic, the LSSVM model, which uses statistically derived features (some of which focus on the downstream only), remained largely unaffected. In our experiments, it could still identify mining traffic with an alert threshold shifted by -0.3 from the default. Although MineShark also processes upstream and downstream features separately during its feature extraction stage, its CNN model, which ultimately processes the combined upstream and downstream feature sequences, still suffered from disruption.

Relying heavily on temporal features and operating with a relatively short detection window, the TELSTM model experienced an increase in False Negative Rate (FNR). This was due to the altered timing patterns and the reduced number of informative packets within the detection window. To mitigate this, we attempted to filter out void packets to increase the density of useful packets in the detection window; the results are also shown in Table 9.

MineShark requires a detection window with more than 50 packets in both upstream and downstream directions. Since many mining samples do not meet this requirement, we omit MineShark's results after filtering.

The experiments demonstrate that while filtering reduces the FNR of TELSTM, it more significantly alters the statistical features of the traffic, leading to an increased FNR for the LSSVM. In summary, we suggest applying the filtering operation only prior to TELSTM. This approach not only decreases the FNR of TELSTM but also reduces the overall detection load.

5. Conclusions

We propose TELSTM, an LSTM model incorporating timestamp embedding that demonstrates a strong generalization ability in mining traffic detection, and apply the LSSVM model to filter out most non-mining traffic to accelerate the overall detection workflow. Besides the mining feature representation and detection model selection, we also investigate the noise patterns in malicious traffic manipulation and the way to against it. Our approach integrates the advantages of both statistical and deep learning models, ensuring high efficiency and accuracy. It remains stable when encountering mining traffic with statistical features divergent from the training set, and robust against evasive traffic manipulation, demonstrating considerable promise for practical deployment.

Acknowledgments

This work was supported by National Key Research and Development Program of China (Grant No. 2023YFC3304704).

References

[1] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, 2008.

[2] SonicWall, 2024 SonicWall Cyber Threat Report, White Paper, 2024 (`https://www.sonicwall.com/medialibrary/en/white-paper/2024-cyber-threat-report.pdf`).

[3] R. Xiao, T. Li, S. Ramesh, J. Han and J. Han, Magtracer: Detecting GPU cryptojacking attacks via magnetic leakage signals, *Proceedings of the Twenty-Ninth Annual International Conference on Mobile Computing and Networking*, pp. 1–15, 2023.

[4] F. Naseem, A. Aris, L. Babun, E. Tekiner and A.S. Uluagac, MINOS: A lightweight real-time cryptojacking detection system, *Proceedings of the Network and Distributed System Security Symposium*, 2021.

[5] S. Eskandari, A. Leoutsarakos, T. Mursch and J. Clark, A first look at browser-based cryptojacking, *Proceedings of the IEEE European Symposium on Security and Privacy Workshops*, pp. 58–66, 2018.

[6] M. Musch, C. Wressnegger, M. Johns and K. Rieck, New kid on the web: A study on the prevalence of WebAssembly in the wild, *Proceedings of the Sixteenth International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 23–42, 2019.

[7] A. Kharraz, Z. Ma, P. Murley, C. Lever, J. Mason, A. Miller, N. Borisov, M. Antonakakis and M. Bailey, Outguard: Detecting in-browser covert cryptocurrency mining in the wild, *Proceedings of the World Wide Web Conference*, pp. 840–852, 2019.

[8] E. Tekiner, A. Acar and A.S. Uluagac, A lightweight IoT cryptojacking detection mechanism in heterogeneous smart home networks, *Proceedings of the Network and Distributed System Security Symposium*, 2022.

[9] A.L. Goodkind, B.A. Jones and R.P. Berrens, Cryptodamages: Monetary value estimates of the air pollution and human health impacts of cryptocurrency mining, *Energy Research & Social Science*, vol. 59, article 101281, 2020.

[10] Digiconomist, Bitcoin Energy Consumption Index, 2024 (`https://digiconomist.net/bitcoin-energy-consumption`).

[11] L. Ferré-Sadurní and G. Ashford, New York enacts two-year ban on some crypto-mining operations, *New York Times*, November 22, 2022 (`https://www.nytimes.com/2022/11/22/nyregion/crypto-mining-ban-hochul.html`).

[12] J. Crawley, Bitcoin mining appears to have survived ban in China, *CoinDesk*, May 17, 2022 (`https://www.coindesk.com/business/2022/05/17/bitcoin-mining-appears-to-have-survived-ban-in-china`).

[13] A. Ashraf, Microsoft bans crypto mining on its online services without permission, *CoinDesk*, December 15, 2022 (`https://www.coindesk.com/tech/2022/12/15/microsoft-bans-crypto-mining-on-its-online-services-without-permission`).

[14] A.Z. Chahoki, H.R. Shahriari and M. Roveri, CryptojackingTrap: An evasion resilient nature-inspired algorithm to detect cryptojacking malware, *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 7465–7477, 2024.

[15] A.Z. Chahoki, H.R. Shahriari and M. Roveri, Unmasking the unseen: CryptojackingTrap, the most resilient evasion-proof cryptojacking malware detection algorithm, *Proceedings of the IEEE International Workshop on Information Forensics and Security*, pp. 1–4, 2024.

[16] Ł. Pietraszek and W. Mazurczyk, A research environment for evaluating file-based cryptojacking detection techniques, *Proceedings of the International Wireless Communications and Mobile Computing Conference*, pp. 915–920, 2024.

[17] O. Sanda, M. Pavlidis and N. Polatidis, A deep learning approach for host-based cryptojacking malware detection, *Evolving Systems*, vol. 15, no. 1, pp. 41–56, 2024.

[18] S. Xi, T. Fu, K. Bu, C. Yang, Z. Chang, W. Chen, Z. Ma, C. Chen, Y. Shen and K. Ren, MineShark: Cryptomining traffic detection at scale, *Proceedings of the Network and Distributed System Security Symposium*, 2025

[19] Y. Feng, J. Li and D. Sisodia, CJ-sniffer: Measurement and content-agnostic detection of cryptojacking traffic, *Proceedings of the Twenty-Fifth International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 482–494, 2022.

[20] K. Lee, S. Oh and H. Kim, Poster: Adversarial perturbation attacks on the state-of-the-art cryptojacking detection system in IoT networks, *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 3387–3389, 2022.

[21] S. Zhang, Z. Wang, J. Yang, X. Cheng, X. Ma, H. Zhang, B. Wang, Z. Li and J. Wu, MineHunter: A practical cryptomining traffic detection algorithm based on time series tracking, *Proceedings of the Thirty-Seventh Annual Computer Security Applications Conference*, pp. 1051–1063, 2021.

[22] Z. Yang, J. Li, F. Cui, J.Q. Liu, Y. Cheng, X.N. Tang and S. Gui, Real-time symbolic reasoning framework for cryptojacking detection based on NetFlow-plus analysis, *Proceedings of the International Conference on Information Security and Cryptology*, Springer, pp. 251–271, 2023.

[23] X. Hu, B. Lin, G. Cheng, R. Li and H. Wu, Detecting cryptomining traffic over an encrypted proxy based on KS test, *Proceedings of the IEEE International Conference on Communications*, pp. 3787–3792, 2023.

[24] H. Li, Y. Hao, M. Lyu, X. Yu, B. Yang and L. Peng, An early stage identification of cryptomining behavior with DNS requests, *Proceedings of the International Conference on Advanced Data Mining and Applications*, Springer, pp. 30–44, 2023.

[25] X. Hu, Z. Shu, X. Song, G. Cheng and J. Gong, Detecting cryptojacking traffic based on network behavior features, *Proceedings of the IEEE Global Communications Conference*, pp. 1–6, 2021.

[26] V. Veselý and M. Žádník, How to detect cryptocurrency miners? By traffic forensics!, *Digital Investigation*, vol. 31, article 100884, 2019.

[27] H. Sun, R. Shi, L. Lan, Z. Peng and C. Wang, A two-stage encrypted cryptomining traffic detection mechanism in campus network, *Proceedings of the IEEE International Conference on Blockchain and Cryptocurrency*, pp. 602–610, 2024.

[28] V. Veselý and M. Žádník, Pcap files and datasets for digital investigation article, GitHub Repository (`https://github.com/nesfit/DI-cryptominingdetection`), 2018.

[29] Y. Feng, J. Li and D. Sisodia, Packet-level cryptomining network traffic dataset, GitHub Repository (`https://github.com/yebof/CJ-Sniffer-Dataset`), 2022.

[30] I. Sharafaldin, A.H. Lashkari and A.A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, *Proceedings of the International Conference on Information Systems Security and Privacy*, vol. 1, pp. 108–116, 2018.

[31] G.D. Gil, A.H. Lashkari, M. Mamun and A.A. Ghorbani, Characterization of encrypted and VPN traffic using time-related features, *Proceedings of the Second International Conference on Information Systems Security and Privacy*, SciTePress, Setúbal, Portugal, pp. 407–414, 2016.

[32] I. Sharafaldin, A. Gharib, A.H. Lashkari and A.A. Ghorbani, Towards a reliable intrusion detection benchmark dataset, *Software Networking*, vol. 2018, no. 1, pp. 177–200, 2018.

[33] DataCon Community, DataCon2022 network traffic analysis: Mining traffic detection dataset, DataCon Open Dataset Platform (`https://www.datacon.org.cn/opendata/openpage?resourcesId=31`), 2023.