MineShark: Cryptomining Traffic Detection at Scale

Shaoke Xi*†, Tianyi Fu*†, Kai Bu*†⋆, Chunling Yang*, Zhihua Chang*, Wenzhi Chen*, Zhou Ma*†,
Chongjie Chen‡, Yongsheng Shen‡, Kui Ren*†,
*Zhejiang University
{shaokexi, tianyifu, kaibu, chly, zhchang, chenwz, mazhou, kuiren}@zju.edu.cn
‡HANG ZHOU CITY BRAIN CO., LTD
{ccj, sys}@cityos.com

***Abstract*—The rapid growth of cryptojacking and the increase in regulatory bans on cryptomining have prompted organizations to enhance detection ability within their networks. Traditional methods, including rule-based detection and deep packet inspection, fall short in timely and comprehensively identifying new and encrypted mining threats. In contrast, learning-based techniques show promise by identifying content-agnostic traffic patterns, adapting to a wide range of cryptomining configurations. However, existing learning-based systems often lack scalability in real-world detection, primarily due to challenges with unlabeled, imbalanced, and high-speed traffic inputs. To address these issues, we introduce MineShark, a system that identifies robust patterns of mining traffic to distinguish between vast quantities of benign traffic and automates the confirmation of model outcomes through active probing to prevent an overload of model alarms. As model inference labels are progressively confirmed, MineShark conducts self-improving updates to enhance model accuracy. MineShark is capable of line-rate detection at various traffic volume scales with the allocation of different amounts of CPU and GPU resources. In a 10 Gbps campus network deployment lasting ten months, MineShark detected cryptomining connections toward 105 mining pools ahead of concurrently deployed commercial systems, 17.6% of which were encrypted. It automatically filtered over 99.3% of false alarms and achieved an average packet processing throughput of 1.3 Mpps, meeting the line-rate demands of a 10 Gbps network, with a negligible loss rate of 0.2%. We publicize MineShark for broader use.**

I. Introduction

Cryptojacking has seen rapid growth over the last few years, accounting for one-sixth of all malware incidents by the end of 2023 [1]. This trend is particularly concerning as cryptojacking malware has compromised critical infrastructure, leading to operational slowdowns, disruptions, and significant public safety risks [2]. Moreover, the legality of cryptomining is shifting. The governments of New York [3] and China [4], along with various organizations [5] are implementing bans to protect public resources and the environment. These changes highlight the critical need for timely and comprehensive detection of unauthorized cryptomining activities.

Among existing detection approaches [6], [7], [8], [9], [10], traffic detection [11] remains the primary method for most organizations to combat cryptomining threats. It provides comprehensive coverage across organizational networks by monitoring network entry and exit points. Specifically, the prevalence of pool mining activities [12], [13], [14], [15], [6], [16] presents intrusion detection systems (IDSes) with opportunities to inspect communications between internal mining devices and external mining pools, which are important signs of compromise.

Traditional IDSes typically rely on rule-based policies [17] or payload inspection [18]. However, rule-based approaches, relying on detection rules crafted by experts post-virus discovery, are not timely and only address known threats [19]. Similarly, payload inspection lacks scalability due to computational overhead and is ineffective against encrypted mining traffic [20]. In contrast, content-agnostic, learning-based detection offers flexibility and effectiveness in identifying emerging threats, even those using encryption [21], [22].

Although existing learning-based approaches to cryptomining traffic detection show promise on balanced labeled dataset [23], [24], [16], [14], their practical usage at scale faces significant challenges. In gateways where IDSes operate, the traffic is unlabeled and imbalanced between mining and non-mining classes. Furthermore, the relatively low volume of mining traffic in comparison to the overall input necessitates efficient detection algorithms. Moreover, the issues of imbalance and efficiency intensify as the scale increases. Without addressing the problems of lacking ground truth labels, class imbalance, and the need for line-rate processing, it remains uncertain whether learning methods can truly enhance IDSes in detecting a broader range of cryptomining threats.

To address these challenges, we propose a novel workflow that automates verification of model outcomes and establishes a feedback loop for continuous accuracy improvement, meanwhile ensuring efficiency. As depicted in Figure 1, incoming traffic is first processed by an inference pipeline to generate predictive labels. Flows labeled as mining are considered suspicious and subsequently evaluated by an automatic confirmation

†The authors are with the State Key Laboratory of Blockchain and Data Security & Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security, Hangzhou, China.
⋆Kai Bu is the corresponding author.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230402
www.ndss-symposium.org

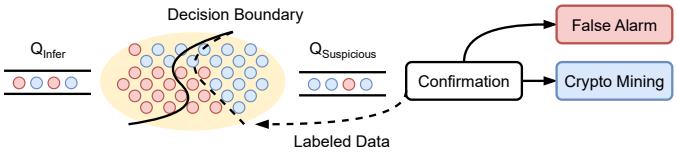


Fig. 1: MineShark employs an end-to-end learning-based detection workflow to resolve unlabeled, imbalanced, and high-speed traffic processing requirements.

module. Using model-independent features, the confirmation module distinguishes between false positives and actual mining activities and also provides reliable labeled data for model refinement. We establish two queues to monitor the processing speed: Q_{Infer} for model inference and $Q_{Suspicious}$ for suspicious-mining confirmation. Guaranteeing no loss in either queue ensures 100% traffic detection coverage. This workflow prepares the learning detection approach for deployment in real-world gateways. We refer to the workflow as a *detection pipeline* in the remainder of the paper.

Challenges in implementing a detection pipeline include: First, addressing imbalanced traffic input necessitates the selection of robust features. With features capturing key mining characteristics, the model can generalize to new mining samples and remain resilient to variations of legitimate traffic and noise in an online environment. Existing studies, however, fail to achieve robust detection due to their utilization of unstable features, as discussed in Section V-C.

Second, managing gateway traffic input requires line-rate inference and automatic confirmation. Inadequate speed could result in the discarding of undetected or unconfirmed flows at the tail of Q_{Infer} or $Q_{Suspicious}$, missing opportunities to identify potential mining flows. Regrettably, existing studies primarily evaluate efficiency in offline settings, and their dependence on manual confirmation cannot ensure no loss in $Q_{Suspicious}$, as detailed in Section VII-C4.

Third, unlabeled traffic input necessitates reliable confirmation. Learning methods are often plagued by a high rate of false alarms. This problem is exacerbated by high-volume and diverse gateway traffic. Providing confident detection results backed by convincing evidence is crucial to reducing operational overhead. Unfortunately, current research lacks in the design of a confirmation mechanism.

Finally, detection in highly imbalanced environments requires continuous model improvement with high-quality training data. To effectively refine the model's decision boundary, post-deployment data collection should be tailored to overcome biases toward the normal traffic class and the underrepresentation of the mining traffic class. However, current approaches depend on manual data collection, which is unsustainable and lacks precision. Additionally, the use of artificially balanced datasets may impair model's generalization ability.

We present MineShark, an online self-improving learning-based cryptomining traffic detection system. Our contributions are summarized as follows:

Robust cryptomining detection (Section V). Through comprehensive mining traffic collection, we find that the repetitive patterns in temporal mining message sequences remain consistent across varying configurations. This observation motivates us to adopt algorithms that identify the regularity of message exchange over sequences. In contrast, previous classifiers, depending on coarse-grained flow-level statistical features or timing features without considering the message order, exhibit poor generalization and are vulnerable to perturbations.

Line-rate inference (Section VI-B). MineShark achieves line-rate traffic detection via a GPU configurable in-memory pipeline, which is optimized for parallel computation, efficient memory usage, and compliance with storage limitations. It allows integration of different models and adaptable resource allocation to manage varying traffic volumes.

Reliable automatic confirmation (Section VI-C). MineShark constructs correlation graphs for suspicious connections in $Q_{Suspicious}$, enabling multifaceted analysis while preserving efficiency. It first filters out apparent false alarms, then exploits graph-level features to rank the remaining addresses. Based on the ranking, it conducts monitoring and active probing to ensure reliable confirmation. Moreover, the correlation graph helps profile the behaviors of mining malware, facilitating the detection of encrypted cryptomining, and aiding in the development of lightweight defenses against potential infections.

Self-improving model (Section VI-D). MineShark supports an iterative workflow to improve model accuracy after deployment. Beginning with an initial model, it periodically learns from labeled data derived from confirmation module, including misclassified benign and mining traffic, to refine the model's decision boundary. Over time, this automated process significantly improves model accuracy in detecting mining flows in the target environment.

Real-world deployment (Section VII). We deployed MineShark [1] to monitor our 10 Gbps campus gateway traffic over ten months (ethical concerns addressed in Section IV). With five CPU cores and one GPU accelerator, MineShark met the line-rate demands with a negligible loss rate of 0.2%. Through automatically removing over 99.3% of false alarms, it identified cryptomining activities associated with 105 mining pools before they were blocked by the concurrently deployed commercial IDSes. In addition, 71.6% of discovered mining addresses are prior to their disclosure by VirusTotal [25]. To the best of our knowledge, we conduct the first large-scale deployment of a learning-based cryptomining traffic detection system. Our work demonstrates the effectiveness of learning methods in augmenting real-world IDSes.

II. Background and Related Work

In this section, we first introduce the basics of cryptomining traffic and learning-based detection approaches. Next, we present MineShark's deployment as a typical use case and detail the key components in its detection pipeline. Finally, we introduce the *detection ratio* metric to assess detection effectiveness in the absence of input traffic labels.

A. Cryptomining Traffic and Learning-based Detection

In pool mining scenarios, a deployed or hijacked mining device communicates with pool servers using mining protocols. Stratum [26], the de facto protocol used by major mining

[1] https://doi.org/10.5281/zenodo.13624057

System	Feature and Detection Robustness	Inference Throughput	Cryptomining Confirmation	Training Data Collection
CJ-Sniffer [23]	Unique inter-packet delay distribution of inbonud traffic, lack accuracy and robustness.	10 Gbps	Manual	Manual
MineHunter [14]	Timing correlation with block generation in the cryptocurrency network, lack accuracy and robustness.	2.8 Gbps	Manual	Manual
IoT-Light [24]	Common statistical features of size and timing on a group of packets, lack robustness.	Not clear	Manual	Manual
Crypto-Aegis [16]	Common statistical features of size and timing on a group of packets, lack robustness.	Not clear	Manual	Manual
MineShark	Repetitive patterns in temporal sequences of packets characterized by unique sizes and timings, accurate and robust.	92 Gbps	Automatic	Automatic post-deployment

TABLE I: Comparison with the state-of-the-art cryptomining traffic detection systems.

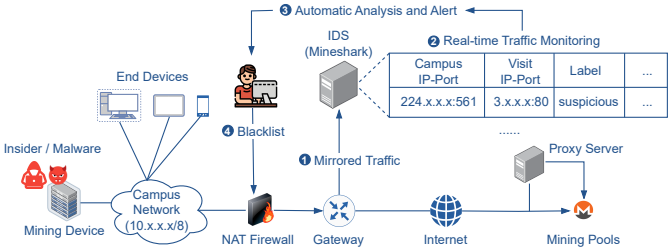


Fig. 2: Cryptomining detection at network gateway.

pools [27], is a JSON-RPC-based TCP protocol that transmits data either in plain text or over TLS encryption. A mining connection typically begins with a service subscription message initiated by the mining device. Following this, pool servers continuously assign computational jobs to the registered miner, who performs the calculations locally and submits the results to the pool server. The pool server acknowledges confirmed results, and the more results are confirmed, the higher the miner's profit. Consequently, mining connections can last for a long period, during which the mining traffic can be inspected by gateway IDSes.

To detect unknown or encrypted mining services, content-agnostic learning-based detection methods are being explored, which leverage distinct characteristics in mining traffic, such as unique packet sizes and message intervals, to identify new threats based on feature similarities. Notably, the learned features remain robust against common evasion techniques, such as traffic encryption or changes in mining pool addresses.

MineShark falls into the category of learning-based detection systems and follows standard IDS deployment practices [28]. As depicted in Figure 2, MineShark can be deployed at any network entry or exit point where traffic can be mirrored for analysis (❶). It employs a two-stage detection strategy: first, it performs fast filtration using a trained model at line rate to identify potential cryptomining activities (❷). Next, it confirms mining activities by eliminating false alarms through correlation techniques, followed by active probing or payload inspection to verify mining services. On identifying mining pool addresses, MineShark notifies network administrators and provides comprehensive visiting logs (❸), enabling them to update denylists and locate internal mining devices (❹).

B. Formulation of Detection Pipeline

We adopt the standard online detection framework to develop MineShark [29], [21]. The detection pipeline processes raw traffic input and reports detected mining addresses.

Traffic input consists of a continuous stream of packets $P_1, ..., P_i$ passing through the gateway. Each packet P_i belongs to a unique bidirectional flow F_{id}, identified by the five-tuple: {`srcip`, `dstip`, `srcport`, `dstport`, `proto`}. The `srcip` and `dstip` denote the source and destination addresses, respectively. When a NAT device is involved, as illustrated in Figure 2, the pair {`srcip`, `srcport`} can be reverse-mapped to pinpoint the internal mining device.

ML inference pipeline includes flow tracking, feature extraction, and model inference. Flow tracking assigns the i-th packet P_i to its belonged flow F_{id}. It extracts features from P_i to update the feature set of F_{id} within the current detection window. A detection window, of size ΔS, comprises a packet sub-sequence $P_m, ..., P_n$ in F_{id}, satisfying $n - m + 1 = \Delta S$. The pipeline generates a new feature vector for every ΔS packets and adds it to $\mathrm{Q}_{\mathrm{Infer}}$ (as depicted in Figure 1), then moves to the next window. This approach efficiently reuses memory allocated per flow across different windows. In cases where a time sliding window is used, a new feature vector is generated every ΔT time units instead of per ΔS packets.

The model retrieves feature vectors from $\mathrm{Q}_{\mathrm{Infer}}$ to compute similarity scores. Each score is compared against a decision threshold to assign a binary label. Thus, the similarity of flow F_{id} to cryptomining is represented by a list of labels and scores. Flows labeled as mining are then moved to $\mathrm{Q}_{\mathrm{Suspicious}}$ for confirmation. A flow is considered complete if the time elapsed since the last packet's arrival surpasses a predefined threshold. This threshold is set by profiling the maximum intervals between mining packets and adjusted during deployment to prevent system overload. To optimize memory usage, expired flows are removed to make space for incoming ones.

Mining confirmation reports confirmed mining addresses and ranks unconfirmed suspicious addresses. It dequeues suspicious flows from $\mathrm{Q}_{\mathrm{Suspicious}}$ and performs model-independent analysis. Flows are categorized as mining, suspicious, or false alarms based on their `dstip`. Mining flows require immediate action. Suspicious flows, while not conclusively dangerous, need further investigation. False alarms, indicating minimal risk, serve as valuable feedback for enhancing the model.

$$\begin{aligned}\text{Detection Ratio} &= \frac{\#\text{Confirmed}_{\text{Mining}}}{\#\text{Flows}_{\text{Mining}}} \\ &= \frac{\#\text{Confirmed}_{\text{Mining}}}{\#\text{Model}_{\text{Infer}}} \times \frac{\#\text{Model}_{\text{Infer}}}{\#\text{Flows}_{\text{Input}}} \times \frac{\#\text{Flows}_{\text{Input}}}{\#\text{Flows}_{\text{Mining}}}\end{aligned} \tag{1}$$

Conventional metrics like precision and recall are unsuitable for evaluation when input traffic lacks ground truth labels. Hence, we propose to assess system effectiveness by *detection ratio*, as defined in Equation 1. Intuitively, the ratio represents the fraction of identified mining flows ($\#\text{Confirmed}_{\text{Mining}}$) relative to the total mining flows present ($\#\text{Flows}_{\text{Mining}}$). It further factors in three elements: First, the accuracy of the detection pipeline, reflected by the ratio of confirmed mining flows to model-inferred flows ($\#\text{Model}_{\text{Infer}}$). This factor may diminish due to misprediction of mining flows (low recall), confirmation errors, or insufficient confirmation speed. Second, the efficiency of inference, indicated by the ratio of inferred flows to all the

system input flows ($\#Flows_{Input}$), which is directly impacted by the model's computational complexity. Third, the prevalence of mining traffic in the monitored environment, represented by the proportion of mining flows within the total input flows.

Equation 1 highlights the importance of selecting models with high inference throughput and high recall when deploying MineShark. This allows examining the maximum number of input flows while effectively filtering out most mining flows, thereby improving the detection ratio.

C. Related Work

Cryptomining(jacking) Detection. We compare MineShark with the state-of-the-art learning-based cryptomining traffic detection systems in Table I. We detail the robustness comparison in Section V-C. Regarding inference efficiency, despite some work estimates the throughput in an offline setting, a notable gap in prior work is the lack of an online detection pipeline, complicating direct runtime comparisons. For instance, systems without in-memory processing necessitate offline traffic storage. However, utilizing 1 TB storage can only capture 13 minutes of traffic on a 10 Gbps link, with no assurance of containing mining flows. Additionally, as reported by [24], predicting a single feature vector takes 100 ms by the algorithm. Given our measurements, as illustrated in Figure 8c, their settings could generate up to 50,000 feature vectors per second online, rendering the algorithm impractical to execute with limited resources, such as on a single server. Furthermore, MineShark is the only system that automates confirmation and training data collection for model refinement post-deployment, meeting the pragmatic demands of production environments.

Several detection tools are designed for specific cryptojacking scenarios without conducting traffic inspection. For instance, Li et al.[6] focus on detecting cryptojacking on Continuous Integration platforms by scanning configuration files, a method not well-suited for broad organizational use. MineSweeper[8] and CMTracker [9] target browser-based cryptojacking, leaving server-oriented attacks unaddressed. Zhang et al. [27] investigate stealthy mining pools, which helps enhance IDSes' denylists, and in turn benefits from IDSes' local detection results to refine probe packet construction and identify scanning ports. In contrast, MineShark provides a versatile solution for diverse organizational gateways and cryptojacking threats, thus complementing the capabilities of existing tools.

ML-based Anomaly Detection. Although existing ML-based detection tools can develop models on given datasets through a general workflow [30], the real-world applicability of these models is not assured [31], [32]. Generalizability problems can arise due to inductive biases within the training data, particularly when there is a significant class imbalance. MineShark bridges this gap with three-tiered efforts: First, it identifies robust features derived from analyzing mining traffic, enhancing the generalization ability from the algorithm side. Second, it distinguishes true mining from false alarms through active probing, tackling the issue of uninterpretable model outcomes. Third, it enriches training data with labeled data from the second step, incrementally improving model accuracy from the data perspective. This iterative workflow extends conventional ML development cycle and provides dependable detection results. Specifically, MineShark approaches the detection task as a binary traffic classification problem, using supervised learning algorithms that are widely-adopted in traffic classification [33], [34], [35], [36]. In contrast, unsupervised learning is typically used for detecting unforeseen threats, such as zero-day attacks [37], [21], [22].

Line-rate detection system. Existing efforts in building line-rate detection systems cannot directly apply to MineShark's detection pipeline, primarily due to the dependence on specialized hardware and limited support for ML models. For example, NetBeacon [38], FlowLens [39], and HorusEye [40] achieve line-rate processing through the use of programmable switches and N3IC [41] relies on SmartNICs. However, hardware programming is challenging [42], and none of them can execute deep learning models. While Retina [29] presents an efficient software framework to filter traffic at line rate for subsequent analysis, it demands substantial co-design of application logic. In contrast, MineShark implements an end-to-end detection pipeline that flexibly supports ML models and is compatible with GPUs for acceleration.

III. Threat Model

MineShark is deployed at network gateways to monitor cryptomining incidents. Administrators can determine whether the mining activity originates from a regular user or an infrastructure device based on the associated internal IP address. Infections on infrastructure devices pose high risks and are blocked immediately. The handling of mining activities by users can vary according to organizational policies.

The operator performs initial data collection and trains a model to integrate into MineShark's detection pipeline. Open-source labeled datasets of mining and non-mining traffic are available for download [24], [23]. Additionally, the operator can collect and label real-world mining and non-mining traffic by connecting varying mining devices to mining pools and sampling benign flows within the operational environment.

Once deployed, MineShark monitors a mirrored copy of gateway traffic, identifying mining flows without affecting the actual traffic. It reports confirmed mining addresses and ranks unconfirmed suspicious addresses to the operator. MineShark independently operates alongside other IDSes within the same network. Different IDSes concurrently detect mining addresses and contribute to a shared denylist, preventing internal access to these blocked addresses. When MineShark is not authorized to update the denylist, the time from its initial detection of a specific mining address to the cessation of any new connections to that address serves as a measure of the delay in addressing mining threats that other IDSes have overlooked.

The attackers can be cryptojacking malware or malicious insiders. They have regular user access to the internal network. They can connect internal (compromised) devices to external mining pools, with the mining traffic being either encrypted or in plain text. To maximize profits, attackers continue their cryptomining activities until detection leads to a ban. Voluntarily terminating a mining connection is uncommon unless it serves as a strategy to avoid detection. Upon experiencing repeated connection failures, an attacker might deduce that a mining address is blocked and consequently switch to alternative, accessible mining addresses. At present, MineShark does not

ensure protection against adversarial attacks aimed at exploiting the model's characteristics. Potential countermeasures are discussed in Section VIII.

IV. Ethics

Similar to commercial IDSes, MineShark needs to process network raw traffic, which contains sensitive user information. We describe detailed steps taken to mitigate the risks.

MineShark is deployed at our campus gateway. Our research plan was thoroughly reviewed and approved by the Information Technology Center (ITC) and the Institutional Review Board. The ITC manages the setup and maintenance of the monitoring server, and traffic mirroring ensures that user services remain unaffected by any malfunction in MineShark. The mirrored traffic is pre-processed to remove any authentication-related personal information, ensuring the confidentiality of sensitive data. Since authentication occurs via internal campus servers and MineShark monitors only traffic directed to the Internet, these two traffic types are topological distinct, making them easy to separate during mirrored traffic configuration. Additionally, dynamic network address translation anonymizes the input traffic to the IDSes. As we only observe public addresses post-translation and are not privy to the reverse mapping rules, the possibility of tracking individual users is mitigated.

MineShark operates locally and ensures that sensitive data never leaves the server. The model, trained on real traffic, focuses solely on non-payload information such as timestamps and packet sizes. During collection of training data, only the TCP five-tuple, packet timestamps, and sizes are recorded. For live traffic analysis, MineShark's in-memory pipeline extracts only timing and size information. In addition, as new packets arrive, old packet data are immediately replaced in the memory buffer, preventing any data persistence in storage.

For suspicious flows requiring further confirmation, we employ correlation techniques to filter out false alarms, relying only on flow-level features like duration, occurrence time, and model score. When false alarms or mining addresses are identified, we record the extracted features of packet timestamps and sizes of a few new flow instances (five in our settings) to aid in refining the model. Such data are automatically collected, analyzed, and deleted after a set period (six weeks in our case).

The only step involving payload inspection in MineShark is the automated keyword scanning of select packets (twenty at most) from suspicious flows, aimed at swiftly confirming plaintext cryptomining activities. This process generates a match result, with human inspection limited to payloads of matched flows only. For matched flows, a set number of raw packets (one hundred in our case) are stored for further analysis. This adheres to the ITC security team's requirement for manual confirmation of detection results by their experts.

To confirm mining addresses, we use non-invasive active probing that emulates real mining software by requesting services from the targeted addresses. To minimize the impact of probing, we perform one-shot probes on suspicious addresses with a daily maximum of approximately one hundred addresses. This is a negligible amount compared to our 10 Gbps detection scale. For confirmed mining addresses, we perform daily liveness tests that do not disrupt normal operations since these addresses are dedicated to providing mining services. The probe packets originate from a dedicated machine that hosts web pages with our contact information. We did not receive complaints during the study.

V. Robust Cryptomining Detection

Existing studies have confirmed the identifiability of cryptomining traffic patterns. However, robust detection is still challenging regarding varying mining configurations. In this section, we begin by characterizing cryptomining traffic. Following this, we outline our data collection process designed to maximize mining traffic variations. We then analyze the instability of current state-of-the-art detection algorithms and introduce MineShark's robust detection features.

A. Cryptomining Traffic Characteristics

Communications between miners and mining pools demonstrate uniform traffic patterns, which is computation-centric, mainly involving messages of job assignment and result submission. We study how diverse mining configurations affect the characteristics of application-layer communication and their network-layer representations, providing insights into the uniformity and variability inherent in cryptomining traffic.

Message size serves as an indicator of the type of messages exchanged in cryptomining. Each type of message consists of unique keyword sets, typically encoded in compact formats like `JSON`, fitting within a single packet. Thus, a mining flow can be distinguished by a pattern of uniquely sized packets. However, sizes are influenced by mining configurations, such as mining algorithms, causing identical messages to use different keyword sets across configurations and result in varied packet sizes. Nonetheless, within a single connection, packet sizes for the same type of message remain consistent.

Message timing (frequency) is influenced by the nature of mining pools and the hash rate of miners. Mining pools schedule job assignment messages to coincide with the growth of specific blockchains, affecting the pace of inbound traffic. Miners impact bidirectional message frequency as higher hash rates lead to more frequent result submissions and quicker receipt of confirmations from mining pools. Thus, bidirectional inter-packet delay is a reliable indicator of mining behavior. Additionally, computation times on mining devices introduce second-level delays, rendering the impact of millisecond-level network transmission delays on overall variance negligible.

Message order reflects the temporal relationships among various semantic messages. For example, an inbound job assignment can be followed by several outbound result submissions, with each submission succeeded by an inbound confirmation. This pattern repeats throughout the mining process and remains consistent across different setups. Furthermore, the packet sequence mirrors messaging patterns, particularly in the relative positioning of uniquely sized packets.

B. Cryptomining Dataset

Our data collection is designed to encompass a broad range of variations in real-world cryptomining traffic regarding message sizes, timing, and orders. We categorize the cryptomining

Category	Time (min)	# Flows	# Packets
Normal cryptomining [24], [23]	51,068	75	2,471,493
Obfuscated cryptomining	15,093	243	1,424,978
Perturbed cryptomining	2,880	48	221,400
Normal traffic [24], Gateway traffic	139,475	22,582	16,424,500

TABLE II: Dataset used throughout the paper.

Protocol	Packet Number	Packet Size	Inter-packet Delay
ShadowsocksR [43]	✔	✔ (Random)	✔
VMess [44]	✗	✔ (Random)	✗
VLESS [44]	✗	✔ (Fixed)	✗
Trojan [44]	✗	✔ (Fixed)	✗
OpenVPN [45]	✔	✔ (Fixed)	✔

TABLE III: Traffic features of obfuscation proxy.

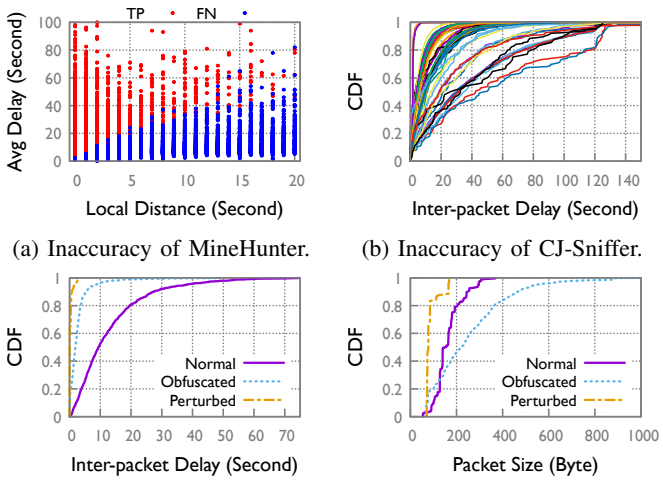
data into three types: normal cryptomining, obfuscated cryptomining, and perturbed cryptomining, as detailed in Table II. In addition to utilizing open-source normal cryptomining datasets from previous work [24], [23], we contribute cryptomining datasets specifically on obfuscation and perturbation.

The normal cryptomining dataset covers a wide range of mining devices, ranging from low-end Raspberry Pis to mid-end personal laptops and desktops, and extending to high-end tower servers and GPU machines. This range introduces over a 125-fold variance in hash rate. Specifically, the dataset in [24] was collected by executing cryptojacking malware, primarily targeting Monero. While the dataset from [23] contains user-initiated cryptomining, including Bitcoin, Ethereum, Ethereum Classic, Zcash, among others to test the generalization capability of detection algorithms.

In addition to configuration variations, intentional traffic manipulation poses challenges for traffic analysis, leading to potential evasion risks. We investigate obfuscation and perturbation threats, both of which lead to altered packet features affecting all detection algorithms listed in Table I. Specifically, we analyze five popular obfuscation proxies often employed for circumventing censorship. Their changes in traffic features are summarized in Table III. For example, `OpenVPN` modifies mining flows by changing packet sizes, reducing inter-packet delays, and disrupting packet orders due to the encapsulation of fixed-length VPN headers and the addition of keep-alive and control channel packets. Detailed explanations of obfuscation mechanisms can be found in Appendix B.

While obfuscation proxies aim to make specific traffic patterns indistinguishable, perturbation methods typically introduce small noise to prevent accurate analysis. Common techniques include inserting `dummy` packets at regular intervals, `padding` packets with a fixed number of bytes, and `splitting` single packets into smaller fragments. While `padding` alters packet sizes, both `dummy` insertion and `splitting` interfere with the timing and sequence of the original flow. Lee et al. [46] reported that the combined use of `dummy & padding & splitting` methods is the most effective at evading learning-based cryptomining traffic detection. We replicate their approach to ensure comprehensive testing.

We followed the traffic collection approach used in prior research [24] to gather proxied and perturbed dataset. We conducted collection on a single server (acting as a mining device) within the campus network, as illustrated in Figure 2. For each parameter combination listed in Table IV, we collected one-hour cryptomining traffic and labeled it as proxied mining. The mining software, set to default configurations, directed data to



(a) Inaccuracy of MineHunter. (b) Inaccuracy of CJ-Sniffer.

(c) Packet interval distribution. (d) Packet size distribution.

Fig. 3: (a) and (b) show the inaccuracy of MineHunter and CJ-Sniffer, when applied to normal cryptomining traffic as outlined in Table II. (c) and (d) show the distribution of packet intervals and sizes, averaged over a five-packet window from one-third randomly sampled traffic of each mining type in Table II. The perturbed mining simultaneously performed `dummy`, `padding`, and `splitting` operations.

the local machine's `SOCKS5` port, which was listened to by the proxy client. This client then connected to commercial proxy servers, which we accessed as paid subscribers. To capture the obfuscated traffic, we used `Wireshark` to filter traffic headed to the destination proxy server's IP and port during mining. In the case of `OpenVPN`, which operates differently from the `SOCKS5` proxy, we isolated its `tun` device within a separate `namespace` to avoid recording any noise traffic from other software. For perturbed cryptomining, we used the code from [46], which uses a similar setup as the `SOCKS5` proxy. We set the server machine within the campus network, which perturbed the traffic in communication with the collection machine yet maintained normal connections to external mining pools. We collected 16 hours of `dummy & padding` traffic and 32 hours of `dummy & padding & splitting` traffic on the collection machine with `Wireshark`, both labeled as perturbed mining.

Finally, we expanded the labeled normal traffic dataset (also referenced from [24]) threefold by sampling gateway traffic in our real-world evaluation environment (detailed in Section VII-A). To avoid potential packet loss on a 10 Gbps network, we opted for `Dpdk-pdump` [47] over `Wireshark` for traffic collection. Across five days, we randomly selected 1,000 flows per hour to ensure a broad diversity of applications in our traffic samples. For each connection, we recorded only packet timestamps and sizes.

C. Unstable Detection of Prior Work

Prior work falls short in robust detection due to their usage of unstable features that lack generalization capabilities. For example, MineHunter [14] and CJ-Sniffer [23] focus mainly on timing features and neglect the analysis of message order, which narrows their detection scope to specific mining configurations. IoT-Light [24] and Crypto-Aegis [16] utilize statistical features aggregated from a set of packets, which are vulnerable

Cryptocurrency	Mining Protocol	Proxy Protocol	Mining Pool	Server Location	Mining Hardware	Mining Software
Ethereum	Stratum+TCP/SSL, Ethproxy+TCP/SSL	SSR, Trojan, VMess, VLESS, OpenVPN	Ethermine, F2pool, Nanopool, 2Miners, Poolin, Flexpool, Herominers, 666pool, Ezil.me, Xnpool, Hiveon, etc.	Japan, Singapore, HongKong, France, America, Russia, Korea, etc.	RTX3090 * 4, RTX3090, RTX2060	NBMiner
Ethereum Classic						
Monero	Stratum+TCP/SSL		Supportxmr, Minexmr, 2miners, Nanopool, Xmrpool.eu, etc.		IntelXeonE5-2678, RTX2060	XMRig

TABLE IV: Summary of obfuscated cryptomining dataset.

to traffic obfuscation and perturbation. We demonstrate the inaccuracy of each algorithm using our collected dataset. A detailed comparison is presented in Section VII-B.

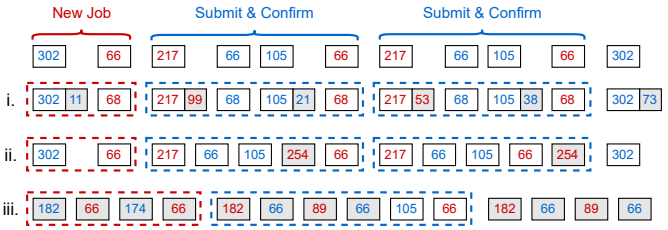
MineHunter evaluates cryptomining flows by examining the delay of the first packet arrival after each new block is generated in a cryptocurrency's blockchain, which is referred to as *local distance*. It employs a time-based detection window synchronized with the blockchain's block generation interval. Within each window, a similarity score is computed, where a higher score suggests a higher likelihood of mining activity. Figure 3a displays the detection results for normal cryptomining traffic from our dataset using the recommended score threshold of 0.6. Each data point corresponds to the outcome of one detection window. The average delay, depicted on the y-axis, is derived by dividing the window size by the total number of packets within that window. We find that MineHunter's heuristic-based scoring method can lead to wrong prediction, particularly when packet arrivals do not consistently follow the new block generation timing, as is often the case for mining flows with low message frequency.

CJ-Sniffer identifies mining flows by matching their inbound inter-packet delay distribution against known mining patterns using the Two-Sample Kolmogorov–Smirnov test. However, as Figure 3b demonstrates, the delay distribution for normal mining traffic exhibits significant variation due to different configurations; each curve corresponds to the delay distribution for an individual mining flow. This variation leads to inconsistent identification: the same flow may be labeled as mining when compared to certain samples in the database, yet benign against others. Increasing the number of comparison samples can result in more false positives, complicating the effort to establish a reliable detection setting.

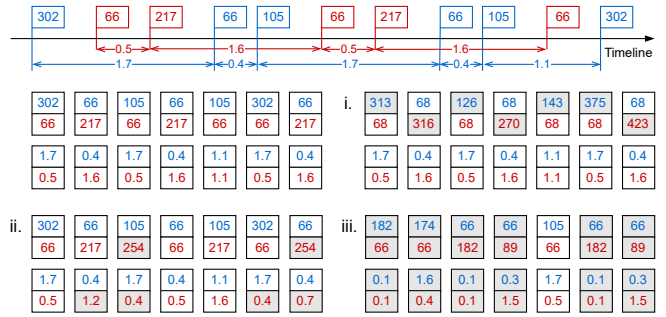
IoT-Light and Crypto-Aegis derive statistical traffic features, such as mean packet delay, from raw packet features in fixed-length windows to train ML models. However, as shown in Figures 3c and 3d, obfuscation and perturbation can significantly alter raw packet features, evading classifiers trained solely on normal mining traffic. While retraining with a broader dataset is a potential defense, it raises concerns about the ability of the classifier to detect unseen mining activities without having been trained on their distinct traffic features.

D. Features of MineShark

MineShark achieves robust detection by analyzing temporal patterns in sequences of packets with unique sizes and timings. It is effective due to the inherent regular patterns in mining communication accross various setups. As shown in Figure 4a, each message in a mining flow corresponds to a deterministic set of bidirectional packets. For instance, an inbound job assignment packet is typically followed by an outbound ACK packet, and an outbound submission packet is succeeded by an inbound confirmation packet, with an interleaving inbound ACK packet. Such inter-message regularity, rooted in mining



(a) Altered features of the original mining flow.



(b) Feature extraction of MineShark.

Fig. 4: Features of the original and manipulated mining flows, including: (i) random padding, (ii) dummy insertion, and (iii) packet split. Blue and red suggest inbound and outbound packets, respectively. Grey boxes represent altered features.

protocol design, persists across diverse mining configurations. We further illustrate the feature consistency of manipulated mining flows. The random padding alters packet sizes but maintains interval regularity. The dummy insertion impacts packet interval and order at insertion points, shifting subsequent original patterns. The packet splitting leads to a feature split, but the regularity across messages remains largely intact.

MineShark utilizes a fine-grained representation to capture message regularity. Specifically, it constructs $4 \times N$ matrices to represent features of packet size, timing, and order, where N indicates the size of the detection window. As illustrated in Figure 4b, the first two rows of the matrix correspond to bidirectional packet sizes, and the last two rows to inter-packet delays. The sequence of features in each row preserves the packet order. This representation effectively aligns with how traffic manipulation impacts feature sets. Random padding, modifies size features at the positions of non-ACK packets. Dummy insertion leads to a shift in the original feature sequence at regular intervals. Packet splitting results in a corresponding feature split for fragmented packets.

MineShark analyzes the feature matrices using a CNN model because of its proficiency in recognizing regular patterns. Although CNN has also been applied to other traffic analyzing tasks, such as website fingerprinting or flow correlation [48], [34], MineShark initiates the first attempt to apply it for cryptomining detection with obfuscation and perturbation

setups. Training details are provided in Appendix A.

VI. MineShark System

In this section, we present MineShark's detection pipeline, as formulated in Section II-B. Concretely, the line-rate inference pipeline addresses efficiency concerns and automatic mining confirmation avoids the overwhelming of false alarms and facilitates mining discovery, especially the encrypted mining traffic. In addition, we apply a self-improving framework to update the model with accumulated labeled data over time.

A. Overview

Figure 5 illustrates MineShark's end-to-end detection pipeline. The inference pipeline drains input traffic by first locating the in-memory connection record for each incoming packet (❶). Then, packet features are appended to the connection's detection window (❷). Once the window reaches its capacity, the pipeline transforms it into a feature matrix and sends the matrix for model inference (❸). Original window is cleared to accommodate new features. A flow is considered suspicious if any of the detection window gets positive prediction result (❹). A number of subsequent packets are sent for keyword scanning, which aims to confirm plain-text cryptomining immediately (❺). Note that keyword scanning cannot keep up with line-rate traffic without adopting the model to select suspicious targets. Meanwhile, subsequent features of suspicious flows are persistent to storage.

An automatic confirmation module employs a set of worker threads to correlate external addresses using saved feature documents (❻). During this process, specific addresses are monitored to uncover existing or emerging mining threats (❼). Apparent false alarms are automatically removed by algorithm. Remaining suspicious addresses are ranked for monitoring as well as human investigation (❽). Active probing is extensively performed in various confirmation operations (❾).

The updating framework gathers detected mining traffic as well as misclassified benign traffic to improve model performance (❿). The label correctness is guaranteed by the confirmation module.

B. Line-rate Inference

We *implement the inference pipeline from the ground up* as generic detection systems do not support deep learning (DL) models. We identify the following key performance contracts to avoid detection failure.

Line-rate feature extraction. No packet loss occurs during feature extraction is crucial as ML models are typically trained on features of complete connections. For example, losing features of job assignment packets may result in misclassification for the corresponding detection window.

Zero loss of model inference. DL models can be computationally expensive. It is important to guarantee that all extracted features are predicted by the model, i.e., no drop of Q_{Infer}.

To ensure efficiency, we utilize the following strategies:

Kernel-bypassing framework. MineShark performs stateless connection tracking to extract packet size and interval features, eliminating overhead exposed by kernel networking stack.

Multi-core parallelism. The pipeline executes feature extraction, model inference, packet inspection, feature recording, and timing modules on separate CPU cores. The mapping between CPU cores and modules supports a many-to-many configuration, enabling elastic resource allocation based on the traffic input. Data sharing is facilitated through zero-copy ring buffers, reducing synchronization costs.

GPU acceleration. We leverage GPU acceleration to address the inefficiency of CPU inference. To demonstrate the bottleneck, we provide a detailed evaluation in Section VII-D.

Efficient IO management. MineShark saves features of suspicious connections into documents. To avoid IO bottlenecks, it stops recording when the positive detection ratio drops below 1%, saving 58% IO bandwidth in deployment.

Time module and system monitoring. The time module (not shown in Figure 5) periodically clears expired flow contexts and monitors system status. The timeout threshold is set to be longer (120 seconds) for suspicious connections to increase their detection ratio as compared to normal connections (60 seconds). To prevent packet loss during feature extraction, we allocate adequate CPU resources to maintain a sufficient extraction speed, avoiding any increase in packet-loss counters in the NICs. In case of any violations of the performance contracts, alarms are triggered to the administrator.

C. Cryptomining Confirmation

MineShark steps toward flexible cryptomining detection by first driving model inference to line rate. However, the inference results still need confirmation. We explore efficient algorithms to maximize mining discovery and minimize false alarms, meanwhile, guaranteeing no drop of $Q_{\text{Suspicious}}$.

Correlation graph. A correlation graph, denoted as $\mathcal{G}$, is shown in Figure 6a. Each leaf node, denoted as $\mathcal{R}$, represents a suspicious connection detected by the inference pipeline. Relative connections are first grouped by internal IP-Port and then external IP-Port. Visits toward the same external IP addresses as well as IP addresses resolved to the same domain are correlated. Note that the internal IP-Port can be a public address on NAT devices (illustrated in Figure 2). Therefore, MineShark reports external IP addresses to block.

MineShark performs monitoring on confirmed mining addresses or risky suspicious addresses, establishing new connections in the correlation graph that assist mining discovery.

Address monitoring. Given a specific monitored address, IP-Port and IP-Domain correlation is performed. Port correlation is effective as mining pools typically open multiple ports for miners of different hash power or employing different setups (e.g., using HTTP or TLS protocol) to connect. Identifying one of them can expose others. This is especially useful in confirming encrypted mining traffic if observing plain-text mining traffic associated to other ports on the same server. Therefore, the inference pipeline records any port visits to the monitored IP and performs keyword scanning and feature recording to all the visit flows. Probing packets are sent to the recorded ports to request mining service. An established mining connection confirms the address hosting a pool server.

Domain correlation is effective when an address links to current or historical domains associated with cryptomining, or

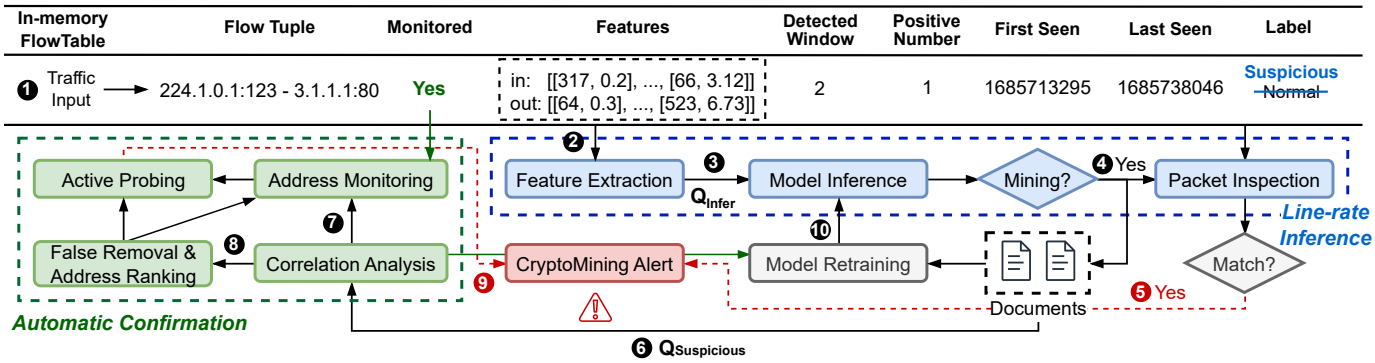
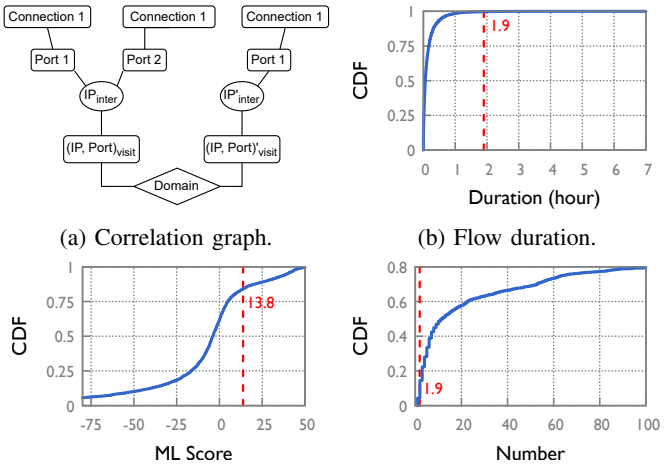


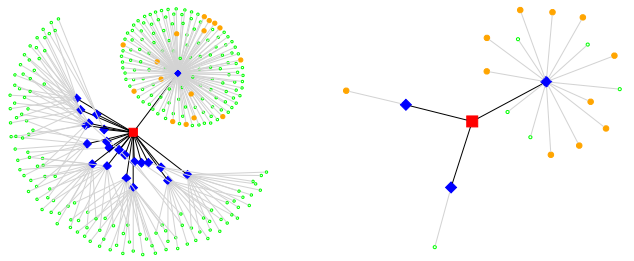
Fig. 5: Workflow of MineShark's detection pipeline.



(a) Correlation graph. (b) Flow duration.

(c) Average score. (d) Parallel interval visits.

Fig. 6: (a) Correlation graph and statistical analysis of 10,000 suspicious connections, including: (b) flow duration, (c) ML score, and (d) parallel internal visits of 1,000 suspicious addresses. Dashed line represents mining average.



(a) Benign address. (b) Mining address.

Fig. 7: Visit pattern of (a) benign and (b) mining addresses. Red and blue nodes represent external and internal addresses. Green and orange nodes represent harmless and risky visits.

when related addresses are involved in mining. Both scenarios increase the suspicion of the monitored address. Therefore, the confirmation module aggregates domain information for the monitored address and probes associated addresses to identify mining pools. Newly confirmed mining addresses are added to the search list, until no further addresses are identified or a predefined time limit is reached. This process forms a defensive denylist, thwarting malware attempts to bypass blocking via domain exploitation. Internal visits hitting the denylist trigger immediate alerts.

Although effective, resource and efficiency constraints limit the monitoring of every suspicious address. For example, domain information collection requires paid services, and correlation graphs require frequent updates to reflect the latest changes. Moreover, matching packets against monitored addresses and extensively logging port visits can decelerate the inference pipeline. Given large amount of suspicious addresses flagged by the model, MineShark first filters easy false alarms by counting harmless visits within correlation graphs.

False alarm removal. Easy false alarms are characterised by exhibiting mining similarity in only one or several detection windows in a connection. We use a metric called *harmless visit ratio* to identify benign addresses, which refers to the proportion of harmless visits to total visits. The idea is that if majority visits exhibit low mining similarity, then the occasional high similarity poses lower risk. A *harmless visit* is a reported connection ($\mathcal{R}$) with only one positive window. We exclude connections with zero positive window in graph construction to reduce noise. As illustrated in Figure 7, the harmless visit ratio (τ) of benign address is significantly higher than that of the mining address. Based on this observation, we consider an address benign if its τ exceeds a predefined threshold ($\mathcal{T}$). Note that τ of each address can change over time. Some addresses may fluctuate between benign and suspicious based on the evolving visit patterns. Active probing is performed to avoid misclassifying mining addresses as benign. Notably, this algorithm reduces false alarms from $\mathcal{O}(10^5)$ to $\mathcal{O}(10^2)$ per day, as evaluated in Section VII-C4.

Finally, MineShark ranks the remaining addresses based on long-term behavioral features exhibited in correlation graphs, which prioritizes more suspicious addresses to monitoring and provide reliable labels to the updating framework.

Address ranking. We characterize suspicious mining connections using features of duration and average ML score. Mining duration tends to be long compared to normal connections. For example, 74% of suspicious connections last less than 10 minutes, as shown in Figure 6b. However, the average mining duration is 1.9 hours. Duration alone is insufficient, as many legitimate applications also maintain long connections. We further calculate the average ML score to reflect long-term mining similarity. As shown in Figure 6c, 62% of connections have a negative average score, while mining connections have

an average score of 13.8 when the model threshold is set as 2 (see Appendix A for parameter details).

Address features are aggregated from associated connections. Specifically, the duration of an address, denoted as $\mathcal{V}$, is the maximal duration among connections, i.e., $\mathcal{V}.duration = \max_{\mathcal{R}\in\mathcal{G}}(\mathcal{R}.duration)$. This intends to highlight long duration behavior. Aggregated ML score is determined by the harmless visit ratio (τ) and the sum of per-connection scores. Formally, $\mathcal{V}.score = (1 \pm \tau) \cdot \sum_{\bar{\mathcal{R}}\in\mathcal{G}} \bar{\mathcal{R}}.score$, where the $+$ is used for a positive sum value and $-$ for a negative one. $\bar{\mathcal{R}}$ suggests counting risky connections only. Hence, the ML score can be penalized by a significant τ, lowering the ranking position of benign addresses. In addition, we exploit timing relations between independent internal visits by counting the number of unique internal addresses that launch connections to a specific suspicious address within a two-hour time window. As shown in Figure 6d, the scale of simultaneous mining to the same pool does not exceed two machines. In contrast, visits toward less than 20% of benign addresses behave similarly.

Finally, we employ DAS ranking [49] on address features, which prioritizes addresses with longer duration, higher score, and lower degree of parallel visits in front. Addresses with better results in any two features are placed next, followed by addresses that are only significant in one feature dimension. Top ranking addresses are prior in monitoring and investigation. We demonstrate well separation between benign and mining addresses using three features in Section VII-C5.

Active probing. We construct both plain-text and encrypted probing packets and target commonly used mining ports under the guidance of work [27]. For confirmed mining addresses, we perform per-day probing to update the address status. For other cases, the probing is one-shot per address. Although probing may misidentify mining addresses as well, for example, targeting wrong ports or using wrong packet format, a successful response can confirm mining activity with high confidence. Therefore, we combine probing with other analysis to jointly judge suspicious addresses.

D. Online Model Improvement

Data collection is pivotal in MineShark's self-improving updating framework. The inherent imbalance in the detection environment results in uneven class representation in the training data, leading to potential prediction bias. The bias becomes visible after operating the model online. Thus, the updating framework aims to improve model accuracy by focusing on its current classification errors in the deployed environment. Specifically, this involves collecting poorly recognized mining traffic and misclassified benign traffic.

Expanding mining traffic samples is crucial, especially to include mining configurations unfamiliar to the current model. Our multifaceted confirmation strategy can accumulate valuable mining samples that are not easily recognized. For example, the model may only identify a fraction of detection windows in a plain-text mining connection, but subsequent keyword scanning confirms the mining activity. In other cases, the model may well detect traffic toward specific mining ports, while additional mining services on other ports are uncovered through correlation techniques and active probing. The updating framework collects mining traffic from detection failures to strengthen future models.

Misclassified benign traffic is also valuable as it is more specific to mining activities. For example, specific pages of a website may exhibit more cryptomining similarity than others, making their traffic more effective for training purposes. However, automatically identifying these subtle differences using a crawler is challenging, while manual selection of relevant web pages trades off efficiency for quality. Fortunately, the confirmation module can facilitate the collection process as it filters out model mistakes from huge amount of gateway traffic and performs mining-specific analysis to ensure label accuracy. We add misclassified benign connections with longer duration and higher ML score than average mining connections into training set, helping the model refine its boundary.

In updating process, we accumulate new training data with the old one. Hyper-parameters are adjusted if accuracy falls short of expectations. We also experiment with various feature scaling methods to balance their influence on learning weights between size and interval features. Updates are scheduled approximately every six weeks, a period chosen to allow adequate observation of the model's performance metrics, such as the rate of newly detected mining addresses. If the number of newly discovered mining addresses in the first three weeks falls behind 20% or the false alarm volume exceeds confirmation throughput (causing drop of $Q_{Suspicious}$), we fall back to the previous model and wait for the next update stage.

VII. Evaluation

We deployed MineShark online at a 10 Gbps gateway in our campus from 3/1/2023 to 12/31/2023 for cryptomining detection (Section VII-A). Specifically, our evaluation answers the following questions:

Robustness: Can MineShark perform robust detection against varying mining configurations? (Section VII-B)

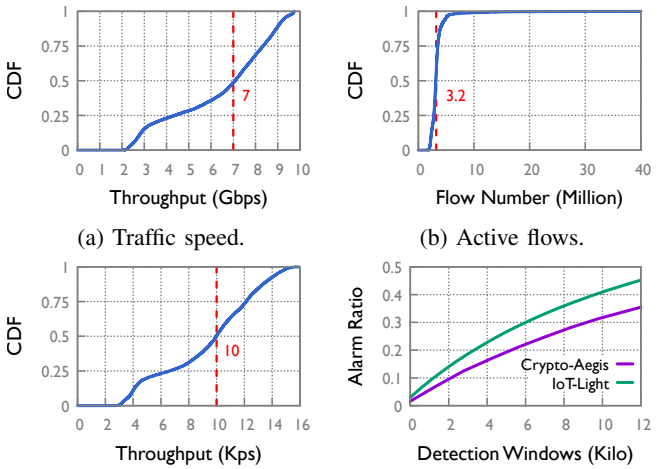
Flexibility: Can MineShark detect cryptomining traffic with ML models in real world? How does its timeliness compare to the rule-based IDSes? (Section VII-C)

Efficiency: How efficient can MineShark process traffic under varying resource constraints? (Section VII-D)

A. Deployment Setup

MineShark was developed in 15K lines of C/C++ and Python code. Its deployed server consists of two Montage Jintide(R) C6248R CPUs with 48 cores, 376 GB of memory, a dual-port 10 GbE Intel X722 NIC, an NVIDIA RTX2060 GPU, and 12 TB of SSD storage. The operating system is Ubuntu 20.04 with GNU/Linux 5.15.0-52-generic. We use DPDK [47] as traffic processing framework and TensorFlow C API (with GPU support) [50] for CNN inference. The server receives mirrored traffic from a 10 GbE SPF+ port, which is connected to the switch mirroring port via an optical fiber.

1) Online Environment Characterization: We profile the deployed network environment in Figure 8, which represents the requirements for detection algorithms regarding efficiency and robustness. First, feature extraction needs to operate at line rate, otherwise, only a small portion of gateway traffic



(a) Traffic speed. (b) Active flows.

(c) Inference workload. (d) Model alarms.

Fig. 8: Profiling of gateway traffic over one-week observation: (a) the average traffic speed is 7 Gbps, (b) the average flow concurrency is 3.2 Million, (c) the average speed of feature generation is 10Kps with a fifty-packet detection window, and (d) the input traffic can raise significant model alarms.

is detected (Figure 8a). Second, algorithms must be memory-efficient due to the huge number of concurrent flows (Figure 8b). Third, model inference must keep pace with feature extraction, otherwise, there is severe loss of Q_{Infer} (Figure 8c). Finally, unreliable detection produces significant model alarms, which overloads the confirmation module and causes loss of $Q_{Suspicious}$ (Figure 8d, explained in Section VII-B).

Results preview. MineShark processed gateway traffic with four CPU cores dedicated to the GPU-accelerated inference pipeline and one core allocated to the confirmation module. It detected 99.8% of input traffic, where the 0.2% loss was due to traffic bursts overwhelming in-memory flow table. No drop was observed in Q_{Infer}, meanwhile, all the inputs to $Q_{Suspicious}$ underwent confirmation. The alarm ratio was less than 3%.

B. Robustness of Detection Systems

We assess the robustness of MineShark against state-of-the-art detection systems using the dataset referenced in Table II. First, we train the baseline model with normal cryptomining and network traffic from [24] (as shown in the first and fourth rows of Table II). This step validates MineShark's performance against established benchmarks, with results shown in the first row of Table V. Next, to evaluate the generalization capabilities of different algorithms, we introduce obfuscated mining traffic (detailed in the second row of Table II) into the test set without including it in the training set. The second row of Table V presents the baseline model's performance on obfuscated traffic without learning obfuscation features. Then, we add obfuscated mining traffic, which includes all proxies' features, to the training and test sets to ascertain if prior knowledge of obfuscation features can enhance the baseline performance. The results are shown in the third row of Table V. In training this enhanced model, we also use gateway traffic (listed in the fourth row of Table II) to simulate the online detection environment. Finally, we test the enhanced model against perturbed cryptomining traffic (as shown in the third row of Table II) by adding it into the test set not training set to evaluate model's resilience. Rows four and five of Table V show the results, where `Dummy & Padding` and `Dummy & Padding & Splitting` traffic are separately added to the enhanced baseline's test set without the model learning the perturbation features.

For ML-based systems, namely IoT-Light, Crypto-Aegis, and MineShark, we set the training-to-test data ratio to 4:1. To ensure a balanced representation, we randomly sampled a subset of normal traffic in the test set to match the quantity of mining samples, making the demonstration of performance variation with different types of inputs clearer. In the case of CJ-Sniffer, which constructs a signature database from the patterns of interval distribution of inbound mining traffic, including all training samples in testing results in a relatively low precision of only 53.5%. To balance between precision and recall, we randomly sampled 20% of mining flows from training set. MineHunter does not undergo a training phase, therefore, all its performance comparisons are made against the initial baseline.

MineShark exhibits high precision and recall across various mining configurations, along with a low false positive rate (FPR). In comparison, CJ-Sniffer and MineHunter demonstrate lower recall, raising concerns about their effectiveness in detecting real-world mining activities. IoT-Light, on the other hand, is prone to high FPR, often incorrectly identifying benign traffic as mining traffic. This is largely attributed to their inappropriate feature extraction approach, which fits an encoder to each feature column and transforms the feature values into integers. However, this results in varying encoding schemes each time the composition of data samples in the test set changes, causing noticeable drops in testing accuracy. Crypto-Aegis, while showing commendable results, struggles to accurately detect obfuscated mining flows and does not generalize to the online environment. As shown in Figure 8d, when we added sampled gateway traffic into the test sets of enhanced Crypto-Aegis and IoT-Light models, limiting the number of detection to a maximum of five windows per flow, the alarm ratio for both models exceeds 30%. The high alarm rate impedes real-world deployment. We provide more detailed analysis of each algorithm's results in Appendix C.

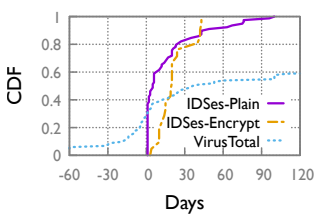
C. Cryptomining Detection in the Wild

MineShark successfully detected 105 mining pools in ten months. We highlight its flexibility in detecting mining threats ahead of commercial IDSes and open intelligence. Timeliness is crucial as mining operations can significantly disrupt normal business functions. Moreover, faster detection limits attackers' profits and prompts earlier bans in community by sharing the detected addresses. We characterize the detected mining traffic to reveal the detection challenges. In addition, we evaluate the effectiveness of address monitoring, false alarm removal, and suspicious address ranking in confirming mining activities and demonstrate model improvement through updating.

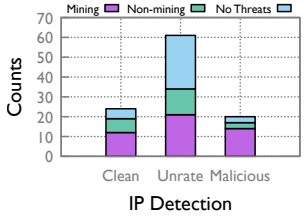
1) Detection Timeliness: MineShark was deployed alongside commercial IDSes without the authority to modify their shared denylist, which is used to block mining connections. Thus, for addresses detected by MineShark before others, we measure the delay from the initial detection to the cessation of connections to the detected addresses. We distinguish cases

Test Case	CJ-Sniffer			MineHunter			IoT-Light			Crypto-Aegis			MineShark		
	Precision	Recall	FPR	Precision	Recall	FPR	Precision	Recall	FPR	Precision	Recall	FPR	Precision	Recall	FPR
Baseline	79.2%	84.4%	2.4%	93.6%	76.5%	9.2%	98.2%	98.9%	3.2%	95.4%	87.9%	0.7%	99.3%	94.8%	0.3%
Obfuscated Mining	82.0%	14.8%	2.4%	95.1%	78.8%	9.2%	50.3%	89.7%	85.2%	99.0%	67.0%	0.7%	99.9%	97.3%	0.3%
Enhanced Baseline	81.8%	40.7%	1.5%	–	–	–	91.1%	94.8%	6.8%	97.6%	96.7%	2.5%	99.8%	98.9%	0.1%
Dummy & Padding	82.0%	38.4%	1.5%	95.5%	78.6%	3.3%	50.2%	92.4%	94.1%	99.1%	98.7%	2.5%	99.8%	98.8%	0.1%
Dummy & Padding & Splitting	81.7%	33.1%	1.5%	95.9%	75.7%	3.3%	7.7%	45.4%	87.4%	99.0%	89.1%	2.5%	99.8%	90.5%	0.1%

TABLE V: Robustness comparison of different detection systems.

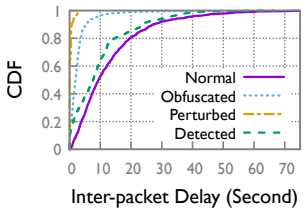


(a) Detection timeliness.

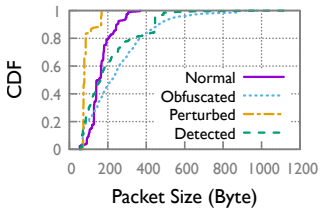


(b) VirusTotal's intelligence.

Fig. 9: MineShark can flexibly detect cryptmining before commercial systems and open intelligence.



(a) Packet interval distribution.



(b) Packet size distribution.

Fig. 10: Comparison of detected cryptomining traffic to the manually collected cryptomining training samples.

where an address was subsequently detected and blocked by other IDSes, evidenced by extended periods of only failed internal connection attempts, from those where the address appeared to be abandoned by attackers rather than being actively banned, indicated by a lack of connection failures before their disappearance. For addresses recognized first by other IDSes, MineShark could no longer detect them due to the lack of active connections for analysis.

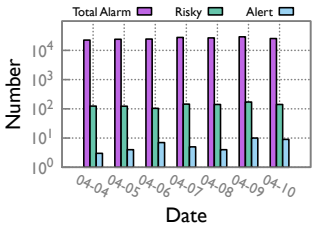
MineShark identified all 105 mining addresses faster than other IDSes. As shown in Figure 9a, the average detection delay was five days for plain-text mining and nineteen days for encrypted mining. Moreover, 13 addresses went completely undetected by other IDSes, with activity duration up to 98 days and a median being 26 days. These addresses demonstrated concerted efforts to conceal their activities. For example, they all evaded open-source intelligence flags and predominantly utilized encrypted connections for mining, obscuring the signatures and patterns that IDSes typically rely on.

We further investigate detected addresses with VirusTotal's IP and graph APIs [25], which are commonly used in security analysis. Figure 9a shows that only 28.4% addresses are associated with mining activities in front, such as communicated by cryptojacking malware. However, 33.3% addresses are reported behind our local detection. Furthermore, 38.3% addresses do not show any mining threats at all until this paper's submission. Figure 9b presents detailed statistics. The x-axis shows labels provided by the IP engine and the stacked bar shows labels from manually analyzing graph engine's reports. For example, the IP engine flags 24 addresses as clean sites, but we find 12 of them are related to mining in reality, e.g., their current or history domain names contain sensitive keywords, such as `xmr` or `pool`. Similarly, 21 unrated sites, which are not examined by the IP engine, are correlated to mining by the graph engine. However, other 40 unrated addresses are only correlated to non-mining threats or exhibit no threats. The above analysis raises concerns regarding timeliness and reliability in applying open intelligence, while MineShark's real-time detection and mining-specific analysis close the gap.

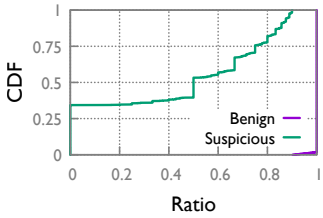
2) Cryptomining Characterization: We analyze the detected cryptomining traffic to understand the detection challenges better. As shown in Figure 10, we find that the inter-packet delay of detected mining traffic resembles that of normal mining. However, the packet sizes align more with obfuscated mining. This difference highlights the need for robust detection algorithms and the importance of continually enriching training datasets to refine models post-deployment. Regarding encrypted mining traffic, which constituted 17.6% of the recorded mining connections, we observe that 69.2% of associated pools adopted TLS 1.3 protocol, which encrypted the server's certificate. Moreover, even among the pools that provided TLS 1.2 services, half did not feature discernible keywords in their Subject Alternative Name fields. These factors complicate the identification of mining activities by analyzing handshake information, a common approach in IDSes.

3) Address Monitoring: We validate the effectiveness of mining address monitoring. We find that 30.8% of the addresses opened more than one service port, with a single address opening up to 13 ports. Through domain correlation, we discover that 41.0% of the identified mining addresses cluster within the top five groups. The size of the defensive denylist, which includes mining addresses identified through domain analysis but not observed in our network, was approximately 69% larger than the list of observed addresses. Note that the total of 105 mining addresses only includes those initially reported by the inference pipeline, excluding addresses identified through correlation analysis. For a comprehensive understanding, a detailed case study that demonstrates the detection workflow is presented in Appendix D.

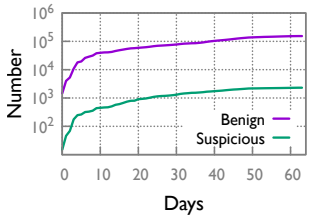
4) False Alarm Removal: We find that even a performant model can produce large amount of false alarms when deployed at scale. Fortunately, the false alarm removal algorithm can reduce the number of alarm flows by two orders of magnitude, as shown in Figure 11a. The *Alarm* bar shows the number of suspicious flows flagged by the model each day. After filtering out false alarms, only around 100 suspicious addresses remain per day, as shown by the *Risky* bar. These addresses are subject to further confirmation actions. Addresses that cannot be confirmed as mining-related await further updates or manual investigation. Operators can manage the investigation workload by prioritizing the most suspicious addresses or applying specific filters, such as the *Alert* bar, which identifies addresses with a positive ML score. Statistics collected at other time periods showed similar results.



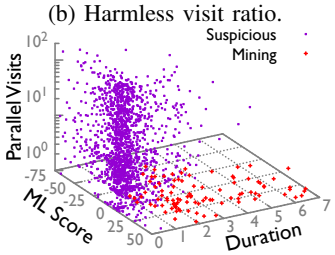
(a) False alarm removal.



(b) Harmless visit ratio.



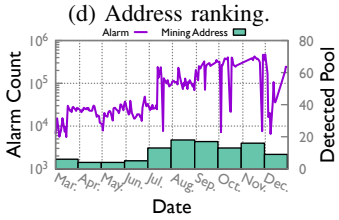
(c) Address label accumulation.



(d) Address ranking.

Data	Precision	Recall	FPR
Baseline	99.9%	82.3%	0.1%
Before Jun.	99.9%	85.0%	0.1%
Jul.	99.8%	90.4%	0.4%
Aug.	99.9%	88.2%	0.1%
Sep.	99.8%	92.5%	0.4%
After Oct.	99.7%	93.3%	0.7%

(e) Model improvement.



(f) Online accuracy.

Fig. 11: Effectiveness of confirmation and updating.

The employed harmless visit ratio threshold ($\mathcal{T}$) is also appropriate. Figure 11b shows that with $\mathcal{T}$ set as 0.9, the τ of 98% of labeled risky addresses was below 0.9. In contrast, over 98% of labeled benign addresses had τ equal to 1. Therefore, the current threshold is unlikely to cause mislabeling problems.

We find the accumulated address labels effectively filter suspicious-but-benign network connections. As shown in Figure 11c, both benign and suspicious labels grew rapidly at initial, but they stabilized as more connections are associated with each address. This trend aligns with our objective of identifying benign addresses by considering visits over time and let the majority judge the address behavior. Interestingly, we observe that turning points occurred around five days after the algorithm starts. This indicates the limited scale of popular services, which can trigger ML model alarms frequently due to their specific cryptomining-like traffic features.

5) Suspicious Address Ranking: We demonstrate the effectiveness of employed address features. Figure 11d illustrates the distribution of feature values of the top 1,864 observed risky addresses. We find that mining addresses (marked by red crosses) have apparent long duration, high ML score, and low parallel visits. However, any single feature is not enough to distinguish them without combining all three features, indicating complex conditions in real world. Notably, the ranking algorithm successfully placed a total number of 43 mining addresses, which conducted encrypted mining, in the top 4% position of the monitoring list on their discovery day, leading to effective confirmation through active probing.

6) Model Improvement: We show accuracy improvement by training with newly discovered mining samples. We first conduct dataset experiments by constructing a new dataset comprising all detected mining samples. The dataset is partitioned into five parts according to the detected time. We sequentially add each part to the training set and test the model on the remaining data, simulating the update process. Figure 11e shows the result. The baseline accuracy is measured using the enhanced model, as detailed in Table V. Starting from the second row, we randomly replaced original samples in the enhanced model's training set with new data, maintaining the unknown data ratio of 8.9% in all tests. The original training set contained 450,990 samples. For each time period, we randomly selected 50,000 samples and divided them into training and test sets using a 4:1 ratio. Five test sets were integrated into the original test set of the enhanced model, which remained unchanged throughout the experiment. We find that the recall metric continuously improves, suggesting the usefulness of learning from unfamiliar features and applying this knowledge to similar targets in the future.

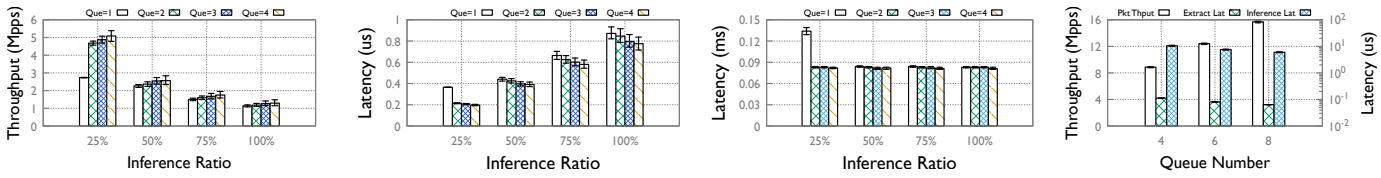
We further report the real-world impact of updates in Figure 11f, illustrating the improvement in the detection ratio (as defined in Equation 1) over time. The line displays daily alarm counts and the bar shows monthly detected mining addresses. The training data comprises both accumulated mining and false alarm traffic. The most interesting update happened on 7/13/2023. Despite an increase in the alarm ratio from 0.2% to 2.2%, the number of discovered mining addresses tripled. The outcomes of other updates are more consistent. In reality, the FPR of that model was as low as 0.1% in offline testing, which reminds us that dataset evaluation is not enough to assess a model's real-world performance. Moreover, optimizing detection ratio requires simultaneous improvements in model accuracy and confirmation speed.

D. System Efficiency

We evaluate the efficiency of feature extraction and model inference, which are key to process traffic at line rate. We test scalability by varying assigned CPU cores and pinpoint the bottleneck in deep learning model inference. We demonstrate GPU acceleration can eliminate the bottleneck. MineShark can readily integrate simpler ML models, such as a backup SVM model in case of GPU failure. We evaluate its performance in Appendix E, offering insights into the trade-offs between performance and cost when employing different models.

We create high-speed traffic more than 10 Gbps for stress testing. We merge traffic dataset (Table II) into one large packet file, then load the packet file into memory and recirculate packets to the highest speed. Our evaluation metrics include: (1) packet throughput, which is the number of processed packets per second, (2) extraction latency, which is the average processing time of one packet, and (3) inference latency, which is the average inference time of one feature matrix.

We scale the input traffic by adding more input queues. Each queue binds with one extraction and one inference core. Since not every flow generates feature matrices (e.g., due to too few packets), we simulate the proportion of inferred features by controlling the *inference ratio*. For example, a 25% ratio indicates one-fourth of extracted matrices are sent for inference. We back propagate the pressure of inference module to extraction module by forcing the extraction core to wait when Q_{Infer} reaches its capacity. This ensures no drop of input packets and features of Q_{Infer}.



(a) Packet throughput. (b) Extract latency. (c) Inference latency. (d) GPU acceleration.

Fig. 12: Efficiency characteristics of MineShark's inference pipeline on CPU (a, b, and c) and GPU (d).

1) CPU performance: We run each case for 15 minutes to ensure system stability. As shown in Figure 12a, the packet throughput increases as the inference ratio decreases, with the peak throughput reaching 5.03 Mpps per queue. However, in most cases, adding input queues (along with more CPUs) does not lead to throughput improvement. This is attributed to the unified TensorFlow execution environment on CPU reaching the inference bottleneck. As further proved in Figure 12c, no decrement in latency after the first bar indicates the inference throughput reaches its peak. Therefore, 0.08 ms per matrix is the optimal inference speed on CPU. Figure 12b shows per-packet processing latency, which is inverse to the throughput. The shortest latency for processing one packet is 0.19 us.

2) GPU acceleration: Aligned with CPU experimenting setup, we set the inference ratio to 100% to enforce maximum stress. As shown in Figure 12d, with 4, 6, and 8 queues, the throughput reaches 8.78 Mpps, 12.29 Mpps, and 15.54 Mpps (equivalent to 92.01 Gbps), respectively. The assignment of 8 queues decreases inference latency to 6.21 us per feature matrix, resulting in a 12.9 times speedup compared to CPU. In summary, GPU acceleration removes the inference bottleneck and facilitates the attainment of line-rate performance.

VIII. Discussion

Defences against black-box adversarial attacks. ML-based network IDSes commonly face adversarial threats, with black-box attacks representing more realistic scenarios [51]. Attackers might use surrogate models to create adversarial features transferable to the target system [52]. However, deploying such attacks on *live* network traffic is challenging [53]. Developing the surrogate models requires accurate real-time predictions of the target model's outcomes. The execution phase involves converting these adversarial features into replayable network traffic and adjusting for network variability. Therefore, our evaluation focuses on simpler attacks that manipulate traffic features through existing obfuscation proxies and perturbation techniques, rather than directly targeting model characteristics. This realistic focus demonstrates effectiveness in degrading the accuracy of current detection systems by altering standard mining traffic patterns. MineShark outperforms existing systems in countering such attacks by leveraging the unconcealed temporal features of mining message sequences. Moreover, by analyzing every packet, MineShark maximizes the potential to detect manipulated mining flows, because even a single accurately identified detection window can initiate a comprehensive analysis within the detection pipeline.

Moreover, MineShark can deploy multiple detection models in parallel to counter adversarial threats targeting specific model features. The distinct feature extraction processes (e.g., employing time-based or sequence-based detection windows) across models and the varied decision boundaries of different models complicate the creation of universal adversarial flows, ensuring robust detection. The current system design requires no changes. Specifically, feature extraction cores can process features for all active models, with adjustable core number to ensure efficiency. Extracted features are shared with inference cores through zero-copy buffers, optimizing memory usage. Inference cores can execute different models and leverage GPUs to accelerate. Mining confirmation and model updates can be performed on a per-model basis.

Acceptable false alarm ratio and consequent workload. MineShark can tolerate a higher false alarm ratio in exchange for improved recall. False alarms are effectively filtered out based on the harmless visit ratio of their destination addresses, with the acceptable alarm ratio depending on the system's confirmation speed. As shown in Figure 11f's alarm count, one CPU core can process over 500,000 false alarms per day. Therefore, by increasing the number of worker threads processing $Q_{Suspicious}$, MineShark can handle a higher alarm ratio. After removing false alarms, MineShark automatically confirms suspicious addresses, significantly reducing the human investigation workload. Although there is some delay before confirmation is completed, this process still provides faster results compared to manual analysis.

IX. Conclusion

We developed MineShark, a learning-based system for detecting cryptomining traffic at scale. It utilizes robust features that are resistant to obfuscated and perturbed cryptomining traffic. It incorporates a line-rate inference pipeline to drain high-speed traffic, an automated confirmation module to provide reliable detection results, and a self-improving updating framework to improve model accuracy post-deployment. We deployed MineShark at our 10 Gbps campus network over a period of ten months, where it successfully discovered 105 mining pool addresses in ahead of other commercial systems. This showcases the robustness, flexibility, and efficiency of MineShark in real-world scenarios.

Acknowledgment

The work is supported in part by National Key R&D Program of China under Grant No. 2023YFB2904000. We would like to sincerely thank NDSS 2025 Chairs and Reviewers for your review efforts and helpful advice.

References

[1] SONICWALL, "2024 sonicwall cyber threat report," 2024, https://www.sonicwall.com/medialibrary/en/white-paper/2024-cyber-threat-report.pdf.

[2] M. Lennon, "Cryptocurrency mining malware hits monitoring systems at european water utility," https://www.securityweek.com/cryptocurrency-mining-malware-hits-monitoring-systems-european-water-utility/, 2018.

[3] L. Ferré-Sadurní and G. Ashford, "New york enacts 2-year ban on some crypto-mining operations," 2022, https://www.nytimes.com/2022/11/22/nyregion/crypto-mining-ban-hochul.html.

[4] J. Crawley, "Bitcoin mining appears to have survived ban in china," 2022, https://www.coindesk.com/business/2022/05/17/bitcoin-mining-appears-to-have-survived-ban-in-china/.

[5] A. Ashraf, "Microsoft bans crypto mining on its online services without permission," 2022, https://www.coindesk.com/tech/2022/12/15/microsoft-bans-crypto-mining-on-its-online-services-without-permission/.

[6] Z. Li, W. Liu, H. Chen, X. Wang, X. Liao, L. Xing, M. Zha, H. Jin, and D. Zou, "Robbery on devops: Understanding and mitigating illicit cryptomining on continuous integration service platforms," in *S&P*, 2022, pp. 363–378.

[7] S. Pastrana and G. Suarez-Tangil, "A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth," in *IMC*, 2019, pp. 73–86.

[8] R. K. Konoth, E. Vineti, V. Moonsamy, M. Lindorfer, C. Kruegel, H. Bos, and G. Vigna, "Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense," in *CCS*, 2018, pp. 1714–1730.

[9] G. Hong, Z. Yang, S. Yang, L. Zhang, Y. Nan, Z. Zhang, M. Yang, Y. Zhang, Z. Qian, and H. Duan, "How you get shot in the back: A systematical study about cryptojacking in the real world," in *CCS*, 2018, pp. 1701–1713.

[10] H. L. Bijmans, T. M. Booij, and C. Doerr, "Inadvertently making cyber criminals rich: A comprehensive study of cryptojacking campaigns at internet scale," in *USENIX Security Symposium*, 2019, pp. 1627–1644.

[11] R. Recabarren and B. Carbunar, "Hardening stratum, the bitcoin pool mining protocol," *arXiv preprint arXiv:1703.06545*, 2017.

[12] J. Rüth, T. Zimmermann, K. Wolsing, and O. Hohlfeld, "Digging into browser-based crypto mining," in *IMC*, 2018, pp. 70–76.

[13] H. L. Bijmans, T. M. Booij, and C. Doerr, "Just the tip of the iceberg: Internet-scale exploitation of routers for cryptojacking," in *CCS*, 2019, pp. 449–464.

[14] 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," in *ACSAC*, 2021, pp. 1051–1063.

[15] E. Tekiner, A. Acar, A. S. Uluagac, E. Kirda, and A. A. Selcuk, "Sok: cryptojacking malware," in *EuroS&P*, 2021, pp. 120–139.

[16] M. Caprolu, S. Raponi, G. Oligeri, and R. Di Pietro, "Cryptomining makes noise: Detecting cryptojacking via machine learning," *Computer Communications*, vol. 171, pp. 126–139, 2021.

[17] M. Kührer, C. Rossow, and T. Holz, "Paint it black: Evaluating the effectiveness of malware blacklists," in *RAID*, 2014, pp. 1–21.

[18] K. Neupane, R. Haddad, and L. Chen, "Next generation firewall for network security: A survey," in *SoutheastCon 2018*. IEEE, 2018, pp. 1–6.

[19] L. Bilge and T. Dumitraş, "Before we knew it: an empirical study of zero-day attacks in the real world," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 833–844.

[20] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, "Deep packet inspection as a service," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 271–282.

[21] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *CCS*, 2021, pp. 3431–3446.

[22] C. Fu, Q. Li, and K. Xu, "Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis," *arXiv preprint arXiv:2301.13686*, 2023.

[23] Y. Feng, J. Li, and D. Sisodia, "Cj-sniffer: Measurement and content-agnostic detection of cryptojacking traffic," in *RAID*, 2022, pp. 482–494.

[24] E. Tekiner, A. Acar, and A. S. Uluagac, "A lightweight iot cryptojacking detection mechanism in heterogeneous smart home networks," in *NDSS*, 2022.

[25] "Virustotal api v3 overview," 2022, https://developers.virustotal.com/reference/ip-object.

[26] "Stratum v1 documents," 2022, https://braiins.com/stratum-v1/docs.

[27] Z. Zhang, G. Hong, X. Li, Z. Fu, J. Zhang, M. Liu, C. Wang, J. Chen, B. Liu, H. Duan *et al.*, "Under the dark: A systematical study of stealthy mining pools (ab) use in the wild," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 326–340.

[28] C. Systems, "Designing and deploying intrusion detection systems," 2003, https://www.cisco.com/c/dam/global/fr_ca/training-events/pdfs/Designing_and_Deploying_ids_technologies.pdf.

[29] G. Wan, F. Gong, T. Barbette, and Z. Durumeric, "Retina: analyzing 100gbe traffic on commodity hardware," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 530–544.

[30] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3366–3383.

[31] R. Beltiukov, W. Guo, A. Gupta, and W. Willinger, "In search of netunicorn: A data-collection platform to develop generalizable ml models for network security problems," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2217–2231.

[32] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville, "Ai/ml for network security: The emperor has no clothes," in *CCS*, 2022, pp. 1537–1551.

[33] E. Papadogiannaki and S. Ioannidis, "A survey on encrypted network traffic analysis applications, techniques, and countermeasures," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–35, 2021.

[34] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *CCS*, 2018, pp. 1928–1943.

[35] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," *arXiv preprint arXiv:1708.06376*, 2017.

[36] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1131–1148.

[37] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[38] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *32nd USENIX Security Symposium (USENIX Security 23). Anaheim, CA: USENIX Association*, 2023.

[39] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications." in *NDSS*, 2021.

[40] Y. Dong, Q. Li, K. Wu, R. Li, D. Zhao, G. Tyson, J. Peng, Y. Jiang, S. Xia, and M. Xu, "{HorusEye}: A realtime {IoT} malicious traffic detection framework using programmable switches," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 571–588.

[41] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco, "Re-architecting traffic analysis with neural network interface cards," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 513–533.

[42] J. Gao, E. Zhai, H. H. Liu, R. Miao, Y. Zhou, B. Tian, C. Sun, D. Cai, M. Zhang, and M. Yu, "Lyra: A cross-platform language and compiler for data plane programming on heterogeneous asics," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 435–450.

[43] (2022) A fast tunnel proxy that helps you bypass firewalls. https://github.com/shadowsocksrr/shadowsocksr.

[44] (2022) Project v github repo. https://github.com/v2fly/v2ray-core.

[45] (2022) A secure tunneling daemon. https://openvpn.net/.

[46] K. Lee, S. Oh, and H. Kim, "Poster: Adversarial perturbation attacks on the state-of-the-art cryptojacking detection system in iot networks," in *CCS*, 2022, pp. 3387–3389.

[47] T. L. F. Projects. (2022) Dpdk programmer's guide. https://doc.dpdk.org/guides/prog_guide/index.html.

[48] M. Nasr, A. Bahramali, and A. Houmansadr, "Deepcorr: Strong flow correlation attacks on tor using deep learning," in *CCS*, 2018, pp. 1962–1976.

[49] G. Ho, A. Sharma, M. Javed, V. Paxson, and D. Wagner, "Detecting credential spearphishing attacks in enterprise settings," *Proc. of 26th USENIX Security*, 2017.

[50] TensorFlow. (2022) Install tensorflow for c. https://www.tensorflow.org/install/lang_c.

[51] K. He, D. D. Kim, and M. R. Asghar, "Adversarial machine learning for network intrusion detection systems: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 538–566, 2023.

[52] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *arXiv preprint arXiv:1605.07277*, 2016.

[53] M. Nasr, A. Bahramali, and A. Houmansadr, "Defeating dnn-based traffic analysis systems in real-time with blind adversarial perturbations." in *USENIX Security Symposium*, 2021, pp. 2705–2722.

APPENDIX A
MODEL DETAILS

Layer	Details
Convolution Layer 1	Kernel num: 20 Kernel size: (2, 20) Stride: (2, 1) Activation: Relu
Max Pool 1	Window size: (1, 5) Stride: (1, 1)
Convolution Layer 2	Kernel num: 100 Kernel size: (2, 20) Stride: (2, 1) Activation: Relu
Max Pool 2	Window size: (1, 5) Stride: (1, 1)
Fully Connected 1	Size: 1200, Activation: Relu, Dropout: 0.8
Fully Connected 2	Size: 500, Activation: Relu, Dropout: 0.8
Fully Connected 3	Size: 100, Activation: Relu

TABLE VI: Hyperparameters of model.

We present the best hyperparameters of our model that works online in Table VI. We configure the kernel size of the first convolution layer as (2, 20). A stride of (2, 1) is used to capture the correlation between bidirectional timing or sizes. In the second convolution layer, we use the same kernel and stride settings to capture the combination of timing and size features from the input of the previous layer. Our second layer differs from the work in [48] as we do not correlate multiple flows in our scenario.

We utilize the Adam optimizer to minimize the loss function and explore different hyperparameters to optimize performance. Our evaluation metrics include true positive rate, false positive rate, accuracy, precision, recall, and F1-score. We set the learning rate to 0.0001. Regarding the kernel size of the convolutional layers, we experiment with values of 5, 10, 15, 20, and find that setting both layers' kernel sizes to 20 yields the best results. For max pooling, we test sizes of 2, 5, 8, 10, and determine that a size of 5 produces the optimal result. In terms of the model's decision threshold, we evaluate values of 0, 2, 4, 6, 8, 10, and examine various packet window sizes of 50, 100, 200, 250, 400. We achieve the best performance with a decision threshold of 2 and a packet window size of 50. The most effective feature scaling method that works for us is to multiply the inter-packet delay feature values by 10 and divide the size feature values by 10. This method outperforms other scaling techniques, such as Min-max scaling or Z-score normalization.

APPENDIX B
OBFUSCATED CRYPTOMINING CHARACTERISTICS

We investigate the obfuscation techniques used by different proxies, drawing implementation details from online documentation in their respective Github repositories. All the proxies listed in Table III are open-source software.

Proxied traffic characteristics are influenced by the type of proxy software, its configuration, and the nature of the original application traffic. The proxy's presence is transparent to the application, meaning that even when using identical proxy software and settings, the resulting traffic varies between applications. For instance, if an application's flow predominantly consists of maximum transmission unit (MTU) sized packets and the obfuscation policy involves random padding of packets smaller than MTU, then the output traffic would not differ significantly. Conversely, for applications with packets smaller than N bytes, the output traffic will range from N bytes up to MTU, altering the original traffic and differing from that of other applications. This principle also applies to obfuscated mining traffic, where the output characteristics deviate from both standard mining traffic and other application traffic.

In summary, Non-VPN proxies tend to mainly affect packet size, whereas VPN proxies more significantly impact packet number and interval. We compare the differences in packet number, size, and interval as follows.

Packet Number. VPN protocols introduce additional packets such as keep-alive and control channel packets, resulting in noticeable changes in the packet count of the cryptomining flow. In contrast, non-VPN proxies employ obfuscation techniques that subtly alter the packet count. For example, the `http_simple` obfuscation plugin in ShadowsocksR (SSR) inserts HTTP GET requests and responses to simulate the HTTP protocol at fixed interval. Similarly, the `tls1.2_ticket_auth` plugin in SSR mimics the original flow as a complete TLS connection, which adds extra handshake packets. These obfuscation techniques can be considered as specific dummy packet insertion attacks.

Packet Size. Non-VPN proxies commonly employ padding techniques to defeat length-based and entropy-based traffic analysis. For example, adversaries can utilize SSR's `protocol_plugin` to add random padding in payload, increasing its size to at most 1440B. The VMess protocol operates similarly, but only adds padding of less than 64B. In contrast, VPN proxies normally add fixed-length headers instead of padding to payload. As a result, the exchange of cryptomining messages can still be identified by observing unique sized packets within the VPN tunnel, because most messages can still fit into a single packet.

Packet Interval. VPN encryption tunnels significantly modify the packet intervals of cryptomining traffic by introducing additional control channel packets. In contrast, Non-VPN proxies typically utilize control channel packets only during connection establishment. After this initial phase, no further control packets are added to the transmission.

Unchanged Features. Although non-VPN and VPN proxies can obfuscate specific traffic features, they cannot entirely mask the inherent temporal patterns in cryptomining flows. For instance, a result submission packet is typically followed by a confirmation packet, and inbound traffic predominantly consists of job assignment packets occurring at a consistent frequency. Machine learning models can capitalize on the regularity of these unaltered features over sequences, enabling effective detection of even obfuscated cryptomining traffic.

APPENDIX C
ROBUSTNESS ANALYSIS

We utilize the open-source implementations of MineHunter and IoT-Light. For CJ-Sniffer and Crypto-Aegis, we develop our own implementations based on the details described in their original papers. For all the algorithms, we apply recommended configurations in training and test. We analyze the results shown in Table V.

CJ-Sniffer struggles with traffic obfuscation and perturbations for two primary reasons. First, its initial rapid filtration phase, which filters out irrelevant traffic flows by comparing packet sizes to standard cryptomining flows, fails to detect mining flows subjected to random padding. Second, the significant alterations in inter-packet delay caused by both obfuscation proxies and perturbations render its signature-based solution less effective.

MineHunter utilizes a coefficient α to calculate the similarity score for each detection window. However, we find this calculation less reliable in reflecting the *intention* of arriving a packet concurrent with block generation in the cryptocurrency network, as described by its authors, particularly for mining traffic with low packet frequency. In such scenarios, local distances does not always significantly shorter than the average packet delay. This leads to an underestimation of α, resulting in the misclassification of mining traffic as low-frequency noise. Interestingly, the introduction of obfuscation and perturbation does not significantly impact MineHunter's performance. This is because MineHunter does not depend on unique packet sizes within a mining flow, rendering padding attacks ineffective. Additionally, these attacks primarily reduce packet intervals, which in turn decrease local distances, inadvertently enhancing the perceived *intention* degree.

IoT-Light applies `LabelEncode` in feature extraction, which affects its ability to generalize to unseen samples. Specifically, it fits a `LabelEncoder` to each feature column and transforms the feature values into integers. However, this method results in varying encoding schemes each time the composition of data samples in the test set changes. For example, introducing obfuscated mining samples into the test set leads to significant alterations in the encoded values of existing samples. Consequently, it causes a noticeable drop in performance, particularly in the precision metric.

Crypto-Aegis utilizes a fixed-size window to calculate the mean and standard deviation of both packet size and interval, and it employs a random forest model for learning from these extracted features. The window slides packet-by-packet in training and testing. It achieves better accuracy compared to other algorithms because it also incorporates learning from sequences. However, it cannot generalize well enough to unseen traffic samples, such as the obfuscated mining samples not included in the training data, primarily due to the limited generalization ability of the applied model. When these new feature types are incorporated into the learning process, the accuracy improves accordingly.

MineShark achieves high precision and recall across various scenarios, attributed to its ability to learn essential mining characteristics over sequences. It employs a detection window comprising fifty packets, sliding forward whenever either direction accumulates fifty packets. This sequence length proves sufficient for recognizing the regularity in the exchange of mining-specific messages. MineShark stands out as the most robust solution against the packet splitting, which is particularly challenging for evasion detection.

APPENDIX D
CASE STUDY OF CRYPTOJACKING DETECTION

We demonstrate the detection workflow with a real example from our network, as shown in Figure 13. First, the inference pipeline identified plain-text mining connections linked to the external address 185.*.*.187, which was added to the monitoring list. Through domain correlation, the confirmation module determined its associated current domain, djkiss.**.download, to be a subdomain of **.download. It also identified two associated subdomains, namely api.**.download and c0me.**.download. By sending probing packets to all resolved hosts on the same port as used by 185.*.*.187, the module confirmed additional mining addresses at 104.*.*.66 and 198.*.*.91, both hosting plain-text mining services. Consequently, these two addresses as well as the four domains were incorporated into the defensive denylist.

Two days after the initial detection, the inference pipeline detected suspicious encrypted mining connections toward address 209.*.*.115, which was subsequently added to the confirmation list. However, initial probing attempts received no response from this address. Given its high rank among suspicious addresses, further correlation analysis were conducted, which revealed that 209.*.*.115 resolved to the already blacklisted domain **.download. Probing packets were then sent to service ports previously identified on 85.*.*.187, leading to responses that confirmed mining service activity associated with 209.*.*.115. Therefore, it was reported to the operator along with detection logs. Further investigation uncovered that the initial probing failure was due to a TLS configuration error. Once corrected, the discovery of new addresses linked to **.download and providing encrypted mining services became more efficient. The module also included the encrypted service port into probing list. By continually updating the monitoring list, more mining addresses related to this domain were added to the defensive denylist, enhancing the security of our network. Meanwhile, features of mining traffic toward confirmed mining addresses are recorded, which serves as training data to enhance the accuracy of the detection model.

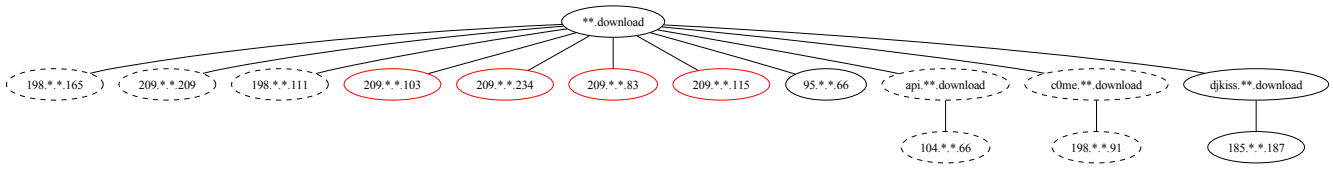
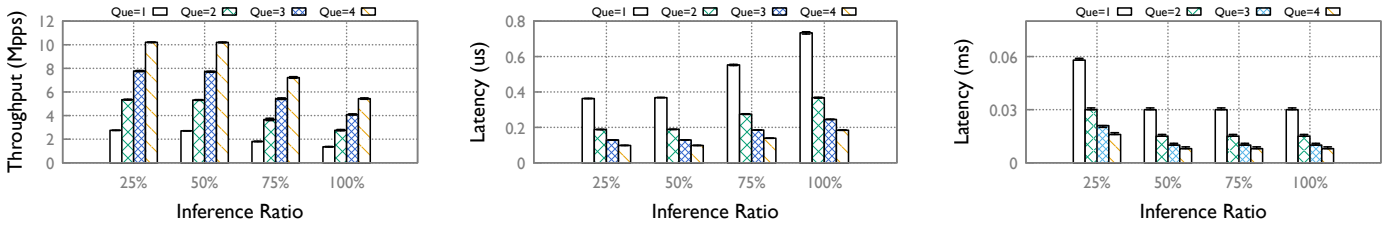


Fig. 13: Domain correlation of monitored mining addresses. Dashed nodes represent addresses on the defensive denylist. Solid nodes indicate confirmed mining addresses, among which red nodes are associated with encrypted mining traffic.



(a) Packet throughput. (b) Extract latency. (c) Inference latency.

Fig. 14: Efficiency characteristics of a simple machine learning model in MineShark's inference pipeline.

Test Case	Precision	Recall	FPR
Baseline	99.0%	99.1%	1.2%
Obfuscated Mining	94.5%	76.2%	1.2%
Enhanced Baseline	93.8%	97.5%	4.8%
Dummy & Padding	16.3%	28.3%	4.8%
Dummy & Padding & Split	13.8%	18.1%	4.8%

TABLE VII: Robustness of SVM.

APPENDIX E
PERFORMANCE TRADE-OFFS

To assess the performance trade-offs of adopting a simpler machine learning model, we replicate the robustness experiment from Section VII-B for our SVM backup model. Table VII indicates that the SVM model struggles with generalizing to unseen samples, which is due to the dependence on statistical features calculated from a group of packets. While Crypto-Aegis offers greater accuracy, it is not selected as a backup because its memory requirements exceed the capacity of our deployment server, given the high volume of concurrent flows at the gateway. In contrast, the SVM's time-based sliding window easily allows for real-time detection.

Despite its limitations in robustness, the simpler machine learning model demonstrates scalable computation on CPU. Using the experimental setup from Section VII-D, we assessed its computational overhead. The results are presented in Figure 14. We find that the SVM computation scales linearly and is fully parallelizable across multi-core CPUs. As illustrated in Figure 14a, the packet throughput increment is in direct proportion to the number of input queues, but independent of the detection ratio. The packet throughput peaks at 16.45 Mpps (97.40 Gbps) with seven queues at a 25% detection ratio (not included in the figure). However, the overall pipeline's throughput is ultimately constrained by model inference. As depicted in Figure 14c, beyond a 50% detection ratio, inference latency becomes irrespective of queue count, indicating that each core is operating at maximum capacity. To further enhance throughput, we need to increase the number of allocated cores, which effectively distributes the detection workload.

In practice, deploying the SVM model necessitates at least four dedicated CPU cores, each handling a distinct pipeline module: feature extraction, feature inference, IO management, and time management. Furthermore, we note that the current model on GPU maintains an average utilization rate of 30% and occupies around 5 GB of memory. Given these resource demands, we consider the costs justifiable in light of the superior performance afforded by deep learning models.

APPENDIX F
ARTIFACT APPENDIX

In this section, we provide information on obtaining the source code and dataset for MineShark. This artifact is designed for a basic setup compatible with a standard server. Researchers using this artifact should be able to produce results related to comparing the robustness of various cryptomining traffic detection methods (Section VII-B) and assessing the efficiency characteristics of MineShark's inference pipeline (Section VII-D). However, the observation obtained from a campus gateway (Section VII-C) cannot be directly reproduced. To facilitate replication in one's own network, this artifact includes the necessary codebase and demonstrates an end-to-end workflow. Additionally, we discuss customization options to extend the artifact's application to broader scenarios.

A. Description & Requirements

1) How to access: The source code and dataset for MineShark (namely `MineShark_AE.tar.gz`), including the files needed to run the evaluation experiments, are available at https://doi.org/10.5281/zenodo.13624057.

2) Hardware dependencies: No specific hardware is required. However, we recommend a system with 10 CPU cores and 16 GB of memory available for the evaluation.

3) Software dependencies: The following software is required to run this artifact:

- Linux OS (tested on x86_64 Ubuntu 20.04 LTS)
- Docker (tested on version 20.10.21)
- DPDK version 20.11.6
- Tensorflow C version 2.8.0
- CppFlow Version 2.0.0
- Python $\geq$ 3.8
- Jupyter Notebook
- MongoDB
- Redis

4) Benchmarks: This artifact includes obfuscated and perturbed cryptomining traffic datasets contributed by this work. For other utilized open-source cryptomining and normal traffic datasets, as well as code implementations, we provide relevant pointers. Specifically, this artifact compares the robustness of MineShark with four state-of-the-art cryptomining traffic detection systems: CJ-Sniffer [23], MineHunter [14], IoT-Light [24], and Crypto-Aegis [16]. We use the source code provided by the original authors for MineHunter and IoT-Light, and we implement CJ-Sniffer and Crypto-Aegis based on the descriptions in their respective papers.

B. Artifact Installation & Configuration

First, follow the official guides (`https://docs.docker.com/engine/install/ubuntu/`) to install Docker on the Linux testbed. Then, pull the MineShark image, where software environments are configured for running the experiments:

```
$ sudo docker pull haers/mineshark:AE
```

Next, download the artifact package using the Zenodo link. Unzip the package into a directory named `AE`. Follow instructions in the `AE/README.md` to configure the host system and start the container. This requires mounting the `AE` path and huge pages memory appropriately. Inside the container, follow the instructions in the `README.md` file under each `experiment*` directory to conduct the evaluation.

C. Major Claims

- (C1): MineShark is more robust than state-of-the-art systems in detecting mining traffic against varying configurations. This is proven by the experiment (E1), whose results are reported in Section VII-B (Table V).
- (C2): MineShark performs end-to-end cryptomining detection with a line-rate inference pipeline and an automatic confirmation module. This is proven by the experiment (E2) whose workflow is described in Section VI and the effectiveness in real-world deployment is reported in Section VII-C.
- (C3): MineShark can process traffic with varying input speeds when allocating different CPU resources. Moreover, the inference speed is the processing bottleneck. This is proven by the experiment (E3), whose results are reported in Section VII-D (Figure 12a~12c).

D. Evaluation

1) Experiment (E1): [Model robustness comparison] [15 human-minutes + 3 compute-hour]: This experiment benchmarks MineShark against state-of-the-art cryptomining traffic detection systems, corresponding to the directory `AE/experiment1`. Each compared system has an associated subdirectory containing the required code and data. Specifically, there are two Jupyter notebook scripts (`*_baseline.ipynb` and `*_enhance.ipynb`) that execute the detection algorithm against five test cases: Baseline, Obfuscated Mining, Enhanced Baseline, Dummy & Padding, and Dummy & Padding & Splitting, as detailed in Section VII-B.

[Preparation] No specific preparation is needed, as extracted features from raw traffic input are saved into data files that can be directly loaded for training and testing. We take this approach because, on one hand, part of the normal traffic data collected in the campus network is prohibited from being shared due to a non-disclosure agreement we signed. On the other hand, feature extraction can be extremely time-consuming. For example, extracting features for `IoT-Light` takes more than twenty-four hours. Instead we provide extraction scripts (`*_extract.py`) in each directory for reference.

[Execution] Navigate to the subdirectories of `cjsniffer`, `minehunter`, `crypto-aegis`, and `mineshark`. Open the Jupyter notebooks and execute all the cells. For `iot`, execute the python scripts in terminal:

```
$ python iot_baseline.py
$ python iot_enhance.py
```

Since training the `iot` models takes more than 8 hours, we have configured the scripts to load our pre-trained models by default. Comment out the loading code to evaluate from scratch. Additionally, MineShark's results may vary slightly across different machines due to the inherent randomness and non-determinism involved in training deep learning models.

[Results] Compare the evaluated metrics printed on the console with expected results in `experiment1/README.md`.

2) Experiment (E2): [End-to-end detection workflow][10 human-minutes + 0.2 compute-hour]: This experiment demonstrates MineShark's detection workflow, corresponding to the directory `AE/experiment2`. The workflow is briefly described as follows: The detection pipeline sequentially receives input packets, which is simulated by looping the replay of recorded traffic, as if they are coming from a real NIC queue. It extracts the timestamp and size of each packet to form feature matrices on a per-flow basis. The extracted features are then inferred by a trained model. Information on suspicious flows is pushed to the confirmation module through WebSocket interfaces. Meanwhile, suspicious flow data is saved on disk.

The confirmation module subscribes to suspicious flow information and uses MongoDB to establish correlation graphs for suspicious addresses. It extracts ranking features of suspicious flows and scans for keywords in packet payloads to confirm plain-text mining traffic. After that, it preserves records of timestamps and sizes of raw traffic information while deleting original packet files. The confirmation module further probes flow destinations to check for the presence of pool mining services. In addition, features of suspicious addresses are maintained in Redis queues, where the ranking algorithm can access the latest information at runtime.

[Preparation] Open three container terminals by executing this command in different terminals on the host:

```
$ sudo docker exec -it MineShark_AE bash
```

[Execution] In the first terminal, launch the director script:

```
$ ./detector.sh
```

In the second terminal, start the confirmation module while the detector is running:

```
$ ./analyser.sh
```

In the third terminal, check the database and the ranking of suspicious addresses during the pipeline execution:

```
$ python detect_info.py
```

[Results] The detector displays real-time throughput and inference statistics, as shown on the left-hand side of `sample_output.png`. The analyser displays the confirmation process of suspicious flows, as shown on the right-hand side of `sample_output.png`. After five minutes of execution, example documents of the `ipinfo` and `suspicious` collections are shown in `ipinfo.json` and `suspicious.json`, respectively. The output of the suspicious address ranking after five minutes should include `167.172.7.190:3331 (plain-text)` and `120.233.7.227:10443`.

3) Experiment (E3): [Efficiency Evaluation][10 human-minutes + 0.5 compute-hour]: This experiment evaluates the efficiency of MineShark in feature extraction and model inference under different CPU resources and identifies the performance bottleneck, corresponding to the directory `AE/experiment3`. A customized version of the detector is built with an additional `-DMONITOR_EXP` flag. This flag enables the detector to vary the inference workload at levels of 25%, 50%, 75%, and 100% by setting different input parameters. For each workload level, the number of core pairs allocated for feature extraction and model inference is varied from one to four. Each case is tested for 2 minutes, totaling 32 minutes to complete the testing of all 16 cases.

[Preparation] Build the detector with the following commands in the current directory:

```
$ mkdir -p build
$ cd build && cmake .. && make
```

[Execution] Start the test with this command:

```
$ python perf.py
```

[Results] After the completion of the test, the results will be available in the `figures` directory. Sample results on our testbed are shown in `sample_extract.png`, `sample_inference.png`, and `sample_throughput.png`. The overall trends should be consistent and align with the description in Section VII-D.

E. Customization

1) Adjustment of required CPU cores: E3 is configured to use up to ten CPU cores by default. If running this experiment on a machine with fewer cores, reduce the `que` number in the `perf.py` script. For details on making this change, see the `Customization` section in `AE/experiment3/README.md`.

2) Acceleration via GPU: To eliminate inference bottlenecks on the CPU, follow these steps to use GPU acceleration.

First, install the NVIDIA Container Toolkit (https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html).

Second, launch the container with GPUs as follows:

```
$ sudo docker run -it --gpus all \
--privileged -v <Hugepage_Path>:/mnt/huge \
-v <MineShark_AE_Code_Path>:/root/AE \
-w /root/AE -e IOVA_MODE=va \
--name MineShark_AE haers/mineshark:AE
```

Third, execute the previous experiments in container with no changes while monitoring the GPU usage with the following command on the host machine:

```
$ watch -n1 nvidia-smi
```

Execution results of *E3* should match Figure 12d in our paper.

3) Deployment in one's own network: Researchers can reuse *E2*'s workflow to analyze traffic of interest in their own networks using a collected packet file. Additionally, they can deploy the system for live network traffic monitoring after setting up the drivers for the network interface to be monitored and configuring the detection file. For details, see the `Scope and Customization` section in `AE/README.md`.

4) Extend to new models and tasks: This artifact can be customized for new models and tasks. Researchers should implement new functions in the detection pipeline to define the feature extraction process required by the target model and the feature calculation process with the model. For other classification tasks, in addition to adapting the model-specific changes, the confirmation mechanism will also need to be redesigned. For details on aligning these changes with specific code segments, see the `Scope and Customization` section in `AE/README.md`.