electronics

MDPI

Article

To (US)Be or Not to (US)Be: Discovering Malicious USB Peripherals through Neural Network-Driven Power Analysis

Koffi Anderson Koffi [1], Christos Smiliotopoulos [2], Constantinos Kolias [1,*] and Georgios Kambourakis [2]

[1] University of Idaho, Idaho Falls, ID 834024, USA; kkoffi@uidaho.edu
[2] University of the Aegean, 83200 Karlovasi, Greece; csmiliotopoulos@aegean.gr (C.S.); gkamb@aegean.gr (G.K.)
* Correspondence: kolias@uidaho.edu

Abstract: Nowadays, The Universal Serial Bus (USB) is one of the most adopted communication standards. However, the ubiquity of this technology has attracted the interest of attackers. This situation is alarming, considering that the USB protocol has penetrated even into critical infrastructures. Unfortunately, the majority of the contemporary security detection and prevention mechanisms against USB-specific attacks work at the application layer of the USB protocol stack and, therefore, can only provide partial protection, assuming that the host is not itself compromised. Toward this end, we propose a USB authentication system designed to identify (and possibly block) heterogeneous USB-based attacks directly from the physical layer. Empirical observations demonstrate that any extraneous/malicious activity initiated by malicious/compromised USB peripherals tends to consume additional electrical power. Driven by this observation, our proposed solution is based on the analysis of the USB power consumption patterns. Valuable power readings can easily be obtained directly by the power lines of the USB connector with low-cost, off-the-shelf equipment. Our experiments demonstrate the ability to effectively distinguish benign from malicious USB devices, as well as USB peripherals from each other, relying on the power side channel. At the core of our analysis lies an Autoencoder model that handles the feature extraction process; this process is paired with a long short-term memory (LSTM) and a convolutional neural network (CNN) model for detecting malicious peripherals. We meticulously evaluated the effectiveness of our approach and compared its effectiveness against various other shallow machine learning (ML) methods. The results indicate that the proposed scheme can identify USB devices as benign or malicious/counterfeit with a perfect F1-score.

Keywords: USB; side channel analysis; power side channel; power fingerprinting; identification; authentication; physical layer; signal processing; autoencoder; long short-term memory networks; neural networks; deep learning; dataset

Citation: Koffi, K.A.; Smiliotopoulos, C.; Kolias, C.; Kambourakis, G. To (US)Be or Not to (US)Be: Discovering Malicious USB Peripherals through Neural Network-Driven Power Analysis. *Electronics* **2024**, *13*, 2117. https://doi.org/10.3390/electronics13112117

Academic Editor: Myung-Sup Kim

Received: 8 April 2024
Revised: 10 May 2024
Accepted: 15 May 2024
Published: 29 May 2024

1. Introduction

USB devices are pervasive in Small Office/Home Office (SOHO), large enterprise networks, and Industrial Control Systems (ICS). In particular, version 3 of the USB protocol is currently dominant, and its global market is projected to reach 6.3 billion USD by 2027 [1]. Today, USB is the most common technology for importing/exporting data to highly secure, air-gapped networks. Therefore, it does not come as a surprise that USB has been in the scope of malicious actors. Interestingly, it has been reported that 30% of all the surveyed ICS security threats in 2020 used a type of USB removable media as an attack vector [2]. An example of this methodology is the high-profile attack against a nuclear enrichment plant in Natanz, Iran [3]. In this case, it is believed that USB thumb drives were initially used to proliferate the Stuxnet malware [3] inside the secure facilities. More specifically, the malware exploited an unknown vulnerability (zero-day) at the time against the MS Windows autorun configuration that allowed code to be executed without permission.

Instead, USB enjoys the trust of users, the majority of whom still appear to be totally unaware of the real dangers that USB peripherals may represent. As demonstrated in [4], 68% of surveyed users took no precaution when connecting USB peripherals to personal or corporate computers. Naturally, this introduces the threat of various USB ransomware infections [5]. At the same time, other peripheral devices such as charging cables [6] and USB hubs [7] can be used to establish covert channels and achieve the leakage of sensitive information even from within air-gapped networks [8].

USB peripherals receive power from the host. When a given peripheral operates normally, the power consumption falls within a profile for the observed operation. However, virtually any type of extraneous activity, say, the operations performed after a peripheral device has been infected, will require additional power. This frequently results in intensive and non-canonical power consumption patterns. The main hypothesis of this work at hand is that such patterns can act as an indicator of anomalous activity. Moreover, we claim that this is valid, regardless of the type of attack and whether it is known or a zero-day, i.e., a type of never-seen-before attack. Finally, the reader should consider that power consumption traces are very hard to conceal and/or manipulate, as extraneous operations *need* to consume additional power, and there is no way to circumvent that.

Our proposed methodology operates at the physical (PHY) layer of the USB protocol stack and aims to provide both the identification/authentication of USB devices and anomaly detection for potentially malicious USB devices. It can be implemented as a USB dongle for seamless integration into the existing computer infrastructure. Our end goal is not only to distinguish between benign peripherals but, at the same time, to detect and prevent USB attacks and/or malicious USB peripherals that pose as benign (e.g., BadUSB). The novelty of the proposed solution lies in the chosen core analysis methodology, namely, side channel analysis (SCA) based on power consumption. In the past, SCA has mainly been employed for offensive purposes [8]. For example, a significant critical mass of research works has been dedicated to the application of SCA for breaking cryptographic keys based on the analysis of power consumption activity [9]. Other works rely on side channels to achieve the leakage of private information [10]. We capitalize on the same principles to design a novel, totally external, and non-intrusive protection system.

By employing an Autoencoder model for effectively extracting discriminate features from power traces in combination with a CNN and LSTM architecture (CNN-LSTM), it becomes possible to simultaneously achieve both tasks, authentication and identification. More specifically, our experiments demonstrate the effectiveness of the proposed system in identifying benign USB devices with perfect scores (F1-score, precision, recall, and ROC-AUC) with noisy data and a limited number of power traces captured by inexpensive off-the-shelf components.

In summary, the contributions of this paper are as follows:

- We provide a novel framework for identifying potentially malicious USB peripherals through the automated analysis of power consumption patterns.
- We experimentally prove that even power signals captured with an inexpensive setup comprising solely off-the-shelf components can lead to the construction of a highly robust detection engine.
- We prove that Autoencoders can effectively extract high-value features out of raw power traces much faster than conventional approaches. These can later be capitalized to train shallow or deep neural network (DNN) models.
- We demonstrate that contrary to popular belief [11], the discriminative CNN-LSTM model can be trained fast with only a handful of signals and still provide perfect scores (F1 score, precision, and recall).

The rest of the paper is organized as follows. The next Section 2 summarizes the most influential works in the area. Section 3 describes the threat model and outlines the main assumptions of the proposed systems. A full description of the proposed framework is given in Section 4, while Sections 5 and 6 detail the experimental setup, the datasets used,

and the results of our evaluation, respectively. The last Section 7 concludes and offers avenues for future work.

2. Related Work

This section provides a taxonomy and summarizes the most important attack methodologies targeting USB peripherals. It also enumerates corresponding protection mechanisms.

2.1. USB Attacks

From a 10,000-foot view, USB attacks can be classified into four major families based on (a) whether the attack relies on sophisticated programmable microcontrollers, (b) the type of the USB peripherals employed, (c) the use of special-purpose hardware, and (d) malware that proliferates via USB. This taxonomy extends the one introduced by Nissim et al. [12]. Hereunder, we describe the most well-known USB attack categories, along with some representative examples per category.

Hidden Programmable Microcontrollers. Certain USB attacks hinge on custom programmable microcontrollers or maliciously reprogrammed ones. Some devices in this category, such as the O.MG cable [6] shown in Figure 1, are often referred to as USB doppelgangers. These devices often contain a camouflaged malicious microcontroller inside their enclosure. Typically, the aim of such approaches is to enable the leakage of sensitive information and perform covert malicious activities. Other examples in this category are the *USB Rubber Ducky* [13], which poses as a benign USB thumb drive that can exfiltrate data from the victim host. Finally, *Evilduino* [14], *USBdriveby* [15], and *Spyduino* [16] are additional examples that are based on Arduino microcontrollers.

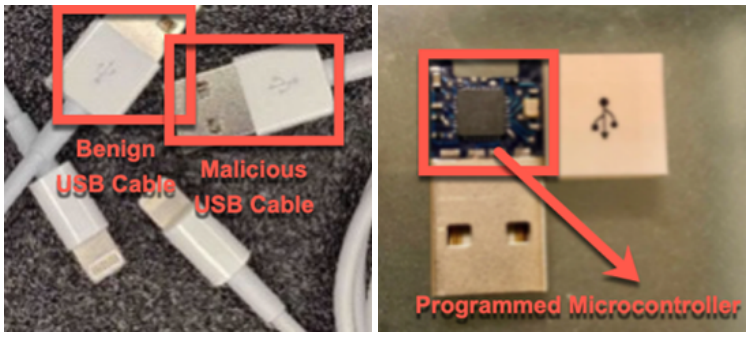


Figure 1. On the surface, an O.MG cable appears to be a simple USB cable (**left**) (Image from [17]); however, such devices have a malicious microcontroller embedded at the connector side (**right**) (Image from [18]).

Maliciously Re-programmed Devices. Attacks in this category usually perform malicious activities against a target system by emulating a USB Human Interface Device (HID), such as a mouse, keyboard, webcam, printer, etc. A prominent example of this situation is the *BadUSB-C* [7], which is based on USB type-C peripherals with re-programmed firmware, allowing it to masquerade as a keyboard to perform malicious activities against the host. The authors in [19] re-programmed a USB flash drive to mount a second hidden partition to the device and exfiltrate the user's data. Another example of such an attack is described in [20]. The researchers relied on a firmware modification of the Logitech G600 mouse against an air-gapped host. Finally, the Kali NetHunter is an Android-based penetration testing tool that can make an Android smartphone impersonate any USB HID and perform BadUSB-like attacks [21].

Custom-Tailored Hardware. This category of attacks is dominated by hardware designed to cause physical damage to the target device's circuitry when connected. This is usually accomplished by delivering a power surge. USBKill V4.0 [22] is a characteristic example of such a device. It can trigger the power surge at command, even remotely, via a smartphone,

at a scheduled time and date, or by detecting changes in the environment near it. This device can effectively perform a permanent denial-of-service attack against hosts.

USB malware. Different from the previous attack methodologies, benign USB devices can be used as mere carriers of malware against hosts. Malware of such a type can be viruses, trojans, backdoors, spyware, or ransomware. For example, the ZCRYPT [5] ransomware infects hosts primarily through USB dongles, flash drives, or other USB accessories. It encrypts all the user's files and requires a ransom payment of 1.2 BTC before decryption.

2.2. USB Defenses

Current USB defense mechanisms include antimalware methods, packet scanning tools, cryptographic methods, and sandboxing techniques. These mechanisms are designed to protect against the aforementioned USB attacks.

Antimalware Methods. Antimalware methods are the most common defense mechanisms against USB attacks. They aim to identify and remove malware and include antivirus software and scanning tools. Modern antivirus software is designed for the identification and removal of malware. Such controls, operating at the application layer, primarily analyze USB storage files to detect and remove malware. Therefore, these methods are not well suited for USB attacks that are based on programmable microcontrollers or repurposed USB hardware. In many cases, the repurposed or re-programmed device can effectively masquerade its malicious components and evade detection. On the other hand, USB scanning kiosks attempt to detect malicious USB storage payloads at the point of entry of organizations' networks. Internally, they may rely on antivirus software to perform the detection. Some of the most well-known devices in this category are the OPSWAT kiosks [23] and OLEA USB kiosks [24] from, respectively, the companies *OPSWAT*, *FL, USA* and *OLEA Kiosks*, *CA, USA*. Their main advantage over simple antiviruses is that kiosks protect against the proliferation of malware in an organization by detecting and removing it from a centralized independent location, not the host. In addition, some kiosks such as the Symantec ICSP Neural [25] can authenticate and authorize USB devices in an organization. Particularly, the Symantec ICSP Neural, (from Symantec Enterprise by *Broadcom Inc.*, *CA, USA*) uses both ML and a signature-based detection system to identify and delete malware from USB storage devices. Nevertheless, USB scanning kiosks share the same weakness as antiviruses, as they fail to protect against reprogrammed microcontrollers and repurposed USB hardware. Both antivirus software and USB scanning kiosks mainly target USB storage devices, and their analysis does not consider the packet-level interaction of the USB device with the host.

Cryptographic Methods. USB tools based on cryptographic schemes and hardware security chips aim to authenticate USB peripherals and authorize their usage. The work in [26] developed a Trust Management Scheme, namely *TSMUI*, for USB storage devices in Industrial Control Systems (ICS) environments. The approach uses a digital signature and a hardware security chip to authenticate USB storage devices and authorize their limited usage in some protected ICS hosts. The authors in [27] implement a three-factor control strategy to safeguard USB connections. This authentication approach merges biometric data, passwords, and smart cards to enhance security during USB mutual authentication. For secure and effective data exchange between the user and the USB server, the protocol utilizes Elliptic Curve Cryptography (ECC) for data encryption. This method stands out by requiring significantly smaller key sizes compared to alternative encryption techniques. Additionally, it minimizes computational demands on the smart card, ensuring efficient data transmission for USB devices.

Sandboxing Techniques. USB sandboxing tools aim to contain the USB peripheral communication with the host in an isolated virtual environment. SandUSB [28] is an example solution that analyzes and scans USB devices to identify malicious attempts without requiring changes to the host. Unlike USBGuard [29], which is a similar tool but is based on the Linux OS, SandUSB is a non-intrusive and platform-independent hardware-based (Raspberry Pi 2 Model B) defense tool, placed between the host and the USB device.

It provides isolation between the USB device and the host by scanning USB packets for malicious payloads and blocking or permitting the connection to the host. SandUSB can protect against most USB-based attacks by warning the user of malicious USB packets or device activities. Although it does not use any cryptographic operation for authentication, SandUSB relies on the user to verify the authenticity of the USB device. Nevertheless, a notable point is that sandboxing techniques do not protect against attacks that are based on altered/repurposed USB hardware.

Packet Scanning Tools. USB packet scanners and filters are tools that monitor the USB packets exchanged between the host and the device. *USB-Watch* is a framework that aims to identify rogue anomalous USB device behavior by analyzing the traffic exchanged with the host [30]. It is custom hardware that uses a Decision Tree anomaly detection classifier. After a USB device has been identified as abnormal, USB-Watch blocks data communication with the host until the user prompts it. On the one hand, both GoodUSB [31] and USBFilter [32] are USB packet-level firewalls capable of providing fine-grained access control to USB peripherals by instrumenting the host's OS. For example, USBFilter can prevent a USB device from masquerading as a keyboard when attempting to access the host's webcam. GoodUSB utilizes Linux virtualization to enforce permissions on the user's expectation of the device functionality.

Existing packet scanning tools like USB-Watch [30], GoodUSB [31] and USBFilter [32] are designed to monitor and control the packet exchanges between the host and the device, aiming to mitigate USB security threats. However, these tools might not be completely effective in preventing intrusions due to the inherent complexities of security-critical systems. Active Directory's consistent management of user and device control complements our approach [33]. In Windows-based environments, Active Directory can work in conjunction with our framework, which can be implemented as a USB dongle with wireless communication capabilities, providing device/peripheral-level authentication (as opposed to a human user). These existing challenges underline the necessity of employing complementary defensive measures. Active Directory, coupled with our proposed USB device authentication approach, which is based on power side-channel analysis, can collectively forge a strong, comprehensive defensive mechanism against USB security threats.

The proposed USB authentication system is designed to address the limitations of existing USB defense mechanisms. Table 1 provides a comparative analysis of our system with various tools and methods used to defend against USB-based attacks. The table highlights the limitations of each defense mechanism and how our system addresses these gaps. Our system offers automated device authentication and detects anomalies in USB device behavior, providing a more robust defense against USB-based attacks.

Table 1. Comparison of USB defense mechanisms vis-à-vis the proposed framework.

Defense Mechanism	Tool/Paper	Target USB Attacks	Intrusive to Host	Limitations	Gaps Addressed by the Our Approach
Antimalware Methods	Antivirus software	Malware in USB storage	Yes	- Limited to known malware signatures - Fails to detect reprogrammed or repurposed USB devices	- Authenticates devices at the hardware level - Detects anomalies in USB device behavior
	USB scanning kiosks (OPSWAT, OLEA, Symantec ICSP Neural)	Malware in USB storage	No	- Centralized solution, not host-specific - Fails to detect reprogrammed or repurposed USB devices	- Detects anomalies in USB device behavior
Cryptographic Methods	TMSUI [26]	Most USB attacks	Yes (requires security chip)	- Designed specifically for ICS environments - Requires external solutions to identify benign USB devices	- Applicable to various environments - Offers an integrated solution for device authentication
	Three-factor control strategy [27]	Unauthorized access	Yes (requires smart card)	- Relies on biometric, password and smart card to authenticate the device authentication	- Requires only the power consumption trace for device authentication
Sandboxing Techniques	SandUSB [28]	Most USB attacks	No	- Relies on user to verify USB device authenticity - Does not protect against altered/repurposed USB hardware	- Provides automated device authentication - Detects anomalies in USB device behavior
	USBGuard [29]	Unauthorized USB devices	Yes	- Requires manual intervention to block unauthorized devices - Does not detect altered/repurposed USB hardware	- Provides automated device authentication - Detects anomalies in USB device behavior
Packet Scanning Tools	USB-Watch [30]	Rogue USB devices	No	- Relies on Decision Tree anomaly detection classifier - Can fail to prevent intrusions due to system complexities	- Uses deep learning for anomaly detection - Detects anomalies in USB device behavior
	USB Filter [32]	Unauthorized USB devices	Yes	- Requires complex setup and rules on the host - Does not detect altered/repurposed USB hardware	- Provides automated device authentication - Detects anomalies in USB device behavior
	GoodUSB [31]	Unauthorized USB devices	Yes	- Requires setting up a policy block unauthorized devices - Does not detect altered/repurposed USB hardware	- Provides automated device authentication - Detects anomalies in USB device behavior

3. Technical Background

In this section, we provide technical background on the USB protocol, side channel analysis, and the deep learning architectures used in the proposed framework.

3.1. The USB Protocol

The USB protocol is a wired communication protocol that facilitates the serial transmission of data, enabling the connectivity of computers and various types of peripheral devices [34]. Apart from data transfer, USB also provides power to connected devices. The first version of the USB protocol, namely, USB 1.0/1.1, has a transmission speed that ranges from 1.5 Mbps to 12 Mbps. That version is primarily used to connect peripherals such as keyboard and mouse input devices. The second version of the protocol, USB 2.0, extends the transmission speed profile to 480 Mbps, which enables the connectivity of webcams and external hard drives [35]. With reference to Figure 2, USB 3.0/3.1 supports two transmission modes: SuperSpeed mode with a speed of 5 Gps and SuperSpeed+ with a speed of 10 Gbps [36]. This latest version of the protocol can also provide up to 100 W of power to devices.

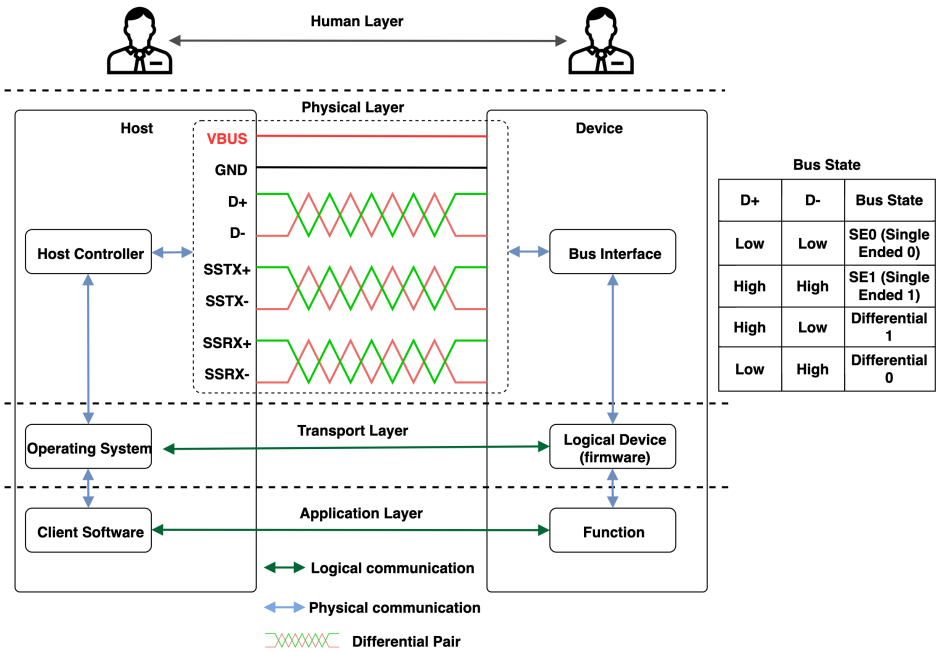


Figure 2. USB 3.0 protocol stack (based on [34,37]).

The organization of devices networked via the USB is according to a tiered star topology, with a host at the root and multiple devices as leaves. The host comprises a root hub and a host controller that is responsible for detecting the attachment and detachment of devices and managing data flow between the host and the connected peripherals. Moreover, the hub provides power to the attached devices and monitors the activity on the bus.

The bus interface and the host controller manage the packet transmission and reception between the host and peripheral devices. The USB controllers, often referred to as Root Hubs, are the hardware components that the OS communicates with to control USB devices. The host controller is the foundation for all USB activity and includes various types, such as UHCI, OHCI, EHCI, xHCI, and WHCI [34] (see Section 7). These controllers initiate and manage USB traffic and are typically configured using Peripheral Component Interconnect (PCI), i.e., a local computer bus that allows hardware devices to be attached to a computer [38]. Furthermore, the protocol operates through a series of processes that occur when a USB device is connected to a host. These processes include device (i) reset, (ii) enumeration, and (iii) device identification, setting the devices in one of the following states.

- **Attached State** The *attached* state corresponds to a device that is connected to the host/hub and does not give power to the VBUS.
- **Powered State** A device is in the powered state when it is attached to the bus, powered, and does not receive a reset signal from the host.
- **Default State** The default state corresponds to a device that is in the powered state and has been reset by the host.
- **Addressed State** An addressed device is in the default state and was assigned a unique address by the host.
- **Configured State** While a configured device is in the addressed state (was given a unique address) and was configured by the host.
- **Suspended State** Finally, a suspended device is in the configured state, but no activity has been observed on the bus for 3 ms.

The host can learn about a device's capabilities after the enumeration process, which is initiated by the host and involves attaching a USB device to the bus and configuring it.

3.2. Side Channel Analysis

A side channel analysis involves the study of the unintended leakage of a system's physical characteristics. The physical characteristics are often in the form of electromagnetic emissions [39,40], electrical power [11], timing [41], sounds [42], and temperature [43]. The analysis can either be offensive or defensive. In the former, side channels are used to break cryptographic keys, achieve the leakage of private information, and so on. In the latter, the side channel is used to protect systems from attacks. Both attack and defense mechanisms leverage analysis techniques that are timing based [44], profile based [45], frequency based [46], statistical [47], or machine learning (ML) [48]. A timing-based analysis, for example, involves the study of the time taken to perform specific operations (e.g., encryption). A profile-based analysis, on the other hand, establishes a baseline profile and compares it with the observed profile. A frequency-based analysis involves the study of a recurring pattern in the side channel data. These patterns can be used to identify the type of operation being performed.

In the case of the USB protocol, a side channel can be exploited to extract sensitive information [10], such as keystrokes or data transmitted through USB devices. As with the present work, ML techniques can be applied to side channel analysis to identify patterns and classify different types of USB peripherals, including malicious ones. The combination of USB side channel analysis and deep learning (DL) can provide a powerful approach to enhancing the security of USB communication and detecting potential threats. By leveraging the technical aspects of the USB protocol and understanding the underlying mechanisms of USB communication, researchers can effectively extract features towards exploiting models such as Autoencoders to mitigate attacks and recognize malicious USB peripherals.

3.3. Deep Learning

DL is a sub-field of ML that focuses on developing algorithms, namely, deep neural networks (DNN), that learn from data [49]. These algorithms are designed to automatically learn to identify patterns from data using artificial neural networks (ANNs). The particularity of a DL network is that it has multiple layers of interconnected neurons that form various architectures. The most common DL architectures include convolutional neural networks (CNNs), recurrent neural networks (RNNs), and long short-term memory (LSTM) networks. Additionally, DNNs can be trained in various ways, including supervised learning, unsupervised learning, and reinforcement learning. Algorithms, such as deep Autoencoders, can make use of unsupervised learning to learn to extract complex features from data without the need for labeled data. In the context of this work, Autoencoders can effectively minimize complex power consumption signals down to the features that are most elemental for discriminative and anomaly detection purposes.

Autoencoders. Autoencoders have garnered significant attention in DL research due to their remarkable ability to capture essential features from raw data [49,50]. They can be described

as a family of dimensionality reduction methods [51] that learn a latent representation by compressing the input data while minimizing a reconstruction loss error (e.g., mean squared error as shown in Equation (1)). These models consist of two primary components, an encoder and a decoder module. The encoder is an encoding function $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^d$ that maps the input data $X \in \mathbb{R}^D$ to a lower-dimensional latent representation $Z \in \mathbb{R}^d$, where $d < D$. On the other hand, the decoder, denoted as $\psi : \mathbb{R}^d \rightarrow \mathbb{R}^D$, reconstructs an approximation $\hat{X} \in \mathbb{R}^D$ of the original input as depicted in Figure 3. Autoencoders have found applications in various domains, including image denoising, anomaly detection, and dimensionality reduction:

$$L(X, \hat{X}) = \frac{1}{N} \sum_{i=1}^{N} (X_i - \hat{X}_i)^2 \tag{1}$$

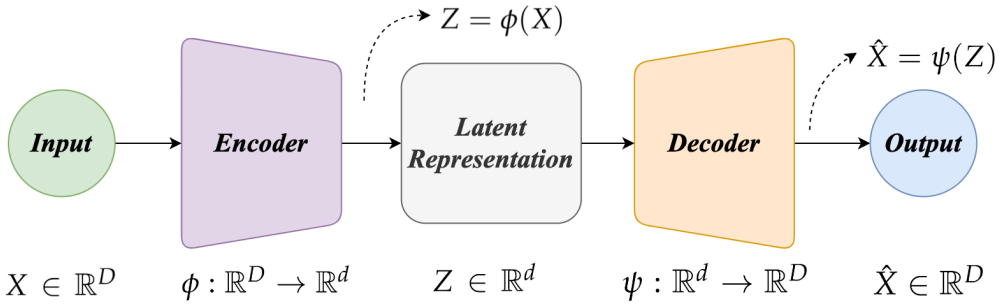


Figure 3. Autoencoder architecture.

CNN and LSTM. They are a class of DL networks that have been extensively used for image- and signal processing tasks. In particular, CNNs are designed to automatically and adaptively learn spatial hierarchies of features from input data. They are composed of layers, such as convolutional, pooling, and fully connected layers. CNN offers several advantages over a fully connected network, including local connectivity, parameter sharing, equivariant representations, spatial hierarchies, and translation invariance [52]. The core operation in a CNN is the convolution operation (or cross-correlation), denoted as $*$ [53]; given an input feature map I and a learnable kernel filter K, the convolution operation is defined according to Equation (2). While CNNs are suitable for learning spatial features, LSTMs were designed specifically for capturing temporal features. An LSTM network is a specific variant of recurrent neural networks (RNNs), which are specifically designed for processing sequential data, e.g., in our case, USB power consumption over time. Although RNNs are capable of modeling sequential dependencies, they suffer from vanishing gradient problems (exploding and diminishing gradients) when dealing with long sequences [54]. LSTM was designed to address the vanishing gradient problem and better capture long-range dependencies in sequential data. Moreover, both CNN and LSTM can be used as layers in the encoder and decoder of an Autoencoder to capture both spatial and temporal features of a USB power consumption time series data. This setup, in particular, could be more suitable for feature extraction and anomaly detection compared to shallow ML approaches:

$$(I * K)(i, j) = \sum_{m} \sum_{n} I(i - m, j - n) \cdot K(m, n) \tag{2}$$

An *attention mechanism* is a technique that allows models to dynamically focus on different parts of their input for making decisions, akin to the way humans pay attention to specific aspects of their environment while ignoring others. Attention mechanisms enhance model flexibility and interpretability by allowing them to dynamically adjust their focus on the input data, leading to improvements in performance and efficiency, especially on complex tasks that require an understanding of contextual relationships within the data.

LSTM models do not inherently incorporate attention mechanisms. However, attention mechanisms can be added to LSTM models as an additional component to enhance their performance on tasks where focusing on specific parts of the input sequence at different times can be beneficial. This combination of LSTMs and attention mechanisms has been particularly successful in various sequence-to-sequence applications.

4. Proposed Framework

In this section, we present the proposed framework for USB device identification and anomaly detection tasks. We outline key assumptions and the threat model and detail the model architecture and the training and testing procedures for the proposed framework.

4.1. Assumptions and Threat Model

The following assumptions are made about the USB devices and the protected system. We assume that the host is free from sophisticated malware infections that can manipulate the power consumption patterns of the connected USB devices. We consider USB cables, connectors, peripherals, and microcontrollers as potential attack vectors, given that malicious actors can repurpose USB hardware to exfiltrate data from a host. In addition, an attacker can reprogram a USB device to act as a USB peripheral, like a mouse, keyboard, or webcam. We assume that components of the protected system can communicate securely by establishing an independent wireless network. Moreover, an attacker may have prior physical access to the USB device or the host. The attacker may also have knowledge of the USB protocol and the host's operating system. The threat model includes the following attack vectors: (1) USB doppelgangers, (2) maliciously re-programmed devices, and (3) custom-tailored hardware. The proposed framework aims to secure legitimate USB users against these attacks by leveraging DNN architectures to analyze USB side channel data and detect potential threats.

4.2. Proposed Framework

The proposed framework leverages DNN architectures, such as CNNs, LSTMs, and Autoencoders with the incorporation of an attention mechanism, to analyze USB side channel data (power). An effective USB security framework should be able to detect and identify USB devices, and to alert the user of potential threats. The proposed framework is designed to achieve these goals by capitalizing on the temporal and spatial characteristics of USB power consumption signals via a novel CNN-LSTM scheme that is enhanced with an attention mechanism. The framework is designed to be flexible and adaptable to different USB devices and to provide a robust and scalable solution for defending against various attacks. It can be implemented on a desktop, laptop, or a Raspberry PI. As illustrated in Figure 4, the most important functions of the proposed framework consist of (1) USB power signal capture, (2) feature extraction, (3) device identification, (4) anomaly detection, (5) device ID storage, and (6) user alerting.

① **USB power signal capture.** The power consumption signal of a connected USB device is captured by a small resistor in series with the GND wire of the USB connector. The voltage drop across the resistor is then measured using an oscilloscope or a data acquisition device. The resulting power consumption signal is digitized and stored for further analysis. To obtain the corresponding label, we initiate the data capture and also tag the signal with its respective label. These labels serve as the ground truth for the later stages of model training and evaluation.

② **Feature extraction.** Next, feature extraction is performed on the power consumption signal. Towards this end, with reference to Section 3.3, the proposed Autoencoder model is applied due to its efficiency in extracting complex features from time series power signals.

③ **Device identification.** The extracted features are used to identify the connected USB device. This step is accomplished via the aforementioned CNN-LSTM model. The model is trained to learn the temporal and spatial characteristics of USB power consumption signals

and to differentiate between multiple USB devices. It then outputs the predicted ID of the connected USB device.

④ **Anomaly detection.** Alternatively, the same extracted features are also forwarded to the anomaly detection model to detect a potential anomalous USB power consumption signal. The model outputs a binary label indicating whether the connected USB device is normal or anomalous.

⑤ **Device IDs Storage.** Once a device is connected to the host for the first time and is identified as normal, it is stored in a database for future reference associated with a unique identifier. This is done to keep track of the connected USB devices and to identify them for future connections easily.

⑥ **User Alert.** If the connected USB device is identified as anomalous, the user is alerted to take appropriate action. This may include blocking the USB device, disconnecting it from the host, or blacklisting it.

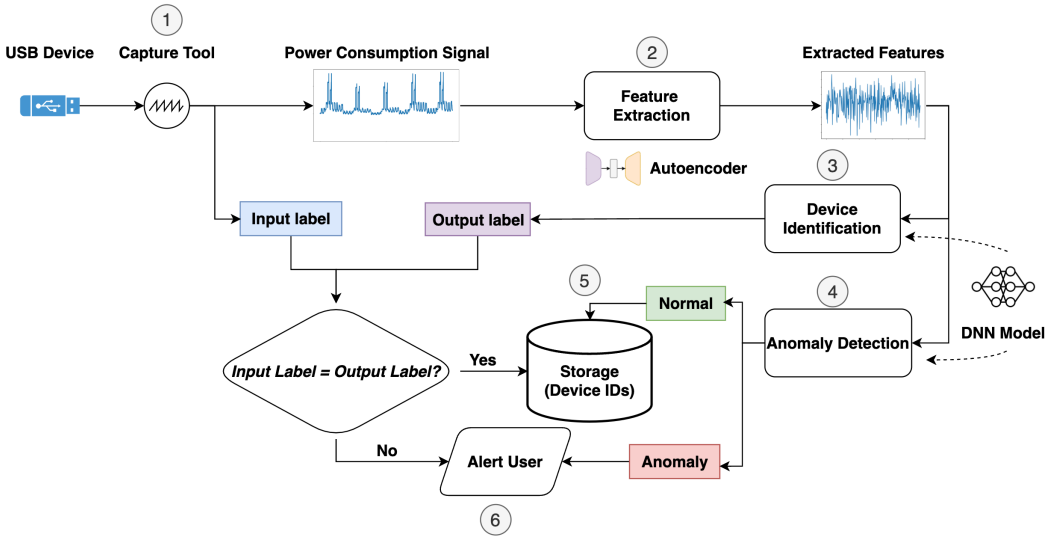


Figure 4. Abstract view of the proposed framework. The framework employs an Autoencoder to extract the features used with the CNN-LSTM model for device identification and anomaly detection.

The proposed framework is summarized in Algorithm 1. The algorithm takes as input the USB power signal and the label ID of the connected USB device. It outputs the device ID and a binary label indicating whether the connected USB device is normal or anomalous. The algorithm consists of three main procedures: feature extraction, device identification, and anomaly detection.

Algorithm 1 Detection algorithm

Input: $X \in \mathbb{R}^D, y_i \in \mathbb{N}$ ▷ USB power signal vector and label ID
Output: $y_o \in \mathbb{N}, y_a \in Normal, Anomaly$ ▷ identification and anomaly detection labels
Procedure:
$F \leftarrow extract\_features(X)$ ▷ Extract $F \in \mathbb{R}^d$ features from X
$y_o \leftarrow identify\_device(F)$ ▷ Identify device
$y_a \leftarrow detect\_anomaly(F)$ ▷ Detect if the device is anomalous
if $y_a = Normal$ and $y_i = y_o$ **then**
    $store\_device\_id(y_i, y_o)$ ▷ Store device ID
else
    $alert\_user(y_i, y_o)$ ▷ Alert the user
end if

4.3. Model Architectures

Core Architecture. The Autoencoder and the DL model used in the proposed framework are based on a core architecture with various orientations of the LSTM and CNN layers. In particular, as shown in Figure 5, based on an *orientation* hyperparameter, the CNN and LSTM are oriented as follows: (a) in series with the CNN layers followed by the LSTM layers, (b) in series with the LSTM layers followed by the CNN layers, and (c) in parallel with the CNN and LSTM layers processing the signals simultaneously and combining their outputs. In addition, an attention mechanism is integrated into the LSTM module to enhance the model's ability to focus on salient features within the time series data. This architecture is utilized in the encoder (with the attention layer) and decoder modules of the proposed Autoencoder model.

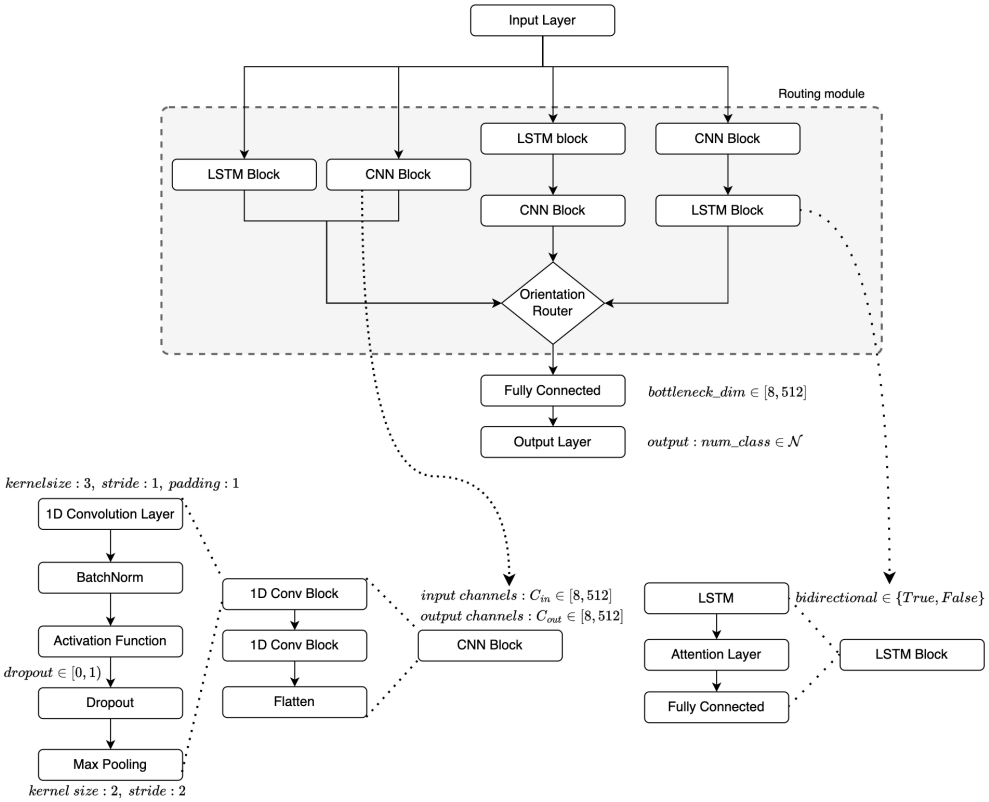


Figure 5. Key architectural elements employed in the Autoencoder and deep learning model. The Autoencoder is composed of encoder and decoder components. The DNN model is built on the integration of an LSTM layer, a CNN layer, and an attention mechanism, designed to optimize feature extraction and temporal pattern recognition, and focus on significant sections of the signal, respectively.

The proposed architecture utilized several building blocks (modules). In particular, the CNN layers block in the architecture is composed of a 1D convolutional layer with kernel size 3, stride 1, and padding 1. It is followed by an optional batch normalization layer and activation function (e.g., leaky ReLU activation), dropout regularization, and a max pooling of size 2. The next building block is an LSTM block that comprises a number of LSTM layers, followed by an attention mechanism and a linear transformation to obtain the encoded bottleneck features. The output of the CNN block and the LSTM block are processed based on the user-specified orientation of the CNN and LSTM layers in the routing module. The output of the routing module is then passed through a fully connected layer. The final building block is a linear layer for classification that predicts the device labels or anomaly class.

Attention mechanism. An attention mechanism was employed within the core architecture to focus on the most useful parts of the input sequence. This mechanism is particularly

useful, given the temporal nature of voltage signals from USB devices, where specific patterns or signal characteristics at certain time steps may be more informative for both reconstruction and classification tasks. The attention module is integrated into the LSTM layer's output, which processes the sequential data. Each LSTM output represents a combination of learned features at each time step. The attention mechanism computes the attention weights by applying a linear transformation followed by a softmax function to the output of the bidirectional LSTM. This normalizes the weights for each time step, indicating the relative importance of each step's features. These attention weights are then used to create a weighted combination of the LSTM outputs, effectively allowing the model to focus on more informative parts of the time series. The resulting context vector, a weighted sum of LSTM outputs, serves as a latent representation of the input sequence, emphasizing crucial temporal features. The attention-enhanced context vector is subsequently passed through another linear layer to ensure dimensionality consistency with the bottleneck representation (*bottleneck_dim*). Furthermore, the inclusion of the attention mechanism enables the model to adaptively prioritize information across different parts of the input sequence, improving the model's sensitivity to pivotal temporal patterns within the USB voltage signals. This approach addresses the challenge of varying signal dynamics across different devices by allowing the model to dynamically adjust its focus, leading to improved feature extraction for both reconstruction and classification performance. By concentrating on relevant time steps, attention not only strengthens the model's interpretability by highlighting which parts of the signal most influence the output but also enhances the model's ability to generalize. This is a key advantage in USB device identification and anomaly detection tasks.

The attention mechanism, pivotal in our proposed framework, enables the model to prioritize the most influential parts of the input signal. This model employs an additive attention mechanism—a simplified alternative to more intricate attention mechanisms often found in natural language processing tasks. As illustrated in Figure 6, it is a simplified version of the Bahdanau Attention proposed in [55]. This mechanism computes the attention weights for each time step in the input signal. These weights are subsequently multiplied with the LSTM layer output to generate the model's ultimate output. Thus, the attention mechanism guides the model to recognize the most significant elements of the input signal for identification and anomaly detection in power consumption data. By concentrating on these crucial areas of the input signal, we enhance the model's efficiency in identifying USB attacks.

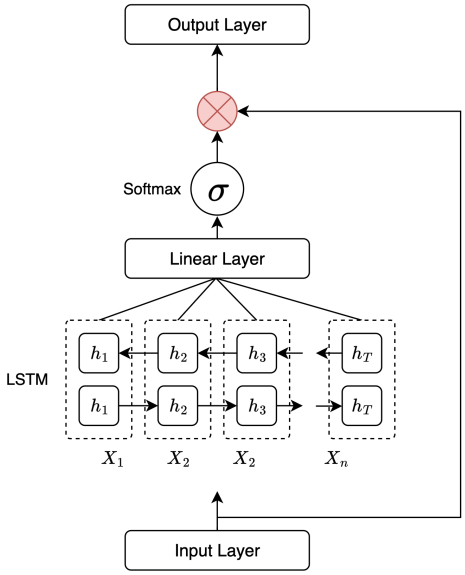


Figure 6. Additive attention mechanism in the proposed framework.

CNN-LSTM Model. The previous building blocks are combined to construct a classifier model for device identification and anomaly detection tasks. In particular, we created a CNN-LSTM model based on the architecture described in Figure 5 to identify USB devices as benign/malicious or classify them based on their user-provided ID. A set of hyperparameters must be carefully chosen to optimize the model's performance on those tasks.

Autoencoder Model. The proposed Autoencoder model, utilizing the core architecture, was trained in a semi-supervised fashion using the power consumption signals as input. The model outputs a reconstructed version of the input signals. The mean squared error (MSE) reconstruction loss function was added to the classification loss function to train the model. The addition of a classification loss enables the Autoencoder to learn latent features that are useful for downstream classification tasks. The encoder module comprises two 1D convolutional layers with kernel size 3, each followed by batch normalization, leaky ReLU activation, max pooling of size 2, and dropout regularization. The CNN outputs were then passed through a bidirectional LSTM layer, and the final hidden states were concatenated and linearly transformed to obtain a total of 128 encoded features.

The decoder maps the encoded state of a fixed-size vector, using a linear layer to transform the encoded features back into the LSTM input dimension. The full temporal sequence is reconstructed from the LSTM input representation via a bidirectional LSTM. Transposed 1D convolutions with kernel size 3, batch normalization, leakyReLU or ReLU activations, and dropout between the layers are leveraged to upsample the LSTM outputs. A final 1D convolutional layer maps the outputs back to the original signal dimension. In parallel to the decoder, the encoded features are passed through a linear classification head to predict the device class probabilities. Note that the key hyperparameters of the model include the dimensionality of the hidden layer, the number of LSTM layers, the number and size of convolutional filters, the dropout rate, the batch size, and the learning rate. To avoid vanishing gradient issues, an activation function such as LeakyReLU may be chosen. Batch normalization can also stabilize learning by normalizing the input of activation functions.

The selection of each deep learning component in this research was strategically made to address the nuances of USB defense. Autoencoders were employed not merely for their general feature extraction and dimensionality reduction abilities but particularly for detecting anomalies in complex power consumption data, which is an essential aspect of defending USB systems. The unique combination of CNN and LSTM was chosen to exploit the spatial and temporal dependencies of power consumption signals, which traditional RNNs lack. RNNs are known to struggle with long-term dependencies [54], which is a common issue in time series data. In contrast, LSTM is specifically designed to handle long-term dependencies, making it a better choice for our time series data.

The data considered in this study are temporal. On the surface, adopting CNN might seem inappropriate, as such models are known to perform exceptionally well on spatial data [56]. However, CNNs are not solely tied to spatial data. CNNs have been shown to be effective in detecting local patterns within sequences [52], and when used with temporal data, they can uncover important local temporal dependencies that may not be visible with LSTM alone [57]. By combining CNN with LSTM in our model, we can leverage the strength of CNNs in extracting complex features across sliding windows of time points - similar to motifs or local shapes in a time series. Both CNN and LSTM help in better capturing an intruder's footprint in the seemingly benign power usage data.

The attention mechanism was integrated to specifically direct the model's focus towards the most critical segments of the input signal. This improves the accuracy of threat detection under the conditions that often accompany USB attacks, such as short-lived yet significant changes in power signals. Therefore, each deep learning component was chosen not just for its standalone function, but primarily for its specific role in enhancing USB defense.

4.4. Training Procedure

Algorithm 2 presents the training procedure for the Autoencoder model with a weighted reconstruction and classification loss, incorporating linear attention. The algorithm takes as input time series data $X \in \mathbb{R}^{N \times T \times D}$ and corresponding labels $Y \in \mathbb{R}^{N \times C}$, along with an encoder $f_\theta(x)$ and decoder $g_\phi(z)$ module, a reconstruction loss function $\mathcal{L}_R$, classification loss function $\mathcal{L}_C$, the loss weights λ_R and λ_C, and a learning rate η. The Autoencoder parameters are initialized using *Kaiming* initialization [58]. In each training epoch, a minibatch of data is sampled, and the encoder maps the input to a hidden representation H. Linear attention weights A are computed and applied to H, resulting in an attended representation $\tilde{H}$. Next, the decoder reconstructs the input from the attended latent representation $\tilde{H}$, and a classifier predicts the labels using those same values. The total loss is a weighted sum of the reconstruction loss (mean squared error between original and decoded signals) and classification loss (cross-entropy) as shown in Equation (3). The gradients are computed with respect to the encoder and decoder parameters, and the parameters are updated using the learning rate (e.g., 0.001). The parameters are optimized using a mini-batch stochastic gradient descent algorithm [59] with the Adam optimizer until a maximum number of training epochs is reached:

$$\mathcal{L}_{\text{total}} = \lambda_R \times \mathcal{L}_R + \lambda_C \times \mathcal{L}_C \tag{3}$$

Here, $\mathcal{L}_R$ is the reconstruction loss function, $\mathcal{L}_C$ is the classification loss function, λ_R is the reconstruction loss weight, and λ_C is the classification loss weight.

Algorithm 2 Autoencoder training with weighted reconstruction and classification loss with linear attention.

Require: $X \in \mathbb{R}^{N \times T \times D}$ ▷ Input time series data
Require: $Y \in \mathbb{R}^{N \times C}$ ▷ Labels
Require: $f_\theta(x)$ ▷ Encoder
Require: $g_\phi(z)$ ▷ Decoder
Require: $\mathcal{L}_R$ ▷ Reconstruction loss function (e.g., MSE)
Require: $\mathcal{L}_C$ ▷ Classification loss function, (e.g., Cross Entropy)
Require: λ_R ▷ Reconstruction loss weight
Require: λ_C ▷ Classification loss weight
Require: η ▷ Learning rate
$\theta, \phi \leftarrow$ Initialize the Autoencoder parameters ▷ Initialize the model parameters with Kaiming initialization
while epoch $\leq$ max_epochs **do**
  $X_{\text{batch}} \in \mathbb{R}^{B \times T \times D}, Y_{\text{batch}} \in \mathbb{R}^{B \times C}$ ▷ Sample minibatch from X, Y
  $H \in \mathbb{R}^{B \times T \times H} \leftarrow f_\theta(X_{\text{batch}})$ ▷ Encoder forward pass
  $A \in \mathbb{R}^{B \times T \times 1} \leftarrow \text{Attention}(H)$ ▷ Compute attention weights
  $\tilde{H} \in \mathbb{R}^{B \times H} \leftarrow \sum_{t=1}^{T} A_{:,t,:} \odot H_{:,t,:}$ ▷ Apply attention
  $\hat{X} \in \mathbb{R}^{B \times T \times D} \leftarrow g_\phi(\tilde{H})$ ▷ Decoder forward pass
  $\hat{Y} \in \mathbb{R}^{B \times C} \leftarrow \text{classifier}(\tilde{H})$ ▷ Classifier forward pass
  $\mathcal{L}_R \leftarrow \mathcal{L}_R(\hat{X}, X_{\text{batch}})$
  $\mathcal{L}_C \leftarrow \mathcal{L}_C(\hat{Y}, Y_{\text{batch}})$
  $\mathcal{L}_{\text{total}} \leftarrow \lambda_R \times \mathcal{L}_R + \lambda_C \times \mathcal{L}_C$ ▷ Compute total loss
  $\nabla_\theta \leftarrow \nabla_\theta \mathcal{L}_{\text{total}}$ ▷ Compute gradients w.r.t. θ
  $\nabla_\phi \leftarrow \nabla_\phi \mathcal{L}_{\text{total}}$ ▷ Compute gradients w.r.t. ϕ
  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{total}}$ ▷ Update θ
  $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_{\text{total}}$ ▷ Update ϕ
end while
return θ, ϕ

4.5. Limitations and Future Work

The proposed framework has a few limitations that should be considered. First, the framework may not be able to detect all types of USB attacks. For example, the framework may not be able to detect attacks that exploit vulnerabilities in the USB protocol itself without causing significant changes in the power consumption patterns. More specifically, if an attacker uses a USB device to exploit a vulnerability in the USB protocol, the framework may not be able to detect the attack. Second, the framework relies on the assumption that the host is free from prior infections from sophisticated malware. In the rare case where it is infected with malware that intentionally causes fluctuations in power consumption, the proposed framework may not be able to detect the malicious activity accurately. Such attacks, however, are sophisticated and require extensive knowledge of the framework's implementation, which is often beyond the capabilities of a common attacker. Nation-state actors could potentially launch such an attack; therefore, future research will focus on enhancing the framework's detection capabilities to identify this possible malware-infected USB device attack effectively. Despite these limitations, the proposed framework is protocol based, thereby extending its applicability beyond desktops to provide a valuable tool for enhancing the security of USB communication and detecting potential threats in all USB-enabled devices. Future research will be devoted to assessing the efficiency of the proposed framework, especially in the fast-evolving domains of mobile and IoT devices. This includes the possibility of integrating sophisticated detection techniques or artificial intelligence to enhance the identification of covert malware infections, thus advancing the framework's efficacy.

5. Experimental Setup

This section details the experimental setup, namely, the hardware and software components, the datasets, and the parameters of the models used to evaluate the efficiency of the proposed framework.

5.1. Testbed

As depicted in Figure 7, the experimental setup comprises the following components: (a) a Raspberry Pi model 4B, (b) a breadboard, (c) USB 3.0 male and female screw terminal connectors (USB breakout), (d) a small resistor, (e) oscilloscope probes, and (f) a PicoScope 3000 series oscilloscope.

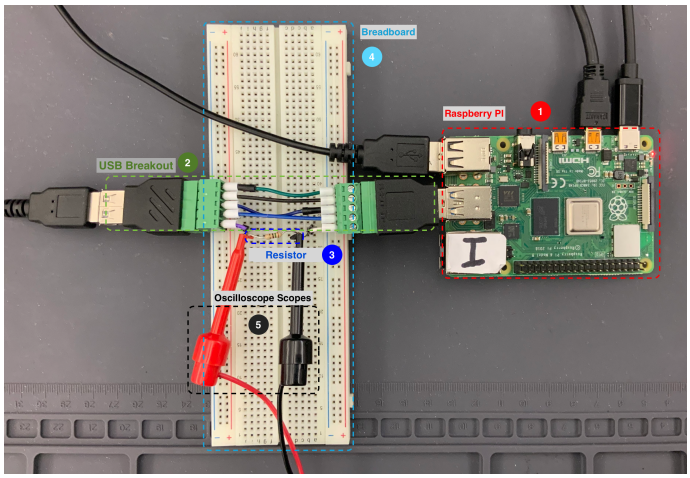


Figure 7. The utilized testbed. The oscilloscope is not shown.

① Raspberry Pi Model 4B. It is a small single-board computer developed and equipped with USB 3.0 ports and a USB-C power connector. The model 4B was selected to simulate a host to which we connect USB devices via the mounted USB screw terminal.

② **USB Breakout Connector.** It is a USB terminal adapter that allows the extension of the USB port, enabling the examination of the USB signals via the various pins on the connector. The screw terminal connector has male and female USB ports and exposes the USB bus signals (VCC, D+, D−, and GND) to the user. It was leveraged to connect the USB devices to the Raspberry Pi and to add a small resistor in series to the ground (GND bus) wire of the USB connector to measure the power consumption.
③ **Small Resistor.** A resistor with a low resistance is often used to measure the current flowing through a circuit. In our setup, the resistor was used to measure the voltage drawn (i.e., power consumption) by the USB devices. The power consumption was measured by attaching the two oscilloscope passive probes to each end of the resistor.
④ **Breadboard.** It was used to hold the resistor and provide a platform for connecting the USB devices to the Raspberry Pi via the USB Breakout connector.
⑤ **Oscilloscope.** It was connected to the Raspberry Pi via a USB cable. It was configured using the PicoScope 6 software [60] . Each voltage signal in the captures contained 100 K samples and was captured at a sampling rate of 1 MS/s (a million samples per second) and a time division of 10 ms/div. The collected samples were derived from five categories of devices, namely, flash drives, keyboards, mice, cables, and microcontrollers. Each device category contained several copies from different brands and devices. All the devices, except the cables, were either in the enumeration or an operating state. The captures were then used to create the experimental datasets.

5.2. Target USB Peripherals

We considered 29 different USB devices. The devices were selected to represent a diverse USB-oriented testbed, including flash drives, keyboards, mice, cables, and microcontrollers. Table 2 recapitulates the USB devices considered in our datasets.

Table 2. USB devices considered in our datasets.

Category	Devices	Brands
Flash Drive	11	4
Keyboard	6	3
Mouse	7	4
Cable	4	4
Microcontroller	1	1
Total	29	16

USB ables. Four USB-C cables were employed as test subjects: a malicious O.MG cable (denoted OMG), a *Just Wireless* cable (denoted J-A), a *heyday* cable (denoted HD), and a *naztec* cable (denoted NT). All cables used in the experiments of Section 6 have similar characteristics in terms of size, length, and dimensions of the connector. However, the OMG cable contains a programmable microcontroller embedded in its connector capable of launching various attacks, including running a Web server. Moreover, the same cable is equipped with an 802.11 radio module (Wi-Fi), which enables an attacker to establish a remote covert channel to receive commands or retrieve data. In this way, the attacker has several options. For example, they can perform remote code execution, log keystrokes, evade antiviruses, or even install backdoors. In our experiments, we chose to perform the fork-bomb attack. This is a type of denial-of-service (DoS) attack that aims to consume the resources of the host by causing a continuous replication of a process.
USB flash drives, keyboards, and mice. Four brands of USB thumb drives were utilized, namely, *Mosdat* USB 3.0 (denoted Mosdat), *Lexar* USB 3.0 (denoted Lexar-1), SanDisk (denoted Sandisk), and *PNY* USB 3.0. In particular, a device was selected from each brand. Additionally, four brands of USB keyboards were considered, namely, four Dell KB522p (denoted Dell-1, Dell-2, Dell-3, and Dell-4), one Perixx (denoted Perixx), one Lenovo KBCM21 (denoted Lenovo), one Perixx PERIDUO-212 (denoted Perixx-1). Also, the following brands of USB mice were used: four Logitech M100 (denoted Logitech-1,

Logitech-2, Logitech-2, and Logitech-4) mice, four Dell MS116 (denoted Dell) mice, and one Lenovo M-U0025-O (denoted Lenovo) mouse.

USB Microcontroller: The Teensy 3.2 microcontroller can be embedded in any USB thumb drive enclosure, giving the impression of benign devices. However, it can be programmed, using the Arduino IDE, for example, to perform several BadUSB attacks; in this experiment, the USBdriveby [15] was selected. The attack is based on modifying the firmware of the Teensy 3.1/3.2 microcontroller in an offline step and launching as soon as the device is connected to the target host. The malicious firmware makes the USB microcontroller behave like a USB Human Interface Device (HID), i.e., USB keyboard, mouse, and webcam. During the attack, the malicious device sends keyboard keystrokes and mouse events to the target host. More specifically, it attempts to open a terminal on the host and download a malicious executable file.

5.3. Datasets Used for the Detection Tasks

We created a set of datasets from the collected power trace samples for each USB identification task and anomaly detection task, e.g., brand identification, BadUSB anomaly detection, etc. The resulting collection consists of six datasets used in the experimental evaluation as presented in Table 3. Each dataset contains the raw power consumption signals, denoted as *Raw*, which are normalized to have zero mean and unit variance. Each signal in the dataset is comprised of 10 K samples (a fixed time window of 10 s), i.e., 400 signals per dataset.

Table 3. Task -specific datasets. Each dataset contains 400 power trace samples of length 10 k samples (10 s).

Dataset	Task	Classes	Signals	Signals/Class
Dataset A	USB Device Category Identification (Flash drive, Mouse, Keyboard, Cable)	Cable, Flash Drive, Keyboard, Mouse	400	100
Dataset B	USB Device Brand Identification (Mouse)	Dell, Lenovo, Logitech 1, Perixx	400	100
Dataset C1	USB Individual Device Identification (Mouse)	Logitech 1, Logitech 2, Logitech 3, Logitech 4	400	100
Dataset C2	USB Individual Device Identification (Keyboard)	Dell 1, Dell 2, Dell 3, Dell 4	400	100
Dataset D1	BadUSB Anomaly Detection (Keyboard)	Normal, Anomaly	400	300 (normal), 100 (anomaly)
Dataset D2	BadUSB Anomaly Detection (Cable)	Normal, Anomaly	400	300 (normal), 100 (anomaly)
Total			2400	

USB device category identification. The purpose of this dataset is to evaluate the classification accuracy of different USB categories. The dataset is denoted as Dataset A. More specifically, the dataset contains signals of the following categories: keyboard, mouse, flash drive, or cable.

USB device brand identification. The aim of this dataset is to evaluate the discriminative capability of algorithms regarding different manufacturers (brands) of the same USB device category (type). The dataset is *Dataset B*, and it contains USB mice of the following brands: Logitech, Perixx, Dell, or Lenovo.

USB device individual identification. The dataset can be used to evaluate the classification accuracy regarding individual devices from the same category and brand. The dataset is denoted as *Dataset C* and it has two subsets. The first contains signals of four alternative copies of Logitech M100 mouses, whereas the second subset contains signals from four Dell KB522p keyboards.

BadUSB anomaly detection. This last dataset aims to detect the presence of a malicious USB device among legitimate USB devices of the same type. This dataset includes two subsets, namely, *Dataset D1* and *Dataset D2*. The former contains signals of a malicious Teensy 3.2 microcontroller as well as signals obtained from benign keyboards. Similarly, the latter contains a malicious OMG cable and legitimate cables.

5.4. Experimental Models

A set of shallow ML models was used to evaluate the feature extraction capabilities of the proposed Autoencoder model, as well as the discriminative capabilities of the LSTM-CNN model that were discussed in Section 4. The following ML classifiers from the *scikit-learn* Python library were used in the experiments: Random Forest, Decision Tree, K-Nearest Neighbors (KNNs), Support Vector Machine (SVM), and Gradient Boosting. The choice to compare against these approaches was made in an effort to provide a close comparison against the results presented in [11], which also considers shallow ML approaches exclusively. Both DNN models were implemented in *Pytorch* and trained using the PyTorch Lightning framework with early stopping based on the validation *monitor_metric*, e.g., validation loss. The Adam optimizer was used with an initial learning rate of 1e-3 and ReduceLROnPlateau scheduling with a learning rate patience of *learning_rate_patience* epochs.

6. Experimental Evaluation

The present section provides the experimental evaluation of the proposed approach and discusses the findings. Three experiments were conducted to evaluate the efficiency of the proposed framework for USB device identification and anomaly detection, using the datasets described in Section 3. In the first experiment, raw signals were fed directly to the alternative discriminative models. In the second, features were extracted via TsFresh in a way similar to [11]. The method yielded a total of 777 features. Finally, in the last experiment, a total of 128 features were obtained from the trained Autoencoder first.

We examined the power consumption signals of the various USB devices used in the experiments. Figure 8 shows the power consumption signals of a normal USB flash drive and a normal USB keyboard. Although both devices might look similar, on the surface, their power consumption signals exhibit distinct patterns. The flash drive signal has a lower amplitude, indicating a lower power consumption compared to the keyboard signal. In addition, the amplitude of the keyboard signal is more erratic, with frequent fluctuations, while the flash drive signal shows a stable power consumption pattern with minor fluctuations. This visual representation highlights the differences in power consumption signals between different types of USB devices.

Figures 9 and 10 show the power consumption signals of a normal USB device and a USB device under attack. As depicted in Figure 9, the normal keyboard device exhibits a stable power consumption pattern, while the malicious USB device (reprogrammed *teensyduino*) shows significant changes in the power consumption signal. The power consumption signal of the USB device under attack is characterized by large spikes and irregular patterns (highlighted in red), which are indicative of a malicious USB device. In contrast, in Figure 10, both the normal USB mouse and the malicious *O.MG* cable display a consistent power consumption pattern. However, the malicious device exhibits an increased amplitude in the power consumption signal (highlighted in red). These visual comparisons demonstrate the existence of subtle differences between normal and abnormal power consumption signals.

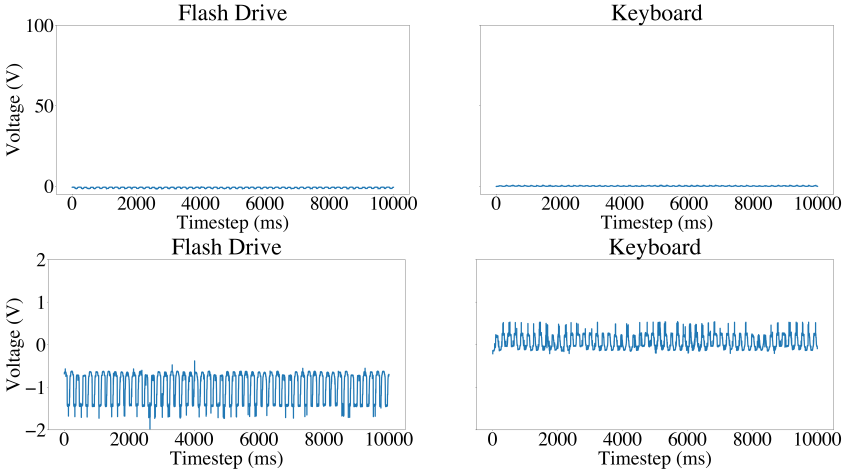


Figure 8. Power consumption signals of a normal flash drive device vs. a normal keyboard device. Zoom-out view on the **top** and zoom-in view on the **bottom**.

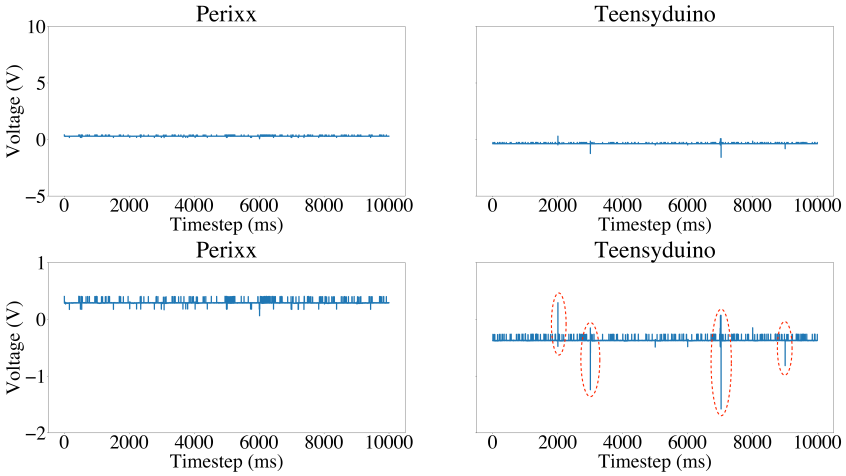


Figure 9. Power consumption signals of a normal USB keyboard device vs. the malicious Teensyduino masquerading as a keyboard. Zoom-out view on the **top** and zoom-in view on the **bottom**. Notice the spikes and irregular patterns in the Teensyduino signal highlighted in red.

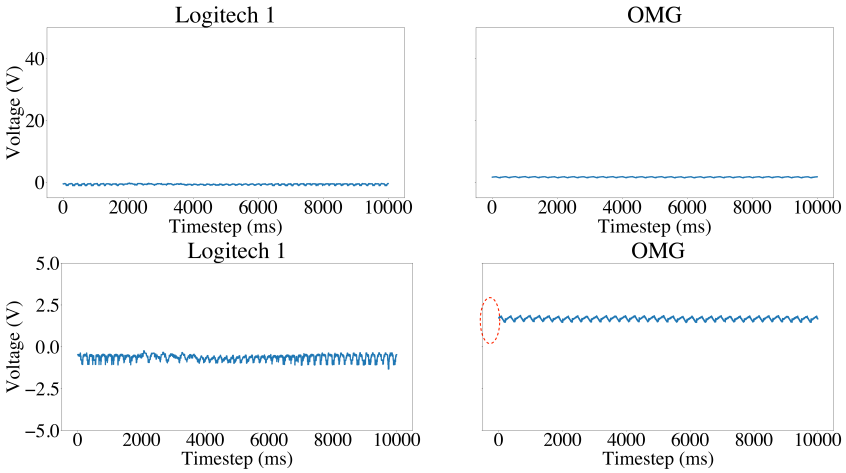


Figure 10. Power consumption signals of a normal USB mouse vs. the malicious O.MG cable. Zoom-out view on the **top** and zoom-in view on the **bottom**. Notice the increased amplitude in the O.MG cable signal highlighted in red.

6.1. Experiment 1: Evaluation of the Raw Power Consumption Signals

The evaluation in [11] indicates that shallow ML approaches can provide near-perfect accuracy for the tasks of both identification and authentication. Let us underline that the devices (and therefore the signals) considered in that work were different. Moreover, the quality of the signals in this work is much lower, as they are obtained by low-cost equipment. Driven by the hypothesis that DNN approaches may be more accurate and efficient [59], we decided to compare various shallow learning algorithms against the DNN models presented in Section 4.3 used in parallel or serial orientation.

For this experiment, we relied solely on the use of raw power consumption signals in an attempt to provide a baseline. We evaluated the models using a 10-fold cross-validation. The metrics considered were F1, precision, and recall. In addition, we reported the training and inference times. The results per dataset are summarized in Table 4.

The results show that both the ML and DNN models achieve high accuracy for the given tasks, while not requiring prohibitively expensive training. Interestingly, while the DNN model consistently achieves a perfect score (1.00) for all the considered metrics, the shallow ML models provide near-perfect accuracy for some tasks but perform poorly for others. Thus, not a single shallow ML model can reliably be adopted for a wide range of tasks. More specifically, as seen in Table 4, Random Forest achieves a precision, recall, and F1 score of 1.00 for USB device category classification, but it yields a score of 0.85 for the same metrics on the USB device brand classification task. An example of the poor performance for shallow classifiers is the results on dataset C1, where, on average, these classifiers achieve scores 0.69, 0.62, and 0.64 for the precision, recall, and F1 scores, respectively. Exceptionally, all the models achieve a perfect score for the task of anomaly detection (Dataset D2). This is not surprising, as the OMG cable exhibits a significantly different power consumption pattern compared to the other cables.

Table 4. Evaluation results per classifier. In this experiment, only raw signals were used (no feature extraction). Perfect scores are in bold.

Model	Precision	Recall	F1 Score	Accuracy	ROC AUC	Training Time (s)	Inference Time (ms)
Dataset A							
Decision Tree	0.88	0.88	0.88	0.88	0.92	0.121	0.21
Gradient Boosting	0.98	0.97	0.97	0.97	**1.00**	35.590	1.35
KNN	0.59	0.55	0.46	0.55	0.78	0.003	182.03
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.282	33.16
SVC	0.93	0.90	0.90	0.90	0.83	2.271	4.77
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	27.989	31.08
Dataset B							
Decision Tree	0.82	0.82	0.82	0.82	0.88	0.225	0.17
Gradient Boosting	0.80	0.80	0.80	0.80	0.94	83.639	1.39
KNN	0.83	0.75	0.73	0.75	0.97	0.003	69.06
Random Forest	0.85	0.85	0.85	0.85	0.97	0.305	27.86
SVC	0.93	0.90	0.90	0.90	0.85	2.416	6.01
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	46.093	30.11
Dataset C1							
Decision Tree	0.39	0.40	0.38	0.40	0.60	0.352	0.18
Gradient Boosting	0.52	0.53	0.52	0.53	0.74	53.956	1.39
KNN	0.33	0.45	0.38	0.45	0.78	0.003	226.60
Random Forest	0.55	0.53	0.53	0.53	0.78	0.275	30.91
SVC	0.69	0.62	0.64	0.62	0.62	5.004	7.13
Series CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	58.704	19.60

Table 4. *Cont.*

Model	Precision	Recall	F1 Score	Accuracy	ROC AUC	Training Time (s)	Inference Time (ms)
Dataset C2							
Decision Tree	0.53	0.60	0.50	0.60	0.73	0.146	0.22
Gradient Boosting	0.52	0.57	0.47	0.57	0.98	51.034	1.38
KNN	0.51	0.53	0.38	0.53	0.68	0.003	233.60
Random Forest	0.80	0.68	0.64	0.68	**1.00**	0.305	31.52
SVC	0.98	0.97	0.97	0.97	**1.00**	2.301	6.03
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	63.579	31.15
Dataset D1							
Decision Tree	0.65	0.72	0.67	0.72	0.52	0.438	0.18
Gradient Boosting	0.65	0.72	0.67	0.72	0.46	19.133	0.47
KNN	0.86	0.82	0.79	0.82	0.64	0.003	54.88
Random Forest	0.86	0.82	0.79	0.82	0.57	0.297	30.42
SVC	0.56	0.75	0.64	0.75	0.71	3.706	6.08
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	12.22	24.00
Dataset D2							
Decision Tree	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.118	0.18
Gradient Boosting	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	11.482	0.45
KNN	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.003	85.00
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.248	31.24
SVC	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.506	1.05
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	48.321	36.10

As expected, the major benefit of most of the shallow models considered in this experiment is that they are significantly faster to train and have a lower inference time. More specifically, except for the Gradient Boosting model, most shallow models were trained in less than 10 s. Not surprisingly, KNN had a minimum training time of 0.003 s for Dataset A. This is because KNN is a lazy classifier; thus, it does not construct a model during the training step, inducing a shorter time but requiring a longer test time (182.03 ms on Dataset A). Similarly, the Decision Tree yielded on average the second-best training time of 0.121 s on Dataset A and the best inference time of 0.17 ms on Dataset B. On the other hand, Gradient Boosting had the longest training time of 83.639 s, significantly slower than Parallel CNN-LSTM, which required 46.093 s for training on Dataset B. It is to be noted that still the considered shallow and DNN approaches have comparable inference times.
Takeaways: The experimental results suggest that both the shallow and DNN models require insignificant-to-reasonable training time and have high performance on all considered tasks. While the shallow models are fast to train, their performance is generally inconsistent. On the other hand, the DNN model consistently achieves a perfect score, although requiring a relatively longer time for training as expected. These results suggest that (a) the DNN model can indeed provide perfect accuracy with a reasonable training time, (b) after training, it is extremely efficient for inference, and (c) the considered CNN-LSTM model is suitable for all the tasks relevant to identification and authentication. Finally, it is possible that further preprocessing, e.g., feature extraction, might be beneficial and lead to minimizing the training time.

6.2. Experiment 2: Evaluation on the TsFresh Time Series Features

Previous work [11] has demonstrated that the time series features extracted by TSFresh can improve the ML models' performance for USB identification and anomaly detection tasks. Consequently, we expect the DNN models to benefit from the same feature extraction procedure. Moreover, we speculate that the inconsistencies in the performance of the shallow models observed in our first experiments might be addressed when trained on

the extracted features. Therefore, we conducted several experiments to test our hypothesis and compare the performance of the shallow and DNN models on the time series features extracted by TSFresh. We utilized the same models, presented in Section 5.4, in the evaluations using the same evaluation method and criteria as in the previous experiment. Table 5 summarizes the evaluation results.

The results of the experiments shown in Table 5 validate that the time series features extracted by TSFresh improve the shallow models' performance. At the same time, the performance of the DNN models remains consistently at perfect levels. Among the shallow models, the Random Forest and Decision Tree give perfect accuracy for all datasets except D2. Still, not a single shallow ML model exists that gives perfect results for all tasks. For example, KNN has an average F1 score of 1.00 for Dataset B, but 0.75 for Dataset D1. Worst, SVM has an average F1 score of 0.92 for Dataset A, but 0.64 for Dataset D2. On the contrary, the DNN model maintains its consistent perfect score of 1.00 on all the tasks' metrics.

Moreover, the results show an overall reduction in the training and inference times of the models, especially for the DNN. A striking example is the training time of the Gradient Boosting classifier, which reduced from 83.639 s to 14.563 s on Dataset B (a reduction factor of 5.7). Similarly, the training time of the Parallel CNN-LSTM model is reduced from 63.580 s to 33.164 s on Dataset C2 (a reduction factor of 1.9).

Takeaways: The results of the second experiment suggest that the time series features extracted by TSFresh significantly improve the performance of the models, with the DNN model maintaining its perfect accuracy on all tasks. The results also demonstrate an overall improvement in the training and inference times of the models, especially for the DNN ones. However, the feature extraction with TsFresh incurs a significant overhead in terms of time. These results suggest that further improvement in the feature extraction method may be beneficial and minimize the training time.

Table 5. Evaluation results using time series features. In this experiment, features extracted from TsFresh were used. Perfect scores are in bold.

Model	Precision	Recall	F1 Score	Accuracy	ROC AUC	Training Time (s)	Inference Time (ms)
Dataset A | Extraction Time: 365.90 s.							
Decision Tree	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.055	0.21
Gradient Boosting	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	11.797	1.32
KNN	0.98	0.97	0.97	0.97	**1.00**	0.002	2.09
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.258	32.81
SVC	0.94	0.93	0.92	0.93	0.86	0.129	0.38
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	69.907	6.49
Dataset B | Extraction Time: 359.70 s.							
Decision Tree	0.98	0.97	0.97	0.97	0.98	0.056	0.17
Gradient Boosting	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	14.563	1.32
KNN	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.002	2.05
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.307	28.84
SVC	0.93	0.90	0.90	0.90	0.92	0.131	0.44
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	180.055	12.92
Dataset C1 | Extraction Time: 359.82 s.							
Decision Tree	0.83	0.82	0.83	0.82	0.88	0.072	0.17
Gradient Boosting	0.93	0.93	0.92	0.93	**1.00**	32.410	1.33
KNN	0.86	0.85	0.85	0.85	0.96	0.002	2.08
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.272	30.49
SVC	0.82	0.70	0.65	0.70	0.19	0.347	0.56
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	28.326	12.20

Table 5. *Cont.*

Model	Precision	Recall	F1 Score	Accuracy	ROC AUC	Training Time (s)	Inference Time (ms)
Dataset C2 | Extraction Time: 360.22 s.							
Decision Tree	0.98	0.97	0.97	0.97	0.98	0.056	0.17
Gradient Boosting	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	11.889	1.33
KNN	0.92	0.90	0.90	0.90	0.97	0.002	2.25
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.286	31.08
SVC	0.79	0.75	0.75	0.75	0.15	0.345	0.56
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	33.164	14.20
Dataset D1 | Extraction Time: 358.14 s.							
Decision Tree	0.93	0.93	0.93	0.93	0.92	0.051	0.18
Gradient Boosting	0.98	0.97	0.97	0.97	**1.00**	5.703	0.45
KNN	0.84	0.80	0.75	0.80	0.64	0.003	2.61
Random Forest	0.95	0.95	0.95	0.95	**1.00**	0.287	32.68
SVC	0.56	0.75	0.64	0.75	0.71	0.155	0.43
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	58.533	7.70
Dataset D2 | Extraction Time: 381.79 s.							
Decision Tree	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.030	0.17
Gradient Boosting	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	3.007	0.47
KNN	0.98	0.97	0.98	0.97	**1.00**	0.003	2.90
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.245	28.80
SVC	0.56	0.75	0.64	0.75	0.99	0.177	0.42
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	42.268	10.60

6.3. Experiment 3: Evaluation of the Autoencoder Features

We hypothesize that the features extracted by an Autoencoder can further improve the performance of ML models, providing more consistent performance, while at the same time reducing their training and inference times. We performed additional experiments to validate these assumptions and compare the performance using the features extracted by the Autoencoder model presented in Section 5.4. The results of the experiments are given in Table 6.

From Table 6, we observe a more consistent and improved performance, especially for the shallow models. This specifically applies to the shallow models, which achieved a perfect score of 1.00 on Datasets A, C1, C2, and D2. Also, Random Forest, Gradient Boosting, and KNN produced an average precision, recall, and F1-score of 1.00 on all the datasets. Even in the worst case, Decision Tree and SVC yielded an average precision, recall, and F1-score above 0.95 on all the datasets. On the other hand, the DNN model maintained its perfect score of 1.00 on all tasks.

The results also show an overall improvement in the training and inference times of the models. For example, the training time of the Gradient Boosting classifier was reduced from 32.411 s to 5.615 s on Dataset C1, namely a reduction of 5.8. Similarly, the training time of the Parallel CNN-LSTM model was reduced from 58.533 s to 9.465 s on Dataset D1 (a factor of 6.2). Furthermore, the inference times of the models benefit from small reduction. For example, the inference time of KNN decreased from 2.08 s to 1.20 s on Dataset C1. Moreover, the inference time of the Parallel CNN-LSTM model had a reduction from 10.60 s to 8.50 s on Dataset D2.

Takeaways: The results regarding this experiment confirm that a complex feature extraction approach, akin to the DNN method, is paramount to providing improved and consistent performance for both shallow ML and DNN models. At the same time, the reduced number of features contributes to a drastic reduction in the training and inference time. Overall,

the Autoencoder feature extraction method has proven to be beneficial in providing a concise and rich set of features for training and inference.

6.4. Discussion

The features considered in the experiments can influence the training and inference times, as well as the performance of the models. However, as suggested by the second and third experiments, not all feature extraction methods have the same impact. As observed from Figures 11 and 12, unsurprisingly, the training and inference times of the models on the raw features are the longest and are followed by the TsFresh method. The figures also clearly indicate that while feature extraction with TsFresh reduces the number of features, and consequently the training and test time, it incurs a significant overhead in terms of extraction time. On the other hand, the feature extraction through Autoencoder yields superior results, as it reduces significantly the training and inference time for almost all shallow ML methods and for all datasets. An exception to that rule is the Random Forest algorithm. Apart from the efficiency in terms of speed, the predictive performance of the models significantly also improves when trained on the Autoencoder-extracted features. In particular, the ML models achieved near-perfect F1 scores on all tasks compared to the other methods as seen in Figure 13. Specifically, the Random Forest achieved a ROC-AUC of 0.64 on both the raw and TsFresh features but a score of 1.00 on the Autoencoder-extracted features. Despite that, comparing the best performing ML classifier, namely the Random Forest, to the DNN model as shown in Figure 14 and Figures A1–A6 clearly state the DNN model's superior performance.

Table 6. Evaluation results using features extracted by the Autoencoder. Perfect scores are in bold.

Model	Precision	Recall	F1 Score	Accuracy	ROC AUC	Training Time (s)	Inference Time (ms)
Dataset A | Extraction Time: 1.35 s.							
Decision Tree	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.011	0.16
Gradient Boosting	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	2.412	1.32
KNN	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.001	1.83
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.260	33.12
SVC	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.011	0.24
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	28.154	13.30
Dataset B | Extraction Time: 1.33 s.							
Decision Tree	0.95	0.95	0.95	0.95	0.97	0.013	0.17
Gradient Boosting	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	3.472	1.28
KNN	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.001	1.20
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.307	31.18
SVC	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.018	0.25
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	29.867	15.90
Dataset C1 | Extraction Time: 1.41 s.							
Decision Tree	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.011	0.16
Gradient Boosting	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	5.615	1.29
KNN	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.001	1.20
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.257	31.81
SVC	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.015	0.24
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	32.926	14.90
Dataset C2 | Extraction Time: 1.44 s.							
Decision Tree	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.011	0.16
Gradient Boosting	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	2.413	1.32
KNN	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.001	1.20
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.245	29.62
SVC	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.014	0.22
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	42.025	15.50

Table 6. *Cont.*

Model	Precision	Recall	F1 Score	Accuracy	ROC AUC	Training Time (s)	Inference Time (ms)
Dataset D1 | Extraction Time: 1.33 s.							
Decision Tree	0.95	0.95	0.95	0.95	0.90	0.007	0.17
Gradient Boosting	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.644	0.45
KNN	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.001	1.86
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.276	30.89
SVC	0.98	0.97	0.97	0.97	**1.00**	0.010	0.23
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	9.465	9.60
Dataset D2 | Extraction Time: 1.33 s.							
Decision Tree	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.007	0.23
Gradient Boosting	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.631	0.45
KNN	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.001	1.32
Random Forest	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.279	28.80
SVC	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**		
Parallel CNN-LSTM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	13.621	8.50

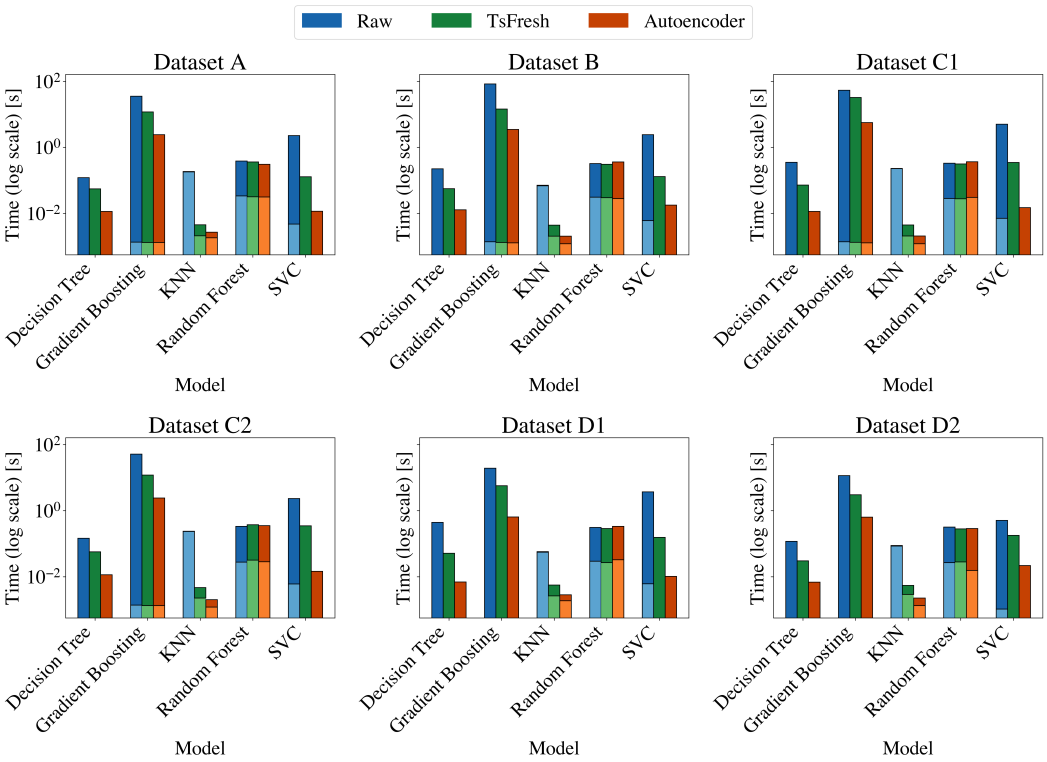


Figure 11. Comparison of the impact of the feature extraction methods on the training times (darker color) and inference times (lighter color) of the shallow models.

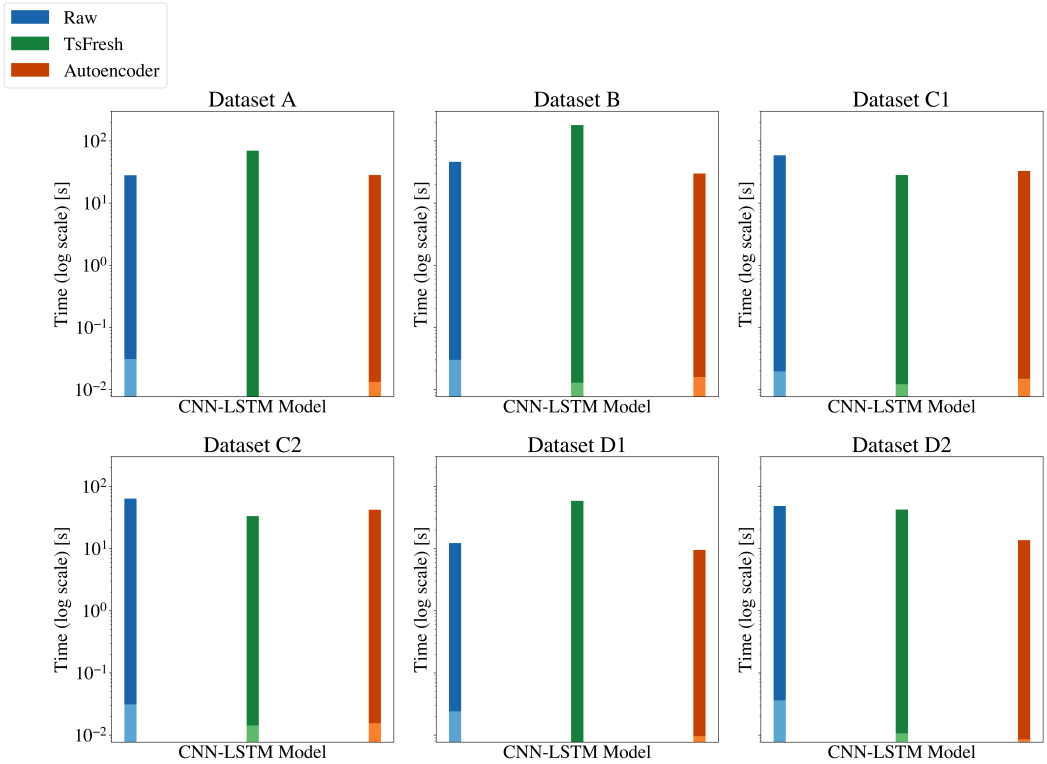


Figure 12. Comparison of the impact of the feature extraction methods on the training times (darker color) and inference times (lighter color) of the CNN-LSTM model.

The experimental datasets used in this research are balanced, except for the two anomaly detection datasets (D1 and D2). Given that some datasets are imbalanced, we opted to use the F1 score as the primary evaluation metric. The F1 score is particularly useful in scenarios where datasets are imbalanced, as it balances precision and recall [61]. When considering a shallow learning model, Random Forest showed itself to be superior in terms of model complexity and 10-fold cross-validation F1 score performance. Conversely, deep learning models were best represented by the Parallel CNN-LSTM model, which surpassed other models in accuracy, precision, recall, and F1-score. Which model is optimal largely hinges on the needs of the USB authentication system. Factors include desired accuracy levels, available computational resources, and model complexity. Through our research, we demonstrated that Parallel CNN-LSTM is a very promising model for USB authentication, given its perfect F1-score on all testing datasets and generally lower requirement for training epochs. However, in situations where computational resources are sparse, the Random Forest model could serve as an excellent substitute. Despite a somewhat lower F1 score, it demands fewer computational resources, therefore exhibiting a faster training speed than CNN-LSTM.

Model selection involves choosing the most suitable model from a pool of candidate models based on certain metrics or selection criteria [62]. Typically, this selection could be guided by resampling techniques or probability-based measures. In the case of probability-based measures, the choice is influenced by model complexity and performance on training data, as seen in methods like Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). Resampling methods, on the other hand, propose model selection based on out-of-sample data, utilizing strategies such as cross-validation and bootstrapping. Affirmatively, we utilized the F1 score on 10-fold cross-validation data as our core selection criteria. The F1 score, reflecting a model's accuracy on a dataset, is computed as the harmonic mean of precision and recall. Its key advantage over straightforward accuracy is its consideration of both false positives and false negatives. This is particularly beneficial in situations with imbalanced datasets—a scenario that was often encountered in our experiments.

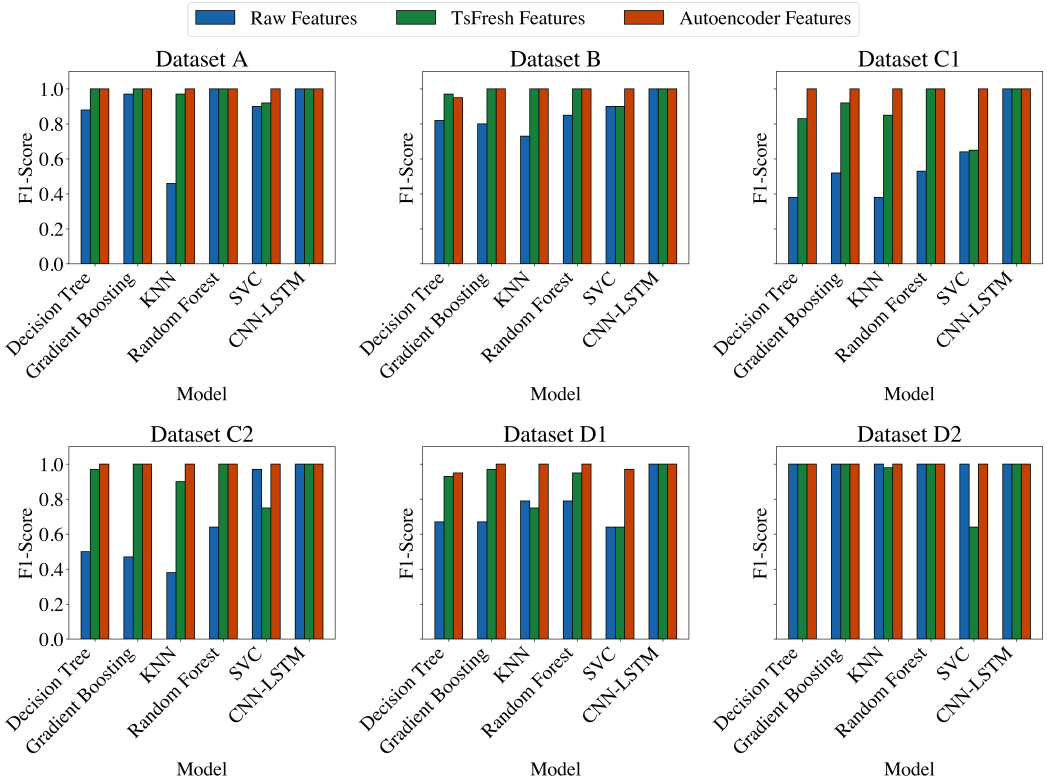


Figure 13. Shallow vs. DDN models in terms of F1 score for raw signals, TSFresh-extracted features, and Autoencoder-extracted features.

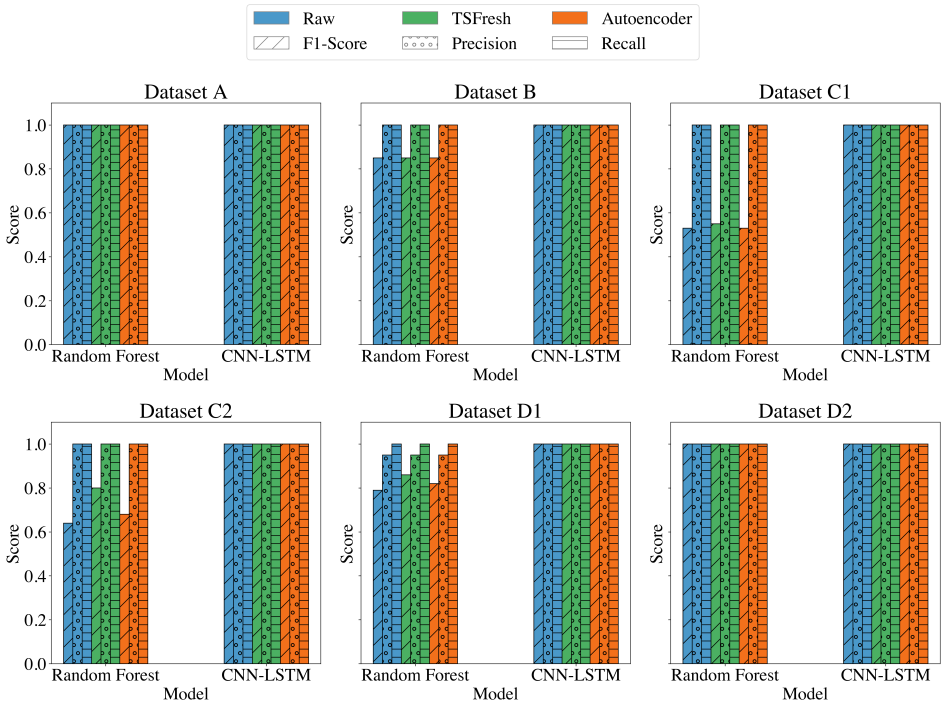


Figure 14. Random Forest vs. CNN-LSTM for raw signals, features extracted by TSFresh, and features extracted by Autoencoder.

7. Conclusions

This work examines the potential of the power consumption side channel as a tool for identification and authentication mechanisms for USB peripherals. We have proven

experimentally that relatively noisy data obtained with inexpensive off-the-shell equipment can be utilized to accurately identify/distinguish individual devices, even those produced by the same manufacturers. At the same time, the same framework can be applied to provide very accurate authentication of USB peripherals. This can potentially protect against a wide range of malicious USB peripherals and USB attacks. More specifically, we have shown that our Parallel and Series LSTM-CNN models, when examining raw signals, can provide significantly more accurate discrimination than any shallow ML classifiers. However, this comes at the cost of a considerable (yet not prohibitive) increase in training and inference time.

We performed an experiment with shallow learning methods and the Parallel LSTM-CNN model on the features extracted with the Autoencoder. The derived results show a slower training time for the DNN model but better overall results compared to when trained on the raw signals. Similarly, the DNN model has the highest overall average performance on the features extracted with the Autoencoder. Although the DNN training time is longer than that of the shallow learning models, it is less than 1 min in all cases.

We performed a comparative set of experiments with the features extracted by both the Autoencoder and those extracted by the feature extraction library TsFresh. Based on the results, the TsFresh feature extraction takes a significant amount of time compared to the feature extraction with the DNN. In particular, the overall average extraction time for the TsFresh is around 6 min, while the Autoencoder feature extraction takes around a second (a speedup of 60×). Furthermore, the performance of the best-performing ML model trained with the Tsfresh feature extraction is comparable to the DNN model, with the DNN having an overall higher average F1 score.

Future work will further explore the reduction in the size of the DNN models while maintaining high accuracy to enable the efficient hardware implementation of the proposed framework. This may potentially be achieved by examining the use of quantization and pruning techniques to reduce the size of the models for prototype implementation on low-power devices.

Author Contributions: Conceptualization, C.K.; Methodology, K.A.K., C.S. and C.K.; Software, K.A.K.; Validation, C.S. and G.K.; Investigation, C.S., C.K. and G.K.; Writing—original draft, K.A.K., C.S. and C.K.; Writing—review & editing, C.K. and G.K.; Supervision, C.K. and G.K.; Project administration, C.K. and G.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All data used in this article can be retrieved from here: https://github.com/kandersonko/usb_side_channel_datasets. All scripts used in this paper can be retrieved from here: https://github.com/kandersonko/usb_side_channel.

Conflicts of Interest: The authors declare no conflict of interest.

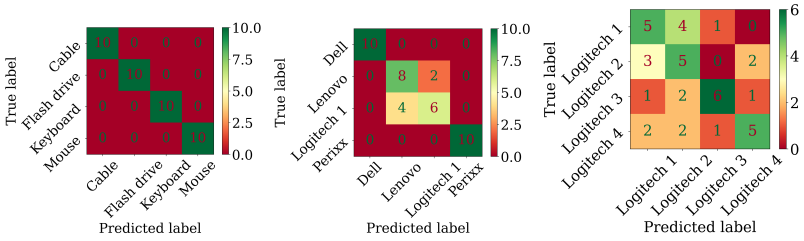
Abbreviations

The following abbreviations are used in this manuscript:

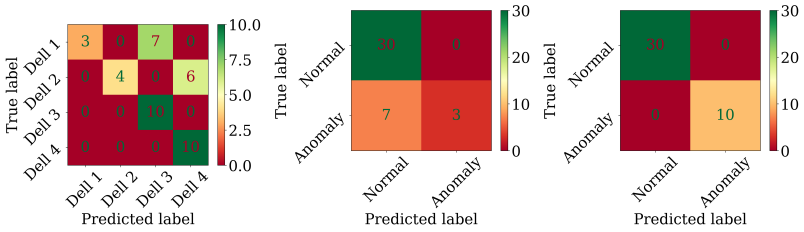
USB	Universal Serial Bus
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory
DNN	Deep Neural Network
ML	Machine Learning
KNN	K-Nearest Neighbors
SVC	Support Vector Classifier
SVM	Support Vector Machine
ROC	Receiver Operating Characteristic
AUC	Area Under the Curve
VBUS	Voltage Bus
GND	Ground
D+	Data Plus

D−	Data Minus
CRC	Cyclic Redundancy Check
SOF	Start of Frame
SOP	Start of Packet
EOP	End of Packet
PID	Packet Identifier
ADDR	Address
SYNC	Synchronization
ENDP	End Point
ACK	Acknowledgment
OHCI	Open Host Controller Interface
UHCI	Universal Host Controller Interface
EHCI	Enhanced Host Controller Interface
XHCI	eXtensible Host Controller Interface
WHCI	Wireless Host Controller Interface
PCI	Peripheral Component Interconnect

Appendix A. Random Forest and CNN-LSTM Confusion Matrices

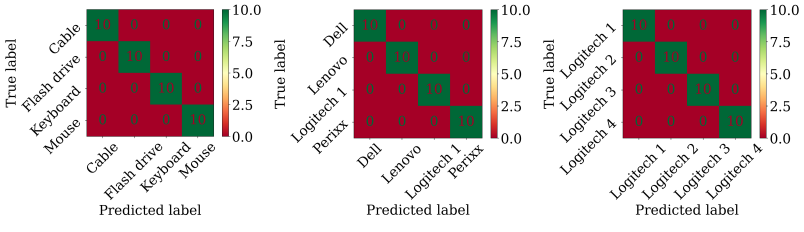


(**a**) Dataset A (**b**) Dataset B (**c**) Dataset C1

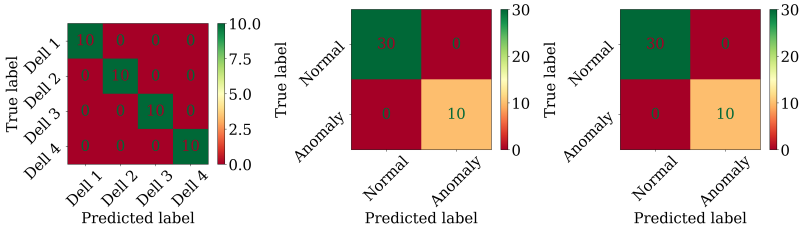


(**d**) Dataset C2 (**e**) Dataset D1 (**f**) Dataset D2

Figure A1. Random Forest confusion matrices on the tasks using the raw signals.

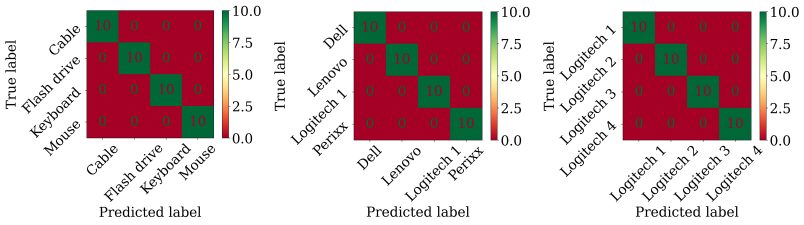


(**a**) Dataset A (**b**) Dataset B (**c**) Dataset C1

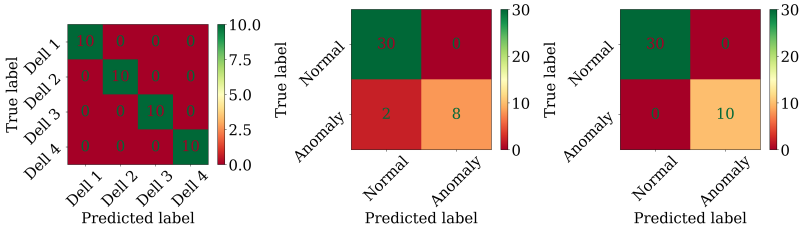


(**d**) Dataset C2 (**e**) Dataset D1 (**f**) Dataset D2

Figure A2. CNN-LSTM confusion matrices on the tasks using the raw signals.

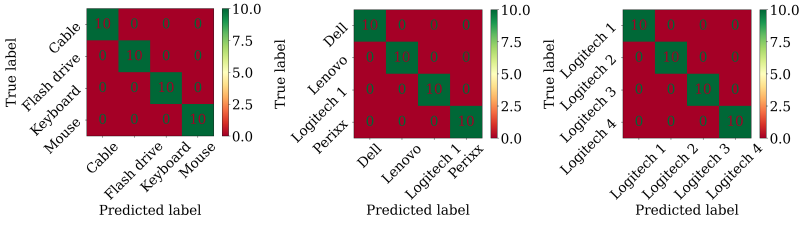


(**a**) Dataset A (**b**) Dataset B (**c**) Dataset C1

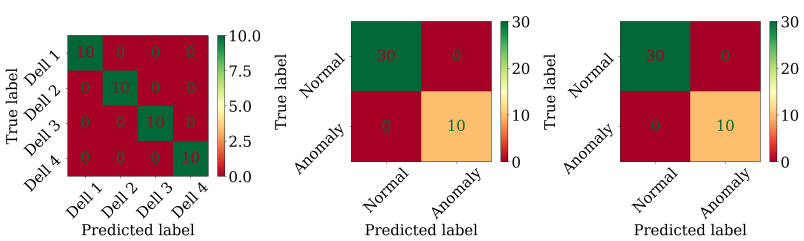


(**d**) Dataset C2 (**e**) Dataset D1 (**f**) Dataset D2

Figure A3. Random Forest confusion matrices for TSFresh-extracted features.

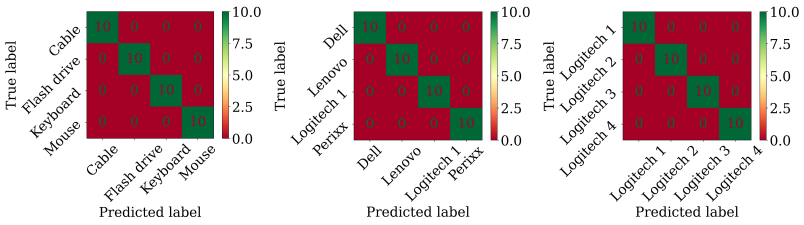


(**a**) Dataset A (**b**) Dataset B (**c**) Dataset C1

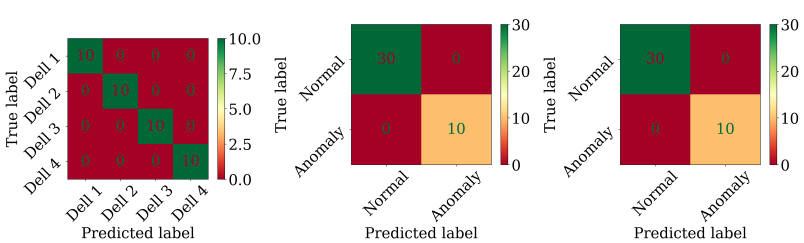


(**d**) Dataset C2 (**e**) Dataset D1 (**f**) Dataset D2

Figure A4. CNN-LSTM confusion matrices for TSFresh-extracted features.

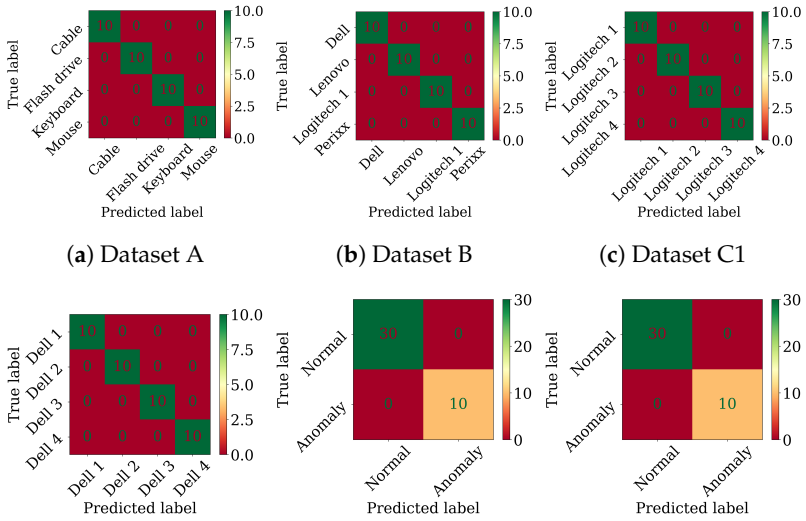


(**a**) Dataset A (**b**) Dataset B (**c**) Dataset C1



(**d**) Dataset C2 (**e**) Dataset D1 (**f**) Dataset D2

Figure A5. Confusion matrices of the Random Forest classifier on the tasks using the features extracted by the Autoencoder.



(**a**) Dataset A (**b**) Dataset B (**c**) Dataset C1

(**d**) Dataset C2 (**e**) Dataset D1 (**f**) Dataset D2

Figure A6. CNN-LSTM confusion matrices on the tasks using the features extracted by the Autoencoder.

References

1. Global USB 3.0 Market to Reach $6.3 Billion by 2027—ResearchAndMarkets.Com. 2020. Available online: https://www.businesswire.com/news/home/20201208005699/en/Global-USB-3.0-Market-to-Reach-6.3-Billion-by-2027---ResearchAndMarkets.com (accessed on 28 March 2024).
2. Cybersecurity USB Threat Report 2021. Available online: https://www.honeywellforge.ai/us/en/campaigns/cybersecurity-threat-report-2021 (accessed on 28 March 2024).
3. Karnouskos, S. Stuxnet worm impact on industrial cyber-physical system security. In Proceedings of the IECON 2011—37th Annual Conference of the IEEE Industrial Electronics Society, Melbourne, Australia, 7–10 November 2011; pp. 4490–4494.
4. Tischer, M.; Durumeric, Z.; Foster, S.; Duan, S.; Mori, A.; Bursztein, E.; Bailey, M. Users Really Do Plug in USB Drives They Find. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2016; pp. 306–319. ISSN: 2375-1207. [CrossRef]
5. Javed Butt, U.; Abbod, M.; Lors, A.; Jahankhani, H.; Jamal, A.; Kumar, A. Ransomware Threat and its Impact on SCADA. In Proceedings of the 2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3), London, UK, 16–18 January 2019; pp. 205–212. [CrossRef]
6. Faife, C. The O.MG Elite Cable Is a Scarily Stealthy Hacker Tool. 2022. Available online: https://www.theverge.com/23321517/omg-elite-cable-hacker-tool-review-defcon (accessed on 2 April 2024).
7. Lu, H.; Wu, Y.; Li, S.; Lin, Y.; Zhang, C.; Zhang, F. BADUSB-C: Revisiting BadUSB with Type-C. In Proceedings of the 2021 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 27 May 2021; pp. 327–338. [CrossRef]
8. Guri, M.; Monitz, M.; Elovici, Y. USBee: Air-gap covert-channel via electromagnetic emission from USB. In Proceedings of the 2016 14th Annual Conference on Privacy, Security and Trust (PST), Auckland, New Zealand, 12–14 December 2016; pp. 264–268. [CrossRef]
9. Randolph, M.; Diehl, W. Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman. *Cryptography* **2020**, *4*, 15. [CrossRef]
10. Yang, Q.; Gasti, P.; Zhou, G.; Farajidavar, A.; Balagani, K.S. On Inferring Browsing Activity on Smartphones via USB Power Analysis Side-Channel. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 1056–1066. [CrossRef]
11. Spolaor, R.; Liu, H.; Turrin, F.; Conti, M.; Cheng, X. Plug and Power: Fingerprinting USB Powered Peripherals via Power Side-channel. In Proceedings of the IEEE INFOCOM 2023—IEEE Conference on Computer Communications, New York, NY, USA, 17–20 May 2023; pp. 1–10. ISSN: 2641-9874. [CrossRef]
12. Nissim, N.; Yahalom, R.; Elovici, Y. USB-based attacks. *Comput. Secur.* **2017**, *70*, 675–688. [CrossRef]
13. Mills, M. How a Rubber Ducky Works and Why It Is So Dangerous | ITIGIC. 2021. Available online: https://itigic.com/how-a-rubber-ducky-works-and-why-it-is-so-dangerous/ (accessed on 28 March 2024).
14. EvilDuino | PPT. Available online: https://www.slideshare.net/Rashidferoz1/evilduino (accessed on 28 March 2024).
15. Samy Kamkar—USBdriveby: Exploiting USB in Style. Available online: https://samy.pl/usbdriveby/ (accessed on 28 March 2024).
16. Karystinos, E.; Andreatos, A.; Douligeris, C. Spyduino: Arduino as a HID Exploiting the BadUSB Vulnerability. In Proceedings of the 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), Santorini Island, Greece, 29–31 May 2019; pp. 279–283. [CrossRef]

17. Lamont, J. This Normal-Looking Cable Actually Helps Steal Data off Your Phone. 2021. Available online: https://mobilesyrup.com/2021/09/03/normal-looking-cable-steal-data-phone-omg-cable/ (accessed on 1 May 2024).
18. Introducing the 'O.MG Cable' That Sends Everything You Type in with the Keyboard to the Outside via Wi-Fi Even Though It Looks like a Normal USB Cable. 2021. Available online: http://gigazine.net/gsc_news/en/20210903-o-mg-cable-leak-key-type/ (accessed on 1 May 2024).
19. Caudill, A. Making BadUSB Work for You—DerbyCon. 2014. Available online: https://adamcaudill.com/2014/10/02/making-badusb-work-for-you-derbycon/ (accessed on 30 April 2024).
20. Maskiewicz, J.; Ellis, B.; Mouradian, J.; Shacham, H. Mouse trap: Exploiting firmware updates in USB peripherals. In Proceedings of the 8th USENIX Conference on Offensive Technologies, San Diego, CA, USA, 19 August 2014; p. 12.
21. Kali NetHunter | Kali Linux Documentation. Available online: https://www.kali.org/docs/nethunter/ (accessed on 28 March 2024).
22. USB Kill Devices for Pentesting & Law-Enforcement. Available online: https://usbkill.com/ (accessed on 28 March 2024).
23. Cyber Security Kiosk—MetaDefender Kiosk. Available online: https://www.opswat.com/products/metadefender/kiosk (accessed on 28 March 2024).
24. Frank. Cybersecurity & Kiosks: Olea's Protective Approach. 2019. Available online: https://www.olea.com/news/kiosks-help-ward-off-cybersecurity-threats/ (accessed on 28 March 2024).
25. IoT in the Age of Everything Connected. Available online: https://symantec-enterprise-blogs.security.com/blogs/product-insights/iot-age-everything-connected (accessed on 28 March 2024).
26. Yang, B.; Qin, Y.; Zhang, Y.; Wang, W.; Feng, D. TMSUI: A Trust Management Scheme of USB Storage Devices for Industrial Control Systems. In *Information and Communications Security*; Qing, S., Okamoto, E., Kim, K., Liu, D., Eds.; Springer: Cham, Switzerland, 2016; pp. 152–168. [CrossRef]
27. Lee, C.C.; Chen, C.T.; Wu, P.H.; Chen, T.Y. Three-factor control protocol based on elliptic curve cryptosystem for universal serial bus mass storage devices. *IET Comput. Digit. Tech.* **2013**, *7*, 48–55. [CrossRef]
28. Loe, E.L.; Hsiao, H.C.; Kim, T.H.J.; Lee, S.C.; Cheng, S.M. SandUSB: An installation-free sandbox for USB peripherals. In Proceedings of the 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, USA, 12–14 December 2016; pp. 621–626. [CrossRef]
29. Home | USBGuard. Available online: https://usbguard.github.io/ (accessed on 28 March 2024).
30. Denney, K.; Babun, L.; Uluagac, A.S. USB-Watch: A Generalized Hardware-Assisted Insider Threat Detection Framework. *J. Hardw. Syst. Secur.* **2020**, *4*, 136–149. [CrossRef]
31. Tian, D.J.; Bates, A.; Butler, K. Defending Against Malicious USB Firmware with GoodUSB. In Proceedings of the 31st Annual Computer Security Applications Conference, New York, NY, USA, 7–11 December 2015; pp. 261–270. [CrossRef]
32. Tian, D.J.; Scaife, N.; Bates, A.; Butler, K.; Traynor, P. *Making {USB} Great Again with {USBFILTER}*; In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10-12 August 2016; pp. 415–430. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tian
33. Ebad, S.A. Lessons Learned from Offline Assessment of Security-Critical Systems: The Case of Microsoft's Active Directory. *Int. J. Syst. Assur. Eng. Manag.* **2022**, *13*, 535–545. [CrossRef]
34. Murphy, R.; Family, A.P. USB 101: An introduction to universal serial bus 2.0. **2014**, *1*, 25–34. Available online: http://kofa.mmto.arizona.edu/stm32/blue_pill/usb/an57294.pdf (accessed on 14 May 2024)
35. USB 2.0 Specification | USB-IF. Available online: https://www.usb.org/document-library/usb-20-specification (accessed on 13 March 2024).
36. USB 3.2 Revision 1.1—June 2022 | USB-IF. Available online: https://www.usb.org/document-library/usb-32-revision-11-june-2022 (accessed on 16 March 2024).
37. Tian, J.; Scaife, N.; Kumar, D.; Bailey, M.; Bates, A.; Butler, K. SoK: "Plug & Pray" Today – Understanding USB Insecurity in Versions 1 Through C. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–23 May 2018; pp. 1032–1047. ISSN: 2375-1207. [CrossRef]
38. Verma, A.; Dahiya, P.K. Pcie bus: A state-of-the-art-review. *IOSR J. VLSI Signal Process. (IOSR-JVSP)* **2017**, *7*, 24–28. [CrossRef]
39. Ibrahim, O.A.; Sciancalepore, S.; Oligeri, G.; Pietro, R.D. MAGNETO: Fingerprinting USB Flash Drives via Unintentional Magnetic Emissions. *ACM Trans. Embed. Comput. Syst.* **2020**, *20*, 8:1–8:26. [CrossRef]
40. Sayakkara, A.; Le-Khac, N.A.; Scanlon, M. Leveraging Electromagnetic Side-Channel Analysis for the Investigation of IoT Devices. *Digit. Investig.* **2019**, *29*, S94–S103. [CrossRef]
41. Zhang, J.; Chen, C.; Cui, J.; Li, K. Timing Side-Channel Attacks and Countermeasures in CPU Microarchitectures. *ACM Comput. Surv.* 2024, *Just Accepted*. [CrossRef]
42. Taheritajar, A.; Harris, Z.M.; Rahaeimehr, R. A Survey on Acoustic Side Channel Attacks on Keyboards. *arXiv* **2023**, arXiv:2309.11012 [cs].
43. Hutter, M.; Schmidt, J.M. The Temperature Side Channel and Heating Fault Attacks. In *Smart Card Research and Advanced Applications*; Francillon, A., Rohatgi, P., Eds.; Springer: Cham, Switzerland, 2014; pp. 219–235. [CrossRef]
44. Karimi, E.; Jiang, Z.H.; Fei, Y.; Kaeli, D. A Timing Side-Channel Attack on a Mobile GPU. In Proceedings of the 2018 IEEE 36th International Conference on Computer Design (ICCD), Orlando, FL, USA, 7–10 October 2018; pp. 67–74. ISSN: 2576-6996. [CrossRef]

45. Wang, C.; Yan, M.; Cai, Y.; Zhou, Q.; Yang, J. Power Profile Equalizer: A Lightweight Countermeasure against Side-Channel Attack. In Proceedings of the 2017 IEEE International Conference on Computer Design (ICCD), Boston, MA, USA, 5–8 November 2017; pp. 305–312. ISSN: 1063-6404. [CrossRef]
46. Song, R.; Song, Y.; Gao, S.; Xiao, B.; Hu, A. I Know What You Type: Leaking User Privacy via Novel Frequency-Based Side-Channel Attacks. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6. ISSN: 2576-6813. [CrossRef]
47. Fei, Y.; Ding, A.A.; Lao, J.; Zhang, L. A Statistics-based Fundamental Model for Side-channel Attack Analysis, 2014. Cryptology ePrint Archive Paper 2014/152, 1 March 2014, Available online: https://eprint.iacr.org/2014/152 (accessed on 14 May 2024).
48. Picek, S.; Heuser, A.; Jovic, A.; Ludwig, S.A.; Guilley, S.; Jakobovic, D.; Mentens, N. Side-channel analysis and machine learning: A practical perspective. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; pp. 4095–4102. ISSN: 2161-4407. [CrossRef]
49. Alom, M.Z.; Taha, T.M.; Yakopcic, C.; Westberg, S.; Sidike, P.; Nasrin, M.S.; Hasan, M.; Van Essen, B.C.; Awwal, A.A.S.; Asari, V.K. A State-of-the-Art Survey on Deep Learning Theory and Architectures. *Electronics* **2019**, *8*, 292. [CrossRef]
50. Bank, D.; Koenigstein, N.; Giryes, R. Autoencoders. In *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*; Rokach, L., Maimon, O., Shmueli, E., Eds.; Springer International Publishing: Cham, Switzerland, 2023; pp. 353–374. [CrossRef]
51. Wang, W.; Huang, Y.; Wang, Y.; Wang, L. Generalized Autoencoder: A Neural Network Framework for Dimensionality Reduction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Columbus, OH, USA, 23–28 June 2014; pp. 490–497.
52. Li, Z.; Liu, F.; Yang, W.; Peng, S.; Zhou, J. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *33*, 6999–7019. [CrossRef] [PubMed]
53. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
54. Essien, A.; Giannetti, C. A Deep Learning Model for Smart Manufacturing Using Convolutional LSTM Neural Network Autoencoders. *IEEE Trans. Ind. Inform.* **2020**, *16*, 6069–6078. [CrossRef]
55. Bahdanau, D.; Cho, K.; Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv* **2016**, [CrossRef]
56. Pouyanfar, S.; Sadiq, S.; Yan, Y.; Tian, H.; Tao, Y.; Reyes, M.P.; Shyu, M.L.; Chen, S.C.; Iyengar, S.S. A Survey on Deep Learning: Algorithms, Techniques, and Applications. *ACM Comput. Surv.* **2019**, *51*, 1–36. [CrossRef]
57. Li, J.; Wang, J.; Tian, Q.; Gao, W.; Zhang, S. Global-Local Temporal Representations for Video Person Re-Identification. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019.
58. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1026–1034.
59. Dong, S.; Wang, P.; Abbas, K. A survey on deep learning and its applications. *Comput. Sci. Rev.* **2021**, *40*, 100379. [CrossRef]
60. PC Oscilloscope, Data Logger & RF Products | Pico Technology. 2024. Available online: https://www.picotech.com/ (accessed on 28 March 2024).
61. Wardhani, N.W.S.; Rochayani, M.Y.; Iriany, A.; Sulistyono, A.D.; Lestantyo, P. Cross-Validation Metrics for Evaluating Classification Performance on Imbalanced Data. In Proceedings of the 2019 International Conference on Computer, Control, Informatics and Its Applications (IC3INA), Tangerang, Indonesia, 23–24 October 2019; pp. 14–18. [CrossRef]
62. Ding, J.; Tarokh, V.; Yang, Y. Model Selection Techniques: An Overview. *IEEE Signal Process. Mag.* **2018**, *35*, 16–34. [CrossRef]