

Article

Integrating Side-Channel Power Signals and Network Traffic for Machine Learning-Based Intrusion Detection in IoT

Felipe Lemus-Prieto ^{1,*} , Alejandro Domínguez Campos ¹, José-Luis González-Sánchez ² 
and Andrés Caro Lindo ² 

¹ COMPUTAEX Foundation, Extremadura Supercomputing Center, 10071 Cáceres, Spain; alejandro.dominguez@computaex.es

² Departamento de Ingeniería de Sistemas Informáticos y Telemáticos, Universidad de Extremadura, Escuela Politécnica, 10003 Cáceres, Spain; jlgs@unex.es (J.-L.G.-S.); andresc@unex.es (A.C.L.)

* Correspondence: felipe.lemus@computaex.es; Tel.: +34-927-049-070

Abstract

The Internet of Things (IoT) is rapidly being integrated into critical infrastructure sectors, such as energy, transportation, healthcare, and industry. This surge of interconnected devices dramatically expands the attack surface and increases the risk of cascading system failures and data breaches. To address these emerging threats, this work proposes an intrusion detection system (IDS) for IoT networks that incorporates machine learning techniques, considering side-channel (power) and network traffic features. We collected power consumption traces and network metrics from IoT devices during normal operation and under diverse cyberattacks (e.g., cryptomining, flooding, port scanning). Time-series machine learning classifiers are trained on this hybrid dataset to differentiate benign versus malicious behavior. The experimental results show that the combined-feature model significantly outperforms models using only one data type, achieving high detection accuracy ($F1 \approx 0.89$) and correctly identifying the attack type. The resulting IDS generalizes to previously unseen attacks, demonstrating robust, adaptive defense capabilities. The novelty of our approach lies in integrating physical side-channel signals into an automated ML framework, enhancing robustness and resilience. This smart, data-driven solution operates in near real time and helps build autonomous, constantly evolving defenses against cyber threats. Overall, our study delivers a state-of-the-art ML-based tool that learns and evolves to counter modern IoT cyberattacks.

Keywords: cybersecurity; intrusion detection system (IDS); Internet of Things (IoT); machine learning; side-channel analysis; network traffic analysis; anomaly detection



Academic Editors: Eduard Popovici,
Dinu Țurcanu, Cătălin Căleanu and
Mircea Scheau

Received: 11 June 2026

Revised: 10 July 2026

Accepted: 13 July 2026

Published: 15 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

The proliferation of the Internet of Things (IoT) has led to the massive interconnection of devices: it is estimated that there are already more than 21 billion connected IoT devices worldwide [1].

Key sectors such as energy, transportation, and healthcare are increasingly incorporating sensors and embedded systems to optimize their efficiency. In the United States, for example, essential services of such key sectors currently rely on connected devices [2]. This massive deployment improves automation and real-time monitoring, but at the same time, it exponentially expands the cyberattack surface: the more connected devices there are, the greater the likelihood of vulnerabilities and cascading failures.

The integration of IoT into critical infrastructure has brought enormous risks. The literature indicates that this widespread adoption of technology introduces new threats to the reliability, security, and integrity of systems [3]. Many IoT devices have limited resources and lack robust protection mechanisms, making them easy for attackers to exploit. In practice, the vulnerability of these devices in sectors such as energy, healthcare, or transportation can cause serious service disruptions. This context demands the adoption of advanced defense mechanisms capable of automatically identifying and neutralizing intrusions.

To mitigate these risks, automated intrusion detection systems (IDSs) based on artificial intelligence have been developed, and they are trained to recognize anomalous behavior in IoT networks. However, most current approaches focus exclusively on the analysis of network traffic and digital events, and they fail to take into account an important source of information: device power consumption. This power signal constitutes a side channel that reflects the internal activity of an IoT device. By incorporating the analysis of this channel alongside network metrics, we could enhance anomaly detection with information extracted directly from the physical behavior of the hardware.

This work proposes a hybrid approach: an IDS that fuses network traffic features with side-channel data (power consumption) in IoT environments. The goal is to leverage the complementarity of both sources to build a more robust and accurate detection model. With this in mind, we pose the following as central research questions:

- RQ1: Can an integrated framework combining power consumption measurements and network traffic information be developed to identify cyberattacks targeting IoT devices?
- RQ2: Is the framework capable of differentiating between legitimate activity and malicious behavior while also accurately classifying the specific attack category?
- RQ3: Can the framework recognize previously unseen attacks that were not represented in the training dataset?
- RQ4: What set of network features would be best to use to maximize the accuracy of the framework?
- RQ5: Does combining network features and power reads improve the accuracy of the framework over using just power reads?
- RQ6: Does using power reads alongside network features improve the accuracy of the framework over using just network features?
- RQ7: Is it possible to identify attacks in real time while maintaining an acceptable detection latency?

The primary contributions of this study are as follows:

1. A new approach to hardening IoT environments based on a hybrid solution that combines network traffic data and device power consumption data to detect and classify cyberattacks with greater accuracy.
2. A transferable methodology that enables the design and development of a security system tailored to the specific characteristics of each IoT environment.
3. A publicly accessible dataset that combines network traffic data with the power consumption of IoT devices; this dataset has not been generated synthetically but rather by setting up a controlled real-world IoT environment.

This paper is structured as follows: Section 2 presents the existing literature in this field. Section 3 introduces the methodology followed in this work, including the designed hardware setup, the behavior of the system, the chosen classification algorithms, the procedures followed for data acquisition and preprocessing, and the training and evaluation of the models. Section 4 explains how the proposed methodology was implemented.

Section 5 exposes the results obtained. Finally, Section 6 presents the conclusions and future research lines.

2. Related Work

The recent literature discusses the use of hybrid detection systems to improve traditional network intrusion detection systems (NIDSs). These approaches are based on the premise that information beyond network statistics can help improve the accuracy achieved by detection systems. The systematic literature review did not reveal many scientific articles proposing hybrid methods as presented in this proposal. The following is a summary of recent work in this area.

Shuxin Shi et al. [4] propose a multimodal hybrid parallel network intrusion detection model (MHPN), which combines the analysis of traffic statistics information and original traffic load. The framework shows an accuracy of 99.98% and outperforms the single modal model in terms of intrusion detection performance.

Kiflay et al. [5] presented a new multimodal machine learning-based network IDS. Their proposal has two subsystems, flow-based and payload-based, using a trained random forest model to classify network traffic in both cases. Then, they used a soft voting scheme to obtain the final decision of the classifier. The resulting system achieves an accuracy of between 98% and 99%.

Tasnimul Hasan et al. [6] propose a distributed multimodal IDS that effectively handles tabular and image-based IoT data. They evaluated the performance of their solution using six contemporary IoT security datasets, achieving greater accuracy than other non-multimodal approaches. It is worth noting that the models were run on a Jetson Nano, making this an ideal solution for edge devices and even end devices.

However, studies that use hybrid intrusion detection models combining network traffic and side-channel techniques are not common. Notable in this regard is the work carried out by Jaafar et al. [7]. They developed a hybrid detection system that combines power consumption and network data to detect botnet attacks. They claim to be the first study to combine both approaches. As far as we are aware, this is correct.

Their study combines different types of IoT devices, such as smart cameras and routers. They established that combining power and network metrics improves detection accuracy, but the improvement varies depending on the device and the algorithm used.

This latest study is the one that most closely resembles the approach outlined in this article, as it proposes a system that combines network data and power consumption data. With this in mind, it was analyzed in detail in order to identify any potential shortcomings and avoid them when designing our own system.

First, the only type of attack tested against the devices is a botnet attack. As proven by studies like [8,9], different attacks can have different kinds of power consumption profiles, so using a single type of attack is not enough to validate that a model would provide good results under other circumstances. This also means that the model is only capable of determining whether an attack is taking place or not. It cannot distinguish the type of the attack.

Second, the study only creates two datasets for each device: one of them while the device is under attack, and another one while it is under normal operation. Each dataset only contains 30 min of data. We believe that this might be insufficient. Moreover, the data is cleanly split into attack and non-attack sections rather than being combined in a single trace, which makes it less realistic.

Third, only six features were chosen to create the dataset, out of which five are network features. These include the number of packets, the source and destination IP/MAC addresses, the protocol, and the length of the packet. We believe that there is room to

include additional network metrics. Performing feature selection over the initial set of features could also be an improvement.

Another potential problem with the metrics is the fact that the source and destination IP addresses are directly included, which could result in the model simply using the address to determine if the packets are coming from an infected device. In a real-world setting, this could pose a problem, since trusted devices can be infected at any time, so using addresses to determine whether an attack is taking place or not is unreliable.

Fourth, in order to build the time series that is passed as input to the machine learning model, the original data is lagged by one and two time units, which results in a size 3 window of temporal data. Such a small window means that the model will have little information about how metrics change over time to make its prediction, potentially lowering its accuracy. It also makes it harder to distinguish random spikes in data caused by normal operation, making the model more prone to false positives.

Fifth, the dataset used for this study is not available, which makes it hard to reproduce. Moreover, the code contains a lot of hard-coded data, such as device names, file paths, or feature names. This also makes it harder to run.

Our proposal presents an IDS-based on well-known models trained to detect and identify multiple attacks in heterogeneous IoT environments by combining network traffic data with power consumption. It is also capable of detecting unknown attacks.

To further contextualize the proposed approach, Table 1 compares it with representative hybrid IDS solutions reported in the literature.

Table 1. Comparison of hybrid intrusion detection systems for IoT.

Reference	Input Data Sources	Dataset Type	Evaluated Models	Side-Channel
Jaafar et al. [7]	Network traffic Power consumption	Custom	3 ML	Yes
Shi et al. [4]	Network statistics Packet payload	Public benchmark	5 DL	No
Kiflay et al. [5]	Network features Packet payload	Public benchmark	2 ML	No
Hasan et al. [6]	Tabular data Image data	Public benchmark	3 ML + 1 DL	No
This work	Network traffic Power consumption	Custom	7 ML	Yes

The comparison focuses on key methodological aspects. The column “Input Data Sources” identifies the types of information combined by each approach, while “Dataset Type” indicates whether the evaluation was performed using a public benchmark or a custom dataset. “Evaluated Models” reports the number and type of machine learning (ML) and deep learning (DL) models assessed in each study, and “Side-Channel” specifies whether side-channel information, such as power consumption, is incorporated into the intrusion detection process.

As shown in Table 1, the proposed approach differs from most representative hybrid IDS solutions in several aspects. While previous studies generally rely on network-derived information or more complex multimodal frameworks, only Jaafar et al. [7] combine network information with power-related side-channel measurements. In contrast, the proposed approach combines power consumption and network-derived information while evaluating a broader set of lightweight machine learning models specifically selected for resource-constrained IoT environments.

The scenario we developed has enabled us to generate a dataset combining legitimate and malicious network traffic in a single trace over a 6 h period, which, based on previ-

ous empirical experience, was considered sufficient for the purposes of this study. We have selected various network metrics as features derived from network traffic statistics, avoiding the use of data that could introduce biases into the models (IP addresses, MAC addresses, exact port numbers, etc.). These features utilize the last 60 s of network data and a sliding window (typically 15 s) to achieve greater sensitivity, continuity, and improved anomaly detection.

All codes and generated datasets are publicly available so that the experiments can be easily reproduced and also so that they can be used in new IoT environments.

3. Materials and Methods

Most publicly available datasets used for training attack detection models primarily rely on network traffic data (KDD CUP [10], NSL-KDD [11], UNSW-NB15 [12], CIC-IDS2017 [13], and CSE-CIC-IDS2018 [14]). There are also datasets specifically designed to train models that detect attacks on IoT networks (BoT-IoT [15] and TON_IoT [16]).

Although the use of these datasets allows for a better comparison of the proposed solution with other proposals, they lack multimodal information (such as power consumption) and do not perfectly align with the specific characteristics of each IoT deployment. Furthermore, some of these well-known datasets are quite old, making them difficult to consider for training current models, especially given the prevalence of modern attacks.

Given the specific requirements the proposed solution (combining network traffic with device power consumption), none of the existing datasets meet these requirements, so we need to build our own dataset. In order to do this, an IoT testbed was designed. This testbed emulates a real IoT environment, capturing network traffic and the power consumption of end devices. It also allows us to launch different attacks and analyze the captured data to detect and/or identify attacks on devices.

We believe that the methodology used to build this specific testbed can serve as a reference for other research teams to set up scenarios tailored to their own particular IoT environment.

3.1. Hardware Setup

The hardware setup used for this study involves multiple devices that work together to generate both power consumption and network data. While this process takes place, a set of software attacks is randomly initiated against the devices, which allows creating a dataset containing both types of generated features and labels indicating which entries correspond to an attack and which ones correspond to normal device behavior.

The setup contains three end devices. These are generic IoT devices that run some predefined behavior, which causes changes in their power consumption. They also perform network connections at irregular intervals, which generates network traffic. Both of these behaviors are designed to generate legitimate power and network signatures.

Attached to the end devices is a power reader, and it is used to collect a trace of their power consumption.

A fourth device, known as the main device, is responsible for launching attacks against the end devices and reading their power consumption data by connecting to the power reader.

Another device present in the setup is the router. It captures all network packets to and from the end devices and dumps them to a file.

The setup also includes a PC that runs a program known as the model server. This server receives power data from the main device and network data from the router and combines them.

Figure 1 graphically displays the hardware setup, including how the devices are connected to each other.

Details about the devices used can be found in Section 4.1.

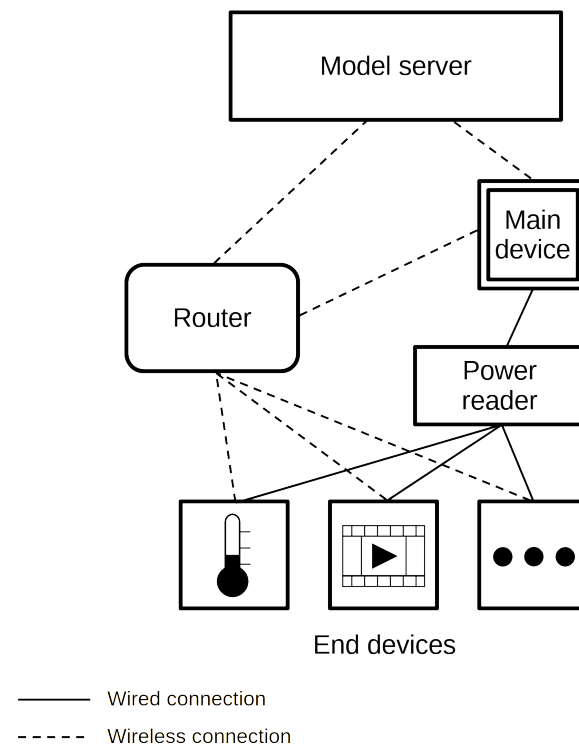


Figure 1. Hardware setup.

3.2. Device Behavior and Attacks

The three end devices were configured to exhibit distinct operational behaviors. The first device operates as a sensor, transmitting an HTTP POST request to a server at 10 min intervals. The second device is programmed to play a video every 10 min, whereas the third device remains in an idle state throughout the experiments.

These behaviors can be used to determine if the resulting attack detection model can be deployed on heterogeneous networks with several kinds of devices without compromising accuracy.

As stated in the previous section, all three end devices also perform random network connections in order to simulate legitimate traffic.

Several software-based attacks were implemented specifically for this study. These attacks are initiated by the main device to target the end devices, with the objective of emulating realistic cyberattack scenarios while ensuring that no actual harm is inflicted on the targeted devices.

In particular, four attacks were used to train the models, and three additional attacks were used during validation to check how the models respond to previously unseen attacks (research question RQ3).

Details about each of the seven attacks can be found in Section 4.2.

3.3. Classification Algorithms

When selecting classification algorithms to use for this study, the latest state-of-the-art approaches to multivariate time-series classification were systematically reviewed. Algorithms were chosen to ensure broad coverage of different methodological approaches. The only type of algorithms we excluded were deep learning-based algorithms, since they

usually have higher resource demands (both when being trained and when being run) that were deemed unfit for IoT environments.

Based on these ideas, the following classification algorithms were initially selected:

- Feature Summary (FS): This approach treats the input as multivariate time-series data and represents each instance using a set of statistical descriptors, namely the mean; standard deviation; and the 5th, 25th, 75th, and 90th percentiles. These extracted statistics replace the original power measurements and are subsequently used as inputs to a random forest classifier.
- Time-Series Forest (TSF) [17]: This algorithm analyzes the data as time series by constructing an ensemble of decision trees using the time-series forest methodology. Rather than operating directly on the raw input data, each tree is built from simple statistical characteristics of the series, including the mean, standard deviation, and slope.
- Supervised Time Series Forest (STSF) [18]: This analyzes the data as a time series and builds a decision-tree ensemble based on the supervised variant of the algorithm. It uses interval-based features (mean, std, and slope), selecting those that best separate the classes instead of relying on purely random choices.
- Time Series Feature Extraction Based on Scalable Hypothesis Tests (TSFresh) [19]: The algorithm processes the input as time-series data by generating a broad set of predefined features. It then employs statistical hypothesis testing to select the subset of features that best discriminates among the target classes.
- Fresh Pipeline with Rotation Forest Classifier (FreshPRINCE) [20]: Processes data as a time series by extracting a large number of features using TSFresh. It then transforms these features into a new representation and trains an ensemble classifier on top of them.
- Random Dilated Shapelet Transform (RDST) [21]: Processes data as a time series by extracting random shapelets with varying lengths and dilations. It transforms the data based on the similarity to these shapelets and uses the resulting features for classification.
- MUSE [22]: Processes data as a time series by extracting symbolic representations from multiple dimensions and window sizes. It builds a bag-of-patterns model and uses these features to train a classifier.
- Random Convolutional Kernel Transform (ROCKET) [23]: The algorithm represents the input as time-series data and generates feature vectors by convolving the series with a large set of randomly initialized kernels. The resulting feature vectors are subsequently used to fit a linear classifier.
- HIVE-COTE [24]: Processes data as a time series by combining multiple classification algorithms. It aggregates their predictions using a weighted ensemble to improve overall performance.

The authors in [25] introduce a taxonomy that can be used to categorize time-series classification algorithms. Under this taxonomy, the algorithms listed above would be classified as follows:

- Distance-Based Algorithms: (None).
- Feature-Based Algorithms: FreshPRINCE, FS, TSFresh.
- Interval-Based Algorithms: TSF, STSF.
- Shapelet-Based Algorithms: RDST.
- Dictionary-Based Algorithms: MUSE.
- Convolution-Based Algorithms: ROCKET.
- Deep Learning-Based Algorithms: (None).
- Hybrid Algorithms: HIVE-COTE.

This classification shows how the selected algorithms span multiple categories.

No other algorithms were included in this paper due to external limitations (some algorithms were not implemented in either of the Python libraries used, some of them would never finish running or took too long to finish, others did not support multivariate series or had compatibility issues, etc.). In addition, not all algorithms were used to produce the final results of this study; some of them were discarded early.

TSFresh was discarded since it had performance similar to that of FS, but its training time was five times longer. Another reason for discarding the model was that FreshPRINCE is a direct upgrade over TSFresh, and it produces better results.

HIVE-COTE was discarded since it took the longest to run but had the lowest F1 score among all models.

Finally, the ROCKET algorithm was run using its MINIROCKET variant [26], since the regular version would consistently crash the laptop used to run the models. For simplicity, the term ROCKET is used throughout the remainder of this manuscript to refer to the MINIROCKET implementation.

As a result, a total of seven algorithms were used in this study.

3.4. Data Collection and Processing

By default, the main device will read power consumption data from the end devices every 0.2 s. These raw power data have certain transformations applied before being passed to the models, as detailed in [9].

The transformed power reads are sent to the model server. When the model server receives a set of power reads, it downloads recent network traces from the router and extracts a set of network features. Both power reads and network features may be passed to the models as input depending on the configuration set by the user. In Section 4.5, a detailed list of network features is presented.

Instead of being directly passed to a prediction model, the collected data can be stored to build a reusable dataset. Once created, the dataset is automatically balanced by applying random undersampling to the majority class, ensuring an equal number of negative (regular behavior) and positive (attack) instances. This approach was preferred over oversampling techniques, such as SMOTE, because it preserves the integrity of the original measurements by avoiding the generation of synthetic samples, which may not accurately represent the power consumption patterns of IoT devices under attack. Moreover, the large number of available non-attack samples makes the reduction in the majority class unlikely to affect the representativeness of normal operating conditions.

Section 4.6 describes the datasets used in this work, including how they were generated and the distribution of instances across the different labels.

3.5. Model Training and Testing

The model server generates the datasets used during the training phase. They contain both power reads from the devices and network features extracted from the network packets captured by the router. During our tests, the last 60 s of both power reads and network features are used. The resulting dataset contains multivariate time series, which are passed as input to the classification algorithms.

A series of experiments was conducted under multiple evaluation scenarios. For experiments requiring both model training and performance evaluation, the available dataset was partitioned using an 80/20 stratified split implemented with *StratifiedShuffleSplit* (*random_state* = 0), ensuring both reproducibility and preservation of the class distribution. The split was performed after the window generation process, and no device-wise separation was applied, as the objective of this work is to compare the contribution of different feature sets under a common evaluation protocol rather than to assess generalization across unseen

devices. After partitioning, the feature scaler was fitted exclusively on the training subset and subsequently applied to both training and testing data to avoid information leakage during feature normalization.

As discussed in Section 5.6, although this partitioning strategy preserves class balance, a chronological split would constitute a more appropriate evaluation protocol for time-series data and will be adopted in future work.

The complete methodology is illustrated in Figure 2, which summarizes each stage of the proposed workflow.

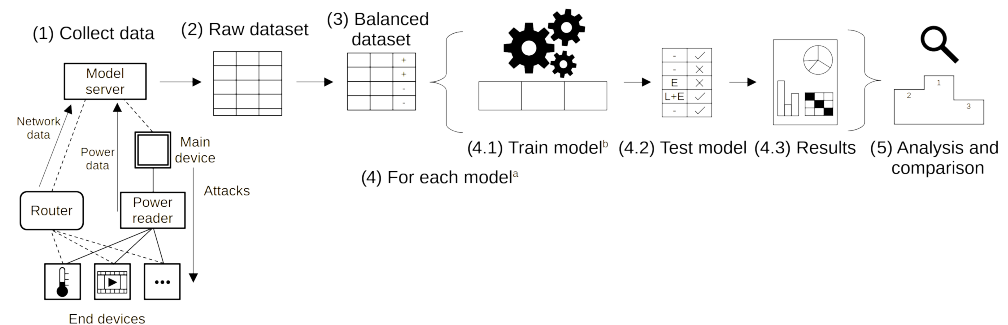


Figure 2. Overview of the proposed methodology. ^a Some experiments require only a single model. ^b Training is not always performed.

3.6. Scenarios

As outlined by the research questions presented in Section 1, a primary objective of this study is to evaluate the proposed system under a variety of operating conditions. To achieve this objective, five experimental scenarios were defined. Although all of them follow the methodology described in the previous section, each scenario incorporates specific characteristics tailored to its corresponding evaluation objective.

3.6.1. Scenario 1—Standard Network Behavior

This scenario constitutes the primary experimental setting and is designed to address research questions RQ1 and RQ2. In this scenario, the main device launches random attacks against the end devices and collects power consumption data. This data is combined with the network data coming from the router, and a dataset is created. The whole process lasts 6 h.

The dataset is then passed to each of the classifiers introduced in Section 3.3. Each model is trained and tested, and several performance metrics are recorded. The results can be used to determine which model performs better.

For this scenario, only six basic network features are used. The list of features is explained in more detail in Section 4.5.

3.6.2. Scenario 2—Validation Attacks

As mentioned in Section 3.2, three of the seven implemented attacks are used for validation.

In this scenario, each model developed in Scenario 1 is evaluated using two short datasets containing traces of the validation attacks. The objective is to determine whether the models can correctly identify these traces as malicious. This evaluation addresses research question RQ3, as none of the three validation attacks were included in the datasets used during the training phase.

This scenario also employs basic network features.

3.6.3. Scenario 3—Feature Selection

Both scenarios 1 and 2 employ basic network features that were arbitrarily selected. However, it is possible to extract many kinds of features out of a trace of network packets. In order to determine which features would be best to use, Scenario 3 is proposed.

A new list of 16 features is created for this scenario: 15 network features and power. Features are selected out of this list. The full list is detailed in Section 4.5.

This scenario has two main objectives: maximizing the performance of the models and minimizing the amount of features used. We therefore decided to approach this scenario as a multi-objective optimization problem.

A genetic algorithm is used to evaluate potential solutions. Each solution consists of a subset of the initial feature list. Solutions are evaluated by taking only the features included in the solution and using them to train four different models. Solutions are evaluated by combining the average performance of the models and the amount of features used.

The algorithm returns a set of non-dominated solutions identified during the search. In other words, for each possible amount of features (1–16), only the solution with the highest performance is printed, or nothing occurs if another solution with better performance and less features was found.

Once the list of solutions is obtained, a full run including all seven models is performed. The accuracy of each model is recorded, alongside the time taken to train them, which is expected to grow with the number of features.

The results of this run can be used to determine which set of features works best for this problem, which provides an answer to research question RQ4.

3.6.4. Scenario 4—Power vs. Network Features

Scenario 4 is aimed at comparing how power and network features affect model performance. It compares the results of running a model trained using only power features, a second one trained using only network features, and a third one trained using both network and power features.

This scenario is used to prove that the combined power + network approach yields better results than only employing one type of features, be it power or network features. It therefore answers research questions RQ5 and RQ6.

3.6.5. Scenario 5—Real-Time Attack Detection

Scenario 5 has some key differences compared to the rest. In this scenario, a single model is evaluated under live operating conditions. The model is deployed on the main device and executed continuously for 30 min while monitoring the end devices for attacks as they occur in real time. No dataset is generated during this evaluation.

Scenario 5 assumes that the prediction model has been selected based on a comparative evaluation of candidate models. Accordingly, the TSF model is employed as the prediction engine. The experimental results supporting this choice are presented in Sections 5.1–5.4, while Section 5.5 summarizes the comparative evaluation and explains the rationale for selecting TSF as the best overall-performing model across the evaluated conditions.

To address research question RQ7, both detection accuracy and detection latency are measured. These metrics make it possible to evaluate not only whether the model can identify attacks in real time but also the time required to detect them.

4. Implementation

This section explains in detail how the proposed methodology has been implemented.

4.1. Hardware Components

All experiments are conducted using a single hardware setup consisting of three end devices, one main device, a power reader, a router, and a laptop running the model server.

Three Raspberry Pi 3 Model B boards (Raspberry Pi Ltd., Cambridge, UK) are used as the end devices. Since they can be powered via their GPIO pins, a power measurement device can be easily integrated to monitor their power consumption.

A power monitoring module incorporating the INA3221 current and voltage monitor (Texas Instruments, Dallas, TX, USA) [27] is used to acquire the electrical measurements. INA3221 provides three independent measurement channels, allowing the simultaneous monitoring of up to three devices. Although the device measures both current and bus voltage, this work uses the measured current as an indicator of the device's power consumption.

A Raspberry Pi 4 Model B (Raspberry Pi Ltd., Cambridge, UK) serves as the main device. It communicates with the INA3221 over the i2c communication protocol to retrieve power consumption measurements, which are subsequently recorded in a file.

The router is a simple home router running the BusyBox Linux distribution and configured using OpenWRT.

The laptop is a work PC running an i7 CPU with a speed of 2.40 GHz and 8 GB of RAM.

4.2. Software Attacks

The following attack scenarios were performed during the collection of datasets subsequently used for model training:

- Mining Attack: A cryptocurrency miner [28] is launched on the target device, causing a significant rise in CPU load. The miner is configured to fully utilize all available CPU cores (four in the experimental platform).
- Login Attack: The main device performs a brute-force SSH attack against the target device by repeatedly attempting to carry out authentication using different password combinations. Numerous concurrent SSH connection requests are generated, requiring the target device to handle multiple authentication attempts simultaneously. The attack has been implemented using Hydra v9.5 [29].
- Encryption Attack: A file encryption program is deployed and executed on the target device. Once launched, it encrypts selected files to reproduce the typical behavior exhibited by ransomware.
- Flood Attack: The main device floods the target device with SYN TCP requests, which produces a large amount of junk network traffic that the target device must respond to.

In accordance with the objective of research question RQ3, three additional attacks were incorporated into the validation stage. None of these attacks were included in the training datasets:

- Password Attack: A script is executed on the remote device to perform a brute-force attack against a password hash by repeatedly computing SHA-256 hashes. To reduce the likelihood of detection, the script introduces random sleep intervals between successive operations.
- Lite Mining Attack: This attack follows the same approach as the mining attack but limits execution to two processing threads instead of four. As a result, its power consumption is reduced, making the attack more difficult to detect.
- Port Scan Attack: The main device attempts to open a connection to each possible software port, normally in an attempt to gather information about which ones are

open. This attack generates a significant amount of network traffic but does not cause a large increase in the target's power consumption.

The datasets generated in this work adopt the attack labeling scheme shown in Table 2. Each attack type is assigned a unique value corresponding to a different bit position by using powers of two. This allows multiple concurrent attacks to be represented by a single label obtained by summing the values associated with each active attack. Since every attack occupies a unique bit position, each possible combination generates a unique label that can be unambiguously decoded. For example, a value of 33 represents the simultaneous occurrence of a mining attack (1) and a flood attack (32).

Table 2. Attack labeling scheme used in the generated datasets.

Attack	Abbreviation	Label Value	Bit
Mining	M	1	2^0
Login	L	2	2^1
Encryption	E	4	2^2
Password	P	8	2^3
Lite mining	LM	16	2^4
Flood	F	32	2^5
Port scan	PS	64	2^6

4.3. Classification Algorithms

All classification algorithms described in Section 3.3 were implemented in Python 3.11.2 using scikit-learn (version 1.2) [30], sktime (version 0.23) [31], and aeon (version 0.4) [32]. For reproducibility, the complete list of Python package dependencies and their corresponding versions is available in the project's pyproject.toml file at the project repository.

All classifiers were trained using the default hyperparameters provided by the corresponding AEON, sktime, or scikit-learn implementations. To ensure reproducibility, a fixed random seed (random_state = 0) was used whenever supported, and parallel execution was enabled through n_jobs = -1 when available. For the ROCKET classifier, the MINIROCKET transform was selected (rocket_transform = "minirocket"), providing substantially lower computational costs than the original ROCKET implementation while maintaining comparable performance as previously mentioned. No additional hyperparameter tuning was performed.

4.4. Data Recording and Formatting

To guarantee that each power consumption sample reflects a newly completed acquisition, the sampling interval was selected based on the timing characteristics of the INA3221 sensor. Power consumption measurements are acquired by the main device from all three end devices every 200 ms. According to Section 7.4.2 of the INA3221 datasheet [27], the duration of a complete measurement cycle depends on the selected ADC conversion times, the averaging configuration, and the sequential acquisition of the three channels. Using the default ADC conversion time of 1.1 ms for both shunt and bus voltage measurements together with a 16-sample averaging configuration results in a complete measurement cycle of approximately 106 ms. Therefore, a sampling interval of 200 ms was selected to ensure that each acquired sample corresponds to a newly completed measurement cycle while avoiding repeated acquisition of the same measurement.

The collected data are stored in individual CSV files, one per device. Each record includes a UNIX timestamp, an integer identifying the active attack, and the measured current value in milliamperes (mA).

Automatic labeling is achieved by continuously monitoring the attack status of each end device on the main device, thereby ensuring that the generated dataset is correctly annotated.

Every 5 s, the main device groups the last 60 s of power data and sends it to the model server.

Meanwhile, the router captures all network packets from and to the end devices and stores them in a cyclic buffer with a set size of 300 KB. The program used to capture the packets is *tcpdump*.

The model server is constantly waiting for data from the main device. When it receives the data, it downloads the entire network data buffer from the router. The last 60 s of network data is processed and turned into the network features specified in the server's configuration file. The power reads and network features are combined to form a multivariate time series. This resulting series is finally added to the output dataset file as a new instance.

4.5. Network Features

For the first two scenarios of this study, six basic network features were handpicked to serve as model input. These features are computed by taking the last 60 s of network data and applying a rolling window with a length of 15 s, which results in a time series.

The list of basic network features is as follows:

- Total number of TCP packets.
- Number of external connection establishments (number of incoming packets with the SYN flag set and the ACK not set).
- Number of packets with the RST flag set.
- Average packet length.
- Number of distinct source ports for incoming packets.
- Percent of packets that contain SSH data.

Starting with Scenario 3, a new list of network features is used. Every feature that was deemed potentially relevant for this study was included, with the intention of using the feature selection algorithm to filter the list.

When creating this list, we wanted to avoid any features that could be used to determine which device the data came from (e.g., IP addresses, MAC addresses) or which software program created the packets (e.g., exact port, IP fragment flags, packet TTL, window size...). This was considered important since, otherwise, the models could learn to recognize attacks by their packet signature rather than their overall network activity, which is not desirable since there is no guarantee that said signature will remain constant across different implementations of the attacks.

For example, the model learning that the port scan attack can be identified by a large amount of distinct source ports is desirable, but the model learning that the flood attack is the only behavior that sets the "do not fragment" flag to 0 is not since that is simply a quirk of the specific implementation used, and other implementations will likely not behave like this.

These features also employ the last 60 s of network data and a rolling window. The size of the rolling window is 15 s by default, but some metrics use a different window size.

The full list of network features is as follows:

1. Average frame length.
2. Average frame length (window size = 5 s).
3. Percent of packets with differentiated services set.
4. Total number of TCP packets.

5. Total number of TCP packets (window size = 5 s).
6. Total number of incoming TCP packets.
7. Total number of outgoing TCP packets.
8. Number of distinct source ports for incoming packets.
9. Number of incoming packets addressed to a non-privileged port (port number ≥ 1024).
10. Number of external connection establishments (number of incoming packets with the SYN flag set and the ACK not set).
11. Number of connection establishments started by the device (number of outgoing packets with the SYN flag set and the ACK not set).
12. Number of connection closures (number of packets with the FIN flag set).
13. Number of packets with the RST flag set.
14. Number of packets with the PSH flag set.
15. Percent of packets that contain SSH data.

4.6. Scenarios

4.6.1. Scenario 1—Standard Network Behavior

The following parameters were used to create the dataset for Scenario 1:

- Event generation seed: 1,376,829,338.
- Attacked devices: sensor (1), video (2), and idle (3).
- Duration: 360 min.
- Features recorded: power reads + six basic network features.
- Time between attacks: 4 ± 1 min.
- Minimum time between attacks: 60 s.
- Attacks: mining (0), login (1), encryption (2), and flood (5).
- Attack duration: 1.5 ± 0.25 min.
- Minimum attack duration: 10 s.
- Multiple attack chance: 10%.
- Short attack chance: 20%.
- Short attack average duration: 15 s.

The values provided for “time between attacks” and “attack duration” correspond to the parameters of the normal distributions governing the generation of inter-attack delays and attack durations.

Table 3 lists the amount of instances of each type in the Scenario 1 dataset.

Table 3. Instance count—Scenario 1 dataset.

Instance Type	Count	Percent
No attack	2662	50%
Mining	666	12.51%
Login	521	9.79%
Mining + Login	26	0.49%
Encryption	668	12.55%
Mining + Encryption	45	0.85%
Login + Encryption	18	0.34%
Flood	592	11.12%
Mining + Flood	45	0.85%
Login + Flood	16	0.3%
Encryption + Flood	57	1.07%
Mining + Login + Flood	5	0.09%
Mining + Login + Encryption + Flood	3	0.06%
Any attack	2662	50%
Total	5324	100%

4.6.2. Scenario 2—Validation Attacks

On this scenario, models were run in the multi-prediction mode. When operating in this mode, the model returns class probability estimates rather than a single prediction. The output includes the probability that a power consumption sample reflects normal behavior together with the probabilities corresponding to each attack combination on which the model was trained.

To assess the classification accuracy of the models, their output needs to be converted to a boolean prediction. This is accomplished through the usage of a threshold value. Whenever the aggregated probability assigned to attack-related classes reaches or exceeds the predefined threshold, the system considers that an attack is taking place.

We set this threshold to 0.4 after manually tuning it through multiple tests. The threshold was selected by evaluating the trade-off between detection sensitivity and the number of false alarms, with the objective of achieving a balanced compromise between false positives and false negatives under the experimental conditions.

In a practical deployment, the detection threshold can be adjusted to achieve the desired trade-off between detection sensitivity and the false positive rate. A higher threshold reduces the number of false positives but increases the risk of missing genuine attacks, whereas a lower threshold increases detection sensitivity at the expense of a higher false positive rate. Therefore, the optimal threshold depends on the operational requirements and the acceptable balance between missed detections and false alarms.

The following parameters were used to generate the three datasets considered in this scenario:

- Event Generation Seed: 1,515,944,703.
- Attacked Devices: sensor (1), video (2), and idle (3).
- Duration: 60 min.
- Features Recorded: Power reads + six basic network features.
- Time Between Attacks: 3 ± 1 min.
- Minimum Time Between Attacks: 60 s.
- Attacks: password (3)/lite mining (4)/port scan (6), depending on the dataset.
- Attack Duration: 2 ± 0.5 min.
- Minimum Attack Duration: 10 s.
- Multiple Attack Chance: 0%.
- Short Attack Chance: 20%.
- Short Attack Average Duration: 15 s.

For each of the three end devices, every dataset contains one hour of recorded data. The class distributions for the lite mining, password, and port scan datasets are provided in Tables 4, 5 and 6, respectively.

Table 4. Instance count—Scenario 2—lite mining dataset.

Instance Type	Count	Percent
No attack	530	50%
Lite Mining	530	50%
Total	1060	100%

Table 5. Instance count—Scenario 2—password dataset.

Instance Type	Count	Percent
No attack	591	50%
Password	591	50%
Total	1182	100%

Table 6. Instance count—Scenario 2—port scan dataset.

Instance Type	Count	Percent
No attack	561	50%
Port Scan	561	50%
Total	1122	100%

4.6.3. Scenario 3—Feature Selection

Scenario 3 employs a multi-objective genetic algorithm and a set of four models to evaluate feature combinations. The four models selected to be used for feature selection were FS, RDST, ROCKET, and TSF. These models were chosen with the intention of using as many types of models as possible (according to the taxonomy described in Section 3.3) while keeping the time taken to run the feature selection algorithm within reasonable limits.

The genetic algorithm chosen for this task was the NSGA-II algorithm [33], a well-known multi-objective genetic algorithm. The algorithm was run with a population of 16 individuals, with eight offspring individuals per iteration, and a bit flip mutation algorithm with a mutation chance of 10% that flips bits with a 25% chance.

The algorithm performed 200 iterations on a dedicated 32-core machine.

A dataset was created containing the 16 basic features to be used for feature selection, and it had the following parameters:

- Event Generation Seed: 1,376,829,338.
- Attacked devices: sensor (1), video (2), and idle (3).
- Duration: 360 min.
- Features Recorded: power reads + 15 network features.
- Time Between Attacks: 4 ± 1 min.
- Minimum Time Between Attacks: 60 s.
- Attacks: mining (0), login (1), encryption (2), and flood (5).
- Attack Duration: 3 ± 1 min.
- Minimum Attack Duration: 10 s.
- Multiple Attack Chance: 10%.
- Short Attack Chance: 10%.
- Short Attack Average Duration: 15 s.

Table 7 lists the number of instances for each type.

Table 7. Instance count—Scenario 3 datasets.

Instance Type	Count	Percent
No attack	2472	50%
Mining	677	13.69%
Login	512	10.36%
Mining + Login	33	0.67%
Encryption	737	14.91%
Mining + Encryption	50	1.01%
Login + Encryption	21	0.42%
Mining + Login + Encryption	1	0.02%
Flood	373	7.54%
Mining + Flood	26	0.53%
Login + Flood	11	0.22%
Encryption + Flood	31	0.63%
Any attack	2472	50%
Total	4944	100%

4.6.4. Scenario 4—Power vs. Network Features

Scenario 4 reuses the dataset from Scenario 3, since it contains both power and network metrics. Depending on which test is performed, a different subset of its features will be used (power only, network only, or both power and network).

4.6.5. Scenario 5—Real-Time Attack Detection

The objective of Scenario 5 is to assess the model's effectiveness in detecting all attack types in real time while also quantifying the corresponding detection latency. Since normal behavior periods are not specially useful for this scenario, a much higher attack rate is used.

Consistent with Scenario 2, the model is evaluated in multi-prediction mode using a threshold of 0.4. A second run with a threshold value of 0.5 is also performed.

Scenario 5 does not have a dataset file. Nevertheless, the attacks executed during the evaluation were configured using a predefined set of parameters, which are summarized below:

- Event Generation Seed: 14,533,676.
- Attacked Devices: sensor (1), video (2), and idle (3).
- Duration: 30 min.
- Features Recorded: power reads + six network features.
- Time Between Attacks: 1.5 ± 0.5 min.
- Minimum Time Between Attacks: 40 s.
- Attacks: mining (0), login (1), encryption (2), lite mining (4), flood (5), and port scan (6).
- Attack Duration: 1 ± 0.25 min.
- Minimum Attack Duration: 30 s.
- Multiple Attack Chance: 10%.
- Short Attack Chance: 0%.

4.6.6. Scenario Summary

Table 8 summarizes the main features of each scenario.

Table 8. Scenario summary.

Scenario	Attacks Launched	Training Attacks	Power	Training Features Network (Basic)	Network	Test Prediction Type	Detection Type	Research Questions
1	M, L, E, F	M, L, E, F	X	X		Exact attack	Offline	RQ1, RQ2
2	P, LM, PS	M, L, E, F	X	X		Boolean	Offline	RQ3
3	M, L, E, F	M, L, E, F		Varies		Exact attack	Offline	RQ4
4	M, L, E, F	M, L, E, F	X		6	Exact attack	Offline	RQ5, RQ6
5	M, L, E, LM, F, PS	M, L, E, F	X		6	Boolean	Real time	RQ7

The “Attacks Launched” column specifies which attacks were used (M: mining; L: login; E: encryption; P: password; LM: lite mining; F: flood; PS: port scan).

The “Training Attacks” column shows which attacks the models were trained with.

The “Training Features” column lists which features were used to train the models. The “Power” column is checked if power reads were used as input. The “Network (basic)” column is checked for those scenarios where the six initial basic network features were used. The “Network” column shows how many of the 15 definitive network features were used.

The “Test Prediction Type” column indicates the prediction strategy adopted during model evaluation. “Exact attack” corresponds to experiments in which the model is both trained and evaluated on a single dataset, requiring it to assign the most probable class label to each test instance. “Boolean” refers to evaluations performed on datasets containing attack labels that were absent during training. In this case, the model is required only to determine whether a given instance corresponds to an attack or to normal operation.

5. Results and Discussion

The results of the training and evaluation procedures using multiple models across the five scenarios are presented in this section.

The following metrics were used to assess model performance:

- **F1 Score:** The F1 score is defined as the harmonic mean of precision and recall, yielding values in the range from 0 (lowest performance) to 1 (highest performance). Because the datasets contain multiple classes, the overall F1 score is computed using the micro-averaging approach, which aggregates contributions from all classes and appropriately accounts for class imbalance, as instances of normal behavior are substantially more frequent than attack instances.
- **TP_c (True Positives, Correct):** Attack samples for which both the presence of an attack and its corresponding attack type are correctly predicted.
- **TN (True Negative):** Samples representing normal device behavior that are correctly identified as such.
- **TP_i (True Positives, Incorrect):** Instances representing attacks that are correctly detected but incorrectly labeled in terms of attack category.
- **FP (False Positive):** Instances representing normal operation that are misclassified as malicious activity.
- **FN (False Negative):** Instances representing malicious activity that are misclassified as normal behavior.

5.1. Test Results—Scenario 1

Table 9 reports the results obtained for all tested models.

Table 9. Scenario 1 results.

Model	F1 Score	TP _c	TN	TP _i	FP	FN	Time (s)
ROCKET	0.9634	505	521	11	12	16	113
STSF	0.9606	500	523	20	10	12	836
RDST	0.9587	498	523	16	10	18	257
MUSE	0.9587	504	517	15	16	13	743
TSF	0.9531	493	522	20	11	19	250
FreshPRINCE	0.9502	500	512	16	21	16	221
FS	0.9408	491	511	21	22	20	107

The F1 scores achieved by the models are very positive, considering that the network features used during this scenario were arbitrarily chosen. The difference between the best and worst performing model is small, which shows that all seven algorithms are good at classifying the attacks. The amount of incorrect true positives (TP_i) is low, which means that the models are capable of not only detecting attacks but also correctly classifying them by type.

Using these results, research questions RQ1 and RQ2 are positively answered.

The time taken to train and test each model is also listed. STSF and MUSE have significantly higher training times, whereas ROCKET and FS have the shortest times. ROCKET is especially remarkable, since it has the second shortest time and the highest F1 score.

5.2. Test Results—Scenario 2

In this scenario, model predictions are determined by comparing the aggregated attack probability to a threshold value of 0.4. This formulation yields a binary decision, and consequently, the TP_i metric is not applied in this case.

Table 10 reports the performance of each tested model when predicting on the dataset containing traces of the lite mining attack. Three additional columns are included: Th_{max} and Th_{min} , which indicate the range of threshold values for which each model achieves its maximum F1 score, and *Best F1*, which corresponds to that maximum value.

Some models do not support the multi-prediction output mode. Those models were run in boolean prediction mode, which means they do not use the threshold value nor the three additional columns listed above.

Table 10. Scenario 2 results—lite mining attack.

Model	F1 Score	Precision	Recall	TP	TN	FP	FN	Th_{max}	Th_{min}	Best F1
ROCKET	0.9887	0.9832	0.9943	527	521	9	3			
STSF	0.9877	0.9778	0.9981	529	518	12	1	0.46	0.45	0.9887
RDST	0.9736	0.9846	0.9623	510	522	8	20			
TSF	0.9651	0.9363	0.9981	529	494	36	1	0.61	0.605	0.9934
FS	0.9443	0.9067	0.9906	525	476	54	5	0.64	0.635	0.9632
FreshPRINCE	0.9189	0.8651	0.9925	526	448	82	4	0.595	0.59	0.9547
MUSE	0.9142	0.9720	0.8528	452	517	13	78			

The F1 scores achieved are similar to the ones seen in Scenario 1's results, and they are sometimes even higher. This proves that the models are capable of detecting the lite mining attack even though it was not present in the training dataset.

ROCKET continues to be the best-performing model. The Th_{max} and Th_{min} columns show that some of the models would benefit from a higher threshold value. In particular, TSF could get the first position if the threshold was raised to approximately 60%.

The high precision and recall values obtained by most models indicate that the lite mining attack can be detected reliably despite not being present in the training dataset.

Table 11 reports the test results of each model on the dataset containing password attack traces.

Table 11. Scenario 2 results—password attack.

Model	F1 Score	Precision	Recall	TP	TN	FP	FN	Th_{max}	Th_{min}	Best F1
MUSE	0.7572	0.9841	0.5228	309	586	5	282			
RDST	0.714	0.9961	0.4298	254	590	1	337			
ROCKET	0.6641	0.9899	0.3316	196	589	2	395			
TSF	0.6616	0.9704	0.3333	197	585	6	394	0.065	0.06	0.9239
STSF	0.6354	0.9706	0.2792	165	586	5	426	0.07	0.065	0.9755
FreshPRINCE	0.5541	0.9103	0.1201	71	584	7	520	0.05	0.045	0.8435
FS	0.5296	0.9730	0.0609	36	590	1	555	0.01	0.005	0.769

The F1 scores on the password dataset are not as good. The password attack is the most difficult attack to detect, since it sleeps at random intervals and barely has any network activity. Although all evaluated classifiers maintain very high precision, indicating that false alarms remain uncommon, their recall is substantially lower, resulting in a considerable number of false negatives. The false negative rate of the best model is $282 / (309 + 282) = 47.72\%$, and the recall is 0.5228, which proves that many of the attack instances were not detected.

The best F1 column shows that some models could greatly improve their F1 scores if the threshold was lowered to almost the minimum possible value, but this is unfeasible, since that would greatly increase the amount of false alarms, rendering the system effectively useless.

These results highlight an important limitation of the proposed approach. Attacks such as password, which exhibit low network activity and intermittent behavior, are inherently more difficult to distinguish from normal operation under a common detection threshold. Consequently, achieving higher detection rates for these attacks may require attack-specific training data or deployment-specific threshold calibration depending on the characteristics of the protected IoT environment.

Table 12 shows the results for each model when making predictions on the port scan attack dataset.

Table 12. Scenario 2 results—port scan attack.

Model	F1 Score	Precision	Recall	TP	TN	FP	FN	Th _{max}	Th _{min}	Best F1
TSF	0.9626	0.9303	1.0000	561	519	42	0	0.435	0.432	0.9679
MUSE	0.943	0.9233	0.9661	542	516	45	19			
FreshPRINCE	0.8289	0.7463	0.9964	559	371	190	2	0.635	0.63	0.8476
FS	0.6979	0.7071	0.6754	379	404	157	182	0.35	0.345	0.8316
ROCKET	0.6943	0.9781	0.3975	223	556	5	338			
STSF	0.6417	0.9297	0.3066	172	548	13	389	0.23	0.225	0.9545
RDST	0.607	0.9348	0.2300	129	552	9	432			

The results of the port scan dataset are slightly worse than those of the lite mining dataset, but they are not as bad as the ones obtained for the password dataset. TSF and MUSE both achieve F1 scores above 90%. In contrast, ROCKET, STSF and RDST maintain high precision but exhibit substantially lower recall, indicating that they produce few false alarms but fail to detect a considerable fraction of the attack instances. This behavior further illustrates that the effectiveness of each classifier depends on the characteristics of the target attack. This proves that no single model is best for every situation, and the specific model to use should be carefully considered depending on the context in which the system is deployed.

The results of this scenario address research question RQ3, showing that it is possible to detect previously unseen attacks, although detection accuracy may decrease depending on the specific characteristics of each attack, the model being tested, and the detection threshold used.

The observed precision–recall trade-offs further demonstrate that different classifiers exhibit different detection behaviors depending on the attack characteristics. Consequently, selecting an appropriate model and detection threshold is essential for the intended deployment scenario.

5.3. Test Results—Scenario 3

The feature selection algorithm was run for a total of 19:47 h. It provided a total of seven non-dominated solutions with a number of features ranging from 1 to 7. Table 13 lists those solutions, alongside the F1 score they obtained during the run. The features are listed with the same number used to identify them in Section 4.5, alongside a short name. Features that were not selected in any of the non-dominated solutions are not shown in the table.

The table has multiple interesting details. First, power was always selected as a feature, which proves that it is the most useful feature when determining which attacks are active on a device. Second, every solution checked by the algorithm that used more than seven features was a dominated solution, which implies that adding more features to the models is not only pointless but also harmful for their accuracy.

Third, there is a significant increase in the F1 score when adding a third feature, and from that point onward, the F1 score only experiences small increases.

Table 13. Features selected by the genetic algorithm.

No. Features	F1 Score	Selected Features							
		1 Power	7 TCP, in	8 TCP, out	9 Distinct Ports, in	11 SYN, in	13 FIN	14 RST	15 PSH
7	0.9689	X		X	X	X	X	X	X
6	0.9684	X	X	X		X		X	X
5	0.9674	X		X		X		X	X
4	0.9671	X	X					X	X
3	0.9649	X						X	X
2	0.9398	X	X						
1	0.9393	X							

Fourth, only 8 out of the 16 features were considered. The other eight features were not helpful in detecting the attacks. These dropped features are the frame length (both short and long windows), the value of the differentiated services field, the total number of TCP packets (both short and long windows), the non-privileged port connections, the number of connections started by the device, and the percent of packets that contain SSH data.

From the perspective of the NSGA-II optimization process, the discarded features did not provide sufficient additional discriminative information to justify their inclusion in the selected subsets. This behavior is consistent with the presence of redundant or only marginally informative features, although no explicit correlation analysis was performed.

These seven sets of features were used to train all seven models in order to better understand how the F1 score is affected by feature changes. An additional run with all 16 features was also performed. The results of these runs are listed in Table 14.

Table 14. Scenario 3 results.

No. Features	F1 Score		Train Time (min.)
	Feature Selection	All Models	
16		0.9649	60
7	0.9689	0.9689	40
6	0.9684	0.9689	39
5	0.9674	0.9682	33
4	0.9671	0.9668	32
3	0.9649	0.9656	27
2	0.9398	0.9581	25
1	0.9393	0.9359	17

The F1 scores obtained when training all seven models are similar. They remain sorted, with more features resulting in an increase in scores up until seven features. It can also be seen how adding all 16 features causes a drop in performance, with an F1 score between the one achieved by the three-feature and two-feature solutions.

As expected, increasing the amount of features also increases the time taken to train the model.

Using this information, research question RQ4 can be answered: The model's F1 score is maximized when using the seven features listed on the first row of Table 13. If the model's train time is a concern, less features can be used without causing a significant drop in the F1 score.

5.4. Test Results—Scenario 4

For Scenario 4, three separate runs are performed. A run that only uses power to train the models, a run that only uses network features, and a run that uses both.

The network-feature-only run was performed using the six network features selected by the seven-feature solution from Scenario 3. That is, feature numbers 8, 9, 11, 13, 14, and 15. Power is added to this list for the run that combines power and network features.

Table 15 shows the F1 score obtained by each model on all three runs.

Table 15. Scenario 4 results.

Model	F1 Score		
	Power Only	Network Only	Power + Network
MUSE	0.9717	0.9323	0.9697
FreshPRINCE	0.8625	0.9141	0.9585
FS	0.8746	0.9120	0.9626
RDST	0.9717	0.9292	0.9656
ROCKET	0.9676	0.9383	0.9767
STSF	0.9585	0.9161	0.9767
TSF	0.9444	0.8898	0.9666
Average	0.93587	0.91882	0.96808

These results prove that the best-performing feature set is the one that combines both power and network features and that it outperforms both the models that only use power as input and the ones that only use network features.

This is a very interesting conclusion, since most studies published to date that attempt to detect attacks against IoT devices focus on network features. These results show that adding features obtained through side-channel techniques could significantly improve the accuracy of these systems.

Nevertheless, this conclusion should be interpreted in the context of the proposed methodology. The network-based models evaluated in this work are intentionally lightweight, as the objective is to develop detection mechanisms suitable for resource-constrained IoT environments. Consequently, direct comparisons with previous studies employing more sophisticated machine learning or deep learning models should be interpreted in light of the different design objectives, and the magnitude of the observed improvement may differ when using such approaches.

These results also indicate that the benefits of feature fusion are model-dependent rather than universal. Although most classifiers improve when combining network and power features, MUSE and RDST achieve slightly better performance using only power-consumption features, suggesting that different learning algorithms exploit multimodal information with varying degrees of effectiveness.

Research questions RQ5 and RQ6 can be deemed solved after the results exposed in this section. Combining power and network features can improve the model's F1 scores over simply using power or network features separately.

Future work should investigate whether the incorporation of side-channel information provides similar benefits when integrated into more sophisticated IDS architectures. Furthermore, evaluating the trade-off between detection performance, computational cost, and deployment feasibility would help determine the most appropriate trade-off between detection accuracy, computational complexity, and deployment feasibility in practical IoT environments.

5.5. Test Results—Scenario 5

In Scenario 5, the model was operated in parallel with ongoing attacks targeting the end devices.

The model chosen to be deployed was *TSF*, since it provided reasonable results in previous scenarios, especially during Scenario 2, where it achieved the best results at

detecting the port scan attack. The only other model that managed to adequately detect this attack was MUSE, but MUSE obtained the worst results for the lite mining dataset.

The model was trained using the best feature combination found during Scenario 3: power and six network metrics, for a total of seven features.

All attacks were considered except the password attack, since Scenario 2 showed that this model is not effective at detecting it.

In addition to the previously introduced metrics, two new measures are defined for this scenario: attack detection delay and attack over detection delay. These metrics quantify the latency required by the model to detect the onset and termination of an attack. Both are expressed in terms of model inference iterations: a value of 0 indicates that the attack is detected in the first model run following its onset, 1 indicates detection in the second run, and so forth.

The result metrics for this scenario are shown on Table 16.

Table 16. Scenario 5 results.

Threshold	F1 Score	TP	TN	FP	FN	Average Attack Detection Delay	Average Attack Over Detection Delay
40%	0.8876	233	337	33	26	0.27 model runs	0.79 model runs
50%	0.9191	233	362	19	22	0.15 model runs	0.35 model runs

The detection delay can be converted into seconds for improved interpretability. Given that the model is executed every 5 s, the expected detection latency is 2.5 s on average after the onset of an attack. If the model then takes $5 \times 0.15 = 0.75$ s to detect the attack, the average time taken to detect an attack after it starts is 3.25 s. Similarly, it needs an average of $2.5 + 5 \times 0.35 = 4.25$ s to detect the end of an attack. These times can be reduced by running the model more frequently.

These results indicate that the model requires more time to detect the termination of an attack than its onset. A further analysis of the script output reveals that most of the reported false positives are associated with delays in detecting the end of an ongoing attack. However, this behavior is not specially concerning, since the main purpose of the model is detecting the start of an attack as soon as possible.

The results also demonstrate the effect that tuning the detection threshold has in the model and how finding a threshold appropriate to the current environment is important to maximize the performance of the model.

In general, the model can detect live attacks with a reasonable delay, which positively answers research question RQ7.

5.6. Limitations and Deployment Considerations

While the experimental results demonstrate the feasibility of the proposed hybrid IDS, several limitations should be considered when interpreting these results and assessing their applicability to practical IoT deployments.

5.6.1. Hardware Limitations

The proposed framework has been experimentally validated using a testbed composed of three Raspberry Pi 3 Model B end devices and one Raspberry Pi 4 Model B acting as the main acquisition node. Raspberry Pi platforms were selected because they provide a flexible and reproducible environment representative of Linux-based IoT edge devices while allowing the controlled execution of cyberattacks and the integration of external power monitoring hardware.

However, this hardware platform does not encompass the full diversity of IoT devices currently deployed in practice. In particular, the selected devices run a full Linux operating

system and offer significantly greater computational resources than highly constrained microcontroller-based platforms. Consequently, the power consumption patterns generated by the evaluated cyberattacks may differ from those observed on lower-power embedded devices with different processor architectures, operating systems, and energy management mechanisms. In such constrained platforms, side-channel signatures may become less pronounced or exhibit different temporal characteristics due to lower computational activity and simpler execution models, potentially reducing the discriminative power of the proposed power-based features.

Therefore, although the experimental findings support the feasibility of combining network traffic with side-channel power measurements for intrusion detection, the proposed methodology should be regarded as a framework for which its deployment requires calibration to the target IoT platform rather than as a universal IDS that can be transferred unchanged across heterogeneous devices. Accordingly, further validation on heterogeneous IoT hardware is required before broader generalization can be claimed. Future work will evaluate the proposed methodology on additional IoT platforms with different computational capabilities and power consumption characteristics to better characterize its applicability across diverse deployment scenarios.

5.6.2. Methodological Limitations

Another limitation concerns the evaluation protocol adopted in this study. For experiments requiring model training and testing, the dataset was partitioned using a stratified random split after the generation of the temporal windows. Although this strategy ensures reproducibility and preserves the class distribution across the training and testing subsets, a strict chronological partition would constitute a more appropriate evaluation protocol for time-series data. Moreover, because the feature extraction process relies on overlapping temporal windows, random partitioning may introduce temporal information leakage between the training and testing sets, potentially leading to optimistic estimates of the reported classification performance.

Despite this limitation, the primary objective of this work is the comparative evaluation of different feature sets and experimental scenarios under a common methodology. In particular, the analyses of unseen attack detection (Scenario 2) and the comparison of power, network, and combined feature sets (Scenarios 3 and 4) were all conducted using the same training and evaluation protocol, allowing meaningful relative comparisons between the evaluated approaches. Future work will repeat the complete experimental campaign using strict chronological partitioning and time-aware validation strategies to obtain leakage-free performance estimates.

5.6.3. Deployment Considerations

From a deployment perspective, the proposed hybrid framework requires the acquisition of both network traffic and side-channel power measurements. While network monitoring can be performed using conventional packet capture mechanisms, incorporating side-channel measurements requires additional sensing hardware. Although commercially available sensors such as the INA3221 provide a low-cost solution, integrating power monitoring increases the complexity of deployment compared to conventional network-based intrusion detection systems.

Furthermore, the proposed architecture has not yet been evaluated in large-scale deployments, where communication overhead, synchronization between data sources, privacy, and data management may become relevant factors.

In this context, distributed privacy-preserving learning paradigms, such as federated learning with secure aggregation, represent a promising direction for enabling collabora-

tive intrusion detection across multiple IoT deployments while avoiding the exchange of sensitive local information [34].

5.6.4. Runtime Performance

Another limitation concerns the computational evaluation of the proposed framework. The experiments focused on detection performance and end-to-end attack detection latency, whereas the inference time of individual classifiers was not analyzed separately. Since data acquisition and feature extraction account for a significant portion of the overall detection delay, the contribution of classifier inference time is expected to be comparatively smaller under the current experimental configuration. Nevertheless, a detailed analysis of inference performance and scalability under larger deployments constitutes an interesting direction for future work.

6. Conclusions and Future Work

Some general conclusions are drawn from the results obtained in Section 5, in addition to the answers to the research questions proposed in Section 1.

The experimental results demonstrate that a hybrid system that combines both power data and network features can successfully detect attacks launched against IoT devices (RQ1). The system is also capable of determining the exact attack that is taking place (RQ2) and can also detect attacks different than the ones used to train the model (RQ3), although accuracy is not as high and is dependent on the model used and the attack launched.

A feature selection procedure was performed. The best set of features found includes power and six network features. It was proven that adding more features negatively impacts the performance of the model (RQ4).

The results obtained also show that combining power and network features significantly improves model accuracy compared to using only power or only network features (RQ5 and RQ6). Considering how uncommon these kind of hybrid approaches are in the existing literature, this could be a good starting point for further improving intrusion detection systems on IoT environments.

The proposed method was shown to be effective for real-time attack detection without taking too long (3.25 s on average) whilst maintaining an acceptable F1 score between 0.89 and 0.92 (RQ7).

Finally, the proposed solution is not intended to be a universal approach, as it requires a set of adaptations that depend on the nature of the IoT devices involved, the functions they perform, and the types of attacks they may face. Rather than providing a one-size-fits-all solution, we envision our work as a methodology for hardening unique and application-specific IoT environments. By leveraging the distinctive characteristics of each deployment, the proposed methodology enables the design of protection mechanisms that can be both more effective and more efficient.

The findings presented in this work should be interpreted within the scope of the experimental validation discussed in Section 5.6. In particular, further validation on heterogeneous IoT hardware and the adoption of time-aware evaluation protocols will be necessary to strengthen the generalizability of the proposed framework and to obtain leakage-free performance estimates. These limitations do not diminish the potential of the proposed hybrid approach but rather identify important directions for extending and consolidating the present work.

Several future lines of research are also proposed, which are listed below:

1. Validate the proposed framework on heterogeneous IoT devices with different hardware architectures and operating environments.

2. Repeat the experimental evaluation using chronological data partitioning and time-aware validation strategies.
3. Investigating federated learning with secure aggregation to improve the scalability and privacy of the proposed IDS in distributed IoT environments.
4. Testing different pre-processing techniques to determine whether the designed models can be improved.
5. Integrating and testing deep learning algorithms to improve the accuracy of the proposed system in IoT environments where security is vital, such as critical infrastructures.
6. Incorporating device power consumption data into existing network-based IDS to determine if the proposed mixed approach can improve attack detection rates.
7. Extend the detection system with a mitigation component in order to make it automatically react to detected attacks.

Author Contributions: Conceptualization, F.L.-P. and A.D.C.; methodology, F.L.-P.; software, A.D.C.; validation, F.L.-P., A.D.C., J.-L.G.-S. and A.C.L.; formal analysis, A.D.C.; investigation, A.D.C.; resources, F.L.-P.; data curation, F.L.-P. and A.D.C.; writing—original F.L.-P. and A.D.C.; writing—review and editing, F.L.-P., A.D.C., J.-L.G.-S. and A.C.L.; visualization, A.D.C.; supervision, F.L.-P., J.-L.G.-S. and A.C.L.; project administration, F.L.-P.; funding acquisition, F.L.-P. and A.C.L. All authors have read and agreed to the published version of this manuscript.

Funding: This work was supported by the Ministry of Education, Science and Vocational Training of the Regional Government of Extremadura (grant numbers: GR24208 and GR24170), the European Regional Development Fund (ERDF). We also thank the Extremadura Supercomputing Center (COMPUTAEX) for allowing us to use the resources of the LUSITANIA supercomputer.

Data Availability Statement: All source code developed for this work, together with the datasets used and the datasets generated during the project, are publicly available in the project's GitHub repository. This material has been released to ensure transparency, reproducibility, and reuse of the results presented in this study. The repository is available at <https://github.com/CenitS-COMPUTAEX/IOT/tree/v2> (accessed on 12 July 2026).

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

References

1. IoT Analytics. State of IoT 2025. Available online: <https://iot-analytics.com/number-connected-iot-devices/> (accessed on 5 May 2026).
2. United States Government Accountability Office. CRITICAL INFRASTRUCTURE Actions Needed to Better Secure Internet-Connected Devices. Available online: <https://www.gao.gov/assets/gao-23-105327.pdf> (accessed on 5 May 2026).
3. Dommari, S.; Jain, A. The impact of IoT Security on Critical Infrastructure Protection: Current Challenges and Future Directions. 2025. Available online: <https://ssrn.com/abstract=5259329> (accessed on 12 July 2026).
4. Shi, S.; Han, D.; Cui, M. A multimodal hybrid parallel network intrusion detection model. *Connect. Sci.* **2023**, *35*, 2227780. [CrossRef]
5. Kiflay, A.; Tsokanos, A.; Fazlali, M.; Kirner, R. Network intrusion detection leveraging multimodal features. *Array* **2024**, *22*, 100349. [CrossRef]
6. Hasan, T.; Tasnim, S. Generalizable Intrusion Detection for IoT: A Cost-Sensitive Multimodal Approach. In Proceedings of the 2025 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT), Lucca, Italy, 9–11 June 2025; pp. 163–170. [CrossRef]
7. Jaafar, F.; Ameyed, D.; Barrak, A.; Cheriet, M. Identification of Compromised IoT Devices: Combined Approach Based on Energy Consumption and Network Traffic Analysis. In Proceedings of the 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), Hainan, China, 6–10 December 2021; pp. 514–523. [CrossRef]
8. Lightbody, D.; Ngo, D.M.; Temko, A.; Murphy, C.C.; Popovici, E. Attacks on IoT: Side-Channel Power Acquisition Framework for Intrusion Detection. *Future Internet* **2023**, *15*, 187. [CrossRef]
9. Domínguez Campos, A.; Lemus-Prieto, F.; González-Sánchez, J.L.; Lindo, A.C. Intrusion detection for IoT environments through side-channel and Machine Learning techniques. *IEEE Access* **2024**, *12*, 98450–98465. [CrossRef]
10. Hettich, S.; Bay, S.D. *The UCI KDD Archive*; University of California, Department of Information and Computer Science: Irvine, CA, USA, 1999. Available online: <http://kdd.ics.uci.edu> (accessed on 12 July 2026).

11. Tavallae, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. A detailed analysis of the KDD CUP 99 data set. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*; IEEE: New York, NY, USA, 2009; pp. 1–6. [\[CrossRef\]](#)
12. Moustafa, N.; Slay, J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS)*, Canberra, ACT, Australia, 10–12 November 2015; pp. 1–6. [\[CrossRef\]](#)
13. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* **2018**, *1*, 108–116. [\[CrossRef\]](#)
14. Communications Security Establishment (CSE); The Canadian Institute for Cybersecurity (CIC). A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018). Available online: <https://registry.opendata.aws/cse-cic-ids2018/> (accessed on 5 May 2026).
15. Koroniotis, N.; Moustafa, N.; Sitnikova, E.; Turnbull, B. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Gener. Comput. Syst.* **2019**, *100*, 779–796. [\[CrossRef\]](#)
16. Moustafa, N. A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets. *Sustain. Cities Soc.* **2021**, *72*, 102994. [\[CrossRef\]](#)
17. Deng, H.; Runger, G.; Tuv, E.; Vladimir, M. A time series forest for classification and feature extraction. *Inf. Sci.* **2013**, *239*, 142–153. [\[CrossRef\]](#)
18. Cabello, N.; Naghizade, E.; Qi, J.; Kulik, L. Fast and accurate time series classification through supervised interval search. In *Proceedings of the 2020 IEEE International Conference on Data Mining (ICDM)*; IEEE: New York, NY, USA, 2020; pp. 948–953.
19. Christ, M.; Braun, N.; Neuffer, J.; Kempa-Liehr, A.W. Time series feature extraction on basis of scalable hypothesis tests (tsfresh—a python package). *Neurocomputing* **2018**, *307*, 72–77. [\[CrossRef\]](#)
20. Middlehurst, M.; Bagnall, A. The freshprince: A simple transformation based pipeline time series classifier. In *Proceedings of the International Conference on Pattern Recognition and Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 150–161.
21. Guillaume, A.; Vrain, C.; Elloumi, W. Random dilated shapelet transform: A new approach for time series shapelets. In *Proceedings of the International Conference on Pattern Recognition and Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 653–664.
22. Schäfer, P.; Leser, U. Multivariate time series classification with WEASEL + MUSE. *Data Min. Knowl. Discov.* **2018**, *32*, 1475–1498.
23. Dempster, A.; Petitjean, F.; Webb, G.I. ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Min. Knowl. Discov.* **2020**, *34*, 1454–1495. [\[CrossRef\]](#)
24. Middlehurst, M.; Large, J.; Flynn, M.; Lines, J.; Bagnall, A. HIVE-COTE 2.0: A new meta ensemble for time series classification. *Mach. Learn.* **2021**, *110*, 3219–3248. [\[CrossRef\]](#)
25. Middlehurst, M.; Schäfer, P.; Bagnall, A. Bake off redux: A review and experimental evaluation of recent time series classification algorithms. *arXiv* **2023**, arXiv:2304.13029. [\[CrossRef\]](#)
26. Dempster, A.; Schmidt, D.F.; Webb, G.I. MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*; Association for Computing Machinery: New York, NY, USA, 2021; pp. 248–257. [\[CrossRef\]](#)
27. Texas Instruments. INA3221: 26-V, Three-Channel Current and Bus Voltage Monitor with Alerts. Revision C. 2015. Available online: <https://www.ti.com/lit/ds/symlink/ina3221.pdf> (accessed on 3 July 2026).
28. Pruvot, T.; Jones, L. Cpuminer. Available online: <https://github.com/tpruvot/cpuminer-multi> (accessed on 5 May 2026).
29. van Hauser Heuse, M. Hydra. 2022. Available online: <https://github.com/vanhauser-thc/thc-hydra> (accessed on 5 May 2026).
30. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
31. Löning, M.; Bagnall, A.; Ganesh, S.; Kazakov, V.; Lines, J.; Király, F.J. sktime: A unified interface for machine learning with time series. *arXiv* **2019**, arXiv:1909.07872.
32. Aeon Contributors. Aeon. Available online: <https://github.com/aeon-toolkit/aeon> (accessed on 5 May 2026).
33. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [\[CrossRef\]](#)
34. Kumar, D.; Pawar, P.P.; Meesala, M.K.; Pareek, P.K.; Addula, S.R.; Shwetha, K.S. Trustworthy iot infrastructures: Privacy-preserving federated learning with efficient secure aggregation for cybersecurity. In *Proceedings of the 2024 International Conference on Integrated Intelligence and Communication Systems (ICIICS)*; IEEE: New York, NY, USA, 2024; pp. 1–8.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.