scientific reports

Check for updates

OPEN

A generalized three-tier hybrid model for classifying unseen (IoT devices) in smart home environments

Quadri Waseem[1], Wan Isni Sofiah Wan Din[1]✉ & Muhammad Aamir[2]

Data drift caused due to network changes, new device additions, or model degradation alters the patterns learned by ML/DL models, resulting in poor classification performance. This creates the need for a generalized, drift-resilient model that can learn without retraining in dynamic environments. To maintain high accuracy, such a model must classify previously unseen IoT devices effectively. In this study, we propose a three-tier incremental architecture (CNN-PN-RF) combining Convolutional Neural Network (CNN) for feature extraction, Prototypical Network (PN) for class embedding, and Random Forest (RF) for robust classification. The model utilizes six aggregated diverse IoT datasets. Two similarly structured datasets (Dataset 1 and Dataset 2) were created from it, differing in training-testing splits, with some device CSV files withheld to test on unseen classification. Phase 1 employs a stand-alone CNN-based model with L2 regularization, dropout, and early stopping, achieving 70.96% accuracy. Phase 2 integrates CNN with RF, using SMOTE for class balancing and PCA for dimensionality reduction, attaining 83.79% accuracy. Phase 3 introduces PN to finalize the CNN-PN-RF model, enhancing classification issue of feature clustering, intra-class separability, and small-class support. Final accuracy, precision, recall, and F1-score were 99.56%, 99.66%, 99.56%, and 99.59% for Dataset 1, and 99.80% for all metrics on Dataset 2. The model was compared with state-of-the-art approaches and validated on unseen IoT subsets of both datasets, showing better generalization capability.

Keywords Data Drift, Unseen IoT devices, Classification, CNN, Prototypical networks, RF

Internet of Things (IoT) adoption is expected to grow to 75 billion devices by 2025[1]. The need for efficient unseen device classifications has become increasingly important for device management, network security, and performance optimization in smart environments[2–4]. The classification task becomes complicated by the need for interoperability and changing network configurations[5–7]. Moreover, the increasing diversity of IoT devices and their data heterogeneity, along with each IoT device having its own set of requirements and Quality of Service (QoS) parameters, further complicates the process of unseen IoT device classification. AI-driven approaches, in particular machine learning (ML) and deep learning (DL), are widely used to process large datasets and networks, and to identify unique device patterns within data for efficient IoT device classification[8,9]. Their ability to detect hidden patterns strengthens anomaly detection, mitigates network congestion, and improves oversight within IoT ecosystems[10,11]. However, traditional ML/DL models often suffer from poor generalization when confronted with unseen/unknown/new IoT devices or exposed to shifting network behaviors, network upgrades, or natural model-accuracy degradation[12,13]. Additionally, these models often use the same data for training and testing. Their architectural design fails to capture real-world variations, leading to performance degradation over time[14,15]. Frequent retraining and continuous updates of ML/DL models are necessary to handle data distribution shifts (commonly known as data drifts)[16]. Failure to address these data drift effects can result in security vulnerabilities, resource inefficiencies, device misbehavior, and higher computational costs, which can prevent their large-scale deployment. Hence, a well-generalized model should be resilient to the data drift effect and must recognize previously unseen devices by leveraging learned patterns without having to constantly retrain or update externally[17].

Recent studies have explored various approaches to investigate generalization in several single ML/DL models to reduce their retraining burdens[18–20]. While these models can perform well in controlled scenarios, they

[1]Faculty of Computing, Universiti Malaysia Pahang Al-Sultan Abdullah, Pekan, Pahang 26600, Malaysia. [2]Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, OX1 3QD Oxford, UK. ✉email: sofiah@umpsa.edu.my

remain ill-equipped to manage the continuous, large-scale data streams generated by diverse IoT environments. Likewise, most of the single ML models[21,22], particularly those relying on manual feature engineering, hamper models scalability and adaptability in dynamic contexts. Eventually, single DL models have addressed a few limitations of ML models (getting control of the overlapping device category issue) by automatically extracting features from raw data, eliminating manual engineering, and providing better classification results[23,24]. Yet, they are still limited in IoT environments where complex nonlinear relationships exist in the data and also classification performance on small samples cannot be avoided. Furthermore, their practicality in real-world IoT deployments is hindered by the overfitting issue. They require large, diverse datasets for training along with robust hyperparameter optimization and significant computational resources[25].

CNNs are capable of learning effectively and efficiently even on non-image tasks, such as network traffic classification in IoT environments. They can learn spatially and temporally correlated patterns of features[26,27] fast and proficiently, especially when data is tabulated and presented as a statistical dataset.The statistical, temporal, and protocol-level features are structured directly into a matrix format for CNN input. This allows the convolutional filters to learn cross-feature interactions and local correlations that are not usually learned by dense networks. Thus, the CNNs can offer intra-class variability robustness, parameter efficiency due to weight sharing, and effective local pattern extraction[28,29]. CNN's, when used as a feature extractor, can facilitate unseen device classification without the need for retraining. Additionally, CNNs are highly effective at extracting subtle features from input data, capturing key spatial and hierarchical patterns that are critical for accurate classification.

Prototypical networks, being a type of metric-based Few-shot learning, are utilized for classification techniques in various IoT scenarios[30] that allow for significant generalization due to the similarity measure, such as the Euclidean distance, where distances are computed to prototypes of the classes[31]. Unlike the traditional models that require huge labeled datasets[32], they can adapt to unknown instances. Likewise, previous authors[33] utilized them in a classification-by-class approach. They reduce retraining costs in the classification of IoT devices to designing generic representations of classes that place unseen data in categories based on their similarity to prototypes, which is accurate, flexible and successful[34].

Random Forest (RF), a machine learning technique that builds an ensemble of decision trees to improve classification performance.They reduce overfitting by employing bagging and random feature selection[35,36]. RF performs well, particularly for IoT device classification, where specific network traffic metrics are extracted and used to differentiate devices[37]. Prior works have shown that RF handles heterogeneous and high-dimensional IoT traffic data with excellent accuracy.

Hybrid models combining ML and DL have also been explored[38–40], showing improved classification in complex datasets. However, they still face issues such as overlapping device categories, nonlinear dependencies, and small-category imbalance[41–43]. Their unseen evaluation is often unjustified, limiting their cross-domain generalization.

To address these gaps, this research proposes a generalized hybrid CNN–PN–RF model that integrates CNN for feature extraction, PN for few-shot generalization, and RF for robust classification. The model is explicitly designed to handle unseen IoT devices and evolving network environments with minimal computational cost, improved scalability, and strong adaptability.

Main objective and contribution

This research aims to enhance unseen IoT device classification through a generalized three-tier hybrid CNN–PN–RF model. The main contributions are:

1. An aggregated dataset was prepared by merging six publicly available datasets to create a generalized and diverse dataset comprising 82 features.
2. An incremental hybrid model was developed starting from a standalone CNN with (70.96% accuracy), then CNN–RF with (83.79% accuracy), and finally CNN–PN–RF with (99.56% and 99.80% accuracy on two aggregated datasets).
3. The proposed model was validated on two datasets Dataset 1 and Dataset 2 with test and train subsets, enabling unseen classification without retraining.
4. Comparative analysis showed that the proposed hybrid model outperforms state-of-the-art across multiple evaluation metrics.

Research questions

The following RQs guide this study:

- RQ1: How to develop an effective three-tier incremental model from phase 1 to phase 2 and then to phase 3 for unseen IoT device classification?
- RQ2: What are the additional tuning processes utilized at each phase, along with incremental evolution?
- RQ3: Does PN between CNN and RF enhance classification performance and generalization in phase 3?
- RQ4:How effective is this hybrid phase 3 CNN–RF-RF in comparison with phase 1 CNN and stage 2 CNN–PN?
- RQ5: How does the proposed model compare with state-of-the-art methods on unseen device classification?

Research motivation and gap analysis

The motivation for this research originates from the limitations in the generalization capabilities of current machine learning (ML) as well as deep learning (DL) models for IoT device classification. They are unable to handle the concept of data drift and performance degradation caused by network changes, new device additions, or configuration updates[12,13,44]. These traditional single ML models tend to need frequent retraining. They are not scalable or adaptable, while as single DL models are capable of better feature extraction[45]. Yet, they fail to achieve perfect generalization, even when used in combined/hybrid models[46].

To overcome these challenges, this study proposes a generalized hybrid model that combines convolutional neural networks (CNN) for powerful feature extraction, prototypical networks (PN) for few-shot generalization, and random forests (RF) for robust and efficient classification. This CNN–PN–RF integration eliminates the need for manual preprocessing, feature engineering, protocol dependence, and frequent retraining, all of which are common shortcomings in prior work (detailed in the next section: **Related Works**). This proposed model is explicitly designed to handle unseen devices and evolving network environments with minimal computational cost and no performance drop, unlike traditional models that suffer performance decay over time[47]. This model was trained on six combined diverse IoT datasets spanning a wide range of device categories and communication protocols and was evaluated separately. It showed strong cross-domain generalization. Additionally, high accuracy was achieved without any dataset-specific tuning. The model not only offers superior scalability and long-term stability but also achieves reliable performance in real-world and resource-constrained IoT deployments. By bridging ML and DL techniques in a unified hybrid model with strong architectural design, this approach addresses a pressing research gap and sets a new benchmark for generalizable unseen IoT device classification, achieving high adaptability, reduced operational overhead, and improved real-world applicability.

Organization of the paper
The remainder of this paper is organized as follows. Section 2 reviews the related work, while Section 3 presents the proposed methodology. Section 4 describes the architectural design, and Section 5 discusses the incremental model development. Section 6 outlines the experimental tools and setup, followed by Section 7, which presents the results. Section 8 provides a comparison with state-of-the-art approaches, and Section 9 discusses the key findings. Finally, Section 10 concludes the paper and highlights future research directions.

Related works
The unseen classification of IoT devices has gained much attention in recent years, and numerous studies have been conducted over a very diverse range of models, including machine learning (ML) models, deep learning (DL) models, and hybrid models. Despite demonstrating high performancein certain limited experimental settings, a large proportion of these studies have experienced unremitting challenges in dealing with unseen classification with robust generalization. The issue includes factors such as low scalability, use of handcrafted features, or lack of dynamic environments, responsiveness of static features, and inadequate generalization of previously unseen devices. These studies have been classified into two main subsections as follows:

A. Feature engineering-based models
Cvitic et al.[48] developed an ensemble-based machine learning solution to IoT device classification with network traffic features in a smart home. They also applied logistic regression and other supervised learning methodologies. They trained their model on a proprietary dataset comprising of 41 devices. Through the application of 13 important characteristics of network traffic, the framework dynamically classified devices and exhibited a high level of accuracy of classification with 99.79%. In spite of its good performance, the method had some weaknesses when it comes to implementation in real-life settings because it was limited by the device diversity, fixed test settings, and the inability to scale to dynamic network settings.

Kostas et al.[49] used standard ML models, along with multistage feature selection and genetic algorithms, to maximize the performance of classification. They recorded an accuracy of 83.30% and 94.30%on the Aalto and UNSW datasets, respectively. The model that they used worked fairly well but was dependent on expertly designed features and did not generalize to previously unseen device types, limiting its applicability and robustness in open-world deployment.

The technique of Aqil et al.[47] proposed a temporally aware method of identifying IoT devices based on robust statistical features. Their model presented an average accuracy of 85 percent 85% on IoT Traffic Traces (2018) and 96 percent 96% on IoT-FCSIT (2022) datasets. However, the suggested method was ineffective when working with encrypted traffic or very dynamic traffic, and they did not present critical evaluation parameters (e.g., precision, F1-score, or recall), needed for comprehensive evaluation.

Xu et al.[50] introduced a fine and lightweight architecture of ML in resource-constrained settings. They obtained 99.08%, 98.15%, and 95.28% percent precision on the CIC, UNSW, and SMPS datasets, respectively. The model focused on efficiency in its performance, yet it did not consider behavior variability and encryption in network traffic. The test was also limited to semi-controlled environments and, therefore was not suitable for realistic implementations.

Fan et al.[51] proposed a semi-supervised learning strategy using convolutional neural networks to reduce reliance on labeled data. On the UNSW dataset, the proposed method achieved 99% accuracy. However, recall and F1-score were not reported as key performance indicators, and therefore it did not demonstrate robustness in open-set conditions.

Niu et al.[52] developed a stacked ensemble learning model trained on the UNSW and TMA-2021 datasets, reporting high performance with an accuracy of over 98% on both datasets. Despite these promising metrics, the models showed high computational complexity, required frequent retraining to include new devices, and lacked scalability for large-scale IoT deployments.

B. Architecture-based models
Kotak et al.[53] studied deep learning approaches for IoT device identification using a public dataset with 10 devices, achieving an accuracy of 99%. Although high accuracy was achieved, the evaluation was constrained

to a small set of known devices and did not address performance across multiple communication protocols or diverse device categories, thereby limiting the model's generalization capabilities.

Deng et al.[54] proposed a hybrid model integrating Transformer-based tokenization with clustering for open-set device identification, achieving 99.89% and 99.68% accuracy on the UNSW and YourThings datasets, respectively. Their model demonstrated strong performance on encrypted traffic. However, it was not validated against previously unseen devices, leaving its real-world applicability uncertain.

Bao et al.[41] introduced a hybrid deep learning model combining supervised and unsupervised learning with clustering and dimensionality reduction. While their results showed an average accuracy between 81.8% and 92.9%, other critical evaluation metrics were not reported. A key limitation was its reliance on easily spoofed features such as MAC addresses. Moreover, the model was data-integrity dependent and computationally intensive, making it unsuitable for real-time applications in resource-constrained environments. It also required frequent data updates to accommodate newly added devices.

Liu et al.[55] proposed a 1D convolutional neural network using directional packet length sequences to reduce manual feature engineering and improve accuracy. Their model achieved promising results (99% across all metrics) while eliminating the need for handcrafted features. However, it relied solely on time-series-based features and did not consider payload, statistical, or header-based features. Additionally, the dataset used had an imbalanced distribution of data instances per device.

Yin et al.[56] introduced GraphIoT, a lightweight identification model based on graph neural networks and incremental learning for IoT classification. With an F1-score of up to 96.37%, the model transformed traffic data into IoT Device Traffic Graph Representations (IoT-DTGRs), utilizing node, edge, and subgraph features for improved classification. The model adapts to new devices without retraining. However, it requires continuous hyperparameter tuning and involves complex graph construction, making it sensitive to changes in edge attributes and thus less practical for large-scale dynamic environments.

Table 1 provides a detailed summary of existing literature.In contrast, our proposed generalized three-tier hybrid CNN–PN–RF model offers full generalization and provides an end-to-end, protocol-agnostic solution for unseen IoT device classification. By integrating the deep feature extraction capabilities of Convolutional Neural Networks (CNN), the few-shot learning strengths of Prototypical Networks (PN), and the robust classification performance of Random Forest (RF), the model achieves strong class-wise identification and generalization. It has been validated on blind test data entirely excluded from training, demonstrating effectiveness in identifying previously unseen devices. Unlike conventional methods, this proposed approach does not require retraining, supports multi-protocol and multi-source data (spanning six diverse datasets), and operates effectively without extensive preprocessing or manual feature engineering.

Methodology

The detailed step-by-step methodology and corresponding subsections used to develop this generalized hybrid model are illustrated in Fig. 1.

Data collection

In this research, six widely used public datasets were utilized, which include:

1. **UNSW Dataset:** https://iotanalytics.unsw.edu.au/iottraces.html
2. **ShIoT Dataset:** ShIoT Dataset Link
3. **Ping-Pong Dataset:** https://athinagroup.eng.uci.edu/projects/pingpong/data/

Paper	Type	Dataset Summary	Key Limitations/Target
A. Feature Engineering-Based Models			
[48](Cvitić et al., 2021)	Ensemble Learning	Personal dataset (Primary and secondary) – 41 devices	Limited device diversity, static dataset, feature selection constraints, only 4 class types, high computational load, and limited real-world validation
[49](Kostas et al., 2022)	Machine Learning	2 Public Datasets – Aalto and UNSW	Relies on handcrafted features, challenges with unseen devices
[47] (Aqil et al., 2024)	Statistical Analysis	2 datasets (Public IoT Traffic Traces (2018) and Private IoT-FCSIT)	Manual feature engineering, not tested on unseen devices
[50] (Z. Xu et al., 2025)	Machine Learning	CIC, UNSW, and SMPS datasets	Lacks generalization across devices with changing behavior or encrypted traffic
[51] (Fan et al., 2020)	Semi-supervised CNN	Public Dataset – UNSW	Doesn't test on unseen devices, limited adaptability to real-world scenarios
[52] (Niu et al., 2024)	Ensemble Learning	Public Dataset – UNSW and TMA-2021	Increased complexity, requires continuous updates
B. Architectural-Based Models			
[53] (Kotak & Elovici, 2021)	Deep Learning	Public Dataset (10 Devices)	Limited to known devices, struggles with multi-protocol environments
[54](Deng et al., n.d.)	Transformer & Clustering	Public Dataset – UNSW and YourThings	Complex architecture, not specifically tested for generalization to unseen devices
[41] (Bao et al., 2020)	Hybrid (DL+ML)	10 IoT device types from various sources	Feature vulnerability, dependency, ongoing updates needed, large feature sets, deep network complexity
[55](X. Liu et al., 2022a)	Deep Learning	3 Datasets – Aalto, UNSW, IoTFinder	Unbalanced dataset, high data requirement, high latency, lack of compatibility, no data augmentation
[56] (Y. Yin et al., 2024)	Graph Neural Network	2 Datasets – real-world and open-source	Graph construction overhead, subgraph complexity, scalability, generalization to unseen behavior, flow/edge feature sensitivity

Table 1. Summary of existing literature.

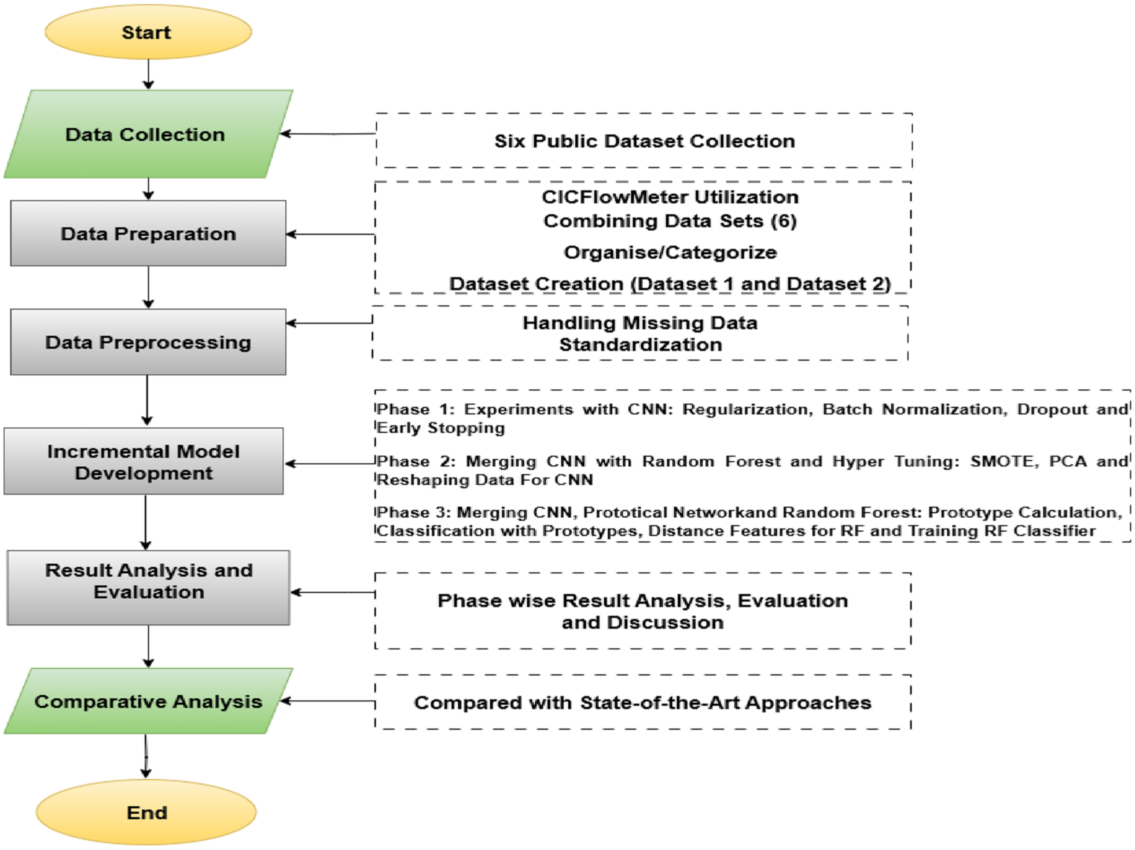
Start
Data Collection
Six Public Dataset Collection
Data Preparation
CICFlowMeter Utilization
Combining Data Sets (6)
Organise/Categorize
Dataset Creation (Dataset 1 and Dataset 2)
Data Preprocessing
Handling Missing Data
Standardization
Incremental Model Development
Phase 1: Experiments with CNN: Regularization, Batch Normalization, Dropout and Early Stopping
Phase 2: Merging CNN with Random Forest and Hyper Tuning: SMOTE, PCA and Reshaping Data For CNN
Phase 3: Merging CNN, Prototical Networkand Random Forest: Prototype Calculation, Classification with Prototypes, Distance Features for RF and Training RF Classifier
Result Analysis and Evaluation
Phase wise Result Analysis, Evaluation and Discussion
Comparative Analysis
Compared with State-of-the-Art Approaches
End

Fig. 1. Research methodology.

4. **IoT Sentinel Dataset:** https://github.com/andypitcher/IoT_Sentinel
5. **IoT Finder Dataset:** https://yourthings.info/data/
6. **Home Mole Dataset:** https://github.com/DongShuaike/iot-traffic-dataset

We analyzed these six datasets one by one and found that they lacked data variety, especially in the type of information presented in their columns. For example, the UNSW dataset mostly uses MAC and IP addresses, many of which are repeated frequently. In numerous packets, the same addresses appear as both the source and the destination. MAC addresses (like "ec:1a:59:79:f4:89", "ec:1a:59:83:28:11") and IP addresses (like "192.168.1.223", "192.168.1.193") show up repeatedly across multiple packets. This repetition limits the data's diversity within individual packets/files and across IoT device files, making it difficult for the model to learn varying network behaviors. Similar issues were observed in the ShIoT, Ping-Pong, IoT Sentinel, IoT Finder, and Home Mole datasets, a common limitation in many IoT-related studies, i.e., the failure to capture the richness of device interactions.

To improve generalization, a multi-dataset approach was adopted by integrating multiple sources to create a more representative dataset while maintaining consistency in common device categories. We selected six IoT device categories with variations in manufacturer specifications, firmware versions, and configurations. The corresponding datasets include raw IoT traffic and PCAP packet traces. For example, although Amazon Echo devices are present in both the UNSW and ShIoT datasets, differences in firmware and hardware result in different MAC and IP addresses for each, adding to dataset diversity. Similarly, IoT Finder's D-Link DSC-50009L camera shares functional similarities with IoT Sentinel's D-Link DayCam, but they are not the same device. By using all six datasets, we created a more diverse, representative, and reliable depiction of IoT network behavior, helping minimize biases and improve generalization in our findings.

Data preparation

To develop a robust and generalizable CNN model for classifying unseen IoT devices, we transformed raw PCAP network traffic into structured data through effective feature extraction. This step was essential for capturing diverse behavioral patterns and avoiding overfitting to specific device signatures. We used CICFlowMeter , an open-source tool widely adopted for flow-based network analysis. It extracts 82 standardized statistical features from packet captures, including flow duration, packet/byte counts, ports, protocol types, and timestamps, and outputs the structured data in CSV format. These features provided a comprehensive representation of each network flow in every dataset and served as the foundation for our analysis. The complete list of features used in our experiments is presented in Table 2.

No.	Name	Description	No.	Name	Description
1	src_ip	Source IP address	42	bwd_iat_tot	Total backward inter-arrival time
2	dst_ip	Destination IP address	43	bwd_iat_max	Maximum backward inter-arrival time
3	src_port	Source port number	44	bwd_iat_min	Minimum backward inter-arrival time
4	dst_port	Destination port number	45	bwd_iat_mean	Mean backward inter-arrival time
5	protocol	Protocol used	46	bwd_iat_std	Std deviation of backward IAT
6	timestamp	Timestamp of the flow	47	fwd_psh_flags	Forward push flags
7	flow_duration	Duration of the flow	48	bwd_psh_flags	Backward push flags
8	flow_byts_s	Flow bytes per second	49	fwd_urg_flags	Forward urgent flags
9	flow_pkts_s	Flow packets per second	50	bwd_urg_flags	Backward urgent flags
10	fwd_pkts_s	Forward packets per second	51	fin_flag_cnt	Count of FIN flags
11	bwd_pkts_s	Backward packets per second	52	syn_flag_cnt	Count of SYN flags
12	tot_fwd_pkts	Total forwarded packets	53	rst_flag_cnt	Count of RST flags
13	tot_bwd_pkts	Total backward packets	54	psh_flag_cnt	Count of PSH flags
15	totlen_fwd_pkts	Total length of forwarded packets	55	ack_flag_cnt	Count of ACK flags
15	totlen_bwd_pkts	Total length of backward packets	56	urg_flag_cnt	Count of URG flags
16	fwd_pkt_len_max	Max forward packet length	57	ece_flag_cnt	Count of ECE flags
17	fwd_pkt_len_min	Min forward packet length	58	down_up_ratio	Downstream to upstream ratio
18	fwd_pkt_len_mean	Mean forward packet length	59	pkt_size_avg	Average packet size
19	fwd_pkt_len_std	Std dev of forward packet length	60	init_fwd_win_byts	Initial forward window size
20	bwd_pkt_len_max	Max backward packet length	61	init_bwd_win_byts	Initial backward window size
21	bwd_pkt_len_min	Min backward packet length	62	active_max	Maximum active time
22	bwd_pkt_len_mean	Mean backward packet length	63	active_min	Minimum active time
23	bwd_pkt_len_std	Std dev of backward packet length	64	active_mean	Mean active time
24	pkt_len_max	Maximum packet length	65	active_std	Std dev of active time
25	pkt_len_min	Minimum packet length	66	idle_max	Maximum idle time
26	pkt_len_mean	Mean packet length	67	idle_min	Minimum idle time
27	pkt_len_std	Std dev of packet length	68	idle_mean	Mean idle time
28	pkt_len_var	Variance of packet length	69	idle_std	Std dev of idle time
29	fwd_header_len	Forward header length	70	fwd_byts_b_avg	Avg bytes bulk rate (fwd)
30	bwd_header_len	Backward header length	71	fwd_pkts_b_avg	Avg packets bulk rate (fwd)
31	fwd_seg_size_min	Min forward segment size	72	bwd_byts_b_avg	Avg bytes bulk rate (bwd)
32	fwd_act_data_pkts	Forward actual data pkts	73	bwd_pkts_b_avg	Avg packets bulk rate (bwd)
34	flow_iat_mean	Mean flow IAT	74	fwd_blk_rate_avg	Avg forward block rate
35	flow_iat_max	Max flow IAT	75	bwd_blk_rate_avg	Avg backward block rate
36	flow_iat_min	Min flow IAT	76	fwd_seg_size_avg	Avg forward segment size
37	flow_iat_std	Std dev of flow IAT	77	bwd_seg_size_avg	Avg backward segment size
38	fwd_iat_tot	Total forward IAT	78	cwe_flag_count	Count of CWE flags
39	fwd_iat_max	Max forward IAT	79	subflow_fwd_pkts	Subflow forward packets
40	fwd_iat_min	Min forward IAT	80	subflow_bwd_pkts	Subflow backward packets
41	fwd_iat_mean	Mean forward IAT	81	subflow_fwd_byts	Subflow forward bytes
42	fwd_iat_std	Std dev of forward IAT	82	subflow_bwd_byts	Subflow backward bytes

Table 2. Common features extracted from each of the six datasets.

This study integrated six publicly available IoT datasets: SHIoT, IoT Sentinel, UNSW, IoT Finder, Ping-Pong, and HomeMole. Initially, these datasets consisted of multiple CSV files corresponding to various IoT devices but lacked a consistent labeling scheme. To unify them, CICFlowMeter was used to extract flow-level features, after which each device was manually categorized into one of six functional categories: Smart Speakers, Media Streaming Devices, Home Automation, Home Security Cameras, Smart Home Hubs/Controllers, and Home Appliances. Following standardization, the datasets were merged while maintaining consistent category labels across all sources. This balanced composition was crucial to ensure that the model learned generalized patterns across device types rather than overfitting to individual devices.

To critically evaluate generalization, we generated two similarly structured datasets (Dataset 1 and Dataset 2) based on the same six source datasets, with identical classes and devices. Theydiffered in their training-testing splits; in each, we randomly withheld some device CSV files for testing only, to assess unseen classification,, but the withheld files were different in the two datasets. This approach tested the model's ability to identify category-level behavioral patterns instead of memorizing device signatures. The whole dataset collection and preparation details are illustrated in Fig. 2.

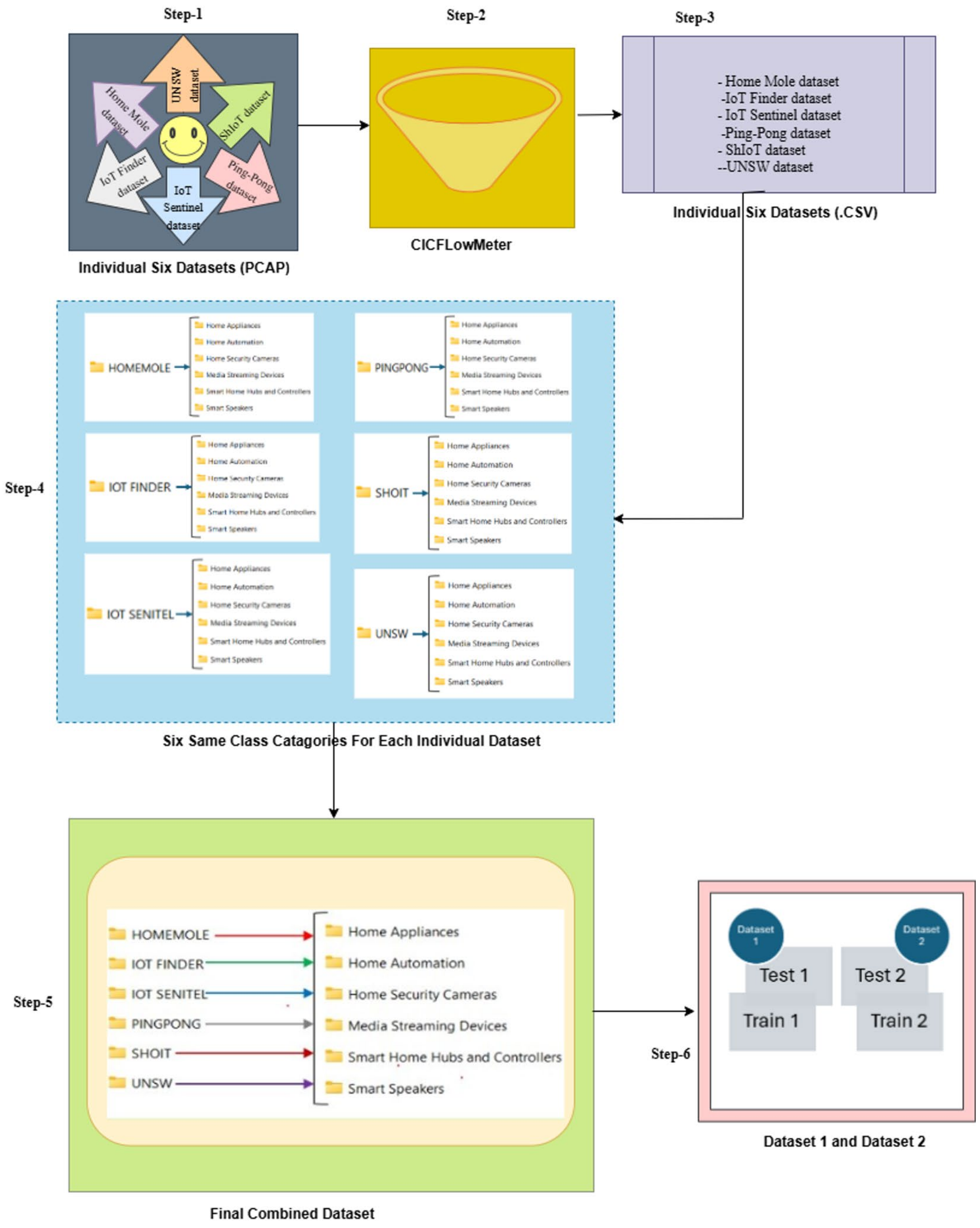

Fig. 2. Dataset Collection and Preparation.

Preprocessing

The first step was to load and merge the data of several CSV files into one data frame. This was crucial in forming a single dataset that can be used for model analysis and training. Data files were maintained in an organized manner in separate folders, reflecting the various categories of devices and a well-structured approach to data management. Reading of each CSV file into a DataFrame was performed by using Pandas functions such as `pd.read_csv()`, after which they were concatenated to form a larger DataFrame. During this process, a column, the label column, indicating the category of each device, was kept, thus retaining the required data for supervised learning.

The `load_data_from_folder()` function reads the contents of each CSV file in a specific folder, iterates through each file, and labels it accordingly. In particular, it uses the library `pandas` to load each file as a `DataFrame`.

Considering an example, a file will be read when it is found in the directory by using $D_i = \text{pd.read\_csv}(\text{file\_path})$.

in which D_i is the DataFrame of the i^{th} file. This organized format allows loading the data efficiently so that further data manipulation and analysis would be more convenient.

On the second stage of data preprocessing, the file paths are used to extract the labels to categorize the files appropriately. As an example, files with the label `'smart_speaker'` are placed together. Once all the individual DataFrames are labeled, a final dataset of shape $(N, M+1)$ where N is the total number of data entries and M is the number of features in each file will be obtained by using `pd.concat()`. To ensure the quality of data, empty files are skipped.

$$S = \{D_i \mid D_i \neq \emptyset\} \tag{1}$$

Rows with missing labels are removed using `dropna()`, ensuring data integrity and resulting in a dataset of shape $(N', M+1)$, where $N' \leq N$.

In this research, missing values were handled using the SimpleImputer function with the strategy='mean' option. Any missing values are replaced by the mean of each numeric feature column using this function. Before this stage, non-numeric values were converted to numeric form and invalid entries such as NaN, inf, and -inf were standardized by replacing them with NaN. By ensuring that the imputation approach could be used consistently and accurately across the dataset, this maintained the data integrity for subsequent preprocessing and model training.

To improve model stability, features are standardized using `StandardScaler()`, ensuring a mean of 0 And a standard deviation of 1. This step helps normalize the data and enhances the performance of machine learning models.

Categorical features are processed by separating the features and labels. The feature matrix is created using $X = \text{combined\_df.drop('label', axis} = 1)$, with a size of (N', M), while the label vector is extracted as $y = \text{combined\_df['label']}$, with a size of $(N', 1)$. Labels are then converted into numerical values using `LabelEncoder()`, transforming categories into unique integers, where

$$y_{\text{encoded}} \in \{0, 1, 2, \ldots, C-1\} \tag{2}$$

and C represents the number of unique categories. For instance, if there are six classes, labels are assigned values from 0 to 5. Non-numeric features are converted into numeric values using `LabelEncoder()`, ensuring that all feature columns are in numerical format. Each non-numeric column is transformed separately, resulting in a fully numeric feature matrix represented as

$$X_{\text{encoded}} \in \mathbb{R}^{(N' \times M)} \tag{3}$$

Table 3 displays a conceptual change made to the dataset during the preparation stage. The word "conceptual" is used because the table is a sample abstraction meant to demonstrate how significant preprocessing steps were implemented, rather than a verbatim duplicate of the raw dataset. A simplified example is given for clarification because the collection's size (more than X million entries) makes it impractical to display every value.

The adjustments shown include imputation of missing data, normalization, and category encoding. This conceptual approach provides a clear illustration of how unprocessed IoT traffic elements were systematically transformed into modeling-suitable inputs.

This consolidated dataset is then structured and formatted to be compatible with machine learning models in the final preprocessing stage. X_{encoded} and y_{encoded} are transformed into numeric representations for training and classification purposes. Specifically, it X_{encoded} takes the form (N', M), where N' is the number of samples and M is the number of extracted features. y_{encoded} is a label vector of shape $(N', 1)$, containing one encoded label per sample. This well-defined structure guarantees that the dataset is ready for input into standard machine learning pipelines.

Feature selection

IoT device classification is heavily influenced by feature selection, which shapes model accuracy and generalization[57]. In this phase, the dataset is refined to contain a well-defined yet complete set of features. For this study, all 82 original features were included, along with a labeling feature, resulting in a total of 83 columns

Feature	Before Preprocessing (Raw Dataset)	After Preprocessing (Processed Dataset)	Transformation Applied
Packet Size	Missing values (NaN) in some records	Missing values imputed with mean (e.g., 1200.0)	Imputation using SimpleImputer (Mean)
Flow Duration	Large heterogeneous values (e.g., 34,000 µs, 150,000 µs)	Standardized values (z-score normalization, e.g., 0.15, –1.20)	Standardization with StandardScaler
Protocol	Text categories: {TCP, UDP, ICMP}	Encoded as integers: {0, 1, 2}	Encoding with LabelEncoder
Device Class (Target Variable)	Semantic labels: {Smart Speaker, Smart Camera, Smart TV,...}	Encoded as integers: {0 = Smart Speaker, 1 = Smart Camera, 2 = Smart TV,...}	Encoding with LabelEncoder
Other Numeric Features	Raw values with varying scales (e.g., Bytes Sent, Packets/sec)	Standardized (mean = 0, std = 1)	Standardization with z-score
Other Categorical Features	Non-numeric labels (e.g., "Established")	Converted to numeric codes (e.g., 0 = No, 1 = Yes)	Encoding with LabelEncoder

Table 3. Dataset transformation: conceptual view of selected features before vs. after preprocessing.

per device file. All features are important, as each brings a different perspective to the data, capturing fine-grained patterns and relationships in IoT behavior. This comprehensive inclusion minimizes information loss and enables the model to learn from the full range of device characteristics. Additionally, exposure to diverse data patterns increases robustness, allowing the model to generalize well to unseen devices, which is crucial in the heterogeneous environment of IoT[58]. Moreover, by preserving all attributes, the model becomes resilient to changes in device behavior (e.g., data drift, new devices, and system upgrades), resulting in better predictive performance across heterogeneous datasets. Maintaining a wide feature set adds adaptability, ultimately improving classification accuracy and generalization, and making the classification more reliable in real-world IoT settings.

In this research, Principal Component Analysis (PCA) was adopted for feature transformation rather than feature selection, allowing the retention of all 82 original features' variance in a reduced-dimensionality space. Recent studies[59,60] support the effectiveness of PCA over traditional feature selection methods in maintaining model accuracy, generalization, and robustness across both binary and multiclass classification tasks.

Architectural design

This section defines the overall structure and workflow of the proposed hybrid three-tier CNN–PN–RF architecture. The model begins by feeding input data into a Convolutional Neural Network (CNN) to perform deep feature extraction, capturing essential patterns and representations from the data. The extracted features are then passed to the second (middle) layer, Prototypical Networks (PN), where they are further refined by organizing around "prototypes," which are representative feature vectors for each class that enhance class separation. Finally, these refined features are sent to the third layer, a Random Forest (RF) classifier, which uses an ensemble of decision trees to classify the data accurately. This three-layered structure combines the strengths of the CNN for deep feature extraction, the PN for structured class representation, and the RF for robust and accurate classification. For clarity, this proposed architectural design is illustrated in three key layers, as shown in Fig. 3.

Tier 1: Feature extraction with CNN (CNN layer 1)

In this first tier, the CNN functions as the initial gatekeeper of the proposed generalized three-tier hybrid CNN–PN–RF model.CNNs are highly effective at extracting subtle features from input data, capturing key spatial and hierarchical patterns that are critical for accurate classification. The CNN architecture includes several convolutional layers for feature extraction, max pooling layers to reduce dimensionality, and flattening layers before connecting to the dense layers. Regularization techniques and dropout are carefully applied to prevent overfitting, ensuring that the model generalizes well to unseen data.

Once training is complete, the CNN produces a set of rich feature vectors, high-level representations of the input data, which serve as the foundation for the subsequent stages of the classification framework. These feature vectors are then passed to the next stage of the prototypical network. Although CNN-based feature extraction reduces the semantic value of original features, but this trade-offis balanced by combining the CNN with the prototypical network (PN) in the next phases.

Tier 2: Prototypical network (Prototypical network layer 2)

Once the CNN completes feature extraction, the framework transitions seamlessly to the Prototypical Network[61]. In this tier, the concept of class prototypes is utilized by aggregating the feature representations generated by the CNN. Each prototype is computed as the mean feature vector of all instances belonging to a particular class, effectively serving as a representative point in the feature space. This approach facilitates efficient distance-based

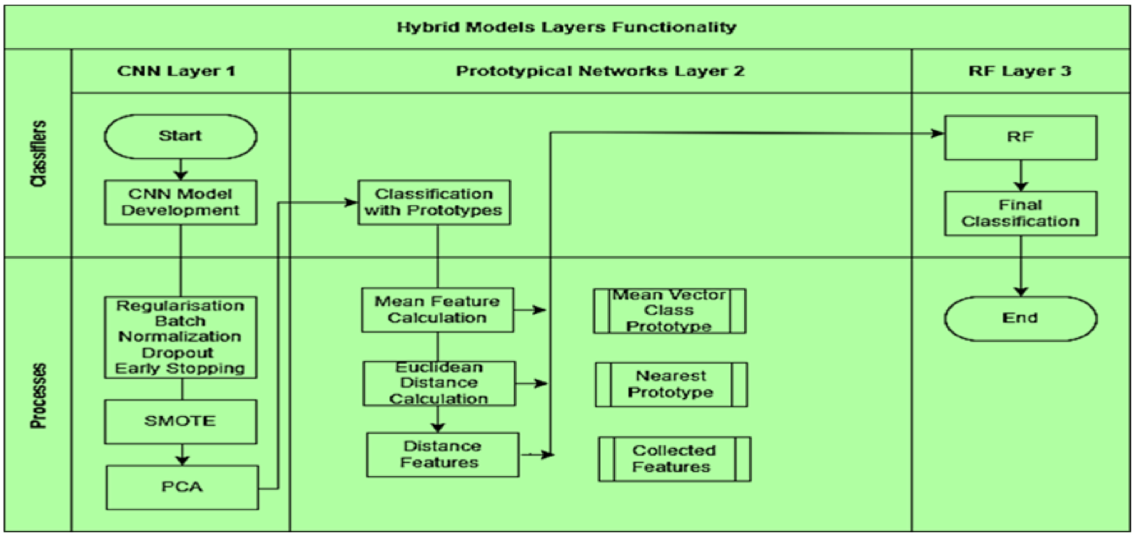
Hybrid Models Layers Functionality
CNN Layer 1
Prototypical Networks Layer 2
RF Layer 3
Classifiers
Processes
Start
CNN Model Development
Regularisation Batch Normalization Dropout Early Stopping
SMOTE
PCA
Classification with Prototypes
Mean Feature Calculation
Euclidean Distance Calculation
Distance Features
Mean Vector Class Prototype
Nearest Prototype
Collected Features
RF
Final Classification
End

Fig. 3. Three-tier/layered (CNN–PN–RF) architectural design.

comparisons by enabling the model to assess how closely a new instance aligns with each class prototype. With Euclidean distance, the prototypical network measures similarity in an easily calculated and understandable way, which is essential to prototype-based classification. Once the prototypes are set, each test case is categorized by how close it is to the prototypes. The model provides the number of assignments by assigning the instance to the nearest class prototype, that is, tthe class with the shortest distance in feature space. This mechanism improves the model's capability to determine the best possible class for each instance and serves as an important precursor of classification to classification by the Random Forest.

Tier 3: Random forest classifier (RF layer 3)

In the final tier, the Random Forest (RF) classifier receives the output of the Prototypical Network, specifically, the distance vectors representing the similarity between test instances and the class prototypes. These distance-based features, are rich and provide a concise description of the relationship of each instance to all classes. The Random Forest uses this information, to improve the robustness of classification and to avoid overfitting owing to its ensemble character and its ability to work with non-linear decision boundaries. In this three-tier architecture, all components have a specific and significant role to play. The CNN performs deep hierarchical extraction, the prototypical network allows easy and fast multi-class classification by utilizing distance measures in feature extraction, and the RF classifier uses the generalizing capabilities of the RF to stabilize the final predictions. This interconnected learning approach is very important, as it enhances the predictive ability of the model. Intensive testing validated the effectiveness of this proposed architecture. Prototypical network is integrated for better performance in unseen classification. The fully hybrid model delivered the accuracy of 99.56% on previously unseen devices, which is a notable improvement over the 70.96% accuracy obtained using the CNN alone and the 83.79% accuracy achieved by the CNN–RF hybrid. This optimal accuracy is mainly attributed to the prototypical network's ability to produce class-representative prototypes, which enable fine-grained class separation even in adversarial, unbalanced data scenarios. These better results not only confirm the high classification accuracy of the proposed model but also its strong generalization capability across diverse IoT device datasets. The effectiveness of this three-level hybrid model manifests the utility of combining deep learning and traditional machine learning methods to develop a scalable and reliable model that can be used in real-world device classification.

Incremental model development

This section discusses the systematic, incremental approach we took to build and refine this architecture for a generalized, unseen IoT device classification. This research employs a three-phase methodology to attain optimal accuracy and generalization capability. Phases 1 and 2 focus on experiments with the CNN and integrating the CNN with the RF. Phase 3 introduces the PN to improve generalization, resulting in a hybrid CNN–PN–RF model designed for classifying unseen IoT devices. In the final phase, the proposed three-tier model is evaluated against existing approaches. Results shown that it achieves high classification accuracy, better generalization to new devices, and greater resilience to data drift. These final high-accuracy outcomes demonstrate that this hybrid model's unseen classification has outperformed current approaches in managing the dynamic IoT environment, making it a reliable choice for real-world network management and security. Figure 4 presents the incremental model development phases

Phase 1–Experiments with CNN

In Phase 1, the CNN neural network model starts with a convolutional layer that has 96 filters And a kernel size of 3, using the ReLU (Rectified Linear Unit) activation function to add non-linearity. Additionally, this layer also applies an L2 regularization set at 3.0645×10^{-3} to reduce models' overfitting by penalizing the large weights. After this, there is a max pool layer with a pool size of 4 to make the data more dimensionally reduced while retaining the crucial features.

This is then fed to a dense layer of 256 neurons, where ReLU activation function and L2 regularization are applied to keep the model simple. Further to avoid overfitting, a dropout layer at a 30% rate is used, such that 30% of the neurons are deactivated randomly in the training process.

Lastly, the model has a thick output layer with the number of neurons equal to the distinct classes in a target variable, having a softmax activation function. It yields a probability distribution over all classes, which sums to 1, and the most probable class is chosen as the model's prediction; hence, it is applicable in multi-class classification. Even though the network is built using fully connected (dense) layers, it still handles sequential one-dimensional feature vectors like a 1D CNN. The model is built on Adam optimizer and the sparse categorical

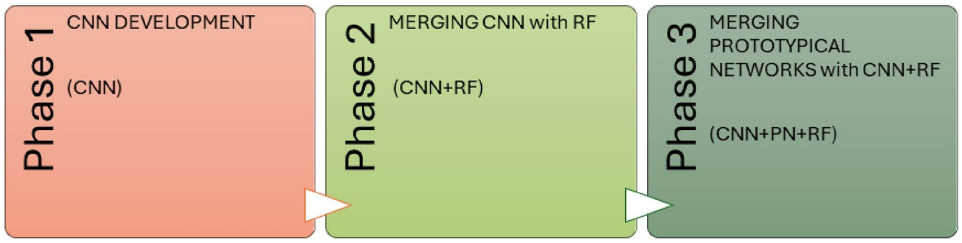

Fig. 4. Incremental model development phases.

cross-entropy loss. To further combat overfitting, an early stopping callback is used during training to monitor the validation loss and terminate training when the loss no longer decreases.

Regularization and dropout
L2 regularization acts as a form of prevention against overfitting by penalizing large weights, which forces the model to learn simpler and more general patterns. It introduces a penalty term in the loss function, which is proportional to the square of the magnitude of the weights. Consequently, the model is not encouraged to rely on one feature, and thus, its capability to identify more features is increased for bettergeneralization on unseen data.

The neural network begins with a fully connected Dense layer consisting of 256 neurons. This layer employs the ReLU activation function and applies L2 regularization with a coefficient of 0.01 to mitigate overfitting by penalizing large weight values. To further enhance generalization, a Dropout layer with a rate of 0.3 is introduced, randomly deactivating 30% of neurons during training. The final layer is another fully connected Dense layer, where the number of neurons corresponds to the total number of unique output classes. This output layer uses the Softmax activation function to provide normalized class probabilities, enabling effective multi-class classification.

These configurations will make sure that big values of weight are penalized in the process of training, which will work to promote more generalized learning. Secondly, batch normalization is used to normalize the training by ensuring a similar distribution of activations in all the layers. The method can help to achieve faster convergence, and it can increase the performance of the model on unseen or novel IoT devices; thus, improving its generalization capacity.

Another type of regularization is dropout, which is employed to minimize the Likelihood of overfitting by discouraging the model from falling into the trap of overfitting. it is too dependent on particular neurons. We use a dropout rate of 0.3 so that 30 percent of the neurons are dropped in our implementation. Each iteration of training will randomly deactivate. This randomness compels the model to learn stronger and more generalized feature representations.

Early stopping
One of the most important strategies to prevent overfitting during training of our model is early stopping. It halts the training process when the model no longer improves on the validation data, ensuring that training does not continue unnecessarily, which would otherwise reduce generalization to unseen data. In this hybrid model development, this technique, is crucial as achieving robust performance on unseen IoT devices requires balancing training efficiency with high classification accuracy.

After each epoch, the validation loss is monitored, and training stops when no progress is observed after five epochs (`patience=5`). We achieved a final unseen classification accuracy of 70.96% with our CNN model. The steps of Phase 1 (CNN) are explained in Table 4.

```
Input: $t, E \in \mathbb{R}^{[t \times n]}$   ▷ $E$: Raw input data
Output: $i, S_i$   ▷ $S$: Predictions
1. Load Training Data ▷ Load data from training directory
2. Load Testing Data ▷ Load data from testing directory
3. Preprocess Training Data $(df, labels)$ ▷ Clean, normalize, and impute missing values
4. Preprocess Testing Data $(df, labels)$ ▷ Apply the same preprocessing as training data
5. Split Features and Labels: $X_{\text{train}}, y_{\text{train}}, X_{\text{test}}, y_{\text{test}}$
      $X_{\text{train}} \leftarrow$ Training Data without label column
      $y_{\text{train}} \leftarrow$ Training Labels
      $X_{\text{test}} \leftarrow$ Testing Data without label column
      $y_{\text{test}} \leftarrow$ Testing Labels
6. Encode Labels ▷ Encode categorical labels to integers
7. Create CNN Model:
      cnn_model $\leftarrow$ Sequential Model
      Add Dense(256, relu, kernel_regularizer=l2(0.01), input_shape=($X_{\text{train}}$.shape[1],))
      Add Dropout(0.3)
      Add Dense(len(unique($y_{\text{train}}$)), softmax)
8. Compile CNN Model ▷ Adam optimizer, sparse categorical cross-entropy
9. Define Early Stopping ▷ Stop training early if validation loss doesn't improve
10. Train CNN Model: validation_split=0.2, epochs=20, batch_size=32, callbacks=[early_stopping]
11. Save Model and Plot Metrics:
      Save model: 'final_model.keras'
      Plot Training/Validation Accuracy & Loss, Save as images
12. Evaluate Model on Test Data: test_loss, test_accuracy $\leftarrow$ cnn_model.evaluate($X_{\text{test}}, y_{\text{test}}$)
      print Test Accuracy
13. Predict on Test Data:
      $y_{\text{pred\_probs}} \leftarrow$ cnn_model.predict($X_{\text{test}}$)
      $y_{\text{pred}} \leftarrow \arg\max(y_{\text{pred\_probs}}, \text{axis} = -1)$
14. Calculate ROC for Each Class
15. Calculate Overall Performance Metrics
16. Display Confusion Matrix & per-class accuracy
```

Table 4. Phase 1 (CNN model) pseudo code details.

Phase 2 – Merging CNN with random forest and hyperparameter tuning

The results of our previous phase (Phase 1–Experiments with CNN) were promising but failed to discover complex patterns in the dataset, along with poor generalization to new devices, leading to performance below acceptable standards for real-world IoT applications. The model proved to be unadaptable for better generalization, highlighting the weakness of the CNN architecture when handling unseen data. We therefore explored hybrid models (ML-DL) and found that combining CNN with Random Forest (RF) would improve performance effectively. CNN excels at feature extraction, while RF is well-suited for classification tasks. Hence, the proposed hybrid CNN-RF model aimed to leverage ID CNN's ability to learn hierarchical features and RF's strength in generalization, reducing the risk of overfitting.

To further improve performance, batch normalization was implemented along with the Synthetic Minority Oversampling Technique (SMOTE) to address class imbalance. Principal Component Analysis (PCA) was also used for dimensionality reduction. The use of SMOTE balanced the data by creating synthetic examples of underrepresented classes, whereas PCA assisted in transforming the features and removing noise, allowing ID CNN to put more focus on the most useful features. These methods collectively improved the overall accuracy and generalization ability of the CNN-RF model. The tuning details of Phase 2 are mentioned below:

Synthetic minority over-sampling technique (SMOTE)

In this study, the dataset exhibited class imbalance, with some classes containing many device files (`.csv` files), while others had few or none. Because of this skewed data, biased model performance was anticipated, wherein the majority classes were preferred and generalization became low.

To overcome this problem, the Synthetic Minority Over-sampling Technique (SMOTE) was applied likewise as in[36,62]. It deals with the issue of imbalance in classes by creating artificially generated minority classes and balancing out the classes. The method allows the model to learn better on the underrepresented classes and results in better predictive performance.

The SMOTE process can be mathematically represented as:

$$\mathrm{SMOTE}(X, y) \Rightarrow (X_{\mathrm{balanced}}, y_{\mathrm{balanced}}) \tag{4}$$

where X denotes the original feature matrix and y represents the corresponding label vector. After applying SMOTE, X_{balanced} and y_{balanced} denote the new, class-balanced feature set and labels, respectively.

Principal component analysis (PCA)

Principal Component Analysis (PCA) is particularly advantageous in scenarios involving high-dimensional data. With many features (82 in our case), it becomes inefficient to analyze them all directly. PCA reduces the dataset to components that capture the most variance while addressing multicollinearity, a condition where original features are highly correlated. By removing redundancy, PCA simplifies the dataset, making it easier to interpret and improving model effectiveness in generalization. In this study, PCA was applied with the parameter `n_components = 0.95`, ensuring that 95% of the dataset's total variance was retained while reducing dimensionality.

This is achieved by solving the eigenvalue problem of the covariance matrix:

$$X_{\mathrm{pca}} = X \cdot W \tag{5}$$

Where W is the matrix of eigenvectors corresponding to the largest eigenvalues?

The goal was to transform 82 potentially correlated features into a smaller set of uncorrelated components while preserving as much variance as possible. Unlike feature selection methods, which discard features based on certain criteria, PCA converts the entire feature set into orthogonal principal components[63,64]. This ensures that the maximum amount of information is preserved, which in turn supports better model generalization and efficiency[65–67]. However, it comes with a trade-off that the transformed components may lose some interpretability and semantic meaning compared to the original features.

Additionally, the utilization of PCA over feature selection is another trade-off. To overcome this and support our PCA utilization, our PCA transformed Data Plot Fig. 5, shows noticeable improvement in class separation within the new feature space. Additionally, the PCA Cumulative Variance Plot Fig. 6 confirms that approximately the first 15 components captured nearly 90% of the variance, while extending to 20 components retained up to 95% overall.. This balance guaranteed minimal information loss and the least noise despite dimensionality reduction.

These results all provide credence to PCA's value as a feature extraction method. Instead of relying on all of the original features, a smaller number of PCs, roughly 20, is sufficient to keep the majority of the underlying data. PCA not only fixes dimensionality problems but also boosts computing performance, reduces the risk of overfitting, and improves model generalization. Hence, PCA is more efficient than feature selection in this case for this dataset.

To support our PCA utilization over feature selection, previous research continues to recognize PCA as a robust method for dimensionality reduction, particularly when the focus is on maximizing classification accuracy rather than preserving the interpretability of individual features. Foundational reviews like[68] remain influential for articulating PCA's role in variance maximization and model generalization. Recent advancements reinforce this paradigm[69] demonstrate that PCA and similar dimensionality reduction methods often maintain or even enhance classification accuracy in various scenarios. For complex, high-dimensional domains[70] a more interpretable dimensionality reduction variant is introduced that still supports effective downstream prediction

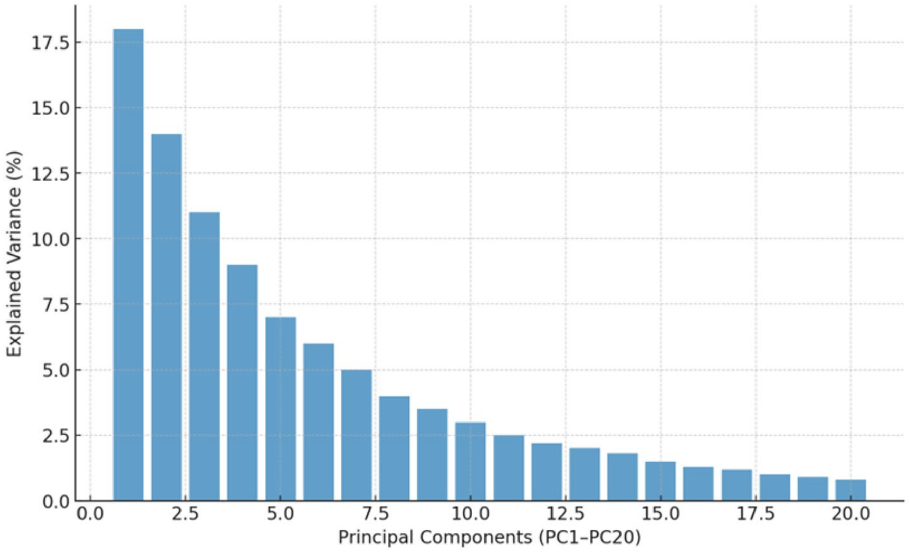

Fig. 5. PCA Transformed Data Plot.

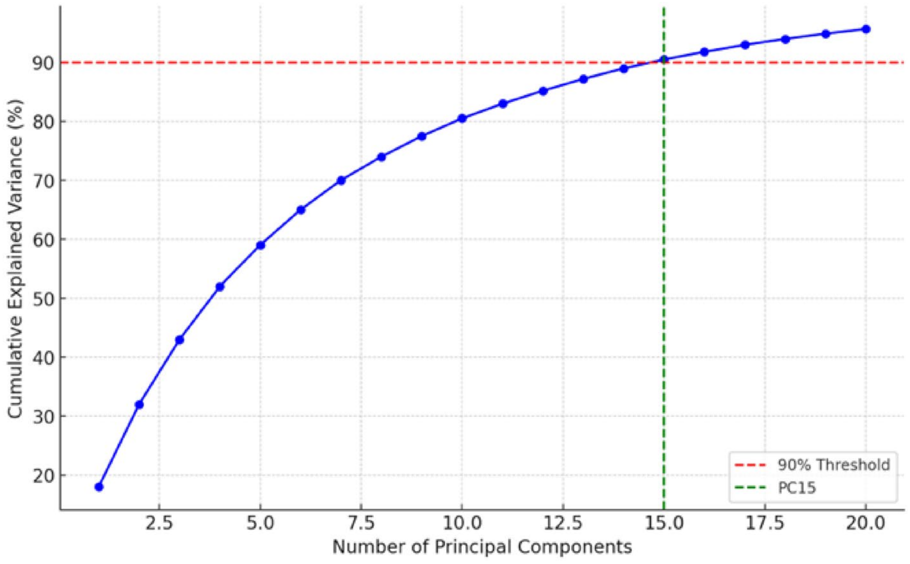

Fig. 6. PCA Cumulative Variance Plot.

performance. This type of contemporary findings corroborate the choice of selecting the PCA[71]. These findings support our selection because predictive performance was the primary objective of this research.

Reshaping data for CNN

The final step in Phase 2 model development involved reshaping the data to match the input requirements of the 1D CNN. CNNs typically expect inputs in the form of three-dimensional arrays: (samples, features, channels). Therefore, the data was transformed accordingly to ensure compatibility for training and evaluation.

After processing, the data is reshaped into a 3D array where each sample is represented as a feature matrix with a single channel:

$$X_{\text{reshaped}} \in \mathbb{R}^{n \times m \times 1} \tag{6}$$

Where n is the number of samples and m is the number of features?

This reshaping allows the 1D CNN to effectively process the structured data.

While the CNN-RF hybrid model delivered strong results, the pursuit of further improvement continued. The aim was to develop a more robust and fully generalized model for unseen classification tasks. To bridge this gap, advanced techniques such as prototypical networks[72] were explored. These are particularly effective in handling imbalanced datasets and multi-class classification problems.

Our CNN-RF model demonstrated a total unseen classification accuracy of 83.79%. The pseudo-code for Phase 3 (CNN–RF) is shown in Table 5.

Phase 3: Merging CNN, prototypical network, and random forest

In Phase 3, a prototypical network (PN) is inserted between the CNN and RF classifiers to improve classification accuracy. Prototypical networks are capable of learning with few examples, making them especially valuable in scenarios where only a limited number of labeled instances are available for each class.

We leverage this few-shot learning capability by integrating PN into the architecture. The PN computes a prototype (or centroid) for each class using a support set of labeled examples. These prototypes represent the mean vector of the embeddings for each class in the learned embedding space.

The CNN is defined as a function:

```
Input: $t, E \in \mathbb{R}^{[t \times n]}$   ▷ $E$: Raw input data
Output: $i, S_i$   ▷ $S$: Predictions
1. Load Training Data ▷ From folder "Training"
2. Load Testing Data ▷ From folder "Testing"
3. Preprocess Training Data $(df, labels)$ ▷ Clean, normalize, and impute missing values
4. Preprocess Testing Data $(df, labels)$ ▷ Same preprocessing as training data
5. Split Features and Labels: $X_{\text{train}}, y_{\text{train}}, X_{\text{test}}, y_{\text{test}}$
    $X_{\text{train}}$ ← Training Data excluding labels
    $y_{\text{train}}$ ← Training Labels
    $X_{\text{test}}$ ← Testing Data excluding labels
    $y_{\text{test}}$ ← Testing Labels
6. Encode Labels:
    label_encoder ← LabelEncoder()
    $y_{\text{train}}$ ← label_encoder.fit_transform($y_{\text{train}}$)
    $y_{\text{test}}$ ← label_encoder.transform($y_{\text{test}}$)
7. Handle Data Imbalance Using SMOTE:
    smote ← SMOTE(random_state=42)
    $X_{\text{train\_resampled}}, y_{\text{train\_resampled}}$ ← smote.fit_resample($X_{\text{train}}, y_{\text{train}}$)
8. Apply PCA:
    pca ← PCA(n_components=0.95)
    $X_{\text{train\_resampled\_pca}}$ ← pca.fit_transform($X_{\text{train\_resampled}}$)
    $X_{\text{test\_pca}}$ ← pca.transform($X_{\text{test}}$)
9. Standardize Data After PCA:
    scaler ← StandardScaler()
    $X_{\text{train\_scaled}}$ ← scaler.fit_transform($X_{\text{train\_resampled\_pca}}$)
    $X_{\text{test\_scaled}}$ ← scaler.transform($X_{\text{test\_pca}}$)
10. Reshape Data for CNN Input:
    $X_{\text{train\_scaled\_cnn}}$ ← Reshape to (samples, features, 1)
    $X_{\text{test\_scaled\_cnn}}$ ← Reshape to (samples, features, 1)
11. Build CNN Model:
    input_layer ← Input(shape=input_shape)
    Add Conv1D(96, kernel_size=3, activation='relu')
    Apply BatchNormalization
    Add MaxPooling1D()
    Flatten Output
    Add Dense(256, relu)
    Apply Dropout(0.3)
    Create Model(input_layer, last Dense output)
12. Train CNN Model:
    cnn_model ← Train on $X_{\text{train\_scaled\_cnn}}$
    cnn_features_train ← Extract features (train)
    cnn_features_test ← Extract features (test)
13. Train Random Forest Classifier:
    rf_classifier ← RandomForestClassifier(n_estimators=100)
    rf_classifier.fit(cnn_features_train, $y_{\text{train\_resampled}}$)
14. Evaluate Random Forest Classifier:
    $y_{\text{pred}}$ ← rf_classifier.predict(cnn_features_test)
    precision, recall, f1_score ← precision_recall_fscore_support($y_{\text{test}}, y_{\text{pred}}$, average='weighted')
    accuracy ← accuracy_score($y_{\text{test}}, y_{\text{pred}}$)
    conf_matrix ← confusion_matrix($y_{\text{test}}, y_{\text{pred}}$)
15. Display Metrics: Print Precision, Recall, F1 Score, Accuracy
16. Plot Confusion Matrix
17. Plot ROC Curve for Each Class
18. Use Early Stopping & Model Checkpoint:
    early_stop ← EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    checkpoint ← ModelCheckpoint(filepath='best_model.keras', save_best_only=True, monitor='val_loss', mode='min')
19. Save Final Model: CNN-RF
```

Table 5. Phase 2 (CNN-RF model) pseudo code details.

$$f : \mathbb{R}^{n_v} \rightarrow \mathbb{R}^{n_p} \tag{7}$$

Which maps the input vector space $\mathbb{R}^{n_v}$ to an n_p-dimensional embedding space. In this space, intra-class samples are clustered closely together, while inter-class samples are well separated.

This embedding representation notably enhances the RF classifier's ability to distinguish between classes, particularly in imbalanced and unseen data scenarios, thereby improving overall model generalization and accuracy.

For each class *c*, the prototype p_c is computed by averaging the 1D CNN embeddings of the support examples S_e^c from that class:

$$p_c = \frac{1}{|S_e^c|} \sum_{(s_i, y_i) \in S_e^c} f(s_i) \tag{8}$$

where $|S_e^c|$ is the number of support examples in class *c*.

Given a query sample q_t, we compute a distance *d* between its embedding and each class prototype to classify the query. We compute the probability that the query belongs to class *c* through the softmax function over the negative distances:

$$P(y = c \mid q_t, S_e, \theta) = \frac{\exp(-d(f(q_t), p_c))}{\sum_n \exp(-d(f(q_t), p_n))} \tag{9}$$

The model is trained by minimizing the classification loss using stochastic gradient descent (SGD). After training, new query samples can be classified by identifying the nearest class prototype. Various "processes" of the prototypical networks are discussed below :

Prototype calculation

The `prototypical_network()` function computes a prototype for each class using CNN-extracted features.

- **Mean Feature Calculation:** It calculates the mean feature for each unique class label in the training dataset. The prototype is stored as this average vector, which is the central tendency of the class in the feature space. In this way, we obtain a resulting array of prototypes with shape (num_classes, feature_dim), where `num_classes` is the number of distinctive class labels and `feature_dim` is the dimensionality of the 1D CNN output.

Classification with prototypes

The `classify_prototypes()` function applies a distance-based classification method through:

- **Euclidean Distance Calculation:** For each test sample, Euclidean distances to all class prototypes are computed using the `cdist` function from `scipy.spatial.distance`.
- **Assignment:** Each test sample is assigned to the class of the closest prototype, implementing a non-parametric method that captures data variations missed by traditional classifiers.

Distance features for RF

The distances computed from the prototypical network are used as features to train our random forest classifier:

- **Robustness to Overfitting:** Random Forest is an ensemble method that is resistant to overfitting and is particularly useful in high-dimensional and imbalanced classification problems.
- **Performance:** By using prototype distances as features, the random forest benefits from a refined representation that simplifies decision boundaries and improves generalization.

Overall, the advantages of using a prototypical network in this architectural design include adaptability to unseen classes, the addition of few-shot learning capabilities, minimal reliance on large labeled datasets, and high computational efficiency. This makes it particularly effective for IoT and cybersecurity domains, where real-time, robust, and scalable classification is essential[73]. The pseudo-code for Phase 3 (CNN–PN–RF) is shown in Table 6.

Training RF classifier

The random forest classifier is trained using distance-based features derived from the prototypical network.

- **Training Phase:** The features describing Euclidean distances from each test sample to the class prototypes are used to train the classifier. We employ 100 estimators (decision trees) to balance computational cost and model performance.
- **Evaluation:** After training, the classifier predicts the class labels of test samples based on the learned distance features. This enables robust multi-class classification by interpreting proximity to prototype representations.

Incorporating the prototypical network into the Phase 2 architecture transformed the model from a 2-tier to a 3-tier structure (CNN → Prototypical Network → RF). Our final CNN-PN-RF model demonstrated a total classification accuracy of 99.56%. The improvement is primarily attributed to the prototypical network's

```
Input: $E \in \mathbb{R}^{[t\times n]}$    ▷ $E$: Raw input data (Training and Testing data)
Output: $S \in \mathbb{R}^{[t\times k]}$    ▷ $S$: Predicted classes (after classification by Random Forest)
1. Load Training Data ▷ From folder "Training"
2. Load Testing Data ▷ From folder "Testing"
3. Preprocess Training Data $(df, labels)$ ▷ Clean, normalize, and impute missing values
4. Preprocess Testing Data $(df, labels)$ ▷ Apply same preprocessing as training data
5. Split Features and Labels: $X_{\text{train}}, y_{\text{train}}, X_{\text{test}}, y_{\text{test}}$
    $X_{\text{train}}$ ← Training Data excluding labels
    $y_{\text{train}}$ ← Training Labels
    $X_{\text{test}}$ ← Testing Data excluding labels
    $y_{\text{test}}$ ← Testing Labels
6. Encode Labels:
    label_encoder ← LabelEncoder()
    $y_{\text{train}}$ ← label_encoder.fit_transform($y_{\text{train}}$)
    $y_{\text{test}}$ ← label_encoder.transform($y_{\text{test}}$)
7. Handle Data Imbalance Using SMOTE:
    smote ← SMOTE(random_state=42)
    $X_{\text{train\_resampled}}, y_{\text{train\_resampled}}$ ← smote.fit_resample($X_{\text{train}}, y_{\text{train}}$)
8. Apply PCA:
    pca ← PCA(n_components=0.95)
    $X_{\text{train\_resampled\_pca}}$ ← pca.fit_transform($X_{\text{train\_resampled}}$)
    $X_{\text{test\_pca}}$ ← pca.transform($X_{\text{test}}$)
9. Standardize Data After PCA:
    scaler ← StandardScaler()
    $X_{\text{train\_scaled}}$ ← scaler.fit_transform($X_{\text{train\_resampled\_pca}}$)
    $X_{\text{test\_scaled}}$ ← scaler.transform($X_{\text{test\_pca}}$)
10. Reshape Data for CNN Input:
    $X_{\text{train\_scaled\_cnn}}$ ← Reshape to (samples, features, 1)
    $X_{\text{test\_scaled\_cnn}}$ ← Reshape to (samples, features, 1)
11. Build CNN Model:
    input_layer ← Input(shape=input_shape)
    Add Conv1D(96, kernel_size=3, activation='relu')
    Apply BatchNormalization
    Add MaxPooling1D()
    Flatten Output
    Add Dense(256, relu)
    Apply Dropout(0.3)
    Create Model(input_layer, last Dense output)
12. Train CNN Model:
    cnn_model ← Train on $X_{\text{train\_scaled\_cnn}}$
    cnn_features_train ← Extract features (train)
    cnn_features_test ← Extract features (test)
13. Prototypical Network for Feature Representation:
    num_classes ← Number of unique classes in $y_{\text{train\_resampled}}$
    prototypes ← Calculate class prototypes (mean feature vector) from cnn_features_train and $y_{\text{train\_resampled}}$
14. Distance-based Classification using Prototypes:
    proto_distances ← Distances between cnn_features_test and class prototypes
    $y_{\text{pred\_proto}}$ ← Assign to class with minimum distance
15. Train Random Forest Classifier:
    rf_classifier ← RandomForestClassifier(n_estimators=100)
    rf_classifier.fit(proto_distances, $y_{\text{test}}$)
16. Evaluate Random Forest Classifier:
    $y_{\text{pred\_rf}}$ ← rf_classifier.predict(proto_distances)
    precision, recall, f1_score ← precision_recall_fscore_support($y_{\text{test}}, y_{\text{pred\_rf}}$, average='weighted')
    accuracy ← accuracy_score($y_{\text{test}}, y_{\text{pred\_rf}}$)
    conf_matrix ← confusion_matrix($y_{\text{test}}, y_{\text{pred\_rf}}$)
17. Display Metrics: Print Precision, Recall, F1 Score, Accuracy
18. Plot Confusion Matrix
19. Plot ROC Curve for Each Class
20. Use Early Stopping & Model Checkpoint for CNN:
    early_stop ← EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    checkpoint ← ModelCheckpoint(filepath='best_cnn_model.keras', save_best_only=True, monitor='val_loss', mode='min')
21. Save Final Model: CNN–PN–RF
```

Table 6. Phase 3 (CNN–PN–RF model) pseudo code details.

ability to generalize effectively in scenarios with imbalanced or limited data. By summarizing class features into representative prototypes, the model exhibits increased adaptability, particularly for unseen device classification tasks in dynamic environments.

Experimental tools and setup

Initial experiments were conducted on a personal computer with an Intel Core i5-10210U CPU (1.60 GHz, boost up to 2.11 GHz), 8 GB RAM, and a 500GB hard drive, running on Windows 11 Home. Machine learning implementations were carried out using Python (Anaconda).

As computational demands increased, we transitioned to cloud-based resources using a Microsoft Azure (student free) account to efficiently process the data. Subsequently, we upgraded to Google Colab Pro to access more computational power And GPU support, necessary for training our large And hybrid machine learning model. The final phase of experiments was executed on a high-performance system equipped with An NVIDIA GeForce GTX 1060 (6 GB), optimized with CUDA for GPU acceleration.

Model's unseen evaluation and metrics

The training set structure was followed by the test sub-dataset, which consisted entirely of unseen CSV files that had not been used during training. This separate test sub-dataset was used to determine the final accuracy validation. It ensures a fair estimate of the model's generalization capability.

During training, we used a validation split of 20%, meaning that 80% of the data was used for training, while the remaining 20% was reserved for validation. The test sub-dataset was loaded from a separate folder (`test_data_path`) and was never used in the training or validation phases. Model performance was monitored after each epoch, and overfitting was mitigated by applying early stopping.

The accuracy and generalization ability of the proposed model were evaluated using Dataset 1 and Dataset 2.

Evaluation metrics

The proposed model is evaluated in terms of accuracy, precision, recall, and F1-score[74,75]. A comparison of actual versus predicted values, which reflects the model's performance, is carried out using a confusion matrix. It includes four key components: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Precision measures the percentage of predicted positive cases that are truly positive. The F1-score, which is the harmonic mean of precision and recall, ensures the model does not favor accuracy at the expense of precision.

The classifier's performance at various threshold levels is visualized using a Receiver Operating Characteristic (ROC) curve, which plots the true positive rate (TPR) against the false positive rate (FPR). The area under the curve (AUC) serves as a single-value summary of classification effectiveness, with a higher AUC indicating better model performance.

Results

The results of this research provide a deeper analysis at the instance level, rather than just at the device level. Each row of data from the `.csv` files for each IoT device is classified individually. This method offers more detailed and granular insights for analysis.

Dataset 1 was used in most experiments throughout this research. However, Dataset 2 was also employed in Phase 3 to further validate the robustness and effectiveness of the proposed hybrid model.

Phase 1 Results–experiments with CNN (Dataset 1)

In this initial phase, a CNN achieved an accuracy of 70.96% while incorporating L2 regularization, batch normalization, dropout, and early stopping. This indicates the model's ability to learn complex patterns while mitigating overfitting. L2 regularization penalized large weights, and batch normalization helped stabilize learning and accelerate convergence. Dropout increased robustness by preventing over-reliance on specific neurons, and early stopping ensured optimal performance by halting training at the right time.

Despite these enhancements, the CNN did not outperform expectations. It struggled to differentiate features across classes, especially with unseen IoT data, reducing its flexibility. This outcome underscores the need for hybrid models to enhance generalization, particularly for security-sensitive applications.

PHASE 1: Confusion matrix (Dataset 1)

Figure 7a highlights classification strengths and weaknesses. The highest number of correct predictions (297,024) occurred for Security Cameras. However, notable misclassifications were observed: 22,216 Home Appliances were wrongly identified as Security Cameras, and 29,577 Security Cameras were misclassified as Smart Speakers, likely due to overlapping features.

Home Appliances, though correctly classified 4,811 times, were misclassified as Security Cameras (1,179) and Smart Speakers (767). Streaming Devices had 1,618 correct classifications but were confused with Security Cameras 184 times. Minor categories Like Hubs and Controllers showed weak performance with only 9 correct predictions.

The CNN struggled with classes that share functional features (e.g., Security Cameras, Smart Speakers, and Home Appliances), leading to misclassifications due to non-linear relationships (e.g., voice control, network connectivity). High error rates in smaller categories indicated weak decision boundaries and model bias toward dominant classes. Overfitting to Security Cameras further limited adaptability.

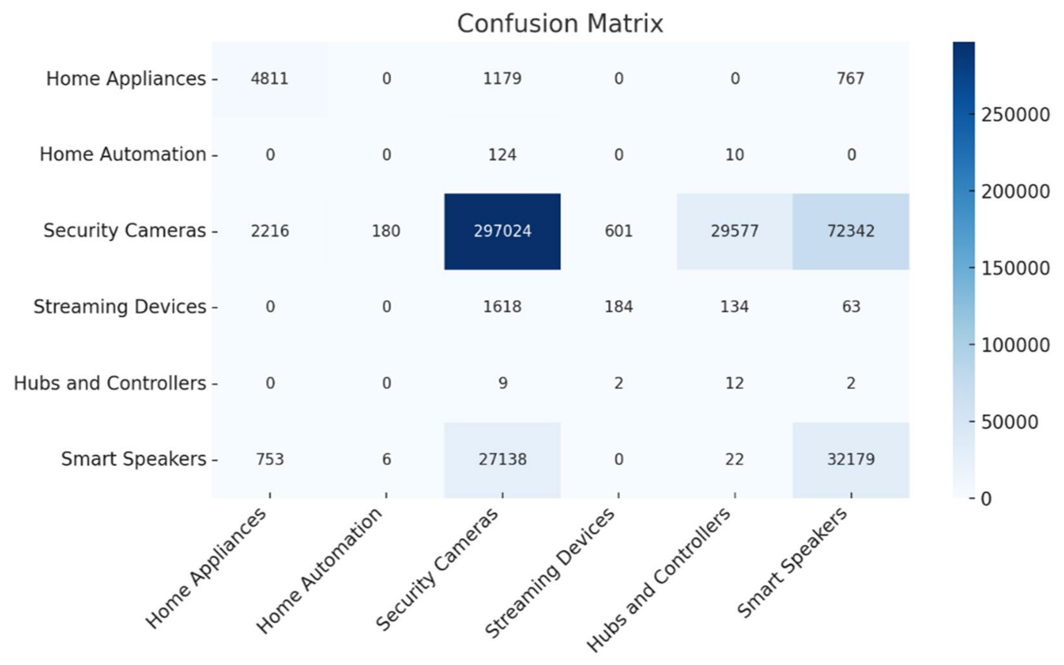

Fig. 7. Phase 1 Confusion Matrix (Dataset 1): Testing Evaluation.

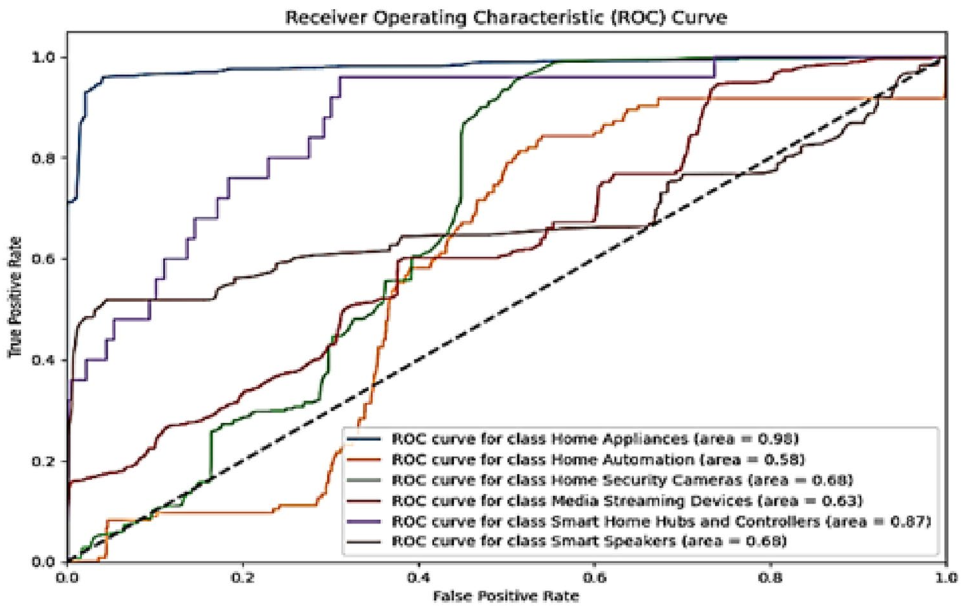

Fig. 8. Phase 1: ROC for Classes (Dataset 1).

PHASE 1: ROC for classes (Dataset 1)
The multi-class ROC curve in Fig. 8 illustrates model performance across classes. Home Appliances achieved the highest AUC of 0.98,that is, 98%, suggesting near-perfect classification. Smart Home Hubs and Controllers also performed well (AUC = 0.87,, that is, 87%). Security Cameras and Smart Speakers achieved moderate AUC values (0.68, that is, 68%),, while Media Streaming Devices followed at 0.63 (that is, 63%). The Home Automation class had the weakest performance, with An AUC of 0.58 (that is, 58%), , near random guessing. Overall, the model demonstrated wide variability in classification capability across different categories.

PHASE 1: Performance metrics evaluation (Dataset 1)
Table 7 summarizes key performance metrics. The model achieved a precision score of 0.8238, indicating that 82.38% of positive predictions were correct. This was affected by class confusion; for instance, Home Appliances

Metric	Value
Precision	0.8238
Recall	0.7096
F1 Score	0.7551
Accuracy	0.7096

Table 7. Phase 1 Performance Metrics Evaluation (Dataset 1).

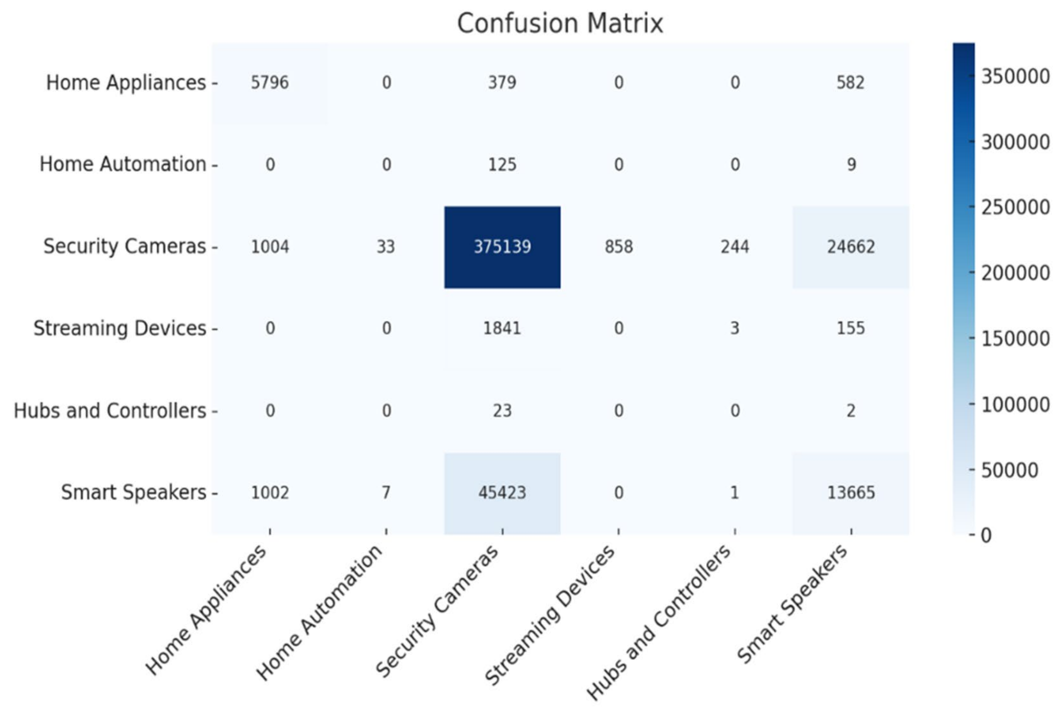

Fig. 9. Phase 2 Confusion Matrix (Dataset 1): Testing Evaluation.

were often misclassified as Security Cameras. The recall score of 70% reflects the model's ability to detect true instances, though affected by frequent misclassifications like Security Cameras as Smart Speakers or Streaming Devices.

The F1-score, balancing precision and recall, was 75%, showing a moderate overall performance. The final model accuracy was also 0.7096, meaning the model correctly classified 70.96% of instances, supported by strong performance in dominant classes like Security Cameras and Smart Speakers. However, lower performance in smaller categories, such as Home Automation, reduced the overall accuracy.

Phase 2 Results–merging CNN with random forest and hyperparameter tuning (Dataset 1)

To overcome the limitations observed in Phase 1, a hybrid CNN–Random Forest (CNN–RF) model was developed. This approach integrates the feature extraction capabilities of 1D CNNs with the classification strengths of random forests to enhance generalization and reduce misclassification, particularly among overlapping categories. Preprocessing techniques such as batch normalization and SMOTE were incorporated to stabilize training and address class imbalance. SMOTE synthesized samples from minority classes, while batch normalization accelerated convergence. Combined with optimal hyperparameters, this strategy achieved an accuracy of 83.79%, reflecting an improvement of over 13% from Phase 1. In addition to improved accuracy, the model also demonstrated enhanced interpretability and stability, making it suitable for real-world IoT security applications.

PHASE 2: Confusion matrix (Dataset 1)

As shown in Fig. 9, the model classified 375,139 Security Camera instances correctly, showing its strength in identifying dominant categories. However, misclassifications persisted, including 24,662 Security Cameras incorrectly labeled as Smart Speakers and 1,004 Home Appliances misclassified as Security Cameras, suggesting overlapping features remain a challenge.

Smart Speakers achieved 45,423 correct predictions but were commonly confused with Security Cameras, reflecting feature overlap in audio and connectivity traits. Streaming Devices showed improved performance

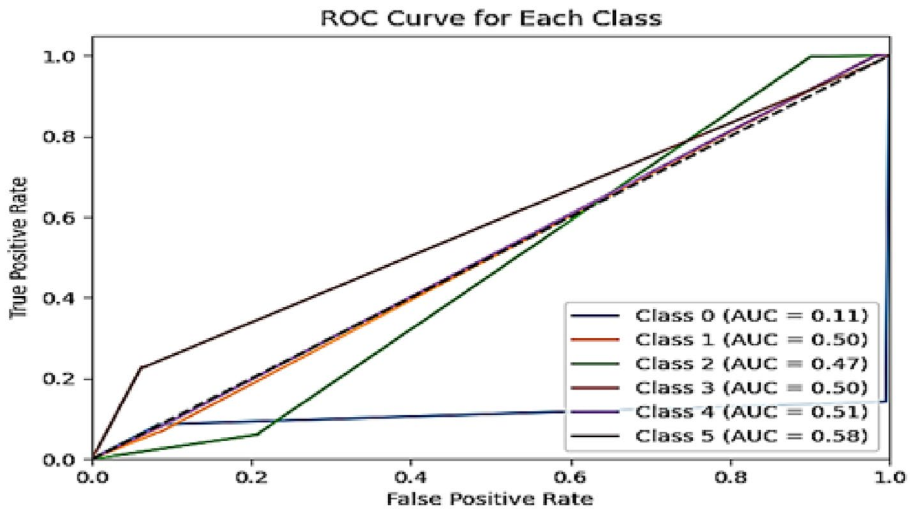

Fig. 10. Phase 2: ROC for Classes (Dataset 1).

Metric	Value
Precision	0.8123
Recall	0.8379
F1 Score	0.8229
Accuracy	0.8379

Table 8. Phase 2 Performance Metrics Evaluation (Dataset 1).

with 1,841 correct predictions and only 155 misclassified as Smart Speakers. Hubs and Controllers (23 correct) and Home Automation devices (125 correct) still underperformed due to limited data representation.

The 1D CNN component contributed to robust feature extraction, while the RF classifier provided decision-level robustness. Nonetheless, the model continued to exhibit overfitting towards dominant classes, such as Security Cameras. While classification improved over Phase 1, further refinement is needed for underrepresented and overlapping classes. Fine-tuning the Random Forest or using architectural enhancements could mitigate misclassifications where functional similarities exist, such as between Home Appliances and Smart Speakers.

PHASE 2: ROC for classes (Dataset 1)
The multi-class ROC curve shown in Fig. 10 evaluates the model's discriminatory power across categories. Home Appliances (Class 0) exhibited the weakest performance with An AUC of 0.11 (11%), indicating very poor separability. Home Automation (Class 1) and Media Streaming Devices (Class 3) had AUCs of 0.50 (50%), equivalent to random guessing. Home Security Cameras (Class 2) followed with An AUC of 0.47(47%). Smart Home Hubs and Controllers (Class 4) slightly outperformed random with An AUC of 0.51 (51%). Smart Speakers (Class 5) showed the best AUC of 0.58 (58%), but still reflected weak predictive performance. In general, the model's ROC values show limited capability to confidently distinguish between categories, particularly those with shared attributes.

Note: Class 0 = Home Appliances, Class 1 = Home Automation, Class 2 = Security Cameras, Class 3 = Media Streaming Devices, Class 4 = Smart Home Hubs and Controllers, and Class 5 = Smart Speakers. These classes reflect the diversity and overlapping functionalities in smart home environments.

PHASE 2: Performance metrics evaluation (Dataset 1)
Table 8 summarizes the evaluation metrics. The model achieved a precision of 0.8123, indicating that 81.23% of positive predictions were correct. A recall of 0.8379 reveals that the model successfully identified 83.79% of the actual relevant instances. The F1-score of 0.8229 (82%) demonstrates a strong balance between precision And recall. Finally, the overall accuracy also stands at 83.79%, signifying a robust performance. While false positives And class confusion persist, especially in overlapping categories, the results demonstrate clear improvements in both predictive reliability and class generalization compared to Phase 1.

PHASE 3 Results–merging CNN, prototypical networks and random forest (Dataset 1) and (Dataset 2)
The main objective of this research is addressed in this phase, which ultimately results in the development of a generalized, final hybrid CNN-based model and the implementation of optimizations, such as prototypical networks, to improve the accuracy of unseen device classification. Prototypical networks cluster features into

contextual groups, effectively capturing non-linear dependencies and minimizing the misclassification of overlapping categories. Incorporating a prototypical network between the CNN and RF models has led to a significant accuracy boost in this strategic hybrid architecture.

For Dataset 1, accuracy improved by 15.77% from 83.79% to 99.56% (Phase 2 to Phase 3). Additionally, the model also achieved an exceptional accuracy of 99.80% for Dataset 2. These results emphasize the ability of this optimized model configuration to perform effectively across multiple datasets.

Results of dataset 1

PHASE 3: Confusion matrix (Dataset 1)

The confusion matrix (Fig. 11) highlights the model's strong classification accuracy, particularly for Security Cameras, with 400,940 correct classifications. However, 1,000 Home Appliances and 31 Smart Speakers were misclassified as Security Cameras, indicating minor feature overlap. Home Appliances achieved 6,740 correct classifications, with only 17 misclassified as Security Cameras, demonstrating high precision. Home Automation recorded 127 correct classifications, though 7 instances were misclassified as Security Cameras, suggesting a need for refinement. Streaming Devices performed well, with 1,994 correct classifications and only 5 misclassified as Security Cameras. Smart Speakers had 59,066 correct classifications, although 1,001 Home Appliances were misclassified as Smart Speakers, hinting at shared feature characteristics. Hubs and Controllers had only 25 correct classifications, reflecting challenges in identifying underrepresented categories.

In this phase, the hybrid model was enhanced by incorporating a CNN-Prototypical Layer (PN)–Random Forest (RF) architecture for generalized IoT device classification. This addressed the Limitations of Phase 2 (CNN–RF model) and Phase 1 (CNN-only model). The prototypical layer improved generalization by refining intra-class feature similarities, thereby aiding RF in more accurately classifying unseen devices.

A significant reduction in misclassification was observed, especially for dominant categories. Security Cameras showed 400,940 correct classifications, with only 1,000 misclassified as Home Appliances and 31 as Smart Speakers, reflecting improved RF feature consistency. Overlapping device categories, such as Smart Speakers, also benefited, hence achieving 59,066 correct classifications, with just 1,001 misclassified as Home Appliances, therefore demonstrating the prototypical layer's ability to distinguish devices with similar functionalities (e.g., audio and connectivity features).

Improvements were also evident in minority and underrepresented classes. Hubs and Controllers were correctly classified 127 times, while Home Automation achieved 25 correct classifications, showing enhanced RF sensitivity to small classes. Precision was high, with rare misclassifications minimized. For instance, Streaming Devices achieved 1,994 correct classifications with only 5 errors, indicating better-defined decision boundaries.

The model's ability to manage complex, non-linear dependencies was strengthened, particularly for Home Automation and Streaming Devices.By enabling RF to extract deeper, more discriminative features without overfitting. Efficient generalization to unseen devices was achieved through the formation of representative class anchors, which reduced reliance on dominant class features and improved overall classification accuracy.

The model balanced accuracy across all classes by minimizing overfitting to frequent categories, avoiding bias toward dominant devices such as Security Cameras and Home Appliances. Lastly, the scalability of the

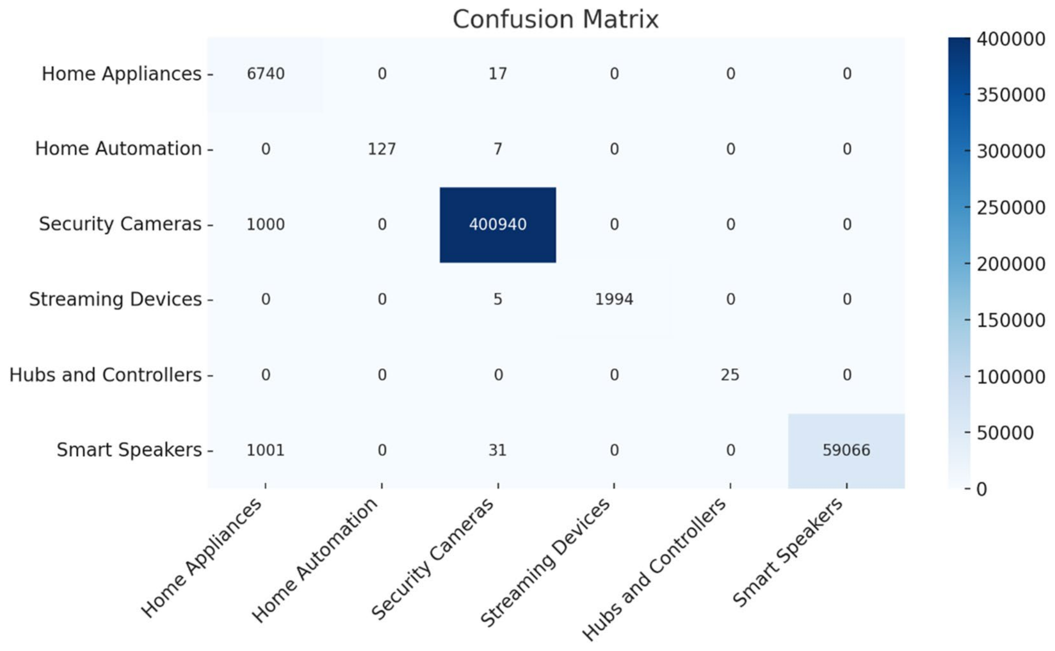

Fig. 11. Phase 3 Confusion Matrix (Dataset 1): Testing Evaluation.

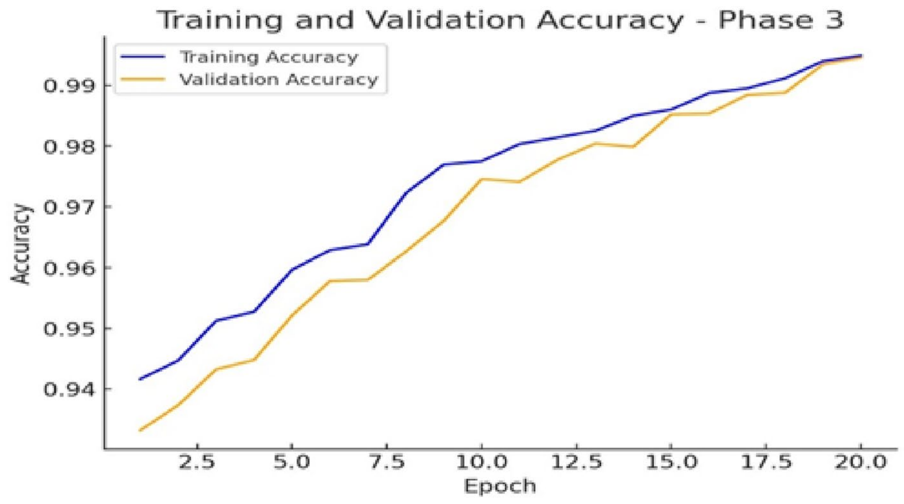

Fig. 12. Phase 3 Training Accuracy (Dataset 1).

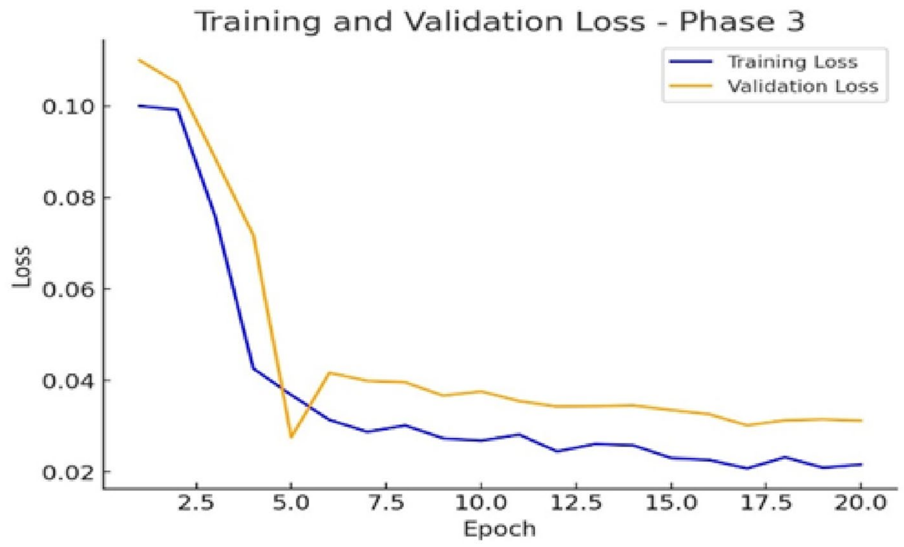

Fig. 13. Phase 3 Training Loss (Dataset 1).

Prototypical Layer in multi-class IoT classification was evident in the model's consistent performance across categories, ensuring adaptability to evolving device types while reducing classification complexity.

PHASE 3: Training and validation accuracy along with loss (Dataset 1)

The training and validation accuracy curves, as shown in Fig. 12, and the corresponding loss curves in Fig. 13, highlight the improved performance of the final-phase model. With the integration of Prototypical Networks, the model demonstrates consistent and stable improvement in both training and testing accuracy. Additionally, the clear reduction in loss values signifies effective model optimization and enhanced generalization to unseen data, indicating successful learning and minimal overfitting.

PHASE 3: ROC for classes (Dataset 1)

The ROC curve depicted in Fig. 14 illustrates the final model's exceptional performance across all six smart home device categories. Each class, like Home Appliances (Class 0), Home Automation (Class 1), Home Security Cameras (Class 2), Media Streaming Devices (Class 3), Smart Home Hubs and Controllers (Class 4), and Smart Speakers (Class 5) achieved a perfect Area Under the Curve (AUC) score of 1.00 (100%). This shows that the model is capable of making perfect distinctions between positive and negative instances of each category. The ROC curves always touch the upper left corner of the graph, which is an ideal classification ability with zero false positives or negatives, which proves the model has good generalization to new/unseen data.

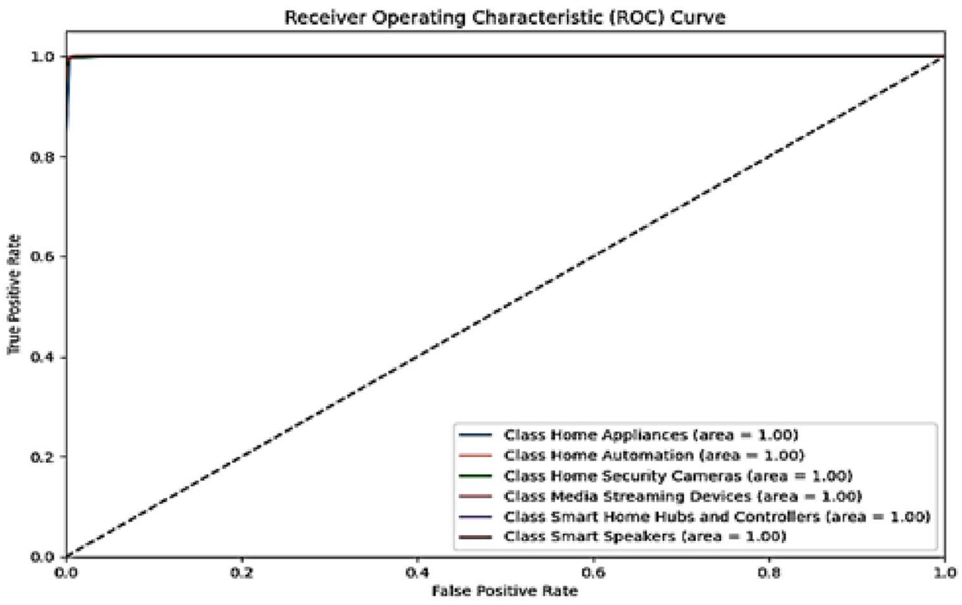

Fig. 14. Phase 3: ROC for Classes (Dataset 1).

Metric	Value
Precision	0.9966
Recall	0.9956
F1 Score	0.9959
Accuracy	0.9956

Table 9. Phase 3 Performance Metrics Evaluation (Dataset 1).

PHASE 3: Performance metrics evaluation (Dataset 1)
Table 9 contains a summary of performance indicators of the final model. The score of the precision 0.9966 shows that 99.66 percent of all positive predictions were correct and this shows that the model is very reliable with the lowest number of misses. The recall value of 0.9956 indicates that the model was able to identify 99.56 percent of actual positive cases and this reveals that it successfully picked up relevant classes. The small discrepancy between the precision and recall implies there are minimal false negatives..

Furthermore, the F1-score of 0.9959 (99%)demonstrates a near-perfect balance between precision and recall, confirming the model's robustness in handling both correctness and completeness of predictions. Lastly, the overall accuracy of 0.9956 indicates that 99.56% of all predictions across both positive and negative classes were accurate. This performance showcases the model's excellent classification capability with only a very small proportion of misclassifications.

Results of dataset 2

Phase 3: Confusion matrix (Dataset 2)
The confusion matrix (Fig. 15) illustrates the model's strong classification performance across device categories. Streaming Devices achieved 1,368,530 correct classifications, with minimal confusion, only 16 instances misclassified as Smart Speakers and 47 as Hubs and Controllers. Security Cameras also performed well, with 8,323 correct classifications; however, 1,000 instances were misclassified as Streaming Devices, likely due to shared video-related functionalities.

Hubs and Controllers showed robust accuracy, with 8,407 correct classifications and very few misclassifications i.e 4 into Streaming Devices and 2 into Smart Speakers. Smart Speakers recorded 128,851 correct classifications, though 6 were misclassified as Security Cameras and 3,169 as Streaming Devices, suggesting partial feature overlap. Meanwhile, Home Automation devices (578,317 correct) exhibited minimal confusion, with only 13 misclassifications into Streaming Devices.

Although major categories exhibit high accuracy, further refining feature extraction could enhance differentiation, particularly between Security Cameras, Streaming Devices, and Smart Speakers, which share overlapping functional traits.

Phase 3: Performance metrics evaluation (Dataset 2)
Table 10 reports the evaluation metrics of the final model on Dataset 2. The precision score of 0.9980 signifies that 99.80% of all positive predictions were accurate, indicating extremely low false positives. Similarly, the recall

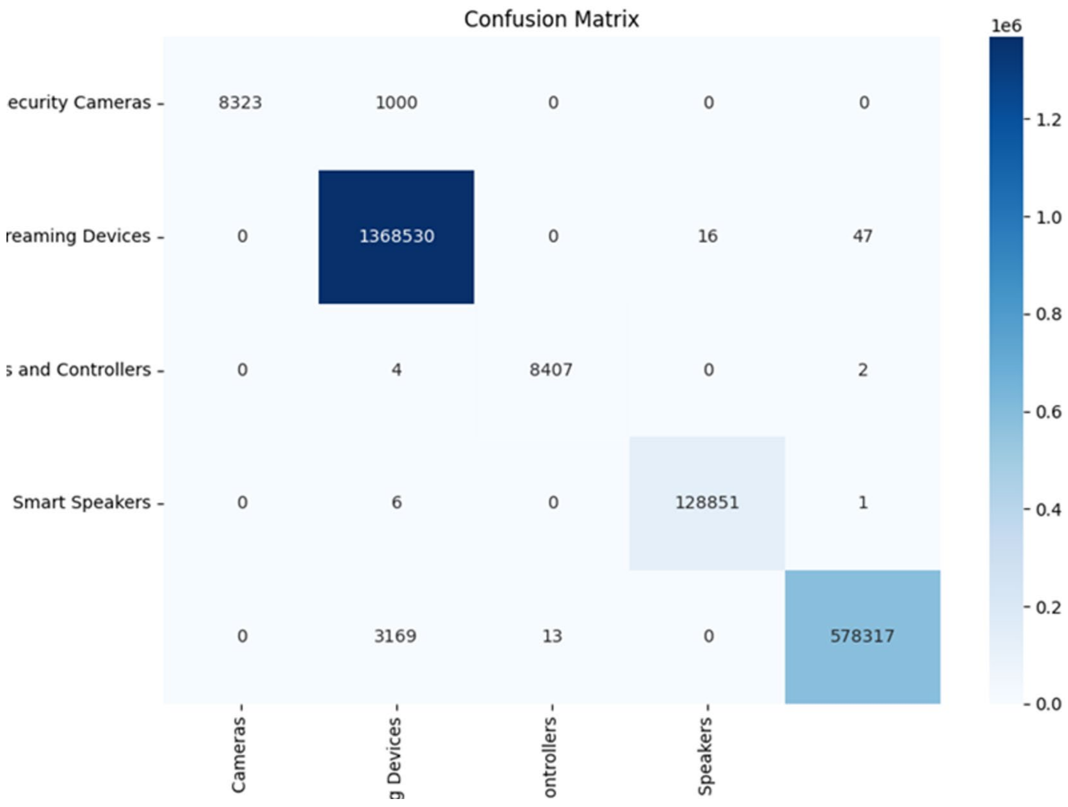

Fig. 15. Phase 3 Confusion Matrix (Dataset 2): Testing Evaluation.

Metric	Value
Precision	0.9980
Recall	0.9980
F1 Score	0.9980
Accuracy	0.9980

Table 10. Phase 3 Performance Metrics (Dataset 2): Testing Evaluation.

of 0.9980 shows that 99.80% of actual positive instances were successfully identified, implying negligible false negatives.

The F1 score of 0.9980 (99%) confirms a near-perfect balance between precision and recall, reinforcing the model's ability to correctly and comprehensively identify each class. The overall accuracy of 0.9980 means that 99.80% of all predictions, whether positive or negative, were correct. These consistently high performance values demonstrate the model's exceptional reliability and robustness in real-world multi-class IoT classification scenarios.

Phase-wise result interpretation in regard with OFSI, NLRI, and UI

The Phase-wise result interpretation regarding OFSI, NLRI, and UI from phase 1 (CNN) to phase 2 (CNN-RF) And then phase 3 (CNN-PN-RF) is presented in Fig. 16.The classification performance across Phases 1, 2, and 3 reveals a progressive evolution of this model's ability to distinguish between complex and overlapping IoT device categories. These phases demonstrate how iterative refinements and architectural adjustments impact the model's learning capability, generalization, and robustness. Three primary challenges were identified and tracked across these three phases:

- **Overlapping Feature Set Issue (OFSI)**[76,77]
- **Non-Linear Relationship Issue (NLRI)**[76,78,79]
- **Underperformance in Smaller Categories Issue (UI)**[80,81]

The challenges associated with feature learning in classification tasks can be mapped to three key issues. Firstly, the Overlapping Feature Set Issue (OFSI) is closely tied to intra-class separability. When intra-class separability is poor, the feature distributions of different classes overlap notably, making it difficult for the model to distinguish

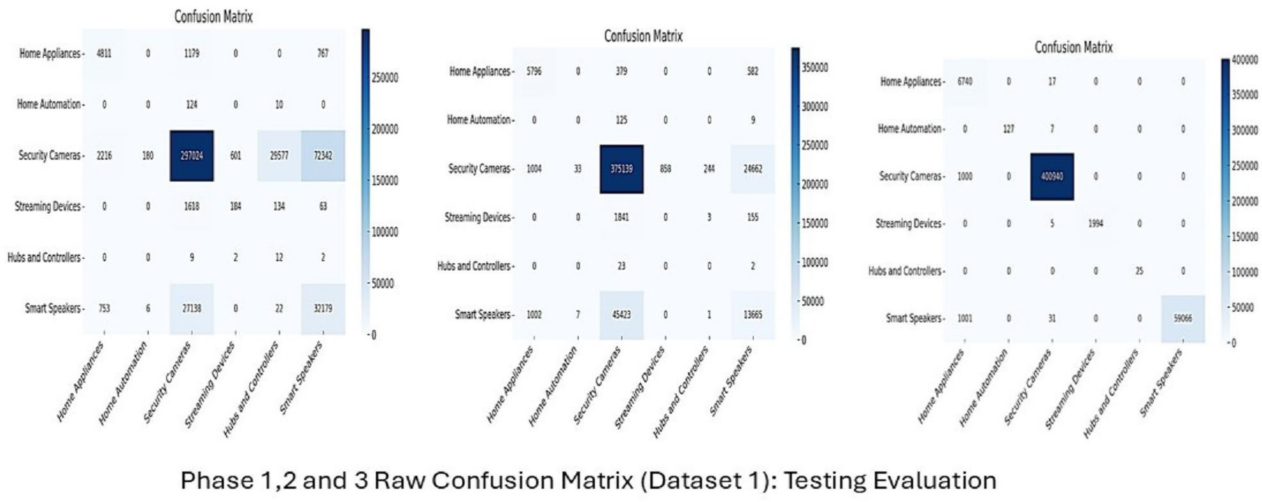

Fig. 16. Phase-wise result interpretation in regard with OFSI, NLRI, and UI.

between them effectively. Secondly, the Non-Linear Relationship Issue (NLRI) relates to feature clustering, where non-linear dependencies within the data disrupt simple clustering patterns. This requires the use of more complex models to accurately capture the underlying data structure. Lastly, the Underperformance in Smaller Categories (UI) issue corresponds to the need for small-class support. Smaller classes often suffer due to data imbalance, which can lead to disproportionately poor performance unless they are given specific attention during training.

Phase 1 interpretation
In the first phase, a standalone CNN model was employed to capture complex patterns and resolve overlapping feature similarities. However, the results indicate that the model faced significant limitations, particularly with Overlapping Feature Similarity Issues (OFSI). Notably, Smart Speakers were heavily misclassified as Security Cameras (27,138 instances), and conversely, Security Cameras were frequently predicted as Smart Speakers (72,342 instances). A similar trend was observed between Home Appliances and Security Cameras. These high bidirectional misclassification rates suggest that the CNN struggled to distinguish between these categories due to highly similar traffic characteristics such as continuous data flows, similar packet sizes, and temporal patterns typically observed in streaming or voice-based devices.

While CNNs are generally strong in handling overlapping classes in image or sequence data, in this case, the lack of spatial or temporal structure in the tabular network traffic features diminished the CNN's ability to learn distinctive representations. Unlike image pixels or time-series segments, these features did not provide meaningful local dependencies for convolutional filters to extract. As a result, the CNN learned general but non-discriminative patterns, leading to blurred class boundaries in the latent space.

The model also exhibited signs of Non-Linear Relationship Inefficiency (NLRI). For example, Security Cameras were incorrectly predicted as Home Appliances (2,216), Home Automation (180), and Streaming Devices (601), indicating that the CNN's current depth and non-linearity were inadequate to disentangle such complex overlaps in feature space. The latent representations lacked sufficient expressive power to form well-separated decision boundaries.

Additionally, Underrepresented Class Instability (UI) was evident in minority categories such as Home Automation and Hubs and Controllers, with only 0 and 12 correct predictions, respectively. Most samples from these underrepresented classes were redirected to dominant categories, a behavior indicative of class imbalance and sparse data learning. The CNN's softmax output layer likely favored the majority classes when handling uncertain or ambiguous feature patterns from these low-frequency groups.

These limitations in Phase 1 highlight the need for architectural enhancements and advanced learning mechanisms to improve class separation, nonlinear feature mapping, and sensitivity to underrepresented categories.

Phase 2 interpretation
In Phase 2, a Random Forest (RF) classifier was integrated with 1D CNN outputs to leverage RF's strength in handling non-linear class boundaries. This hybrid model demonstrated noticeable improvements over Phase 1, particularly in mitigating Overlapping Feature Similarity Issues (OFSI). The bidirectional confusion between Smart Speakers And Security Cameras improved notably. Misclassifications of Smart Speakers as Security Cameras increased from 27,138 to 45,423, while the reverse happened in the case of Security Cameras, which were misclassified as Smart Speakers, decreasing from 72,342 to 24,662. This reduction of nearly 40% in one direction illustrates RF's effectiveness in refining decision boundaries that 1D CNNs alone could not adequately distinguish in high-dimensional feature spaces. The enhancement likely stems from RF's ability to generate non-

linear decision paths and capture complex feature interactions beyond what 1D CNN convolutional layers can isolate.

Furthermore, Phase 2 addressed Non-Linear Relationship Inefficiency (NLRI) with measurable progress. Misclassifications of Security Cameras into Home Appliances (1,004), Streaming Devices (858), and Hubs (244) were all notably lower than in Phase 1, indicating better non-linear feature discrimination. This improvement is attributed to RF's ensemble-based decision process, which complements 1D CNN's feature extraction but overcomes its linear classification constraints.

Despite these gains, Underrepresented Class Instability (UI) remained problematic. Home Automation, for instance, had 125 instances incorrectly labeled as Security Cameras and 9 as Smart Speakers, with zero correct predictions. Hubs and Controllers were also poorly predicted, with only 2 correct classifications out of 25 samples. These persistent misclassifications reflect the limitations of RF in dealing with imbalanced class distributions, where dominant categories continue to overshadow minority ones.

Overall, Phase 2 marked substantial progress in handling OFSI and NLRI, yet it revealed the need for further architectural improvements and data-level techniques to improve stability and accuracy for minority classes.

Phase 3 interpretation

Phase 3 introduced a CNN–Prototypical Network–Random Forest (CNN–PN–RF) hybrid architecture, notably resolving the shortcomings of the earlier phases. Most notably, the Overlapping Feature Similarity Issue (OFSI) between Smart Speakers and Security Cameras was nearly eliminated. Where Phase 1 had 27,138 Smart Speakers misclassified as Security Cameras and 72,342 Security Cameras misclassified as Smart Speakers, Phase 3 reduced these to just 31 and 0 instances, respectively. This dramatic improvement is attributed to class-specific enhancements such as deeper feature extraction, prototype-based learning, and possible data augmentation strategies that enabled the model to more precisely differentiate overlapping classes.

Substantial gains were also observed in handling Non-Linear Relationship Inefficiency (NLRI). In Phase 2, Security Cameras were misclassified into unrelated categories like Home Appliances (1,000) and Streaming Devices (858). In contrast, Phase 3 achieved 400,940 correct predictions for Security Cameras, with only minor errors (1,000 to Home Appliances and 5 to Streaming Devices). These results suggest the model effectively captured complex nonlinear dependencies using the prototypical layer's ability to structure inter-class distances more meaningfully for RF to interpret.

Finally, Underrepresented Class Instability (UI) was substantially mitigated. Home Automation, which previously saw near-zero correct predictions, was correctly classified in 127 of 134 instances. Hubs and Controllers improved from just 2 correct predictions in Phase 2 to 25 out of 25 in Phase 3. This enhanced sensitivity to minority classes likely resulted from data balancing techniques (e.g., SMOTE, class-weighted loss functions) and the model's increased capacity to learn from sparse data distributions.

Overall, Phase 3 demonstrates significant progress across all key challenges (overlapping categories, non-linear separability, and class imbalance). The CNN–PN–RF framework successfully disentangled complex feature overlaps, improved boundary sharpness in high-dimensional space, and increased recognition of minority categories. These results underscore the value of integrating deeper neural architectures with prototype learning and ensemble classification to boost overall model robustness and generalizability.

Comparison with state-of-the-art approaches

The proposed hybrid CNN-PN-RF model outperformed and was more efficient than the state-of-the-art approaches. On Dataset 1, it obtained 99.56% accuracy, 99.66% precision, 99.56 % recall, And 99.59% percent F1-score. It achieved 99.80% on Dataset 2 on all measures, reflecting its adaptability and robustness with different network conditions.

Although the accuracy of most of the existing research works[48,49,51,53–55] is high (ranging from 95 to 99 percent), their effectiveness usually depends on known or fixed device sets, fewer training samples, and extensive feature engineering. For example, [48], considers 41 devices and 4 types of classes only, and does not do not validate on unseen devices. [49], and[51] rely on publicly available data which deos not generalize to unobserved IoT behaviors. Several models, such as[55] and[54], employ deep or transformer networks but they are limited in flexibility or scalability due to latency or excessive complexity or the lack of support for diverse data sets and protocols.

On the contrary, our CNN-PN-RF hybrid model uniquely fills these gaps, adopting generalization directly, tested on completely unseen IoT devicesand utilizing 6 datasets, including the highest number of device classifications (> 316). It classifies devices into 6 categoriesand is independent on feature engineering. The model manages behavioral drift effectively, is protocol-agnostic and sustaining state-of-the-art. Moreover, it sustains zero retraining and scales well to dynamic settings, achieving high performance with small volumes of labeled data, making it real-life IoT device identification solution.

As shown in Table 11, the CNN–PN–RF model consistently outperforms existing approaches. On Dataset 1, it achieved an accuracy of 99.56%, precision of 99.66%, recall of 99.56%, and F1-score of 99.59%, reflecting balanced and reliable performance. On Dataset 2, the model maintained outstanding consistency with an accuracy of 99.80%, precision of 99.80%, recall of 99.80%, and F1-score of 99.80%, demonstrating robustness across varying network conditions and device behavior.

Discussion

The progressive evaluations across Phases 1 through 3 clearly illustrate how strategic architectural and data-centric interventions can effectively address real-world classification challenges inherent to IoT environments. The enhancements implemented in each of the phases were phase specific, gradually increasing model

Method	Precision	Recall	F1 Score	Accuracy	Dataset Diversity	Model Type	Model Design	Unseen Evaluation	Others
State-of-the-Art Approaches									
Cvitic et al[48].	99.7–99.9%	99.7–99.9%	99.7–99.9%	99.79%	Medium- Only 41 IoT Devices Utilized	Ensemble ML	Feature Engineering Innovation-based	No	Only 4 classes/ Category Classification
Bao et al[41].	–	–	–	81.8–92.9%	Low (Only 10 devices used)	Hybrid Supervised and Unsupervised Learning	Architectural Innovation-based	No	Only 1 Class/ Category classification
Liu et al[55].	99%	99%	99%	99%	Medium- Only 3 Datasets Utilized with total 18 devices	DL Approach	Architectural Innovation-based	No	7 Classes/ Category Classification
Proposed Hybrid CNN–PN–RF Model									
This Proposed Model	99.66%, 99.80%	99.56%, 99.80%	99.59%, 99.80%	99.56%, 99.80%	High- Total 6 Datasets utilized, including more than 316 IoT devices	Incremental Hybrid (CNN-PN-RF)	Architectural Innovation-based	Yes–cross-checked on two diverse datasets (Dataset 1 And 2)	6 Class/ Category classification

Table 11. Comparison with state-of-the-art approaches.

expressiveness, minimizing misclassifications, and addressing the key weaknesses, including overlapping feature space and poor representation of minority classes.

The phase 3 result is achieved by a combination of deeper learning architectures, combined methods, and class-specific mechanisms. The model demonstrated robust generalization capabilities,effectively not only the dominant categories but also weak and minority device types that were posing challanges in the previous phase.

By lowering the data dimensionality using CNN feature extraction and PCA, the CNN–PN–RF model minimizes computational overhead and attains low cost training. Training stability is also ensured via batch normalization, dropout, and prototype-based feature grouping, which mitigate overfitting and improve convergence. The approach enhances security, device management, and network efficiency while being appropriate for resource-constrained real-world deployments. Scalability is attained by the hybrid design, in which the prototype network enables the classification of unseen devices without retraining, while the Random Forest generates dependable predictions over a range of datasets.

The final configuration brings out the compound effect of methodical improvement in the precision and stability of this proposed model. In addition, this proposed research proved to be successful in critical situations where there are data imbalance and feature overlap conditions that are common in heterogeneous IoT networks. Results gathered at the first phase are used as guidelines for the creation of the second phase, and then lead to the development of the third phase accordingly, which is scalable, precise, and generalizable in nature. This model's high application to the IoT sector demonstrates its relevance and effectiveness for real, dynamic IoT implementations.

After analyzing Fig. 17,we observe improved values from phase 1 to phase 2 and then phase 3 (Dataset 1). The transition from Phase 1 to Phase 2 using Dataset 1 reveals notable improvements across all performance metrics, emphasizing the advancements made in the proposed classification model.

An evaluation of precision revealed that the model achieved a value of 0.8238 in Phase 1, which slightly declined to 0.8123 in Phase 2. This marginal drop of approximately 1.15 percentage points suggests that, while the model continued to perform well, its ability to correctly identify relevant instances experienced a slight dip.

In terms of recall, the model demonstrated a notable improvement, rising from 0.7096 in Phase 1 to 0.8379 in Phase 2. This gain of around 12.83 percentage points indicates a significant enhancement in the model's capacity to detect true positives, leading to better identification of relevant cases.

The F1 score, which provides a balanced measure of precision and recall, increased from 0.7551 to 0.8229 between Phase 1 and Phase 2. This 6.78 percentage point improvement illustrates the model's strengthened performance in maintaining balance between identifying relevant instances and avoiding false positives.

Regarding accuracy, the model achieved its most substantial improvement in this phase, growing from 0.7096 to 0.8379. This represents a rise of 12.83 percentage points, reflecting the overall effectiveness of the enhancements applied in Phase 2 and confirming the model's suitability for practical deployment.

Further improvements were evident in the transition from Phase 2 to Phase 3 using Dataset 1. Across all performance metrics, the enhanced classification model demonstrated significant gains.

Precision increased markedly from 0.8123 in Phase 2 to 0.9966 in Phase 3, marking an improvement of 18.43 percentage points and showcasing the model's heightened ability to accurately identify relevant items.

Similarly, recall rose from 0.8379 to 0.9956, An improvement of 15.77 percentage points, indicating a stronger capability to capture true positive cases.

The F1 score also advanced notably, moving from 0.8229 in Phase 2 to 0.9959 in Phase 3. This 17.30 percentage point increase reflects the model's improved balance between precision and recall, further confirming its reliability.

Finally, accuracy improved from 0.8379 to 0.9956, also representing a gain of 15.77 percentage points. These results collectively highlight the effectiveness of the applied techniques and confirm the model's strong potential

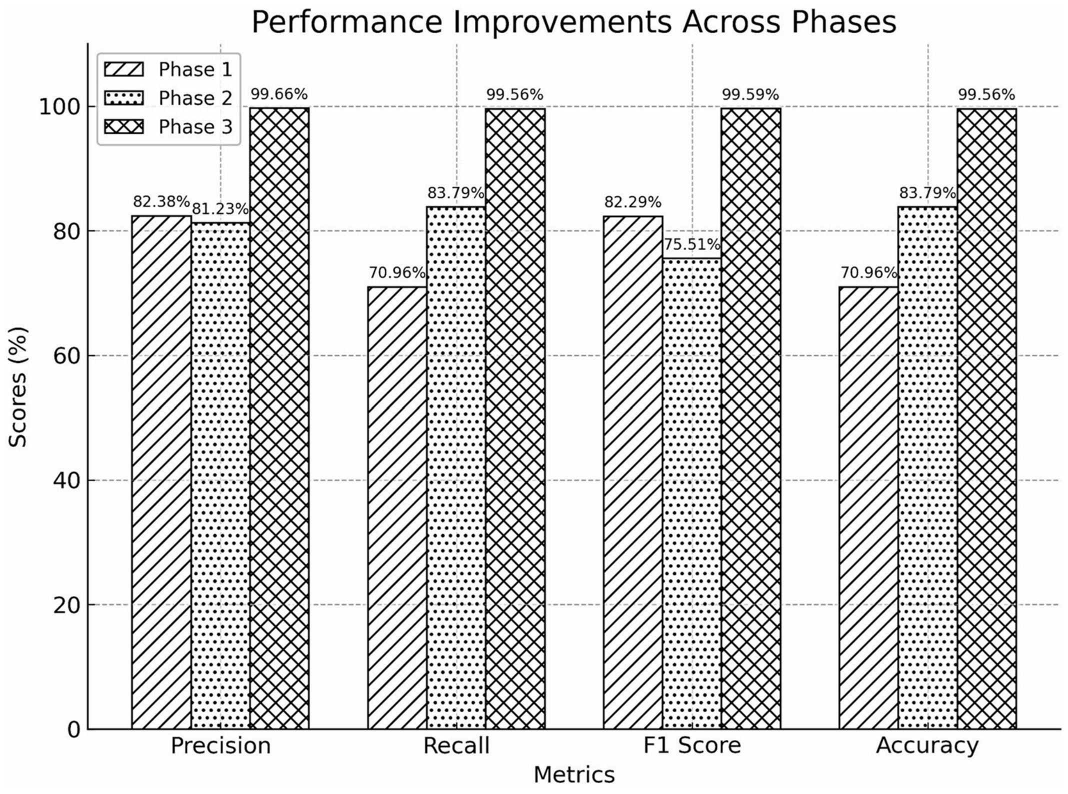

Fig. 17. Phase-wise performance improvement (Dataset 1).

Metric	Phase 1 ⟶ Phase 2	Phase 2 ⟶ Phase 3
Precision	82.38% ⟶ 81.23% (−1.15%, decreased)	81.23% ⟶ 99.66% (+18.43%, increased)
Recall	70.96% ⟶ 83.79% (+12.83%, increased)	83.79% ⟶ 99.56% (+15.77%, increased)
F1 Score	75.51% ⟶ 82.29% (+6.78%, increased)	82.29% ⟶ 99.59% (+17.30%, increased)
Accuracy	70.96% ⟶ 83.79% (+12.83%, increased)	83.79% ⟶ 99.56% (+15.77%, increased)

Table 12. Performance evaluation improvement from Phase 1 to Phase 3.

for real-world implementation, demonstrating impressive progress across Phases 1 to 3. Further details of the improvements are presented in Table 12.

Conclusion and future directions

This study presented a high-accuracy, generalized hybrid model (CNN-PN-RF) for classifying unseen IoT devices, demonstrating significant improvements across three evaluation phases. By integrating convolutional neural networks, prototypical networks, and a random forest classifier, our model effectively captured inter-class similarities and improved feature clustering, resulting in superior generalization capabilities.

Model performance improved from 70.96% accuracy in Phase 1 to 83.79% in Phase 2, and finally to 99.56% in Phase 3, An absolute accuracy gain of 28.6% on Dataset 1. Similar improvements were observed on Dataset 2, further validating the model's generalizability. The focus of this study was the classification of IoT devices using a generalized hybrid model, with particular emphasis on its ability to handle unseen device classification.

Threats to validity

Despite the promising results of the proposed model, many limitations are still there, which we consider as threats to validity. First of all, the dataset might not correctly reflect the range of real IoT scenarios due to bias imposed by its manual structuring. Second, the evaluation relied on self-prepared benchmark datasets that might not accurately reflect all traffic patterns and anomalies found in real-time implementations. Third, while unseen device generalization was tested under controlled conditions, performance may differ in large-scale, highly heterogeneous networks. These factors may affect the external validity of the results. Future studies should consider larger, more diverse datasets and real-world testing.

Future directions

Several future directions exist for improving and expanding our approach to overcome these threats to validity concerns. These include:

Refining manual data structuring and class formation
In our data preparation phase, we manually structured the dataset by grouping devices into classes based on domain knowledge. For instance, all camera-based devices were grouped under "Home Security Cameras," and devices used for control functions were classified under "Home Automation." However, this approach may not always be precise, as certain devices can belong to multiple categories.

For example, a "Smart Home Hub" could reasonably be classified under both "Home Automation" and "Home Security." This ambiguity can lead to misclassifications. Future research could explore automated, data-driven class formation, possibly using unsupervised learning techniques, which would allow the model to define logical groupings based on inherent device characteristics.

Exploring unstructured data and semi-supervised learning
Currently, our model relies on structured data with predefined classes. However, real-world data is often unstructured, presenting both a challenge and an opportunity for more realistic evaluation. Future studies could investigate unsupervised or semi-supervised learning approaches to handle such data directly. Techniques like clustering or graph-based learning could enhance the model's ability to adapt to unseen devices and traffic behaviors without extensive labeling, improving flexibility and reducing reliance on manually curated class definitions.

Investigating meta-learning and few-shot learning
While hybrid models combining deep learning and classical machine learning methods offer high performance, they often introduce computational complexity. As an alternative, future research could explore meta-learning and few-shot learning approaches either in isolation or with lighter models. These methods promise faster training, lower resource usage, and the ability to quickly adapt to new, unknown devices with minimal retraining, potentially making them more suitable for constrained or real-time environments.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Received: 21 July 2025; Accepted: 8 September 2025
Published online: 09 October 2025

References

1. Naik, K. & Patel, S. An open source smart home management system based on iot. *Wireless Networks* **29**, 989–995. https://doi.org/10.1007/s11276-018-1884-z (2023).
2. Chen, J., Liu, Y., Zhang, S., Chen, B. & Han, Z. A survey of traffic classification technology for smart home based on machine learning. In *Proceedings of the International Conference on Artificial Intelligence and Security*, 544–557, https://doi.org/10.1007/978-3-031-06791-4_43 (Springer, 2022).
3. Mainuddin, M., Duan, Z., Dong, Y., Salman, S. & Taami, T. Iot device identification based on network traffic characteristics. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 6067–6072, https://doi.org/10.1109/GLOBECOM48099.2022.10001639 (IEEE, 2022).
4. Niakanlahiji, A., Orlowski, S., Vahid, A. & Jafarian, J. H. Toward practical defense against traffic analysis attacks on encrypted dns traffic. *Computers-Security* **124**, 103001. https://doi.org/10.1016/j.cose.2022.103001 (2023).
5. Bzai, J. et al. Machine learning-enabled internet of things (iot): Data, applications, and industry perspective. *Electronics* **11**, 2676. https://doi.org/10.3390/electronics11172676 (2022).
6. Chi, H. & Chi, Y. Smart home control and management based on big data analysis. *Computational Intelligence and Neuroscience* **2022**, 3784756. https://doi.org/10.1155/2022/3784756 (2022).
7. Khandait, P., Hubballi, N. & Mazumdar, B. Iothunter: Iot network traffic classification using device specific keywords. *IET Networks* **10**, 59–75 (2021).
8. Xiong, Y., Dong, S., Liu, R., Shi, F. & Jing, X. Iot network traffic classification: a deep learning method with fourier transform-assisted hyperparameter optimization. *Frontiers in Physics* **11**, 1273862. https://doi.org/10.3389/fphy.2023.1273862 (2023).
9. Rafique, S. H., Abdallah, A., Musa, N. S. & Murugan, T. Machine learning and deep learning techniques for internet of things network anomaly detection-current research trends. *Sensors* **24**, 1968. https://doi.org/10.3390/s24061968 (2024).
10. Jang, H.-C. & Chiu, C.-J. Using deep q-network in bandwidth allocation of smart homes. In *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 0098–0101, https://doi.org/10.1109/IEMCON53756.2021.9623084 (IEEE, 2021).
11. Jmila, H., Blanc, G., Shahid, M. R. & Lazrag, M. A survey of smart home iot device classification using machine learning-based network traffic analysis. *IEEe Access* **10**, 97117–97141. https://doi.org/10.1109/ACCESS.2022.3205023 (2022).
12. Kolcun, R., et al. Revisiting iot device identification. *arXiv preprint* arXiv:2107.07818https://doi.org/10.48550/arXiv.2107.07818 (2021).
13. Pinheiro, A. J., Bezerra, J. D. M., Burgardt, C. A. & Campelo, D. R. Identifying iot devices and events based on packet length from encrypted traffic. *Computer Communications* **144**, 8–17. https://doi.org/10.1016/j.comcom.2019.05.004 (2019).
14. Yang, L. & Shami, A. Iot data analytics in dynamic environments: From an automated machine learning perspective. *Engineering Applications of Artificial Intelligence* **116**, 105366. https://doi.org/10.1016/j.engappai.2022.105366 (2022).
15. Abusitta, A. et al. Deep learning-enabled anomaly detection for iot systems. *Internet of Things* **21**, 100656. https://doi.org/10.1016/j.iot.2023.100656 (2023).

16. Yang, L. & Shami, A. A lightweight concept drift detection and adaptation framework for iot data streams. *IEEE Internet of Things Magazine* **4**, 96–101. https://doi.org/10.48550/arXiv.2104.10529 (2021).
17. Yousefnezhad, N., Malhi, A. & Framling, K. Automated iot device identification based on full packet information using real-time network traffic. *Sensors* **21**, 2660. https://doi.org/10.3390/s21082660 (2021).
18. Wang, J., Zhong, J. & Li, J. Iot-portrait: Automatically identifying iot devices via transformer with incremental learning. *Future Internet* **15**, 102. https://doi.org/10.3390/fi15030102 (2023).
19. Andrews, A., Oikonomou, G., Armour, S., Thomas, P. & Cattermole, T. Iot firmware version identification using transfer learning with twin neural networks. *arXiv preprint* arXiv:2501.06033https://doi.org/10.48550/arXiv.2501.06033 (2025).
20. Kolcun, R. *et al.* The case for retraining of ml models for iot device identification at the edge. *arXiv preprint* arXiv:2011.08605 (2020).
21. Chung, Y., Haas, P. J., Upfal, E. & Kraska, T. Unknown examples-machine learning model generalization. *arXiv preprint* arXiv:1808.08294 (2018).
22. Keipour, H., Hazra, S., Finne, N. & Voigt, T. Generalizing supervised learning for intrusion detection in iot mesh networks. In *International Conference on Ubiquitous Security*, 214–228, https://doi.org/10.1007/978-981-19-0468-4_16 (Springer, 2021).
23. Mahadevan, A. & Mathioudakis, M. Cost-aware retraining for machine learning. *Knowledge-Based Systems* **293**, 111610. https://doi.org/10.1016/j.knosys.2024.111610 (2024).
24. Charyyev, B. & Gunes, M. H. Locality-sensitive iot network traffic fingerprinting for device identification. *IEEE Internet of Things Journal* **8**, 1272–1281. https://doi.org/10.1109/JIOT.2020.3035087 (2020).
25. Pramod, A., Naicker, H. S. & Tyagi, A. K. Machine learning and deep learning: Open issues and future research directions for the next 10 years. *Computational analysis and deep learning for medical care: Principles, methods, and applications* 463–490, https://doi.org/10.1002/9781119785750.ch18 (2021).
26. Kalwar, J. H. & Bhatti, S. Deep learning approaches for network traffic classification in the internet of things (iot): A survey. *arXiv preprint* arXiv:2402.00920 (2024).
27. Moreira, R., Rodrigues, L. F., Rosa, P. F. & Silva, F. d. O. Improving the network traffic classification using the packet vision approach. *arXiv preprint* arXiv:2412.19360 (2024).
28. Deshmukh, A. & Ravulakollu, K. An efficient cnn-based intrusion detection system for iot: Use case towards cybersecurity. *Technologies* **12**, 203 (2024).
29. Xiang, Q., Wu, S., Wu, D., Liu, Y. & Qin, Z. Research on cnn-bilstm network traffic anomaly detection model based on mindspore. *arXiv preprint* arXiv:2504.21008 (2025).
30. Bovenzi, G., Di Monda, D., Montieri, A., Persico, V. & Pescape, A. Classifying attack traffic in iot environments via few-shot learning. *Journal of Information Security and Applications* **83**, 103762 (2024).
31. Zhu, H. & Koniusz, P. Transductive few-shot learning with prototype-based label propagation by iterative graph refinement. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 23996–24006 (2023).
32. Tan, Z., Guo, R., Ding, K. & Liu, H. Virtual node tuning for few-shot node classification. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2177–2188 (2023).
33. An, W. et al. Generalized category discovery with decoupled prototypical network. *In Proceedings of the AAAI Conference on Artificial Intelligence* **37**, 12527–12535 (2023).
34. Lim, J. Y., Lim, K. M., Lee, C. P. & Tan, Y. X. Ssl-protonet: Self-supervised learning prototypical networks for few-shot learning. *Expert Systems with Applications* **238**, 122173 (2024).
35. Zouhri, H., Idri, A. & Ratnani, A. Evaluating the impact of filter-based feature selection in intrusion detection systems. *International Journal of Information Security* **23**, 759–785 (2024).
36. Zouhri, H., Idri, A. & Hakkoum, H. Assessing the effectiveness of dimensionality reduction on the interpretability of opaque machine learning-based attack detection systems. *Computers and Electrical Engineering* **120**, 109627 (2024).
37. Chowdhury, R. R., Idris, A. C. & Abas, P. E. Identifying sh-iot devices from network traffic characteristics using random forest classifier. *Wireless networks* **30**, 405–419 (2024).
38. Altunay, H. C. & Albayrak, Z. A hybrid cnn+ lstm-based intrusion detection system for industrial iot networks. *Engineering Science and Technology, an International Journal* **38**, 101322. https://doi.org/10.1016/j.jestch.2022.101322 (2023).
39. Ali, M. et al. Hybrid machine learning model for efficient botnet attack detection in iot environment. *IEEE Access* https://doi.org/10.1109/ACCESS.2024.3376400 (2024).
40. Nazir, A. et al. A deep learning-based novel hybrid cnn-lstm architecture for efficient detection of threats in the iot ecosystem. *Ain Shams Engineering Journal* **15**, 102777. https://doi.org/10.1016/j.asej.2024.102777 (2024).
41. Bao, J., Hamdaoui, B. & Wong, W.-K. Iot device type identification using hybrid deep learning approach for increased iot security. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, 565–570, https://doi.org/10.1109/IWCMC48107.2020.9148110 (IEEE, 2020).
42. Nguyen, X.-H. & Le, K.-H. Robust detection of unknown dos/ddos attacks in iot networks using a hybrid learning model. *Internet of Things* **23**, 100851. https://doi.org/10.1016/j.iot.2023.100851 (2023).
43. Bao, J. Network traffic-driven hybrid learning for classifying seen and unseen iot device types. *Missing* (2020).
44. Chen, Q. *et al.* Iot-id: robust iot device identification based on feature drift adaptation. In *2021 IEEE Global Communications Conference (GLOBECOM)*, 1–6, https://doi.org/10.1109/GLOBECOM46510.2021.9685693 (IEEE, 2021).
45. Kimanzi, R., Kimanga, P., Cherori, D. & Gikunda, P. K. Deep learning algorithms used in intrusion detection systems–a review. *arXiv preprint* arXiv:2402.17020https://doi.org/10.48550/arXiv.2402.17020 (2024).
46. Benmalek, M. & Seddiki, A. Particle swarm optimization-enhanced machine learning and deep learning techniques for internet of things intrusion detection. *Data Science and Management* https://doi.org/10.1016/j.dsm.2025.02.005 (2025).
47. Aqil, N. et al. Improved temporal iot device identification using robust statistical features. *PeerJ Computer Science* **10**, e2145. https://doi.org/10.7717/peerj-cs.2145 (2024).
48. Cvitic, I., Perakovic, D., Perivsa, M. & Gupta, B. Ensemble machine learning approach for classification of iot devices in smart home. *International Journal of Machine Learning and Cybernetics* **12**, 3179–3202. https://doi.org/10.1007/s13042-020-01241-0 (2021).
49. Kostas, K., Just, M. & Lones, M. A. Iotdevid: A behavior-based device identification method for the iot. *IEEE Internet of Things Journal* **9**, 23741–23749. https://doi.org/10.1109/JIOT.2022.3191951 (2022).
50. Xu, Z., Lu, Q., Chen, F. & Xian, H. Lfiotdi: A lightweight and fine-grained device identification approach for iot security enhancement. *Computer Communications* **237**, 108149. https://doi.org/10.1016/j.comcom.2025.108149 (2025).
51. Fan, L. *et al.* An iot device identification method based on semi-supervised learning. In *2020 16th International Conference on Network and Service Management (CNSM)*, 1–7, https://doi.org/10.23919/CNSM50824.2020.9269044 (IEEE, 2020).
52. Niu, K. *et al.* Ensiot: A stacking ensemble learning approach for iot device identification. In *2024 IEEE-ACM 32nd International Symposium on Quality of Service (IWQoS)*, 1–6, https://doi.org/10.1109/IWQoS61813.2024.10682848 (IEEE, 2024).
53. Kotak, J. & Elovici, Y. Iot device identification using deep learning. In *13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020) 12*, 76–86, https://doi.org/10.1007/978-3-030-68288-6_8 (Springer, 2021).
54. Deng, L., Gu, D. & Lin, Z. IoT Device Identification Method Based on Transformer and Clustering. *Available at SSRN* https://doi.org/10.2139/ssrn.5211585 (2024). SSRN ID: 5211585.
55. Liu, X., Han, Y. & Du, Y. Iot device identification using directional packet length sequences and 1d-cnn. *Sensors* **22**, 8337. https://doi.org/10.3390/s22218337 (2022).

56. Yin, Y. et al. Graphiot: Lightweight iot device detection based on graph classifiers and incremental learning. *IEEE Transactions on Services Computing* https://doi.org/10.1109/TSC.2024.3466854 (2024).
57. Li, J., Othman, M. S., Chen, H. & Yusuf, L. M. Optimizing iot intrusion detection system: feature selection versus feature extraction in machine learning. *Journal of Big Data* **11**, 36. https://doi.org/10.1186/s40537-024-00892-y (2024).
58. Kingeski, R., Henning, E. & Paterno, A. S. Fusion of pca and ica in statistical subset analysis for speech emotion recognition. *Sensors* **24**, 5704. https://doi.org/10.3390/s24175704 (2024).
59. John, A., Isnin, I. F. B., Madni, S. H. H. & Muchtar, F. B. Enhanced intrusion detection model based on principal component analysis and variable ensemble machine learning algorithm. *Intelligent Systems with Applications* **24**, 200442 (2024).
60. Talukder, M. A. et al. Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction. *Journal of big data* **11**, 33. https://doi.org/10.1186/s40537-024-00886-w (2024).
61. Guo, W. et al. Malfsldf: A few-shot learning-based malware family detection framework. *International Journal of Intelligent Systems* **2025**, 7390905. https://doi.org/10.1155/int/7390905 (2025).
62. Zouhri, H. & Idri, A. A novel ctgan-enn hybrid approach to enhance the performance and interpretability of machine learning black-box models in intrusion detection and iot. *Future Generation Computer Systems* 107882 (2025).
63. Awotunde, J. B. et al. An enhanced convolutional neural network with principal component analysis for pneumonia diagnosis classification from medical images. *Procedia Computer Science* **258**, 1496–1505 (2025).
64. Berrich, Y. & Guennoun, Z. Eeg-based epilepsy detection using cnn-svm and dnn-svm with feature dimensionality reduction by pca. *Scientific Reports* **15**, 14313 (2025).
65. Gao, J., Hu, W. & Chen, Y. Revisiting pca for time series reduction in temporal dimension. *arXiv preprint* arXiv:2412.19423 (2024).
66. Mehrabinezhad, A., Teshnehlab, M. & Sharifi, A. A comparative study to examine principal component analysis and kernel principal component analysis-based weighting layer for convolutional neural networks. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization* **12**, 2379526 (2024).
67. Zhu, M. et al. semg-based lower limb motion prediction using cnn-lstm with improved pca optimization algorithm. *Journal of Bionic Engineering* **20**, 612–627 (2023).
68. Gewers, F. L. et al. Principal component analysis: A natural approach to data exploration. *ACM Computing Surveys (CSUR)* **54**, 1–34 (2021).
69. Sadegh-Zadeh, S.-A. et al. Comparative analysis of dimensionality reduction techniques for eeg-based emotional state classification. *American Journal of Neurodegenerative Disease* **13**, 23 (2024).
70. Heinrichs, F. Gt-pca: Effective and interpretable dimensionality reduction with general transform-invariant principal component analysis. *arXiv preprint* arXiv:2401.15623 (2024).
71. Gyimadu, M. & Bell, G. A comparative analysis of principal component analysis (pca) and singular value decomposition (svd) as dimensionality reduction techniques. *arXiv preprint* arXiv:2506.16663 (2025).
72. Zhou, F., Wang, P., Zhang, L., Wei, W. & Zhang, Y. Revisiting prototypical network for cross domain few-shot learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 20061–20070, https://doi.org/10.1109/CVPR52729.2023.01921 (2023).
73. Thein, T. T., Shiraishi, Y. & Morii, M. Few-shot learning-based malicious iot traffic detection with prototypical graph neural networks. *IEICE TRANSACTIONS on Information and Systems* **106**, 1480–1489. https://doi.org/10.1587/transinf.2022OFP0004 (2023).
74. Maran, P. *et al.* Comparison of machine learning models for iot malware classification. In *International Conference on Computer, Information Technology and Intelligent Computing (CITIC 2022)*, 15–28 (Atlantis Press, 2022).
75. Salman, O., Elhajj, I. H., Chehab, A. & Kayssi, A. A machine learning based framework for iot device identification and abnormal traffic detection. *Transactions on Emerging Telecommunications Technologies* **33**, e3743. https://doi.org/10.1002/ett.3743 (2022).
76. Akrout, M., Feriani, A., Bellili, F., Mezghani, A. & Hossain, E. Domain generalization in machine learning models for wireless communications: Concepts, state-of-the-art, and open issues. *IEEE Communications Surveys & Tutorials* **25**, 3014–3037. https://doi.org/10.1109/COMST.2023.3326399 (2023).
77. Panthi, M. & Das, T. K. Intelligent intrusion detection scheme for smart power-grid using optimized ensemble learning on selected features. *International Journal of Critical Infrastructure Protection* **39**, 100567. https://doi.org/10.1016/j.ijcip.2022.100567 (2022).
78. Verma, R. D. et al. Optimal partitioning of unbalanced datasets for bgp anomaly detection. *In Telecom* **6**, 25. https://doi.org/10.3390/telecom6020025 (MDPI (2025).
79. Romo-Chavero, M. A., Alatorre, G. D. L. R., Cantoral-Ceballos, J. A., Perez-Diaz, J. A. & Martinez-Cagnazzo, C. A hybrid model for bgp anomaly detection using median absolute deviation and machine learning. *IEEE Open Journal of the Communications Society* https://doi.org/10.1109/OJCOMS.2025.3550010 (2025).
80. Altalhan, M., Algarni, A. & Alouane, M.T.-H. Imbalanced data problem in machine learning: A review. *IEEE Access* https://doi.org/10.1109/ACCESS.2025.3531662 (2025).
81. Abbasi, M., Shahraki, A., Prieto, J., Arrieta, A. G. & Corchado, J. M. Unleashing the potential of knowledge distillation for iot traffic classification. *IEEE Transactions on Machine Learning in Communications and Networking* **2**, 221–239. https://doi.org/10.1109/TMLCN.2024.3360915 (2024).

Acknowledgements
The authors would like to thank Universiti Malaysia Pahang-Al Sultan Abdullah (UMPSA), Malaysia, for providing financial assistance for this research work. This work was supported by Research Grant Scheme (FRGS) of Universiti Malaysia Pahang-Al Sultan Abdullah under Grant no: FRGS/1/2023/ICT02/UMP/02/7 and RDU Grant no: RDU230144.

Author contributions
1. Quadri Waseem (Q.W): Conceptualization, Literature, software, Writing the original draft. 2. Wan Isni Sofiah Wan Din (W.I.S.W.D): Methodology and Supervision 3. Muhammad Aamir (M.A): Investigation and Validation.

Funding
Open access funding provided by Universiti Malaysia Pahang-Al Sultan Abdullah (UMPSA), Malaysia.

Declarations

Competing interests
The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to W.I.S.W.D.

Reprints and permissions information is available at www.nature.com/reprints.