GeMID: Generalizable Models for IoT Device Identification

Kahraman Kostas, Rabia Yasa Kostas, Mike Just, and Michael A. Lones, *Senior Member, IEEE*

***Abstract*—With the proliferation of Internet of Things (IoT) devices, ensuring their security has become paramount. Device identification (DI), which distinguishes IoT devices based on their traffic patterns, plays a crucial role in both differentiating devices and identifying vulnerable ones, closing a serious security gap. However, existing approaches to DI that build machine learning models often overlook the challenge of model generalizability across diverse network environments. In this study, we propose a novel framework to address this limitation and evaluate the generalizability of DI models across datasets collected within different network environments. Our approach involves a two-step process: first, we develop a feature and model selection method that is more robust to generalization issues by using a genetic algorithm with external feedback and datasets from distinct environments to refine the selections. Second, the resulting DI models are then tested on further independent datasets in order to robustly assess their generalizability. We demonstrate the effectiveness of our method by empirically comparing it to alternatives, highlighting how fundamental limitations of commonly employed techniques such as sliding window and flow statistics limit their generalizability. Our findings advance research in IoT security and device identification, offering insights into improving model effectiveness and mitigating risks in IoT networks.**

***Index Terms*—IoT security, device identification, machine learning, generalizability.**

I. Introduction

THE Internet of Things (IoT) seamlessly integrates our cyber world with the physical world and is becoming increasingly prevalent in daily life. According to published statistics, the number of IoT devices has exceeded 15 billion and is projected to reach approximately 30 billion by 2030 [1].

Despite the rapid proliferation of devices and providers, security remains a critical issue. IoT devices can be more challenging to secure than conventional devices due to their diverse hardware, software, and varied manufacturer profiles [2]. Moreover, unlike conventional devices, many IoT devices have non-standard interfaces which impair user interaction and make it difficult to mitigate against vulnerabilities [3]. The NETSCOUT Threat Intelligence Report [4] indicates that a new IoT device on the network typically faces its first attack within 5 hours and becomes a specific attack target within 24 hours, and that most attacks exploit vulnerabilities in IoT devices [5]. Beyond being targets in attacks, compromised IoT devices can serve as tools in botnet attacks, exemplified by Mirai, URSNIF, and BASHLITE [6], where captured devices are used to orchestrate high-volume Distributed Denial of Service (DDoS) attacks. While various methods have been developed to detect and mitigate ongoing attacks, primarily Intrusion Detection Systems (IDS), proactive measures such as device-specific updates, internet access restrictions or isolation can prevent these attacks before they happen. Given the variety, quantity and unfamiliar interfaces of IoT devices, it is impractical to rely on users to implement these solutions. As a result, DI systems have been developed to automatically identify devices based on their activity and make it possible to apply appropriate security measures to create a secure IoT ecosystem.

Most existing approaches to DI involve the construction of Machine Learning (ML) models. When developing ML models, it is important to ensure that they generalize beyond the specific environment in which they were trained. In DI, this means that models trained on data from one device should be able to detect other devices of the same make and model operating within different network environments. Achieving this requires accounting for the domain shifts and variations typical of network and IoT environments [7]. In this regard, a prominent limitation of previous work is that most studies have relied upon a single dataset collected in a single network environment for both developing and testing their models [8]–[10]. This has resulted in an incomplete, and potentially misleading, picture of DI generalizability. Where multiple datasets have been used, the focus has been on the generalizability of methodologies and feature sets rather than of model instances [11]–[13].

In this study, we explicitly focus on developing DI model instances that are demonstrably generalizable across network environments. We measure this using multiple datasets containing the same devices operating in different network environments. We use a two-stage process to develop the models. The first stage involves feature and model selection, and focuses on identifying device-independent features that do not display over-dependence on their network environment. In the second stage, we use an independent dataset to train device-specific model instances. We then use a further dataset, containing the same devices operating in a different network environment, to evaluate the generalizability of these device-specific models.

We make the following contributions to the field of DI:

1) **Comprehensive evaluation of model generalizability**: We provide an extensive assessment of ML-based DI models across diverse network environments, highlighting key factors that influence generalizability.
2) **Novel study framework for assessing generalizability**: We introduce a two stage framework specifically

Corresponding author: Kahraman Kostas. Kahraman Kostas is with the Ministry of National Education, Türkiye. Rabia Yasa Kostas is with Gümüşhane University, Türkiye. Mike Just and Michael A. Lones are with the Department of Computer Science, Heriot-Watt University, Edinburgh EH14 4AS, U.K. (e-mail: kahraman.kostas@meb.gov.tr; rabia.yasakostas@gumushane.edu.tr; m.just@hw.ac.uk; m.lones@hw.ac.uk).

designed to evaluate the generalizability of DI models across varied conditions, offering a more rigorous methodology for testing model robustness.

3) **Insight into feature selection and its impact on generalizability**: We demonstrate how the method of feature selection and construction critically impacts the generalizability of DI models. In particular, we show that features derived from individual packet characteristics lead to superior generalizability compared to those based on flow or window statistics.
4) **Validation of packet-based approaches**: Through empirical comparisons, we validate our packet-based method against other approaches, demonstrating that models built on individual packet features consistently outperform methods that use flow or window-based statistics in terms of generalizability.
5) **Commitment to transparency and reproducibility**: To foster transparency and encourage further research, we openly share our code and analysis[1], providing the community with the resources to replicate and build upon our findings.

This paper is organized as follows: Section II reviews related work, Section III covers data selection and feature extraction, Section IV details the process used for selecting and evaluating features, Section V presents model selection and the results of our model generalizability study, Section VI discusses limitations, and Section VII concludes.

II. Related Work

The field of IoT DI has been developing in earnest since 2017. Among the pioneering works is IoTSentinel [14], which builds models using 23 features extracted from packet headers. Packet headers, analogous to envelopes for data traveling over a network, contain information for routing and processing the data. By analyzing these headers, various useful features for network tasks can be extracted. Fig. 1 illustrates a sample network packet and lists the header features present in it. Extractable features from packet headers include source/destination IP/MAC addresses, protocol, Time-to-Live (TTL), flag information, port numbers, sequence/acknowledgment numbers, and checksum. A significant contribution of this study was the introduction of Aalto University IoT device captures [15], one of the most widely used open-access datasets in the IoT DI domain. Subsequent studies, such as IoTSense [9], expanded the feature sets by incorporating payload features (such as payload size and entropy) alongside header-derived features. Concurrently, other research focused on statistics derived from headers, emphasizing the relationship between packets within a specified range (e.g., TTL [8], packet size [16], time [16]).

In IoTSentinel and IoTSense, features are derived from individual packet headers, but ML models are trained using features collected from multiple packets, known as fingerprints. These fingerprints consist of 12 packets in IoTSentinel and 20 packets in IoTSense. Despite using multiple packets, these fingerprints simply concatenate individual packet features, rather

[1]Source code available at github.com/kahramankostas/GeMID

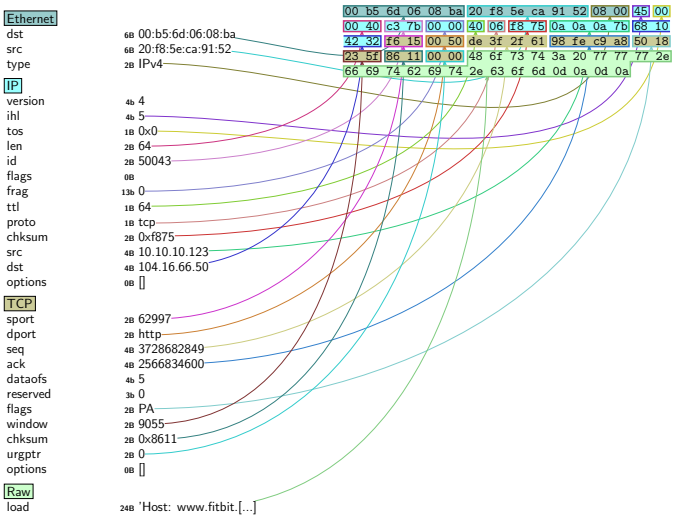


Fig. 1. Raw bytes and headers of a network packet from the Fitbit Aria WiFi enabled weighing device.

than generating inter-packet features. Alternatively, SysID [10] and IoTDevID [13] create fingerprints based on individual packet features, avoiding the need for merged data features. Obtaining fingerprints from an individual packet addresses the transfer problem that occurs when multiple devices share the same MAC address, a common issue due to the use of MAC addresses in the preassembly of packets [13].

Deriving features from flow data instead of packet headers is also common. A flow refers to a sequence of packets sent from a specific source to a specific destination, considered as part of a single session or connection. The size of the flow can be a predetermined number of packets or packets within a specific time/protocol interval. These features can include basic flow information (source, destination, protocol), time-related metrics (start-end time, duration), packet and byte counts (total, average, max, min sizes), rate metrics (packet, byte, flow rate), flag statistics, inter-arrival times (average, max, min), and flow direction indicators (packets/bytes sent between source and destination). Sivanathan et al. [17] used features obtained from network flows, such as flow volume, flow duration, and average flow rate. They also introduced the widely used UNSW-DI dataset. Many other DI studies have also used flow features [18], [19].

Features can also be obtained from packet headers or flows through sliding windows of various sizes. For example, features such as port ranges, packet size, packet quantity, availability time and inter-arrival time can be subjected to statistical analysis within the window to derive features such as maximum, minimum, mean and standard deviation [20]. This inherently involves features based on multiple packets. However, an issue with features derived from flow/network statistics and windowing methods is that they are likely to capture implicit characteristics of the network environment, rather than just the characteristics of individual devices. In this paper, we argue that this makes them fragile, and demonstrate that they are not a reliable basis for carrying out DI.

The use of raw data is also possible. As shown in Fig. 1,

a network packet can be represented as raw bytes. In this approach, the first n bytes of the packet are provided as input to the ML models. Instead of being used explicitly as features, this data is fed into neural network-based learning models, in particular Convolutional Neural Networks (CNNs), where each byte is treated as an image pixel [21]–[25]. If the number of bytes exceeds n, truncation is applied; if less, padding is used. This method is conceptually similar to the packet header method; but its fields are not strictly separated.

However, using raw data for DI presents other problems that have not yet been adequately addressed. In studies where the entire raw packet data was used [22], [24], this included headers, payloads, and metadata, making it difficult to filter out identifying information such as MAC/IP addresses or string identifiers, both of which can significantly impair model generalizability. Whilst this dependency can be reduced by solely using packet payloads [23], [26], caution is needed because much of today's internet traffic is encrypted. In practice encrypted payloads are likely to be ineffective for device identification models as the data appears random and opaque, providing no useful information about the contents. Thus, the added complexity of raw data is unlikely to contribute to the model's performance and may even hinder it.

A more general issue with existing DI studies is the lack of cross-dataset validation, which involves evaluating a model trained on one dataset against a different dataset to assess its generalizability and robustness. While some studies use multiple datasets [11]–[13], they typically apply their methods separately to each, training and testing within the same dataset (e.g., training on a portion of Dataset A and testing on the remainder). This approach raises potential concerns about their performance in unseen environments. It also exacerbates the specific issues raised above surrounding the use of statistical features, since models using these features would — in effect — be tested within the same network environment as the one they were trained in. In this study, we demonstrate the importance of cross-dataset validation in developing generalizable models.

III. Materials And Methods

A. Data Selection

DI research typically relies on datasets comprising data collected from real devices. These datasets commonly feature over 20 devices and necessitate prolonged observation periods to gather sufficient data. However, the data collection process for IoT devices entails substantial resource allocation, including personnel and space, making it burdensome. Consequently, many researchers opt to utilize publicly available DI datasets, thereby enhancing reproducibility and transparency and facilitating comparative analysis.

In our study, we use data from two dataset families, UNSW and MonIoTr. Each family consists of multiple datasets that are useful for our model building and evaluation. We use the UNSW datasets for feature and model selection, and the MonIoTr datasets to test the robustness and generalizability of the resulting models.

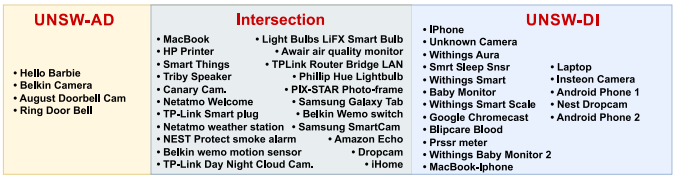


Fig. 2. Devices represented in the UNSW-DI (blue) and UNSW-AD (yellow) datasets, along with their intersection (grey).

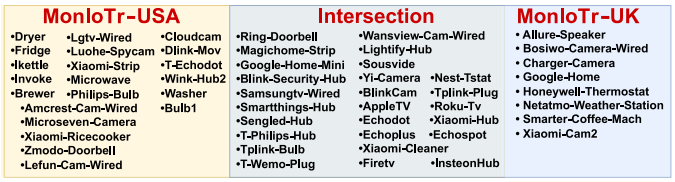


Fig. 3. Devices represented in the MonIoTr dataset from both UK (blue) and USA (yellow) sites, along with their intersection (grey). Adapted from [28].

1) UNSW: For feature and model selection, we utilized the IoT Attack Traces—ACM SOSR 2019 (UNSW-AD) [27] and IoT Traffic Traces—IEEE TMC 2018 (UNSW-DI) [17] datasets. The UNSW-DI dataset is tailored for DI tasks, comprising data from 24 benign devices collected over a 60-day period in 2016. In contrast, the UNSW-AD dataset, originally designed for anomaly detection (AD), contains 27 days of benign data and 17 days of both benign and malicious data from 28 devices collected over a 44-day period in 2018. For the UNSW-AD dataset, we used only the benign data.

Notably, despite both datasets containing most devices in common (see Fig. 2), they were collected at different times, in distinct site environments with varying network characteristics, and for disparate purposes, likely by different users. We leverage these datasets for feature and model selection, employing UNSW-DI as training data and UNSW-AD as test data, and vice versa in different runs/iterations, to enhance model robustness and generalizability.

Each dataset contains a number of sessions with one session per day (UNSW-DI: 60 sessions, UNSW-AD: 27 sessions). The resultant datasets are very large, and some sessions lack certain devices. To address this we selected and merged some sessions to maximize the number of devices available, and then used these merged sessions. Specifically, in the DI dataset, sessions 16-10-03 and 16-11-22 were combined to form S1, while sessions 16-09-29 and 16-11-18 were combined to form S2 (names are in yy-mm-dd date format). In the AD dataset, sessions 18-10-13 and 18-06-14 were combined to form S1, and sessions 18-10-16 and 18-06-11 were combined to form S2.

2) MonIoTr: After selecting the features and models, to remove any bias, we evaluate their effectiveness on a different dataset, IoT Information Exposure—IMC'19 (MonIoTr) [28]. The MonIoTr dataset consists of 81 devices. An important characteristic of this dataset is that data collection took place at two separate sites, one in the UK (MonIoTr-UK) and the other in the USA (MonIoTr-USA). Of the 81 devices, 26 devices were present in both sites, with the USA site having 47 devices (including 21 unique devices) and the UK site

having 34 devices (including 8 unique devices). Fig. 3 shows the distribution of devices by country.

Data collection in these sites spanned 112 hours, covering various scenarios:

Idle Data collected during an 8 hour period at night when devices were inactive.

Power Data collected for 2 minutes immediately after device power-on, without interaction.

Interaction Data collected through various interactions, including physical button presses, voice commands, phone app usage on the same network, communication with the device through a phone on a different network utilizing cloud infrastructure, and interaction with the device via Alexa Echo Spot.

Additionally, each device was connected via a VPN to the other site, enabling data collection across both sites for every condition. The same conditions were repeated using VPN tools to establish connections between the sites. For example, idle data was collected by connecting to the UK site from the USA via VPN and vice versa.

Many devices interact heavily with the local network, applications, tools, and cloud services during startup, so we merged power and interaction data into one *active* class. In a previous study [29], we observed that when both active and idle data are available, active data is much more effective for DI. Consequently, we excluded idle data from this study.

B. Feature Extraction

For feature extraction, we used Tshark, a network protocol analyzer, integrated with Python. We focused on features contained in the protocol headers of DNS, HTTP, ICMP, STUN, TCP, UDP, DHCP, EAPOL, IGMP, IP, NTP, and TLS. We eliminated features that consisted of string expressions or contained MAC and IP addresses. After this preliminary elimination, over 300 features remained from 4600 features.

IV. Feature Selection

During the feature selection phase, we aimed to identify the most effective features for DI. For this, we used the four UNSW *data partitions* described in Section III-A1 to guide feature selection. These comprise two sessions (S1 and S2) from each of UNSW-AD and UNSW-DI (referred to hereafter according to their dataset-session number: *AD-S1*, *AD-S2*, *DI-S1*, and *DI-S2*).

The generalizability of features is assessed in the following three ways, in order of increasing strictness:

5-fold cross-validation (CV) within a partition, which is common in the literature, e.g., within *AD-S1*.

Session versus session (SS) where generalizability is measured between two partitions of the same dataset, e.g., *AD-S1* vs *AD-S2*.

Dataset versus dataset (DD) where generalizability is measured between partitions of different datasets, e.g., *AD-S1* vs *DI-S2*.

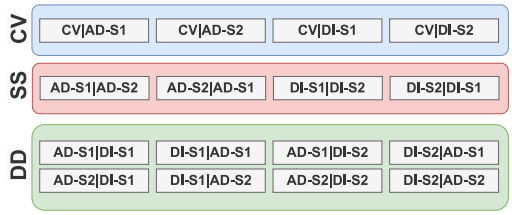


Fig. 4. Data usage from UNSW datasets across different evaluation contexts. CV, SS, and DD steps are visualized in shades of blue, red, and green, respectively, throughout the paper for clarity. In the nomenclature, *CV|AD-S1* denotes cross-validation on the AD dataset session 1. For cases other than CV, the first dataset is used for training and the second for testing. For example, *AD-S1|DI-S2* means the first session of the UNSW-AD dataset is used for training and the second session of the UNSW-DI dataset is used for testing.

A. Predictive Features

First, each feature is evaluated individually using each relevant combination of partitions, leading to 16 distinct evaluation contexts (see Fig. 4). Generalizability is assessed by training a decision tree (DT) model using the feature, with DT selected for speed and explainability. The utility of each feature is measured using the *kappa* metric, which adjusts for chance agreement.

Fig. 5 shows the resulting utility values. Notably, the scores measured under CV are much higher for some features than those measured under DD and SS scores. This indicates that information leaks may be occurring due to train and test folds being taken from the same partition, and suggests that CV is not a reliable basis for selecting features. Hence we only use DD and SS from now on.

A voting system was used for feature selection, with each non-zero kappa value (with a tolerance of $\pm 5\%$, so ≥ 0.05), indicating the feature had a positive contribution, resulting in a vote. Fig. 6 illustrates this system. Features receiving four or more votes from either DD (green) or SS (red) categories, with at least one DD vote, advanced to the next stage. Features not meeting this criterion were eliminated. The requirement for at least one DD vote is based on its greater strictness in assessing generalizability. As a result of voting, we identified 46 of the 332 features as being individually predictive.

B. Feature Interactions

We then considered feature interactions, using a wrapper method, specifically a Genetic Algorithm (GA), to find the optimal combination of individually predictive features which we validated across multiple datasets. The global search characteristic of a GA tends to find good solutions, and avoiding exhaustive search enables us to obtain potential feature sets within an acceptable timeframe. However, in general there is no guarantee of a GA finding an optimal subset, meaning it may include uninformative features in the feature set. Additionally, GA-selected features may develop a bias towards the data used for selection, potentially reducing their performance on other datasets.

To address this, all 46 features that passed individual voting were combined into a single feature set for selection using a genetic algorithm (GA). A decision tree (DT) model was employed for evaluation, with the fitness of each generation

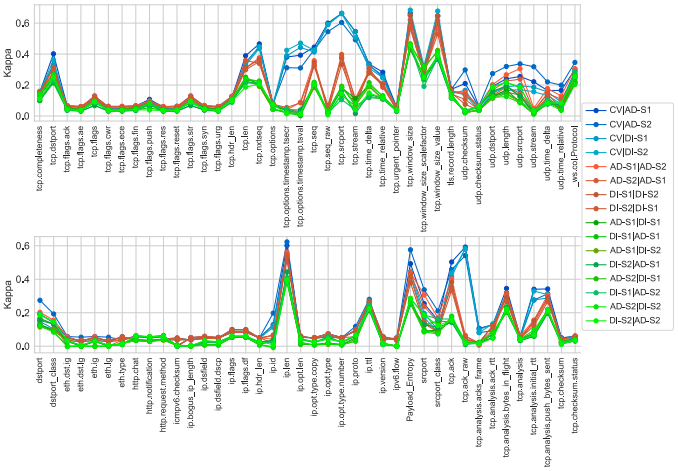


Fig. 5. Comparison of feature utility in UNSW datasets measured using CV (blue) and isolated methods, SS (red) and DD (green). CV tends to overestimate feature utility, with higher scores for many attributes. SS and DD produce more realistic evaluations. The discrepancy highlights the potential for information leakage in cross-validation and the importance of using isolated validation methods for assessing feature utility in ML-based DI models.

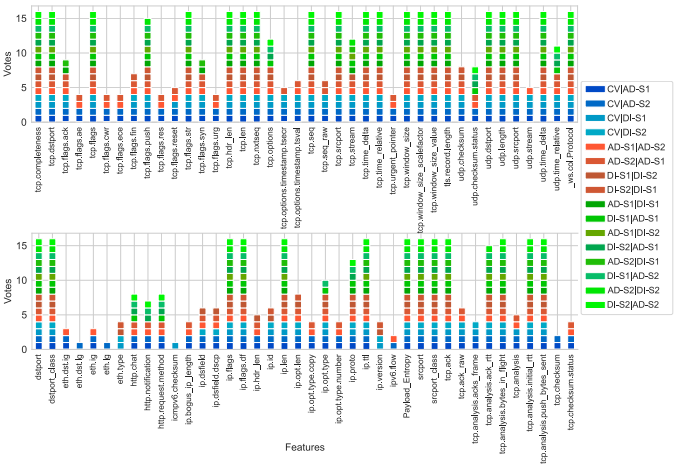


Fig. 6. Features identified as being predictive within one or more evaluation contexts (CV: blue, SS: red, DD: green), based on non-zero kappa scores of DT models. Note that the CV votes are not used to select features, but are shown here for completeness.

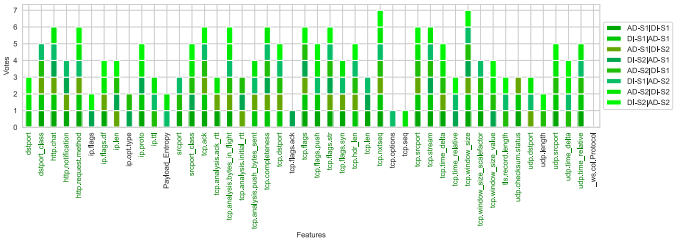


Fig. 7. List of intersecting features identified across eight dataset versus dataset (DD) cases using Genetic Algorithm (GA). Feature names highlighted in dark green were selected for model building after further post hoc analysis.

assessed using the F1 score across 8 additional DD datasets (see Fig. 4). This external validation provides feedback to the GA, ensuring that feature selection is based on success across different datasets and not dependent on only one. For each of the eight DD cases, the GA produced slightly different feature sets, so we then identified the intersections between them. Fig. 7 depicts this, showing that some features (e.g., tcp.window_size, tcp.ack) were chosen multiple times, some (e.g., tcp.seq, tcp.flags.ack) only once.

C. Deriving a Final Feature Set

To identify the most effective combination of features from the GA results, we conducted two post hoc analyses. First, the feature set derived from each DD case (each GA run) is reevaluated on each other DD case. F1 scores of the resulting DT models are presented in the left part of Fig. 8. These show how well the feature sets from each GA run generalize.

The second analysis uses a voting mechanism based on intersections of the eight GA feature sets (Fig. 7), with a feature receiving votes proportional to how often it appears. Features are then grouped based on thresholds, i.e., whether they appear in 2 or more feature sets (Vote+2), 3 or more (Vote+3) etc., and DT models are trained using these feature groups. The results are shown on the right side of Fig 8.

From the mean F1 scores shown in the final row of Fig. 8, it can be seen that feature sets based on grouping generally lead to better performance, at least for voting thresholds up to 4. For larger thresholds, the feature sets become very small, explaining the lower scores. Generally, the feature sets from single runs lead to models with lower F1 scores, suggesting that voting adds more robustness. Consequently, we use the Vote+3 feature set — which has the highest F1 score — in the next section. The selected features are highlighted in Fig. 7.

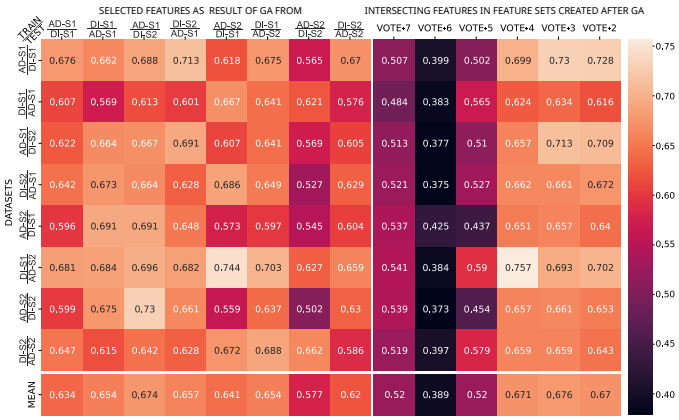


Fig. 8. Comparison of feature set performance across dataset versus dataset (DD) cases. The left side of the heatmap displays the results of applying feature sets obtained from the first step of the Genetic Algorithm (GA) to all DD data, yielding 64 results. On the right, the performance of the features grouped according to their frequency, focusing on the intersection of features obtained from the output of the GA algorithm.

V. Model Evaluation

Next, we construct and evaluate ML models using the selected features and compare their performance against established baselines using an independent dataset.

A. Model Selection

We experimented with various ML modeling approaches commonly used in DI to identify the most effective one

for this purpose, in terms of both predictive success and inference time — the latter being an important consideration when scanning network packets. The models we considered were Logistic Regression (LR), Decision Trees (DT), Naive Bayes (NB), Support Vector Machines (SVM), Random Forest (RF), Extreme Gradient Boosting (XGB) Multi-Layer Perceptron (MLP), K-Nearest Neighbors (KNN), Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), and Bidirectional Encoder Representations from Transformers (BERT).

We used random search for hyperparameter optimization and applied each model to the DD datasets — see Supplementary Material (SM) Section 4.4 for details. The F1 scores and average inference times are shown in Fig. 9. From this, it is clear that the most successful models are RF and XGB, with F1 scores of 0.799 and 0.780, respectively. While their predictive performance is similar, RF runs about 6 times faster than XGB, so we chose this as our preferred model. Notably, DT has the fastest inference time, but its predictive performance lags behind RF by about 10 percentage points.

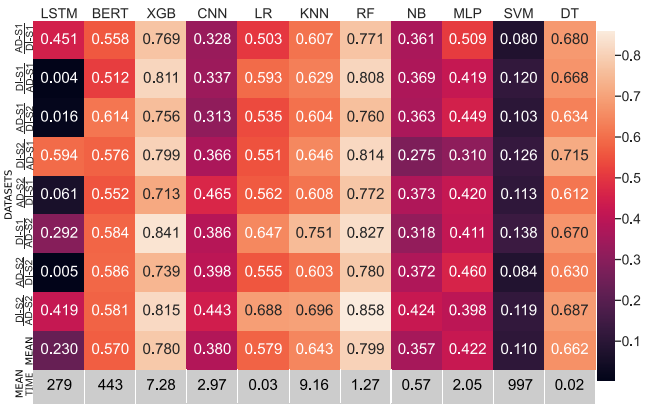


Fig. 9. F1 scores and average inference times of various ML algorithms applied to DD datasets. RF and XGB demonstrate the highest F1 scores of 0.799 and 0.780, respectively, while DT show the fastest inference time.

B. Evaluating Model Generalizability

In this section, we measure the generalizability of our method (GeMID) and other methods using the MonIoTr dataset, using the US, US-VPN, UK, and UK-VPN partitions of the MonIoTr dataset within three different evaluation contexts. In CV, we carry out cross-validation within a single partition. In SS, we train models using non-VPN data and test them using VPN data, and vice versa. In DD, we use data collected from one geographical site to train models, and data collected from the other geographical site to test them.

As comparison baselines, we use flow-based features generated by CICFlowmeter [30] and window-based features generated by Kitsune [31]. These are well-known tools that were used in previous DI studies. To ensure a fair comparison, feature and model selection were repeated independently (see SM Section I and II for results). We also compare against IoTDevID [13], another packet header-based DI method that aimed to build generalizable DI models and demonstrated improved performance over alternative methods.

TABLE I
COMPARISON OF DI METHODS ON MONIOTR DATASET. F1 SCORES.

	Dataset	GeMID	CICFlwM	IoTDevID	Kitsune
CV	UK	0.960	0.857	0.900	0.931
	UKVPN	0.965	0.853	0.908	0.924
	US	0.965	0.904	0.910	0.903
	USVPN	0.954	0.904	0.923	0.944
	Mean	**0.961**	0.879	0.910	0.925
SS	UK|UKVPN	0.895	0.644	0.824	0.741
	UKVPN|UK	0.877	0.555	0.802	0.710
	US|USVPN	0.861	0.606	0.821	0.712
	USVPN|US	0.894	0.616	0.839	0.687
	Mean	**0.882**	0.605	0.822	0.712
DD	UK|US	0.760	0.475	0.703	0.464
	UK|USVPN	0.769	0.490	0.666	0.435
	UKVPN|US	0.748	0.516	0.654	0.392
	UKVPN|USVPN	0.746	0.454	0.709	0.444
	US|UK	0.778	0.454	0.752	0.541
	US|UKVPN	0.780	0.517	0.694	0.455
	USVPN|UK	0.811	0.515	0.710	0.439
	USVPN|UKVPN	0.816	0.464	0.731	0.521
	Mean	**0.776**	0.486	0.702	0.461

The results are shown in Table I. All methods achieved high scores for CV, close to or above 0.90. However, a decrease is observed in all methods for SS and DD, with this decrease being particularly dramatic for the statistics-based methods (CICFlowMeter and Kitsune). For DD, GeMID maintains 81% of its CV F1 score and IoTDevID 77%, compared to only 55% for CIC and 50% for Kitsune. This significant drop supports our claim that flow or window-based statistical methods, which focus on relationships between devices within the network rather than individual device activities, are not suitable for building generalizable DI models due to their poor transferability across heterogeneous network environments.

The two methods — GeMID and IoTDevID — based on packet headers both perform significantly better, with GeMID outperforming IoTDevID within all evaluation contexts. Table II shows the per-device results for GeMID within the DD evaluation context. It can be seen that most of the devices are identified with a high level of accuracy. A notable exception is the *echo* family of devices, which, presumably due to underlying similarities, are easily misclassified with each other (see SM Figs. 12 and 13 for confusion matrices). There is also a degree of misclassification between devices from different manufacturers that perform the same function, e.g., between *roku-tv* and *samsungtv-wired*, and between *yi-camera* and *wansview-cam-wired*. A third category of poor performance is due to data paucity, e.g., for *sousvide*, there are only 70 US samples, compared to 15,000 for UK. The challenges in classifying these devices are not specific to GeMID but are also observed in other methods (IoTDevID, CICFlowMeter, and Kitsune). These methods also encounter similar difficulties due to the inherent complexity of distinguishing between devices with similar characteristics.

C. Feature Usage

GeMID and IoTDevID demonstrate better generalization in model performance compared to those that use statistical features. Of these two, GeMID is approximately 8 percentage

TABLE II
F1 SCORES PER DEVICE IN ALL 8 DD CASES FOR MONIOTR DATA, WITH GREEN FOR HIGHER AND RED FOR LOWER SUCCESS.

Devices	Train→ / Test→	UK US	UK USVPN	UKVPN US	UKVPN USVPN	US UK	US UKVPN	USVPN UK	USVPN UKVPN
appletv		0.868	0.879	0.851	0.830	0.658	0.858	0.614	0.802
blink-camera		0.963	0.983	0.952	0.985	0.951	0.929	0.945	0.932
blink-security-hub		0.909	0.889	0.498	0.775	0.259	0.213	0.649	0.599
echodot		0.161	0.093	0.106	0.053	0.521	0.083	0.575	0.313
echoplus		0.263	0.387	0.430	0.242	0.402	0.590	0.479	0.413
echospot		0.629	0.542	0.633	0.532	0.704	0.798	0.760	0.739
firetv		0.629	0.695	0.646	0.694	0.638	0.809	0.651	0.805
google-home-mini		0.806	0.885	0.761	0.873	0.873	0.895	0.909	0.903
insteon-hub		0.909	0.842	0.648	0.537	0.806	0.712	0.958	0.962
lightify-hub		0.951	0.941	0.946	0.936	0.839	0.924	0.969	0.979
magichome-strip		0.837	0.857	0.905	0.905	0.977	0.945	0.978	0.952
nest-tstat		0.879	0.803	0.870	0.866	0.959	0.945	0.957	0.956
ring-doorbell		0.997	0.998	0.997	0.999	0.996	0.991	0.995	0.991
roku-tv		0.337	0.490	0.482	0.507	0.692	0.693	0.448	0.563
samsungtv-wired		0.834	0.848	0.941	0.908	0.808	0.820	0.659	0.645
sengled-hub		0.670	0.758	0.954	0.922	0.525	0.455	0.533	0.552
smartthings-hub		0.853	0.880	0.910	0.907	0.932	0.904	0.934	0.911
sousvide		0.422	0.394	0.128	0.092	0.992	0.982	0.997	0.994
t-philips-hub		0.987	0.988	0.987	0.987	0.956	0.826	0.977	0.971
t-wemo-plug		0.935	0.844	0.938	0.841	0.957	0.979	0.957	0.982
tplink-bulb		0.941	0.977	0.852	0.946	0.904	0.857	0.994	0.987
tplink-plug		0.783	0.905	0.836	0.929	0.832	0.855	0.808	0.886
wansview-cam-wired		0.875	0.822	0.876	0.841	0.914	0.917	0.908	0.913
xiaomi-hub		0.825	0.879	0.830	0.870	0.753	0.924	0.986	0.989
yi-camera		0.729	0.654	0.719	0.673	0.596	0.589	0.646	0.657
Mean accuracy		0.857	0.825	0.861	0.830	0.893	0.888	0.898	0.889
Mean F1 score		0.760	0.769	0.748	0.746	0.778	0.780	0.811	0.816

points higher in accuracy. Given that the model building stages are similar, this is presumably due in part to a more rigorous feature selection process, using two data sets rather than just one to ensure that the selected features are generalizable.

Notably, IoTDevID's feature set is rich in features related to BOOTP, DNS, and EAPOL protocols, whereas GeMID lacks these. This may be due to a dataset-specific bias, owing to the reliance on a single dataset — Aalto — for IoTDevID's feature selection process. This is supported by the observation that while GeMID outperforms IoTDevID in experiments using the UNSW and MonIoTr datasets, IoTDevID shows markedly better performance when applied to the Aalto dataset [13] (see SM Table VIII).

However, it is worth noting that both share the features dstport_class, ip.flags.df, ip.len, ip.ttl, and tcp.window_size, suggesting that these are particularly important for DI.

Another reason for the difference in performance may be the larger initial pool of extracted features in GeMID (332 features) compared to IoTDevID (112 features). Although both studies generally use the same protocols, GeMID captures more sub-features within each protocol, providing finer-grained information in the feature set. In terms of selected features, GeMID offers more comprehensive protocol coverage, starting with a richer set of HTTP features and extending to broader coverage of TCP and UDP protocols. This includes features such as sequence numbers, timing metrics, and window size information, which enhance its ability to capture more detailed network behavior (see Fig. 7). Unlike in the IoTDevID study, we retained features containing session-based identifiers. Whilst these can lead to over-optimistic metric scores [32], this is only the case when sessions are split between test and train sets, e.g., when using CV. When this is controlled for using stricter evaluation contexts (as in this study), the inclusion of these features can enhance model performance by capturing meaningful relationships between packets.

In practice, the accuracy of packet-level classification methods can be improved by aggregating predictions from individual packets. The IoTDevID study introduced an aggregation algorithm that enhances performance by combining multiple packet-level predictions. This approach is particularly effective for packet-level methods such as GeMID, IoTDevID, and Kitsune (despite Kitsune focusing on the relationship between the window system and packets, as it still relies on packet-based labels). In contrast, CICFlowMeter operates at the flow level, where such aggregation is not applicable. Our results in SM Tables VI and VII show the impact of this approach, with GeMID achieving a mean F1 score of 0.892 in the DD evaluation context.

VI. LIMITATIONS

The primary limitation of this work is the availability and quality of public datasets. Although we used well-regarded datasets and cross-dataset validation, there will always be a limit in terms of the diversity, representativeness and sampling biases of any dataset.

Future work in DI would benefit from larger datasets designed specifically for this purpose, containing device samples that reflect realistic usage situations. Whilst we already consider devices situated in two different network environments, collecting data from the same devices operating in a broader range of networks would inevitably provide a broader perspective on generalizability. There is also a need for more data from non-IP protocols such as ZigBee and Z-Wave to test how well DI models of non-IP devices generalize.

Whilst this study (in common with previous studies) focuses on benign data, future DI studies would also benefit from using attack or malicious data, since this would be more representative of the actual environments within which DI models are deployed.

VII. CONCLUSIONS

Our work presents a novel approach to improving the generalizability of IoT DI models. Using a two-stage framework involving feature and algorithm selection followed by cross-dataset validation, we show that models trained on datasets in one environment can effectively identify devices in another environment. This addresses an important gap in existing methodologies, which often do not validate across different datasets.

Comparative analysis of DI methods showed that packet header-based features performed better than network statistics and window methods. Packet features offer a more consistent and reliable basis for model training because they are less influenced by network-specific variables. This finding highlights the importance of selecting features that inherently capture device-specific attributes.

Our results also highlight the limitations of relying only on single datasets for model training and validation. Such approaches run the risk of overfitting and may fail to generalize to different network environments. In contrast, our methodology provides robustness and adaptability, which are crucial for practical IoT security applications.

REFERENCES

[1] V. L. Sujay, "Number of internet of things (IoT) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030," 2023, accessed: 2023-09-08, https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[2] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in internet of things," *Journal of Network and Computer Applications*, vol. 84, pp. 25–37, 2017.

[3] M. Williams, J. R. Nurse, and S. Creese, "Privacy is the boring bit: user perceptions and behaviour in the internet-of-things," in *2017 15th Annual Conference on PST*. IEEE, 2017, pp. 181–18 109.

[4] H. Modi, "Dawn of the terrorbit era," Feb 2019, accessed: 2024-04-16, Available at https://www.netscout.com/blog/dawn-terrorbit-era.

[5] A. Marzano, D. Alexander, O. Fonseca, E. Fazzion, C. Hoepers, K. Steding-Jessen, M. H. Chaves, Í. Cunha, D. Guedes, and W. Meira, "The evolution of bashlite and mirai IoT botnets," in *ISCC, IEEE*, 2018.

[6] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *USENIX*, 2017.

[7] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, "Domain generalization: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, pp. 4396–4415, 2022.

[8] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, and Y. Elovici, "Detection of unauthorized IoT devices using machine learning techniques," *arXiv preprint: 1709.04647*, 2017.

[9] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, "Behavioral fingerprinting of IoT devices," in *Proc. of the 2018 Workshop on Attacks and Solutions in Hardware Security*, 2018, pp. 41–50.

[10] A. Aksoy and M. H. Gunes, "Automated IoT device identification using network traffic," in *IEEE Int. Conf. on Comm. (ICC)*, 2019, pp. 1–7.

[11] R. R. Chowdhury, S. Aneja, N. Aneja, and E. Abas, "Network traffic analysis based IoT device identification," in *Proc. of the 2020 4th Int. Conf.e on Big Data and Internet of Things*, 2020, pp. 79–89.

[12] J. Ortiz, C. Crawford, and F. Le, "DeviceMien: network device behavior modeling for identifying unknown IoT devices," in *Proc. of the Int. Conf. on Internet of Things Design and Implementation*, 2019, pp. 106–117.

[13] K. Kostas, M. Just, and M. A. Lones, "IoTDevID: A Behavior-Based Device Identification Method for the IoT," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 23 741–23 749, 2022.

[14] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "IoT sentinel: Automated device-type identification for security enforcement in iot," in *37th ICDCS*. IEEE, 2017.

[15] S. Marchal, "IoT devices captures, aalto university," 2017, accessed: 2023-06-09, https://research.aalto.fi/en/datasets/iot-devices-capture.

[16] H. Kawai, S. Ata, N. Nakamura, and I. Oka, "Identification of communication devices from analysis of traffic patterns," in *2017 13th CNSM*. IEEE, 2017, pp. 1–5.

[17] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2018.

[18] K. Kostas, "Behaviour-based security with machine learning on IoT networks," Ph.D. dissertation, Heriot-Watt University, 2023.

[19] H. Jmila, G. Blanc, M. R. Shahid, and M. Lazrag, "A survey of smart home IoT device classification using machine learning-based network traffic analysis," *IEEE Access*, vol. 10, pp. 97 117–97 141, 2022.

[20] N. Msadek, R. Soua, and T. Engel, "IoT device fingerprinting: Machine learning based encrypted traffic analysis," in *2019 IEEE wireless communications and networking conf. (WCNC)*. IEEE, 2019, pp. 1–8.

[21] J. Kotak and Y. Elovici, "IoT device identification using deep learning," in *13th International Conference on Computational Intelligence in Security for Information Systems*. Springer, 2021, pp. 76–86.

[22] T. B. Hoang, L. Vu, and Q. U. Nguyen, "A data sampling and two-stage convolution neural network for IoT devices identification," in *2022 Int. Conf. on Comp. and Comm. Tech. (RIVF)*. IEEE, 2022, pp. 458–463.

[23] X. Hu, H. Li, Z. Shi, N. Yu, H. Zhu, and L. Sun, "A robust IoT device identification method with unknown traffic detection," in *Wireless Algorithms, Systems, and Applications: 16th Int. Conf., WASA 2021, Proceedings, Part I 16*. Springer, 2021, pp. 190–202.

[24] S. Liu, X. Xu, Y. Zhang, and Y. Wang, "Autonomous anti-interference identification of IoT device traffic based on convolutional neural network," in *2022 IJCNN*. IEEE, 2022, pp. 1–8.

[25] R. Yao, N. Wang, P. Chen, D. Ma, and X. Sheng, "A cnn-transformer hybrid approach for an intrusion detection system in advanced metering infrastructure," *Multimedia Tools and Applications*, pp. 1–24, 2022.

[26] J. Kotak and Y. Elovici, "IoT device identification based on network communication analysis using deep learning," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–17, 2022.

[27] A. Hamza, H. H. Gharakheili, T. A. Benson, and V. Sivaraman, "Detecting volumetric attacks on loT devices via SDN-based monitoring of MUD activity," in *Proc. ACM Symp. SDN Res.*, 2019, pp. 36–48.

[28] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure for consumer IoT devices: A multidimensional, network-informed measurement approach," in *Proc. of the Internet Measurement Conference (IMC)*, 2019.

[29] K. Kostas, M. Just, and M. A. Lones, "Externally validating the IoTDevID device identification methodology using the CIC IoT 2022 dataset," *arXiv preprint arXiv:2307.08679*, 2023.

[30] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, A. A. Ghorbani *et al.*, "Characterization of tor traffic using time based features." in *ICISSp*, 2017, pp. 253–262.

[31] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," in *Network and Distributed Systems Security (NDSS) Symposium*, 2018.

[32] K. Kostas, M. Just, and M. Lones, "Individual packet features are a risk to model generalisation in ml-based intrusion detection," *arXiv preprint arXiv:2406.07578*, 2024.

Kahraman Kostas is a researcher at the Ministry of National Education in Türkiye. He earned his PhD in Computer Science from Heriot-Watt University in 2023 and holds an MSc in Computer Networks and Security from the University of Essex, obtained in 2018. His research interests encompass cyber security, computer networks, IoT, indoor positioning, and machine learning. For more information, visit https://kahramankostas.github.io/.

Rabia Yasa Kostas is a lecturer at Gümüşhane University. She earned her PhD in Computer Science from the University of Edinburgh in 2023 and holds an MSc in Advanced Computer Science from the University of Essex, obtained in 2018. Her research interests include Computational Linguistics, Natural Language Processing, Semantic Analysis, and Cognitive Models in Neuroscience. For more information, visit her website: https://rykostas.github.io/.

Mike Just is an Associate Professor at Heriot-Watt University. He received both MCS and PhD degrees from Carleton University. He applies human-computer interaction and machine learning techniques to solve computer security problems. You can find more information at http://www.justmikejust.wordpress.com/.

Michael A. Lones (M'01—SM'10) is a Professor of Computer Science at Heriot-Watt University. He received both MEng and PhD degrees from the University of York. He carries out research in the areas of machine learning and optimization, where he has a particular interest in biologically-inspired approaches and improving ML practice. Application areas of his work include medicine, robotics and security. You can find more information at http://www.macs.hw.ac.uk/∼ml355.

GeMID: Generalizable Models for Device Identification in IoT Networks — Supplementary Material —

Section 1. CICFlowmeter Feature Selection Process and Results

In this section, we apply the same feature selection process to the CICFlowMeter features as we did with GeMID. Table S1 lists all features included in the CICFlowMeter method. Figure S1 compares feature utility in the UNSW datasets, using both cross-validation and isolated evaluation methods (SS and DD). The voting process for selecting the most suitable features, based on the utility of each feature, is illustrated in Figure S2. Candidate sub-features generated by applying a genetic algorithm to each dataset series are presented in Figure S3. The testing results for these features and their intersections across all UNSW-DD datasets are shown in Figure S4. The best-performing feature set, as determined by these tests, is detailed in Table S2. The results of testing the final features with various machine learning and neural network-based techniques to evaluate model performance are shown in Figure S5.

TABLE S1
FEATURES OBTAINED AFTER APPLYING THE CICFLOWMETER TOOL TO THE PCAP FILE.

CICFlowmeter Full Feature List		
Idle Min	total Length of Fwd Packet	Flow Bytes/s
Idle Max	total Length of Bwd Packet	Flow IAT Std
Idle Std	Packet Length Variance	Flow IAT Max
Idle Mean	Fwd Packet Length Min	Flow IAT Min
Active Min	Fwd Packet Length Max	Fwd IAT Mean
Active Max	Fwd Packet Length Mean	Bwd IAT Mean
Active Std	Bwd Packet Length Mean	Flow duration
Fwd IAT Min	Fwd Packet Length Std	Flow IAT Mean
Fwd IAT Max	Bwd Packet Length Min	Bwd IAT Total
Fwd IAT Std	Bwd Packet Length Max	Fwd PSH flags
Bwd IAT Min	Bwd Packet Length Std	Bwd PSH Flags
Bwd IAT Max	Fwd Segment Size Avg	Fwd URG Flags
Bwd IAT Std	Bwd Segment Size Avg	Bwd URG Flags
Active Mean	Average Packet Size	FWD Packets/s
Fwd Header Length	Bwd Packet/Bulk Avg	Bwd Packets/s
Bwd Header Length	Packet Length Mean	down/Up Ratio
Packet Length Max	Fwd Packet/Bulk Avg	Flow Packets/s
Packet Length Std	Subflow Fwd Packets	FIN Flag Count
Bwd Bulk Rate Avg	Subflow Bwd Packets	SYN Flag Count
Subflow Fwd Bytes	Packet Length Min	RST Flag Count
Subflow Bwd Bytes	Fwd Bytes/Bulk Avg	PSH Flag Count
Fwd Act Data Pkts	Fwd Bulk Rate Avg	ACK Flag Count
total Fwd Packet	Bwd Bytes/Bulk Avg	URG Flag Count
Fwd IAT Total	Fwd Init Win bytes	CWR Flag Count
Fwd Seg Size Min	Bwd Init Win bytes	ECE Flag Count
	total Bwd packets	

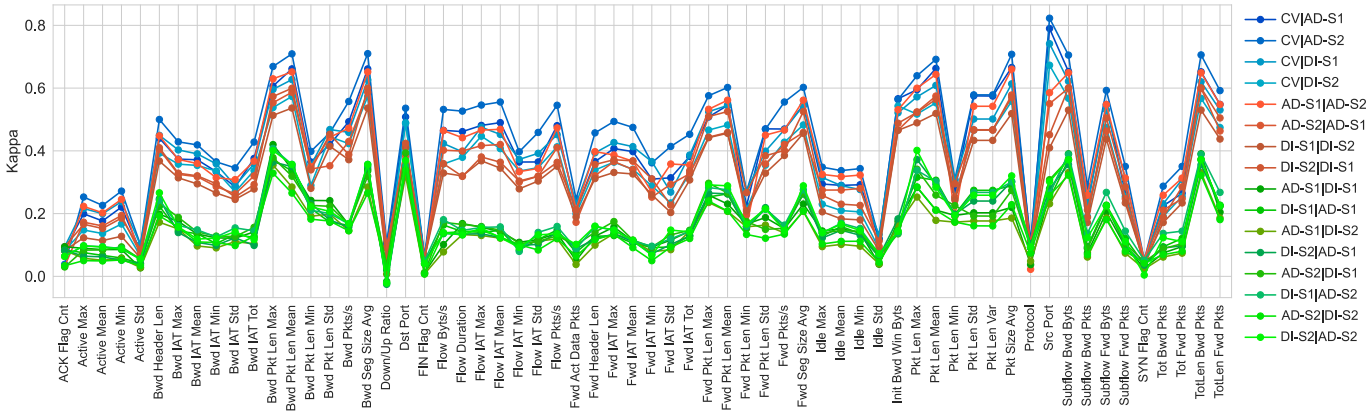


Fig. S1. Comparison of kappa scores for CICFlowmeter features in UNSW datasets using CV (blue) and isolated methods, SS (red) and DD (green). The CV method tends to overestimate feature utility, showing higher scores for many attributes, while SS and DD produce more realistic evaluations.

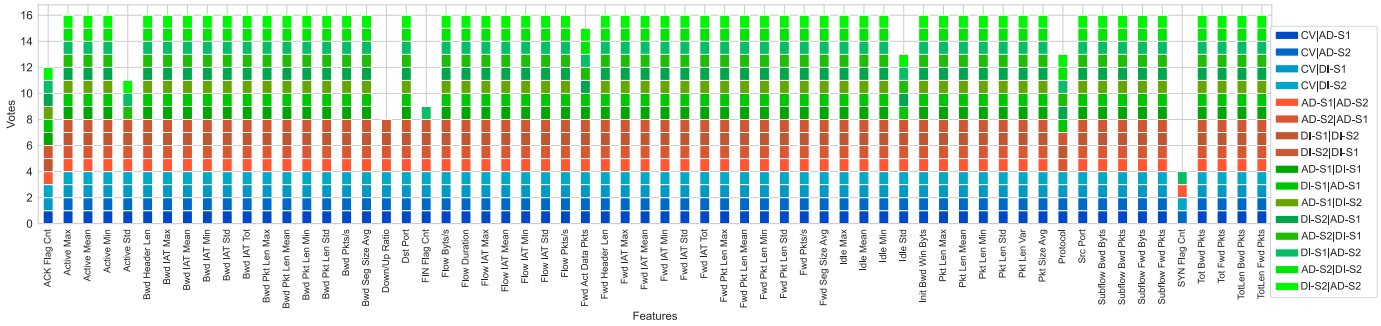


Fig. S2. A voting system for CICFlowmeter features that gives one vote to each feature with a kappa value different from 0 that is individually predictive within CV (blue), SS (red) & DD (green).

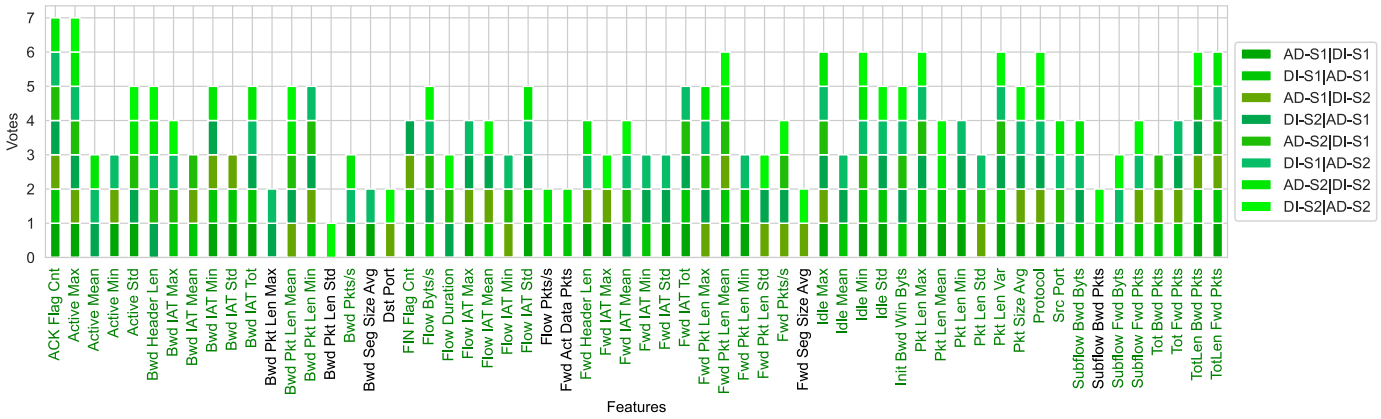


Fig. S3. List of intersecting CICFlowmeter features identified across eight dataset versus dataset (DD) cases using Genetic Algorithm (GA).

TABLE S2
FINAL LIST OF FEATURES OBTAINED AS A RESULT OF MULTI-STEP FEATURE SELECTION.

CICFlowmeter Final Feature List		
ACK Flag Cnt	Flow IAT Mean	Init Bwd Win Byts
Active Max	Flow IAT Min	Pkt Len Max
Active Mean	Flow IAT Std	Pkt Len Mean
Active Min	Fwd Header Len	Pkt Len Min
Active Std	Fwd IAT Max	Pkt Len Std
Bwd Header Len	Fwd IAT Mean	Pkt Len Var
Bwd IAT Max	Fwd IAT Min	Pkt Size Avg
Bwd IAT Mean	Fwd IAT Std	Protocol
Bwd IAT Min	Fwd IAT Tot	Src Port
Bwd IAT Std	Fwd Pkt Len Max	Subflow Bwd Byts
Bwd IAT Tot	Fwd Pkt Len Mean	Subflow Fwd Byts
Bwd Pkt Len Mean	Fwd Pkt Len Min	Subflow Fwd Pkts
Bwd Pkt Len Min	Fwd Pkt Len Std	Tot Bwd Pkts
Bwd Pkts/s	Fwd Pkts/s	Tot Fwd Pkts
FIN Flag Cnt	Idle Max	TotLen Bwd Pkts
Flow Byts/s	Idle Mean	TotLen Fwd Pkts
Flow Duration	Idle Min	
Flow IAT Max	Idle Std	

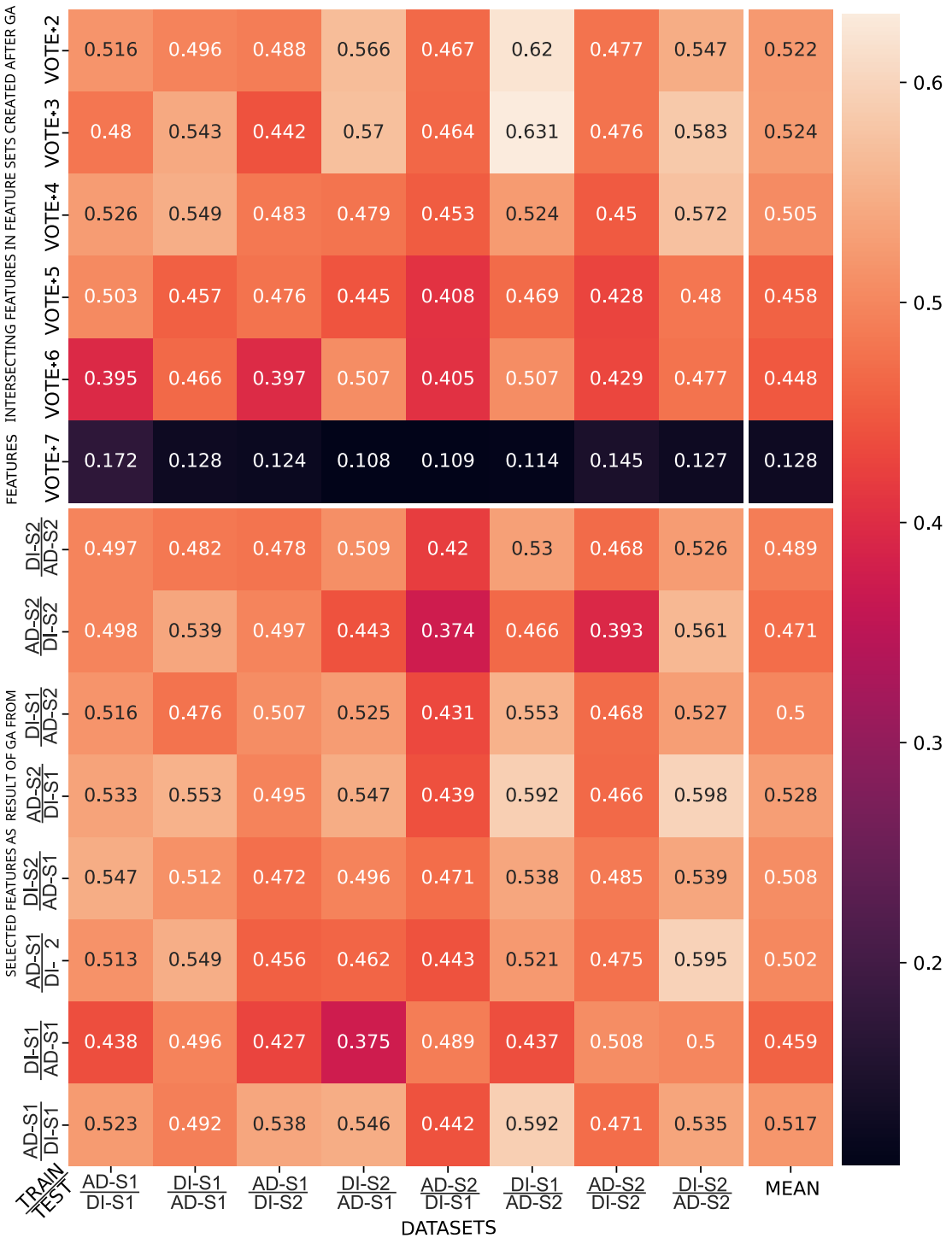


Fig. S4. Comparison of CICFlowmeter feature set performance across dataset versus dataset (DD) cases. The left side of the heatmap displays the results of applying feature sets obtained from the first step of the Genetic Algorithm (GA) to all DD data, yielding 64 results. On the right, the performance of the features grouped according to their frequency, focusing on the intersection of features obtained from the output of the GA algorithm.

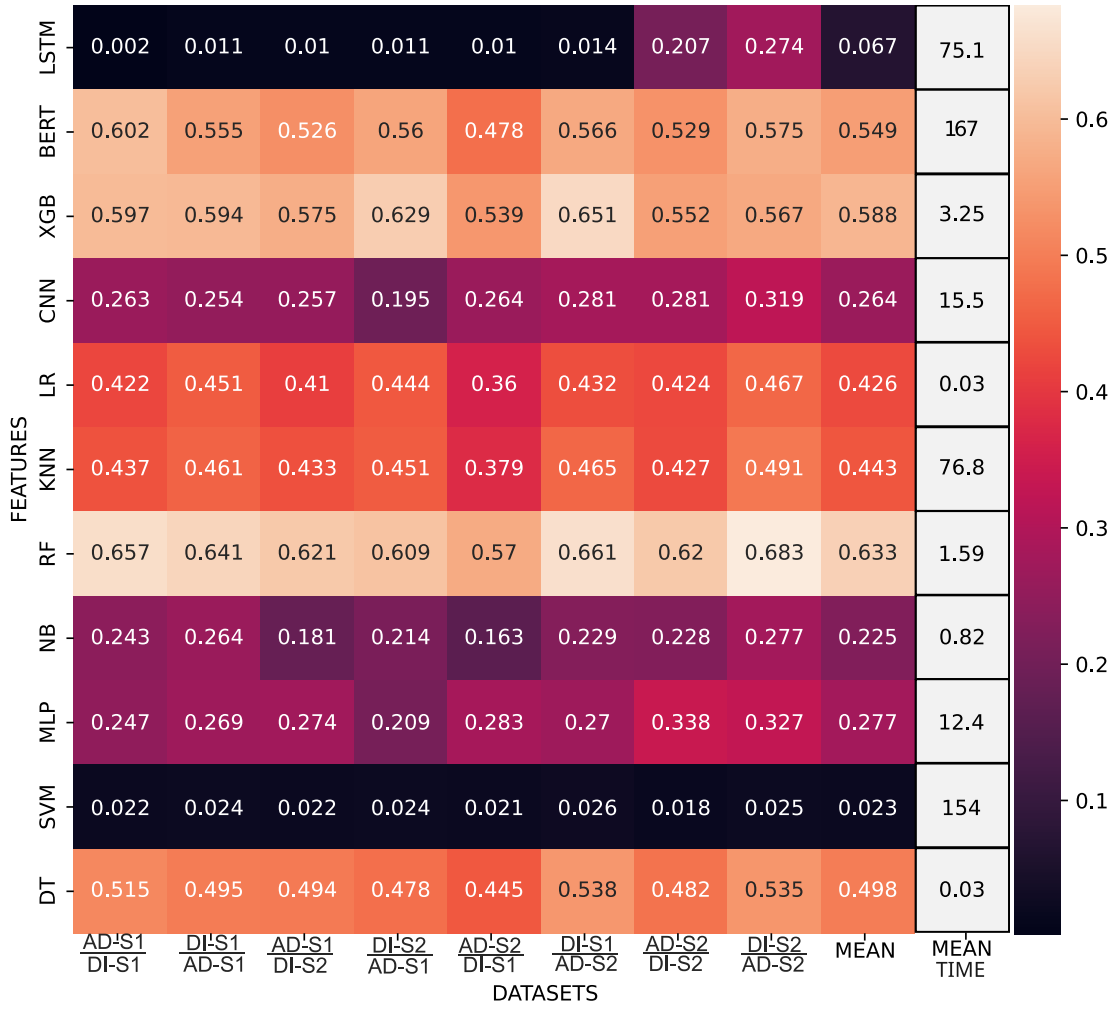


Fig. S5. F1 scores and average inference times of various ML algorithms applied to DD datasets with Kitsune features. The Random Forest (RF) and XGB methods demonstrate the highest F1 scores of 0.633 and 0.588, respectively, while Decision Trees (DT) show the fastest inference time.

SECTION 2. KITSUNE FEATURE SELECTION PROCESS AND RESULTS

In this section, we apply the same feature selection process to the Kitsune features as we did with GeMID. Table S3 lists all features included in the Kitsune method. Figure S6 compares feature utility in the UNSW datasets, using both cross-validation and isolated evaluation methods (SS and DD). The voting process for selecting the most suitable features, based on the utility of each feature, is illustrated in Figure S7. Candidate sub-features generated by applying a genetic algorithm to each dataset series are presented in Figure S8. The testing results for these features and their intersections across all UNSW-DD datasets are shown in Figure S9. The best-performing feature set, as determined by these tests, is detailed in Table S4. The results of testing the final features with various machine learning and neural network-based techniques to evaluate model performance are shown in Figure S10.

TABLE S3
FEATURES OBTAINED AFTER APPLYING THE KITSUNE TOOL TO THE PCAP FILE.

Features obtained after applying the Kitsune tool to the pcap file.		
HH_5_std_0	HpHp_0.01_pcc_0_1'	HpHp_0.01_mean_0
HH_3_std_0	HH_5_covariance_0_1	MI_dir_0.1_weight
HH_1_std_0	HH_3_covariance_0_1	HH_0.1_radius_0_1
HH_5_mean_0	HH_1_covariance_0_1	HH_jit_0.1_weight
HH_3_mean_0	HpHp_0.1_radius_0_1	HpHp_5_radius_0_1
HH_1_mean_0	HH_0.1_magnitude_0_1	HpHp_3_radius_0_1
MI_dir_5_std	HpHp_5_magnitude_0_1	HpHp_1_radius_0_1
MI_dir_3_std	HpHp_3_magnitude_0_1	HpHp_0.1_weight_0
MI_dir_1_std	HpHp_1_magnitude_0_1	MI_dir_0.01_weight
HH_5_pcc_0_1	HpHp_0.01_radius_0_1	HH_5_magnitude_0_1
HH_3_pcc_0_1	HH_0.1_covariance_0_1	HH_3_magnitude_0_1
HH_1_pcc_0_1	HH_0.01_magnitude_0_1	HH_1_magnitude_0_1
HH_0.1_std_0	HpHp_5_covariance_0_1	HH_0.01_radius_0_1
HH_jit_5_std	HpHp_3_covariance_0_1	HH_jit_0.01_weight
HH_jit_3_std	HpHp_1_covariance_0_1	HpHp_0.01_weight_0
HH_jit_1_std	HH_0.01_covariance_0_1	HH_0.01_pcc_0_1
HpHp_5_std_0	HpHp_0.1_magnitude_0_1	HH_jit_5_weight
HpHp_3_std_0	HpHp_0.1_covariance_0_1	HH_jit_3_weight
HpHp_1_std_0	HpHp_0.01_magnitude_0_1	HH_jit_1_weight
MI_dir_5_mean	HpHp_0.01_covariance_0_1	HH_jit_0.1_mean
MI_dir_3_mean	HpHp_0.01_std_0	HH_jit_0.01_std
MI_dir_1_mean	MI_dir_0.01_mean	HpHp_5_weight_0
HH_5_weight_0	HH_0.01_weight_0	HpHp_3_weight_0
HH_3_weight_0	HH_jit_0.01_mean	HpHp_1_weight_0
HH_1_weight_0	HpHp_0.1_pcc_0_1	HpHp_0.1_mean_0
HH_0.1_mean_0	MI_dir_0.1_std	MI_dir_5_weight
HH_0.01_std_0	HH_0.1_pcc_0_1	MI_dir_3_weight
HH_jit_5_mean	HH_0.01_mean_0	MI_dir_1_weight
HH_jit_3_mean	HH_jit_0.1_std	MI_dir_0.1_mean
HH_jit_1_mean	HpHp_5_pcc_0_1	MI_dir_0.01_std
HpHp_5_mean_0	HpHp_3_pcc_0_1	HH_5_radius_0_1
HpHp_3_mean_0	HpHp_1_pcc_0_1	HH_3_radius_0_1
HpHp_1_mean_0	HpHp_0.1_std_0	HH_1_radius_0_1
	HH_0.1_weight_0	

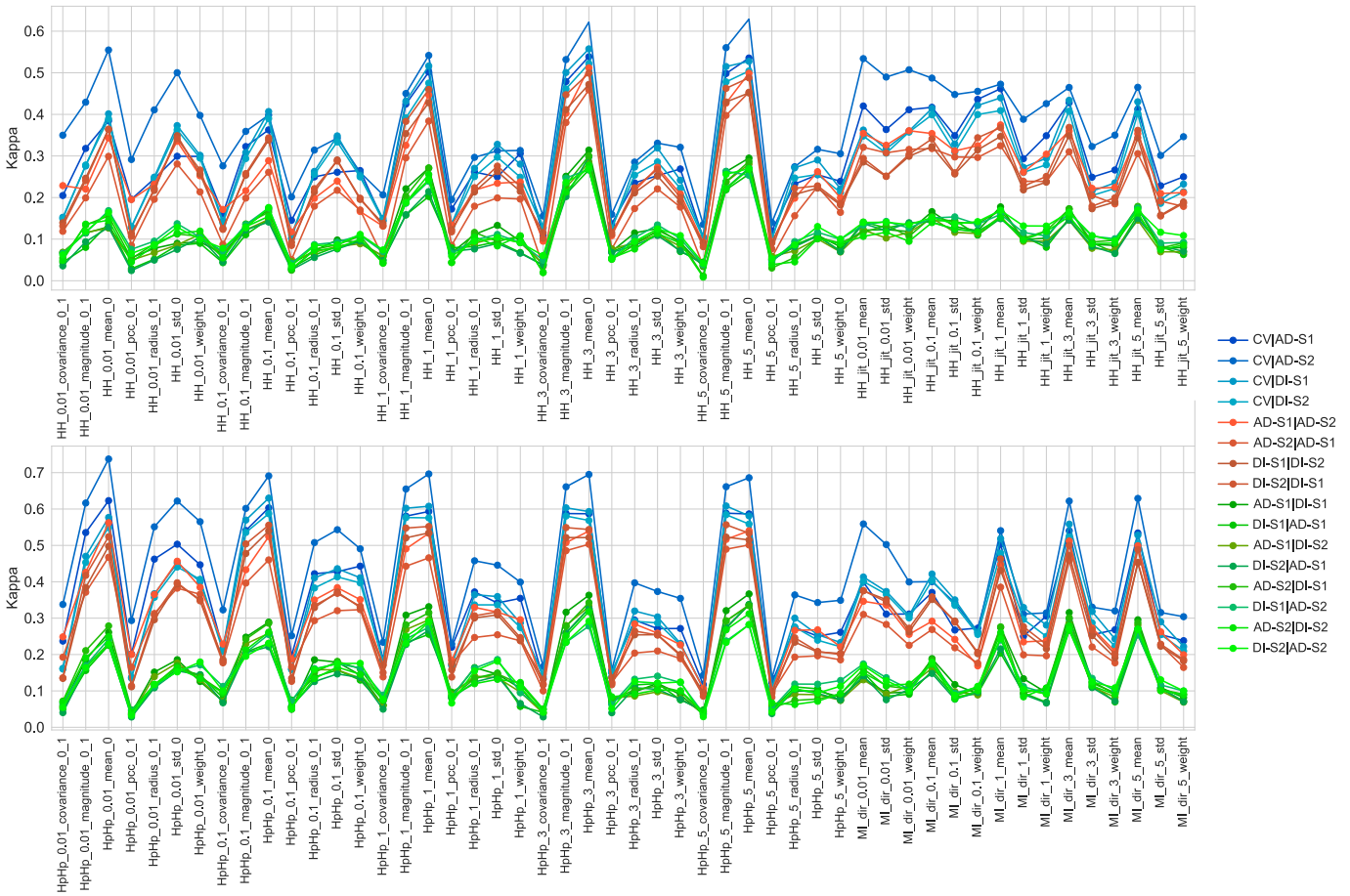


Fig. S6. Comparison of kappa scores for Kitsune features in UNSW datasets using CV (blue) and isolated methods, SS (red) and DD (green). The CV method tends to overestimate feature utility, showing higher scores for many attributes, while SS and DD produce more realistic evaluations. This discrepancy highlights the potential for information leakage in cross-validation and the importance of using isolated validation methods for assessing feature utility in ML-based DI models.

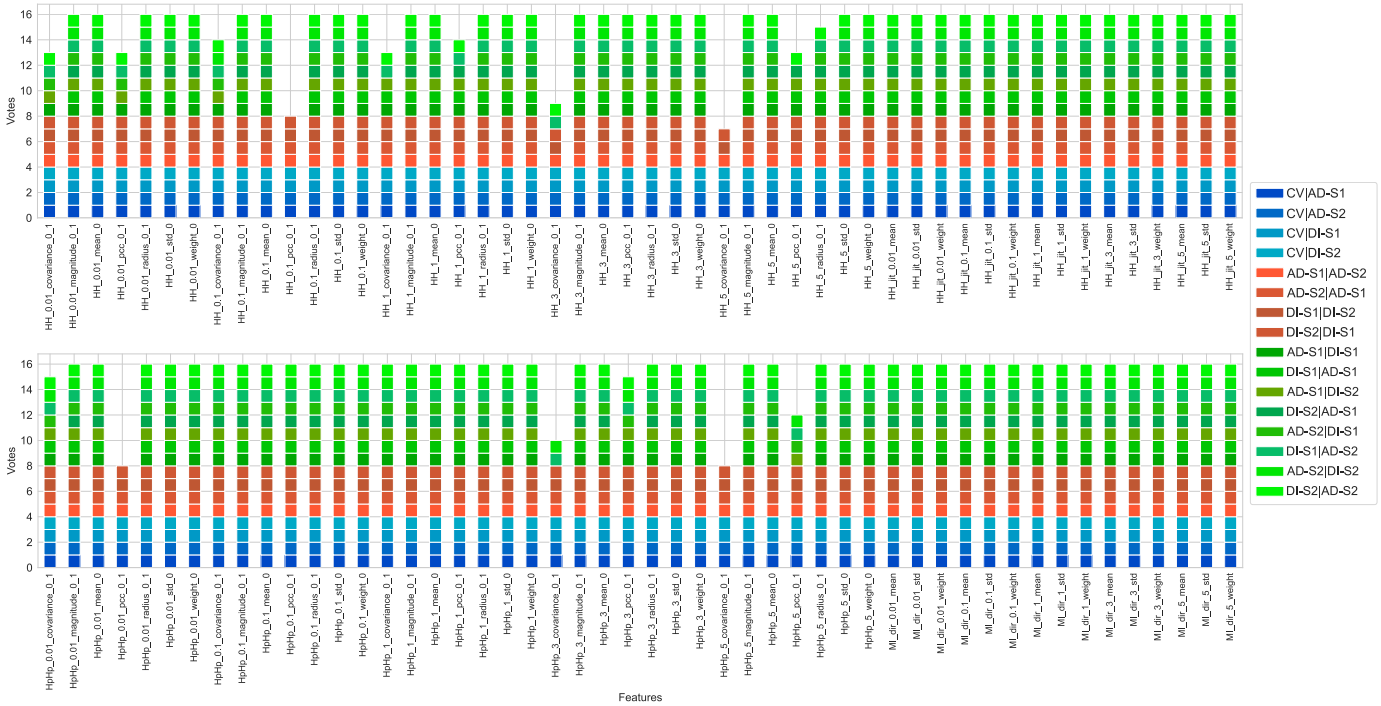


Fig. S7. A voting system for Kitsune features that gives one vote to each feature with a kappa value different from 0 that is individually predictive within CV (blue), SS (red) & DD (green).

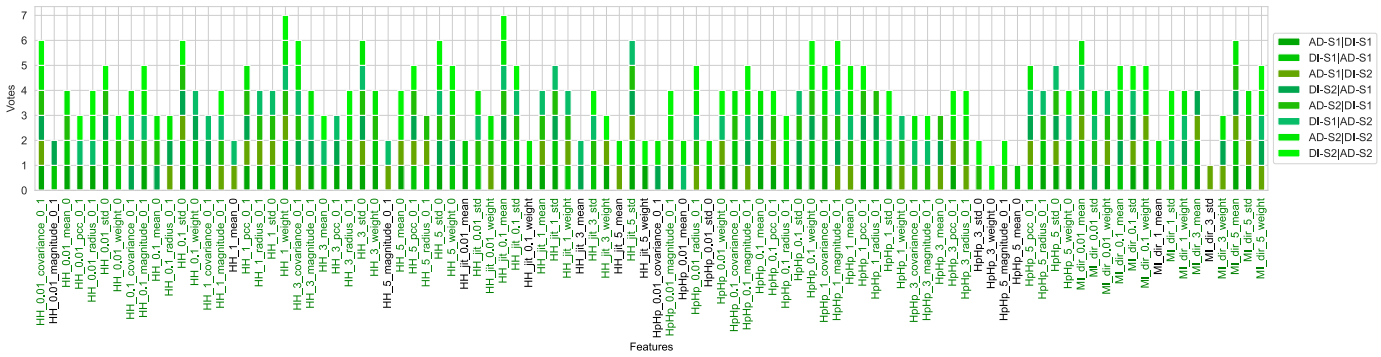


Fig. S8. List of intersecting Kitsune features identified across eight dataset versus dataset (DD) cases using Genetic Algorithm (GA).

TABLE S4
FINAL LIST OF FEATURES OBTAINED AS A RESULT OF MULTI-STEP FEATURE SELECTION.

Kitsune Final Feature List		
HH_0.01_covariance_0_1	HH_5_pcc_0_1	HpHp_1_pcc_0_1
HH_0.01_mean_0	HH_5_radius_0_1	HpHp_1_radius_0_1
HH_0.01_pcc_0_1	HH_5_std_0	HpHp_1_std_0
HH_0.01_radius_0_1	HH_5_weight_0	HpHp_1_weight_0
HH_0.01_std_0	HH_jit_0.01_std	HpHp_3_covariance_0_1
HH_0.01_weight_0	HH_jit_0.01_weight	HpHp_3_magnitude_0_1
HH_0.1_covariance_0_1	HH_jit_0.1_mean	HpHp_3_mean_0
HH_0.1_magnitude_0_1	HH_jit_0.1_std	HpHp_3_pcc_0_1
HH_0.1_mean_0	HH_jit_1_mean	HpHp_3_radius_0_1
HH_0.1_radius_0_1	HH_jit_1_std	HpHp_5_pcc_0_1
HH_0.1_std_0	HH_jit_1_weight	HpHp_5_radius_0_1
HH_0.1_weight_0	HH_jit_3_std	HpHp_5_std_0
HH_1_covariance_0_1	HH_jit_3_weight	HpHp_5_weight_0
HH_1_magnitude_0_1	HH_jit_5_std	MI_dir_0.01_mean
HH_1_pcc_0_1	HpHp_0.01_magnitude_0_1	MI_dir_0.01_std
HH_1_radius_0_1	HpHp_0.01_radius_0_1	MI_dir_0.01_weight
HH_1_std_0	HpHp_0.01_weight_0	MI_dir_0.1_mean
HH_1_weight_0	HpHp_0.1_covariance_0_1	MI_dir_0.1_std
HH_3_covariance_0_1	HpHp_0.1_magnitude_0_1	MI_dir_0.1_weight
HH_3_magnitude_0_1	HpHp_0.1_mean_0	MI_dir_1_std
HH_3_mean_0	HpHp_0.1_pcc_0_1	MI_dir_1_weight
HH_3_pcc_0_1	HpHp_0.1_radius_0_1	MI_dir_3_mean
HH_3_radius_0_1	HpHp_0.1_std_0	MI_dir_3_weight
HH_3_std_0	HpHp_0.1_weight_0	MI_dir_5_mean
HH_3_weight_0	HpHp_1_covariance_0_1	MI_dir_5_std
HH_5_mean_0	HpHp_1_magnitude_0_1	MI_dir_5_weight
	HpHp_1_mean_0	

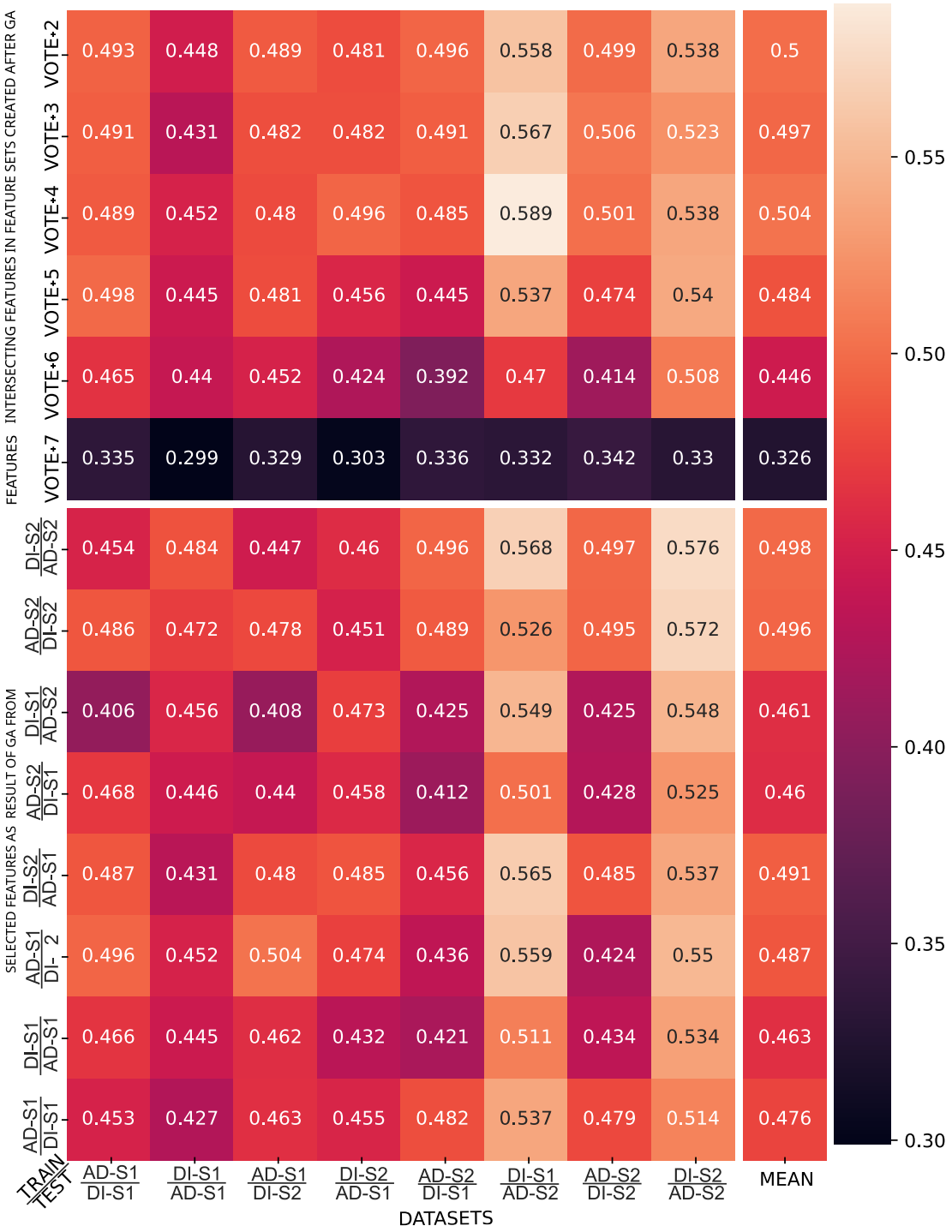


Fig. S9. Comparison of Kitsune feature set performance across dataset versus dataset (DD) cases. The left side of the heatmap displays the results of applying feature sets obtained from the first step of the Genetic Algorithm (GA) to all DD data, yielding 64 results. On the right, the performance of the features grouped according to their frequency, focusing on the intersection of features obtained from the output of the GA algorithm.

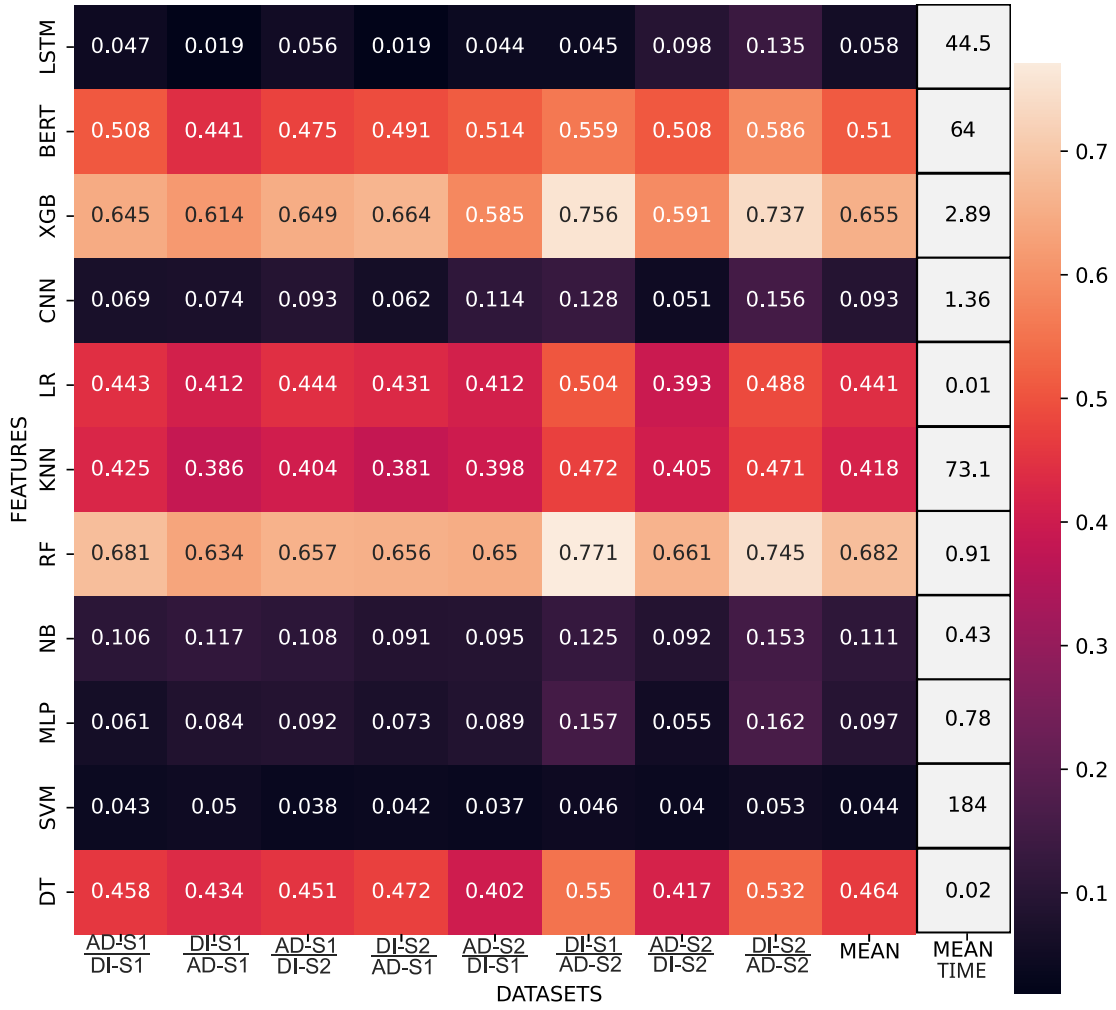


Fig. S10. F1 scores and average inference times of various ML algorithms applied to DD datasets with Kitsune features. The Random Forest (RF) and XGB methods demonstrate the highest F1 scores of 0.682 and 0.655, respectively, while Decision Trees (DT) show the fastest inference time.

SECTION 3. IOTDEVID RESULTS

Since the original IoTDevID study included its own feature selection process, we did not perform an additional feature selection in this analysis. Instead, we used the features selected in the IoTDevID study for model selection. Therefore, unlike other methods, this section does not include the feature selection steps. The scores obtained during model selection are presented in Figure S11, along with the models and datasets used.

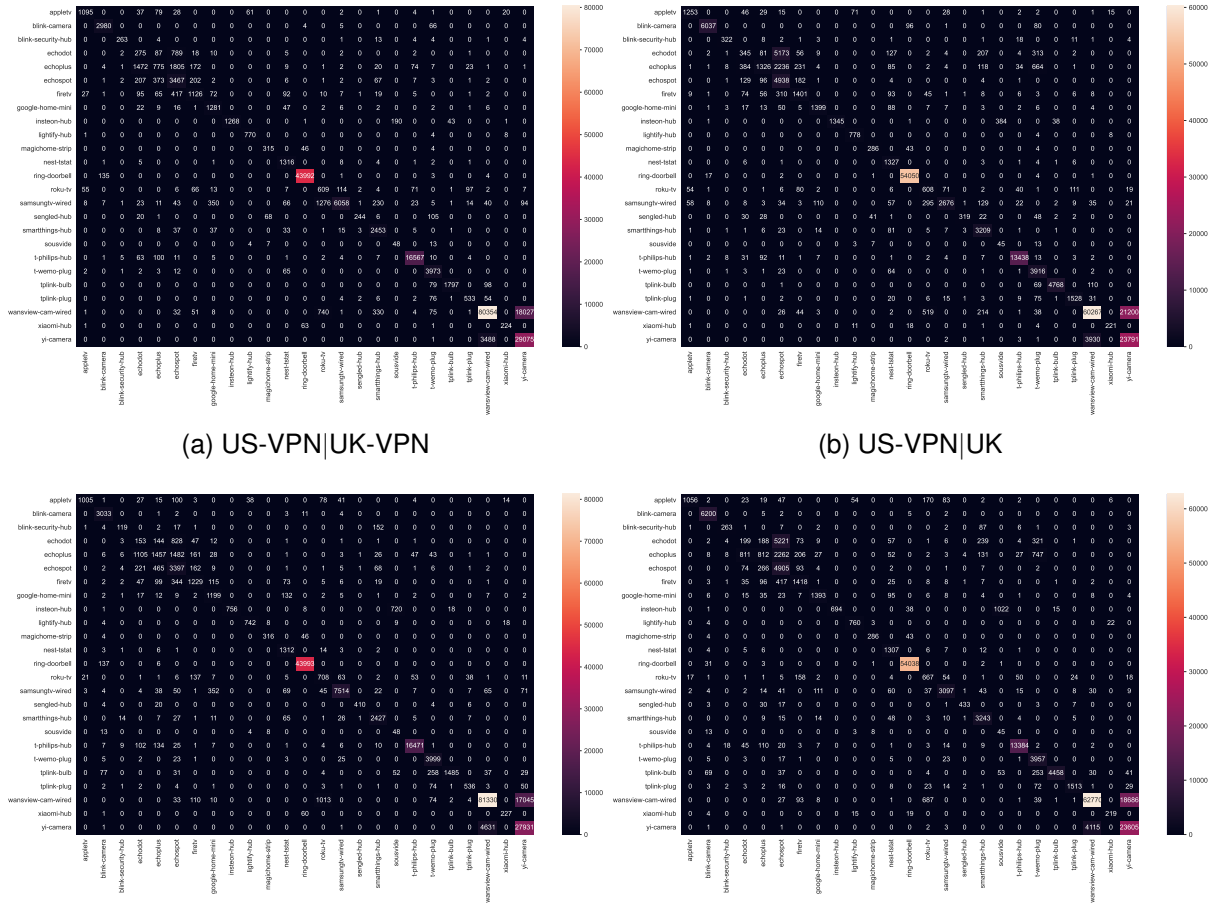
Fig. S11. F1 scores and average inference times of various ML algorithms applied to DD datasets with IoTDevID features. The Random Forest (RF) and XGB methods demonstrate the highest F1 scores of 0.713 and 0.708, respectively, while Decision Trees (DT) show the fastest inference time.

SECTION 4. GEMID METHOD ADDITIONAL TABLES AND FIGURES

The following tables and figures provide additional insights into the GeMID methodology and results, detailing feature selection, evaluation metrics, and model performance.

A. Final Evaluation Confusion Matrices

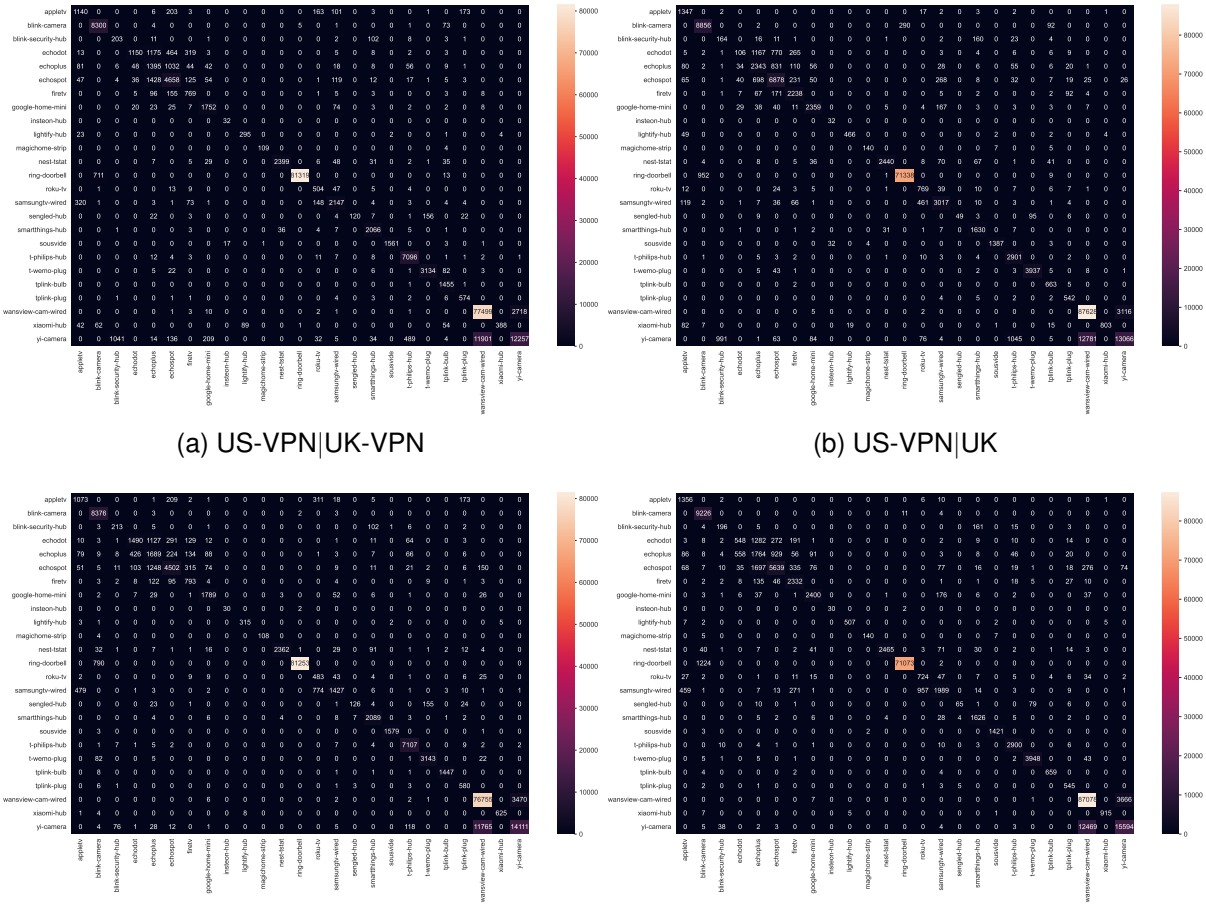
This section presents the confusion matrices for the individual results shared in Table II of the main paper.



(a) US-VPN|UK-VPN (b) US-VPN|UK

(c) US|UK-VPN (d) US|UK

Fig. S12. Confusion matrices depict scenarios where distinct laboratories were employed in the MonIoTr dataset. UK data is utilized for training, while US data is utilized for testing.



(a) US-VPN|UK-VPN (b) US-VPN|UK

(c) US|UK-VPN (d) US|UK

Fig. S13. Confusion matrices depict scenarios where distinct laboratories were employed in the MonIoTr dataset. US data is utilized for training, while UK data is utilized for testing.

B. Other Supporting Tables

TABLE S5
DEVICES COMMON TO UK AND US SITES AND THEIR PACKET COUNTS, WITH GREEN INDICATING HIGHER AND RED INDICATING LOWER PACKET NUMBERS.

Device Names	Packet (Sample) Number			
	US	US-VPN	UK	UK-VPN
appletv	13319	14685	17977	13775
blink-camera	30560	62151	83855	92434
blink-security-hub	2983	3771	3352	3863
echodot	11955	63302	31471	23483
echoplus	43742	51061	27440	35799
echospot	43504	53605	65150	83536
firetv	19479	20274	10487	25935
google-home-mini	13992	16102	19209	26735
insteon-hub	15067	17724	33	33
lightify-hub	7837	7916	3274	5245
magichome-strip	3667	3336	1138	1521
nest-tstat	13438	13508	25651	26854
ring-doorbell	441388	540796	820444	723017
roku-tv	10606	10095	5876	8900
samsungtv-wired	82579	34790	27148	37328
sengled-hub	4464	4967	3370	1642
smartthings-hub	25970	33576	21271	16856
sousvide	73	69	15842	14271
t-philips-hub	167841	136289	71515	29364
t-wemo-plug	40616	40155	32546	40069
tplink-bulb	19753	49492	14580	6715
tplink-plug	6812	16968	5965	5588
wansview-cam-wired	996276	823173	802384	907544
yi-camera	325660	277292	261276	281210
xiaomi-hub	2897	2588	6394	9298
Average	93779	91907	95106	96841

TABLE S6
GEMID VS IOTDEVID ON THE AALTO DATASET WITH OVERALL SCORES

	Accuracy	Precision	Recall	F1 Score
GEMID	0.729±0.003	0.705±0.004	0.592±0.002	0.624±0.003
IoTDevID	0.686+0.000	0.730+0.005	0.665±0.000	0.683±0.002

C. Results from Aggregation Algorithm

The aggregation algorithm proposed in the IoTDevID study is designed to enhance performance by consolidating the machine learning results of individual packets originating from the same source.

TABLE S7
COMPARISON OF DI METHODS ON MONIOTR DATASET WITH AGGREGATION ALGORITHM. F1 SCORES.

	Dataset	GeMDID	IoTDevID	Kitsune
CV	UK	1.000	0.970	1.000
	UKVPN	1.000	0.976	0.997
	US	0.998	0.988	0.960
	USVPN	0.996	0.995	1.000
	Mean	**0.998**	0.982	0.989
SS	UK—UKVPN	0.962	0.943	0.894
	UKVPN—UK	0.951	0.903	0.869
	US—USVPN	0.925	0.911	0.896
	USVPN—US	0.968	0.978	0.895
	Mean	**0.952**	0.934	0.889
DD	UK—US	0.894	0.851	0.638
	UK—USVPN	0.885	0.815	0.536
	UKVPN—US	0.882	0.804	0.618
	UKVPN—USVPN	0.838	0.862	0.637
	US—UK	0.916	0.940	0.723
	US—UKVPN	0.889	0.856	0.605
	USVPN—UK	0.931	0.880	0.616
	USVPN—UKVPN	0.900	0.846	0.643
	Mean	**0.892**	0.857	0.627

TABLE S8
GEMDID F1 SCORES PER DEVICE IN ALL 8 DD CASES FOR MONIOTR DATA WITH AGGREGATION ALGORITHM.

Devices	Train→ / Test→	UK US	UK USVPN	UKVPN US	UKVPN USVPN	US UK	US UKVPN	USVPN UK	USVPN UKVPN
appletv		0.999	1.000	1.000	1.000	0.987	1.000	0.950	0.978
blink-camera		1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
blink-security-hub		1.000	1.000	0.660	0.952	0.914	0.697	0.940	0.802
echodot		0.025	0.000	0.000	0.000	0.621	0.001	0.788	0.140
echoplus		0.141	0.320	0.559	0.116	0.587	0.785	0.814	0.675
echospot		0.699	0.510	0.741	0.502	0.913	0.961	0.990	0.956
firetv		0.969	0.992	0.977	0.992	0.979	0.995	1.000	0.996
google-home-mini		0.999	1.000	0.997	1.000	1.000	1.000	1.000	1.000
insteon-hub		0.986	0.839	0.725	0.273	1.000	1.000	1.000	1.000
lightify-hub		1.000	1.000	0.999	1.000	0.965	1.000	1.000	1.000
magichome-strip		0.969	0.969	0.999	1.000	1.000	1.000	1.000	1.000
nest-tstat		0.999	1.000	1.000	0.999	1.000	1.000	1.000	1.000
ring-doorbell		1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
roku-tv		0.949	0.984	1.000	0.997	0.988	0.993	0.744	0.787
samsungtv-wired		0.994	0.997	1.000	1.000	0.998	0.998	0.912	0.917
sengled-hub		0.768	0.869	1.000	1.000	0.450	0.261	0.491	0.561
smartthings-hub		1.000	0.999	0.979	0.993	0.989	0.950	0.991	0.964
sousvide		0.905	0.723	0.441	0.186	1.000	1.000	1.000	1.000
t-philips-hub		1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
t-wemo-plug		0.987	0.981	0.999	0.975	0.966	0.983	0.967	0.988
tplink-bulb		1.000	1.000	0.998	1.000	0.993	1.000	1.000	1.000
tplink-plug		0.996	0.997	1.000	1.000	0.989	1.000	0.990	1.000
wansview-cam-wired		0.998	0.983	0.998	0.991	0.928	0.930	0.942	0.950
xiaomi-hub		0.972	1.000	0.972	1.000	0.955	1.000	1.000	1.000
yi-camera		0.995	0.954	0.993	0.975	0.686	0.678	0.767	0.796
accuracy		0.975	0.941	0.978	0.939	0.934	0.932	0.950	0.945
macro avg		0.894	0.885	0.882	0.838	0.916	0.889	0.931	0.900

D. Hyperparameter Optimization

This section presents the hyperparameter ranges and the selected values for both classical machine learning (ML) and neural network-based models. Tables S9 and S10 outline the hyperparameter tuning process, highlighting the optimal parameter settings identified during experimentation to achieve the best model performance.

TABLE S9
HYPERPARAMETER SELECTION - VALUE RANGES AND SELECTED PARAMETERS FOR CLASSICAL ML ALGORITHMS

Method	**Parameter**	**Value Range**	**GEMID**	**CICFwMtr**	**IoTDevID**	**Kitsune**
DT	Criterion	gini, entropy	Entropy	Entropy	Entropy	Entropy
	Max Depth	1 - 32	14	18	19	16
	Max Features	1-Feature num.	21	6	24	33
	Min Samples Split	2 - 10	5	6	5	5
RF	Bootstrap	True, False	False	False	False	False
	Criterion	gini, entropy	Entropy	Gini	Gini	Entropy
	Max Depth	1 - 32	17	29	31	20
	Max Features	1 - 11	3	6	10	2
	Min Samples Split	2 - 11	5	5	2	2
	N Estimators	1 - 200	71	88	136	79
XGB	N Estimators	100, 500, 900, 1100, 1500	900	1100	1500	1100
	Max Depth	2, 3, 5, 10, 15	3	3	3	2
	Learning Rate	0.05, 0.1, 0.15, 0.20	0.15	0.15	0.15	0.2
	Min Child Weight	1, 2, 3, 4	1	1	1	1
KNN	Algorithm	auto, ball_tree, kd_tree, brute	kd_tree	ball_tree	ball_tree	ball_tree
	Leaf Size	1 - 50	39	44	35	44
	N Neighbors	1 - 64	5	4	13	2
	Weights	uniform, distance	Distance	Distance	Distance	Distance
NB	Var Smoothing	10^{0} - 10^{-9}	1.52×10^{-6}	8.11×10^{-9}	2.31×10^{-6}	8.11×10^{-9}
LR	C	10^{-5} - 100 (log-uniform)	0.0809264	0.10342	0.689136	0.071026
	Penalty	none, L1, L2, elasticnet	L2	L1	L1	L1
	Solver	newton-cg, lbfgs, liblinear	newton-cg	liblinear	liblinear	liblinear
SVM	Gamma	0.001, 0.01, 0.1, 1	0.001	0.001	1	0.001
	C	0.001, 0.01, 0.1, 1, 10	1	1	10	10

TABLE S10
HYPERPARAMETER SELECTION - VALUE RANGES AND SELECTED PARAMETERS FOR NEURAL NETWORK NEURAL NETWORK MODELS

Method	Parameter	Value Range	GEMID	CICFwMtr	IoTDevID	Kitsune
CNN	Filters	[32, 64, 96, 128]	64	32	32	32
	Kernel Size	[3, 4, 5]	3	3	3	3
	Num Dense Layers	[1, 2, 3]	1	1	1	1
	Dense Units	[32, 64, 96, 128]	32	128	96	32
	Dropout	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]	0.0	0.0	0.0	0.0
	Learning Rate	[1e-2, 1e-3, 1e-4]	0.01	0.01	0.01	0.01
	Epochs	[10]	10	10	10	10
	Batch Size	[32]	32	32	32	32
LSTM	Units	[32, 64, 96, 128]	64	128	32	32
	Dropout 1	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]	0.1	0.3	0.2	0.1
	Num Dense Layers	[1, 2, 3]	1	2	1	1
	LSTM Units	[32, 64, 96, 128]	64	96	96	128
	Dropout 2	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]	0.3	0.4	0.2	0.4
	Units Last	[32, 64, 96, 128]	128	128	128	96
	Dropout Last	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]	0.4	0.4	0.2	0.3
	Learning Rate	[1e-2, 1e-3, 1e-4]	0.01	0.01	0.01	0.0001
	Epochs	[10]	10	10	10	10
	Batch Size	[32]	32	32	32	32
ANN	Units Input	[32, 64, 96, 128]	128	128	32	32
	Num Dense Layers	[1, 2, 3]	1	3	2	1
	Units	[32, 64, 96, 128]	96	128	128	32
	Dropout	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]	0.0	0.1	0.0	0.0
	Learning Rate	[1e-2, 1e-3, 1e-4]	0.01	0.01	0.01	0.001
	Epochs	[10]	10	10	10	10
	Batch Size	[32]	32	32	32	32
	Validation Split	[0.3]	0.3	0.3	0.3	0.3
BERT	Batch Size	[16, 32, 64]	16	32	16	16
	Learning Rate	[1e-5, 3e-5, 5e-5]	1e-5	3e-5	1e-5	3e-5
	Epochs	[3, 5, 7]	7	3	3	7
	Weight Decay	[0, 0.01, 0.1]	0	0.01	0	0.01

TABLE S11
SPECIFICATIONS OF THE EXPERIMENTAL PLATFORM

Component	Details
Processor	12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz
Installed RAM	16.0 GB (15.7 GB usable)
System Type	64-bit operating system, x64-based processor
Edition	Windows 11 Home / Version 23H2

TABLE S12
DESCRIPTION OF NON-CORE PYTHON LIBRARIES USED IN THE PROJECT

Library	Description
numpy	Fundamental package for numerical computing in Python.
tqdm	Library for adding progress bars to Python code.
zipfile	Module to read and write ZIP files (standard library).
scapy	Library for packet manipulation and network analysis.
pandas	Data manipulation and analysis tool for structured data.
pickle	Module for serializing and de-serializing Python objects.
scikit-learn	Machine learning library for Python.
tabulate	Library for creating nicely formatted tables in Python.
pyximport	Module for importing Python modules written in C/C++.
seaborn	Statistical data visualization library based on Matplotlib.
transformers	Library for natural language processing with pre-trained models.
torch	Deep learning framework providing GPU acceleration.
keras_tuner	Library for hyperparameter tuning in Keras models.
tensorflow	End-to-end open-source platform for machine learning.

SECTION 5. ADAPTING NEURAL NETWORK-BASED LEARNING MODELS FOR IOT DEVICE IDENTIFICATION

This section provides an in-depth explanation of how specific machine learning models—Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), and Bidirectional Encoder Representations from Transformers (BERT)—were adapted to classify IoT devices. While these models are commonly applied to distinct data types (e.g., image data for CNNs, sequential data for LSTMs, and natural language processing for BERT), their adaptability to IoT device data required tailored preprocessing and structuring. Here, we detail the approach taken for each model, including preprocessing techniques and model architecture adjustments, to fit them to the IoT device identification task.

A. Input Representation in BERT-based Model

1) Tabular Data and Feature Engineering: Our input features consist of numerical and categorical data (e.g., port numbers, IP flags, TCP header lengths), commonly used in IoT traffic analysis. Unlike BERT's traditional text-tokenized inputs, these features require a tailored embedding approach.

2) Customized Input Handling: To adapt these tabular features for BERT, we input them directly into the model through a fully connected layer (`fc1`, implemented as `torch.nn.Linear`). This layer projects feature vectors from the original input space into a 768-dimensional embedding space, aligning with the BERT model's expected hidden layer size.

```
x = torch.relu(self.fc1(x))
x = self.transformer(inputs_embeds=x.unsqueeze(1)).last_hidden_state
```

3) Utilizing BERT as a Transformer Backbone: We utilize `inputs_embeds` instead of tokenized input IDs, allowing embedded numerical features to pass directly into BERT.

```
x = self.transformer(inputs_embeds=x.unsqueeze(1)).last_hidden_state
```

Typically, BERT processes sequences of token embeddings. By passing our feature embeddings via `inputs_embeds`, we adapt tabular data as sequences of length one. Although unconventional, each row in the IoT dataset becomes a distinct input sequence.

4) Pooling and Prediction: After processing the features through BERT, we apply mean pooling to aggregate hidden states across the input sequence. The pooled output is then passed through a classification layer (`fc2`) to generate predictions.

B. Contrasting Standard and Custom BERT Workflows

Standard BERT workflow:

```
Input text → Tokenization → Token Embeddings → Transformer → Classification Layer
```

Custom Workflow in This Study:

```
Tabular Data → Linear Embedding → Transformer → Pooling → Classification Layer
```

C. Input Representation in CNN-based Model

While Convolutional Neural Networks (CNNs) are typically used for image classification, they can also be applied to other data formats, such as time-series or tabular data. In this study, a **1D CNN** was used for classifying IoT devices based on network traffic features.

Tabular IoT data usually comprises numerical features (e.g., packet length, header flags). Although CNNs are generally applied to spatial data (like images), they are also capable of capturing **local patterns** in sequential or structured data, making them useful for:

- **Time-series analysis**, where the sequence of inputs matters.
- **Feature correlations** in structured datasets, like network traffic logs.

1) Input Shape for 1D CNN: Each row in the dataset corresponds to one instance (e.g., an intercepted traffic session) with multiple features. Let:

- n: Number of instances (rows).
- m: Number of features (columns).

The input shape for a 1D CNN is transformed to:

$$(n, m, 1)$$

where 1 is the **channel dimension**, analogous to the RGB channels in image data but here signifying a **single feature channel**.

2) Steps to Prepare Data for CNN: Before feeding the data into the model, the following steps are performed:

1) **Data Scaling:** A Min-Max scaler is applied to normalize the feature values between 0 and 1:

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

2) **Reshaping:** The scaled data is reshaped to:

$$X_{\text{train}} \in \mathbb{R}^{n_{\text{train}} \times m \times 1}, \quad X_{\text{test}} \in \mathbb{R}^{n_{\text{test}} \times m \times 1}$$

3) CNN Architecture for IoT Device Classification: The CNN model is defined as follows:

```
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',
input_shape=(m, 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.0))
model.add(Dense(96, activation='relu'))
model.add(Dropout(0.0))
model.add(Dense(21, activation='softmax'))
```

- **Conv1D Layer:** Applies convolution over the feature dimension (of size m) to extract **local patterns** between adjacent features.
- **MaxPooling1D Layer:** Reduces the dimensionality of the output and helps prevent overfitting.
- **Flatten Layer:** Converts the 2D tensor output of the previous layer into a 1D vector for further processing.
- **Dense Layers:** Fully connected layers for learning non-linear combinations of the extracted features.
- **Softmax Layer:** The final dense layer with 21 units corresponds to the 21 device classes, and the `softmax` activation outputs class probabilities.

4) Feeding the Input to the CNN: The `input_shape` argument of the first `Conv1D` layer is defined as:

$$\texttt{input\_shape} = (m, 1)$$

This indicates that each input sample consists of m features arranged along the feature axis, with 1 channel per feature.

During training, batches of data with shape:

$$\text{batch\_size} \times m \times 1$$

are fed into the CNN. For example, if the batch size is 32, the input will have the shape:

$$(32, m, 1)$$

D. Input Representation in LSTM-based Model

This appendix details the key steps and design choices made to adapt a Long Short-Term Memory (LSTM) neural network model for IoT device identification. While LSTMs are traditionally used in text and sequential data analysis, their ability to capture temporal dependencies makes them a viable option for analyzing IoT traffic patterns and device identification.

1) Data Preparation: The data preprocessing workflow involves several key steps to ensure compatibility with the LSTM model's input requirements:

- **Data Loading:** Device feature data and corresponding labels are imported from CSV files, enabling a structured format for the model input.
- **Feature Normalization:** The `MinMaxScaler` from the scikit-learn library is applied to scale feature values to a standardized range, which improves convergence and stability during training.
- **Input Reshaping:** To align with the LSTM's expected 3D input shape (samples, time_steps, features), each 1D feature vector is reshaped with time_steps = 1, accommodating the sequential nature of LSTM processing.

2) LSTM Model Architecture: The architecture design leverages the LSTM's capabilities for temporal feature extraction and sequence learning:

- **LSTM Layers:** The model comprises multiple LSTM layers with varying units (32, 64, 96), structured to progressively extract higher-level temporal features. Each layer uses `return_sequences=True`, enabling the passage of sequential outputs across layers.
- **Dropout Regularization:** Dropout layers interspersed between LSTM layers help reduce overfitting, improving model generalization on unseen data.
- **Output Layer:** A Dense output layer with 21 units and a softmax activation function produces a probability distribution over the device classes, effectively handling the multi-class classification problem.