Lumen: A Framework for Developing and Evaluating ML-Based IoT Network Anomaly Detection

Rahul Anand Sharma[†], Ishan Sabane[§], Maria Apostolaki[†◇], Anthony Rowe[†], Vyas Sekar[†]

[†]Carnegie Mellon University, [§]IIT Madras, [◇]Princeton University

ABSTRACT

The rise of IoT devices brings a lot of security risks. To mitigate them, researchers have introduced various promising network-based anomaly detection algorithms, which oftentimes leverage machine learning. Unfortunately, though, their deployment and further improvement by network operators and the research community are hampered. We believe this is due to three key reasons. First, known ML-based anomaly detection algorithms are evaluated –in the best case– on a couple of publicly available datasets, making it hard to compare across algorithms. Second, each ML-based IoT anomaly-detection algorithm makes assumptions about attacker practices/classification granularity, which reduce their applicability. Finally, the implementation of those algorithms is often monolithic, prohibiting code reuse. To ease deployment and promote research in this area, we present Lumen. Lumen is a modular framework paired with a benchmarking suite that allows users to efficiently develop, evaluate, and compare IoT ML-based anomaly detection algorithms. We demonstrate the utility of Lumen by implementing state-of-the-art anomaly detection algorithms and faithfully evaluating them on various datasets. Among other interesting insights that could inform real-world deployments and future research, using Lumen, we were able to identify what algorithms are most suitable to detect particular types of attacks. Lumen can also be used to construct new algorithms with better performance by combining the building blocks of competing efforts and improving the training setup.

CCS CONCEPTS

• **Computing methodologies** → *Machine learning*; **Machine learning**; • **Networks** → **Network security**; **Network security**; • **Security and privacy** → *Intrusion detection systems.*

ACM Reference Format:

Rahul Anand Sharma[†], Ishan Sabane[§], Maria Apostolaki[†◇], Anthony Rowe[†], Vyas Sekar[†] , [†]Carnegie Mellon University, [§]IIT Madras, [◇]Princeton University, . 2022. Lumen: A Framework for Developing and Evaluating ML-Based IoT Network Anomaly Detection. In *The 18th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '22), December 6–9, 2022, Roma, Italy.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3555050.3569129

CoNEXT '22, December 6–9, 2022, Roma, Italy

ACM ISBN 978-1-4503-9508-3/22/12.
https://doi.org/10.1145/3555050.3569129

1 INTRODUCTION

IoT devices are everywhere, with an estimated 75 billion devices deployed by 2025 [2]. While IoT devices have huge potential, they come with attendant security challenges. In fact, IoT devices often become entry points into critical infrastructures and/or leak sensitive information [16, 23, 36–38]. Traditional security approaches (e.g., antivirus, software patches) are often not suitable for IoT devices due to vendor security, management practices [3, 8] and constrained hardware.

Network-layer anomaly detection [11, 13, 18, 27, 40, 41] tailored for IoT devices offers a pragmatic alternative for IoT security. This is based on two key insights. First, network gateways (e.g., IoT hubs) act as a natural "chokepoint" for inspecting traffic to/from IoT devices. Second, by nature, IoT devices exhibit fairly constrained normal behavior, unlike general-purpose user computers, and this normal behavior can potentially be captured using machine learning (ML) models. Indeed, we have seen a plethora of ML-based *network-layer* IDS (NIDS) for IoT devices [11, 13, 15, 18, 20, 24, 26–28, 30, 40].[1]

We qualitatively and quantitatively analyze a wide spectrum of prior work in this space and identify three key shortcomings:

- *Limited evaluation:* ML-based IoT IDS are typically evaluated only on a couple of datasets, and thus there is no guarantee that they will generalize. While limited evaluation is a well-known problem for any ML-based system, it is further amplified in IoT due to the lack of public datasets. In fact, we find that most of the datasets used in published works are private and created by the authors of the corresponding work.
- *Implicit deployment assumptions:* Each ML-based IoT anomaly-detection algorithm makes unique assumptions about the attacker's practices; thus, there is little common ground for comparison. For example, some IDS are tailored to only detect certain types of attacks or assume that the attacker's traffic has a common source IP prefix [15].
- *Inextensible:* Most IDS are designed in a monolithic manner; thus, there is little chance for code reuse and further optimizations. Worse yet, there are few open-source deployments that one can build upon or reproduce results.

Taken together, these shortcomings impact both the state of academic research and that of practice. For instance, practitioners cannot quickly test existing algorithms in practice or have well-defined playbooks to deploy these ML approaches in new settings. At the same time, researchers cannot rapidly prototype new approaches or faithfully compare those with prior work.

[1]For the rest of the paper, we use the term IDS and NIDS interchangeably, noting that our scope is restricted to network-layer IDS.

These factors motivate us to develop a common *development framework* and a *benchmarking suite*. The common development framework will facilitate rapid prototyping, code reuse, and allow easy comparison between algorithms. The benchmarking suite will facilitate fair and comprehensive comparison among algorithms under common assumptions that will be easier to interpret. The combination of the two will give the research community clarity on the precision of the various algorithms, their shortcomings, and the most promising research directions going forward.

Although intuitive, designing a development framework is not trivial due to the heterogeneity of algorithms and datasets. As an intuition, today's algorithms are too complex to be expressed using existing general-purpose development frameworks such as netML [39]. Moreover, publicly available datasets contain various attacks, are collected in heterogeneous networks, and are labeled at varying granularities. In effect, selecting a relevant and compatible dataset to test an algorithm requires understanding both the algorithm's working and the dataset collection process.

To solve this problem, we rely on two key insights. First, although most algorithms have separate workflows, we identify a common abstraction and use this to create a *modular* development framework. This modularity facilitates efficient deployment and testing. Concretely, we optimize multiple algorithms at the same time by optimizing each module (used by multiple algorithms) independently. Moreover, we construct the evaluation pipeline such that intermediate results are shared across algorithms. Second, we observe that an ML-based anomaly detection algorithm can only be faithfully tested against a dataset that is of the same classification granularity (or can be transformed into that granularity).

We demonstrate the usability of our framework by implementing 16 state-of-the-art algorithms. Using this implementation and our benchmarking suite, we compare the algorithms and report multiple actionable insights. Finally, we demonstrate the power of our modular framework by using it to automatically synthesize new algorithms by combining modules from existing work. Our automatically generated algorithm can achieve higher precision than any of the previously proposed works.

In summary, our paper makes the following contributions:

(1) A detailed literature search on ML-based IoT anomaly detection systems and identification of the factors that hinder systematic comparison across them.
(2) The design and implementation of Lumen: a modular framework to develop ML-based IoT anomaly detection algorithms and a benchmarking suite to evaluate them. The combination of the two facilitate *(i)* fast implementation of ML-based IoT anomaly algorithms; *(ii)* faithful evaluation on multiple datasets; *(iii)* comparison across algorithms; and *(iv)* guidance for future algorithm design.
(3) An implementation and head-to-head comparison of 16 algorithms on 15 datasets using Lumen.
(4) The design of two new Lumen-guided heuristics which demonstrates better precision compared to existing algorithms. The heuristics combine modules from existing algorithms and leverage an optimized training setup.

2 PRIOR WORK & LIMITATIONS

IoT ML-based anomaly detection has been a very active research area in recent years. There exist multiple algorithms that aim to identify anomalous IoT behavior using machine learning [11, 15, 18, 20, 24, 26, 27, 30]. In this section, we first describe a taxonomy of ML-based anomaly detection algorithms. Next, we put ourselves in the shoes of an operator and identify three shortcomings that hinder the deployment and further improvement of existing algorithms.

2.1 Taxonomy of prior work

The goal of an IoT anomaly detection algorithm is to distinguish between malicious and benign IoT traffic. To this end, a typical ML-based anomaly detection algorithm defines a feature-engineering pipeline and an ML training and testing pipeline. The feature-engineering pipeline consists of operations to convert network traffic to a feature vector. Popular methods for feature engineering include applying statistical aggregates (e.g. mean packet inter-arrival time, median packet length), and grouping packets (e.g. based on source IP or destination IP). Each feature vector is classified as benign or malicious (label). The training pipeline takes as input the set of feature vectors along with their corresponding ground-truth labels (in the case of supervised classification) and learns a model that can predict if a given feature vector corresponds to benign or malicious traffic. The ML model can be as simple as passing feature vectors over to a Decision Tree or more complicated e.g. autoencoders or deep neural networks.

Despite their common high-level goal and approach, we find that IoT ML-based anomaly detection algorithms can greatly differ in their *(i)* scope (e.g. deployment type, targeted attacks); *(ii)* solution specifics (e.g. features, ML model, classification granularity); and *(iii)* evaluation methodology (e.g. datasets, testing pipeline).

While one could study these algorithms considering any of those differentiators, we review them grouped by their classification granularity. As we explain in §2.2, the different classification granularity hinders faithful comparison across algorithms and is critical to our work. We distinguish three common granularities *(i)* packet; *(ii)* unidirectional flow and *(iii)* connection.

Packet-level algorithms independently classify each packet as malicious or benign. This is the finest granularity possible. While more flexible such algorithms can be less scalable (as they might need per-packet inference or rules). Representative examples of this category include nPrint [20], which proposes a new unified packet representation or feature that can be used for any ML task. It builds a feature vector using packet fields from various layers. SmartHome [11] converts each packet to a packet description markup language (PDML). Kitsune [27] is a classical anomaly detection algorithm that generates packet-level features by grouping packets based on criteria such as source IP address, channel, socket, and computing 1D and 2D statistical features. ML_DDoS [18] introduces an ML model built using IoT network variables such as bandwidth, packet interval, protocols, packet size, and destination address for DoS attack detection.

Unidirectional-flow-level algorithms classify each unidirectional flow as anomalous or benign. A unidirectional flow is a set of packets with common tuple of srcIP, dstIP, srcPort, dstPort, and protocol [24, 30]. While this classification granularity is less flexible, it is

Algorithm	ML Model	Granularity	Datasets	Reported Performance
ML for DDoS [18]	Ensemble of RF, SVM, DT and KNN	Packet	Custom	Precision: 99.9%
Efficient One-Class SVM [40]	OCSVM and GMM	Packet	CTU IoT, UNB IDS, MAWI	AUC: 62 - 99%
Kitsune [27]	Stacked Auto-Encoders	Packet	Custom Camera Traffic	Precision: 99%
Nprint [20]	AutoML	Packet	CICIDS2017, netML	Balanced Precision: 86-99%
Smart Detect [24]	Random Forest	Unidirectional Flow	CICIDS2017, CIC-DoS	Precision: 80 - 96.1%
Network Centric Anomaly Detection [15]	Auto Encoder	Flow: srcIP, dstIP	Custom [2]	Precision: 99%
Industrial IoT [41]	Random Forest	Connection	Custom	Sensitivity: 97%
Smart Home IDS [11]	Random Forest	Packet	Custom	Precision: 97%
Ensemble [30]	NB,DT, RF and DNN	Unidirectional Flow	UNSW NB-15, NIMS	Precision: 98.29-99.54%
Bayesian Traffic Classification [28]	Bayes Classifier	Connection	Custom	Precision: 96.29%
Zeek Logs [13]	RF	Connection	CTU	Precision: 97%

Table 1: List of network-layer ML-based anomaly detection algorithms for IoT devices. The heterogeneity in the algorithms' design (e.g. classification granularity) and evaluation (e.g. used datasets) make directly comparing the reported precision values meaningless.

intuitive and practical as it assumes access to a single direction of traffic. SD-IoT [24] proposes to build a DDoS detection algorithm using features from IP and TCP layer fields. Ensemble[30] proposes to build features from the MQTT, DNS, and HTTP protocols.

Connection-level algorithms classify and build features at the connection level (*i.e.,* bidirectional flow). This is the coarsest granularity for which we find algorithms. For instance, OCSVM [40] proposes creating a feature for each flow based on the first hundred packets' inter-arrival times and lengths. IIoT [41] creates an IDS system customized to SCADA systems by building features based on packet time, length, raw bytes, bandwidth, packet loss, and jitter. BayesianIDS [28] proposes building features from 248 per-flow discriminators combined with a naive Bayes classifier.

Classification algorithms naturally work with a coarser granularity than their original but not with a finer one. For example, a packet-level algorithm classifies at a packet granularity and thus can be trained with a flow-granularity dataset as we can propagate the flow label to all packets of each flow. Yet, a connection-level algorithm cannot be trained with a packet-granularity dataset because there will be connections that contain packets with both labels; thus, one would need to change the ground-truth data.

2.2 Limitations of prior work

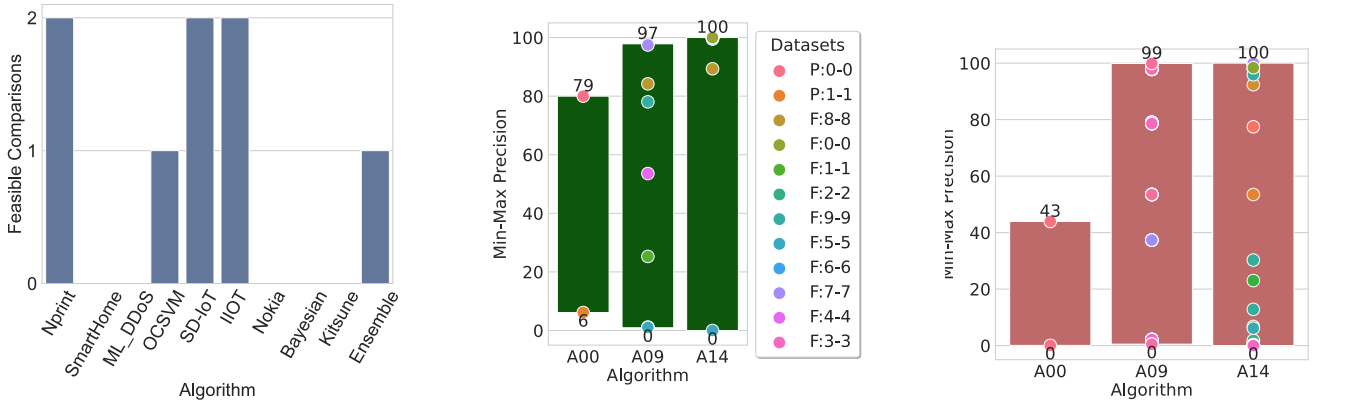
In this subsection, we explain the inefficiencies of previous ML-based IoT anomaly detection algorithms that hinder deployment, comparisons, and future advances. To illustrate our point, we put ourselves in the shoes of a network operator who wants to deploy such an algorithm.

Example Scenario Consider an operator who wants to implement an anomaly detection algorithm in their small business to detect brute force and DoS attacks on IoT devices. To this end, the operator turns to research work in the field, with the aim of finding the most suitable algorithm for their needs. Assume that the operator performs a literature search resulting in a comparative table similar to Table 1. Next, we explain the challenges the operator would face to identify and build upon the most accurate algorithm.

Literature search is inconclusive, meaning that an operator reviewing previous works has no guarantee that a given algorithm will be accurate in practice in their deployment or at least more accurate than others. As we see in Table 1, each algorithm is evaluated on a few datasets; thus, there is no guarantee that it generalizes to different conditions (on different datasets). Worse yet, many of those datasets are private and/or introduced by the same paper which introduces the algorithm. Moreover, there is very little reuse of datasets in papers. As an illustration, Fig. 1a shows the number of direct comparisons that are possible for each given algorithm. A comparison is possible when two algorithms have been evaluated in at least one common dataset. Notably, for half of the algorithms that we reviewed, there is no possible comparison.

Implicit assumptions in the algorithm design hinder their faithful comparison. As literature search alone does not allow comparisons, an operator could try to implement and compare different algorithms internally. At first sight, such a methodology is straightforward and will allow the operator to compare various algorithms on the same dataset. In practice, however, evaluating various algorithms is extremely challenging and time-consuming. First, most of the works do not have publicly available codes. To address this, the operator could focus only on evaluating open-source algorithms, despite the risk of dismissing superior algorithms. Unfortunately, that does not make the evaluation easier, as many authors have not publicly released their datasets. To address this, the operator could use publicly available datasets. Unfortunately, doing so is not straightforward because each algorithm is designed assuming that training and testing are done at a particular granularity, as we explain in §2. For instance, if the labels of a dataset are per packet, but the algorithm classifies traffic at the connection granularity, then training is not possible. Indeed, one would need to either modify the labels of all packets corresponding to a single connection to match or remove all connections whose packets do not hold the same label over time (either benign or malicious). In

[2] uses publically available benign traces and private attack traces



(a) Comparing algorithms using the literature-reported precision is not feasible because they are not evaluated on the same datasets.

(b) The precision of the tested algorithms varies widely across datasets, showing their lack of generality, even when trained and tested on data from the same dataset.

(c) The variance in precision of the tested anomaly detection algorithms further degrades (compared to Fig.1b) when they are tested and trained on different datasets.

Figure 1: An operator cannot faithfully compare (even open-sourced) IDS algorithms due to their limited evaluation, implicit assumptions, and/or monolithic system designs.

either case, one would be forced to evaluate an algorithm on a modified dataset, effectively influencing the results.

Monolithic design prevents reusing, improving, and/or learning from existing algorithms. Even after the operator has found an implementation of some algorithms and a couple of datasets to evaluate them, comparing them can be a rather confusing task. As an illustration, Figures 1b,1c show the precision of various algorithms when evaluated under a pair of training and testing datasets from the same and different sources, respectively. Unfortunately, even after all this work, the operator cannot decide which algorithm is more suitable. First, each algorithm displays a wide range of precision values. Second, some algorithms (e.g. A00) are good when trained and tested on the same dataset (Fig. 1b), but their performance significantly degrades if they are trained and tested on different datasets (Fig. 1c). Finally, the operator might not be able to evaluate how relevant each dataset is to their own deployment.

Overcoming these limitations requires the operator to debug some of the cases or understand the workings of each algorithm and extract their key insights. The operator's burden could be reduced if the algorithms were implemented and evaluated more systematically such that the operator would be able to reason about their differences, root causes for performance degradation, and improvements.

3 FRAMEWORK: LUMEN OVERVIEW

As we explain in § 2.2 various inefficiencies impact both the state of academic research and that of practice. For example, an operator or researcher cannot answer seemingly straightforward questions such as: *Which is the most accurate IoT ML-based IDS? Does it generalize to other deployments? How does my new algorithm compare to the state-of-the-art?* In this section, we first define the output of Lumen (§3.1), before we elaborate on its workings.

3.1 Overview

As illustrated in Figure 2, Lumen consists of a development framework and a benchmarking suite. Lumen works in three logical steps, which can run one after the other or independently.

The first step facilitates rapid prototyping and is useful if the user wants to invent a new algorithm or extend the framework. Lumen takes as input the programmer's specification of a new anomaly detection algorithm as a sequence of operations among those that Lumen has defined and optimized. As a result, the use of Lumen can speed up the implementation of a new algorithm while reducing the burden of debugging, testing, and optimizing the individual components.

The second step facilitates fair comparison amongst algorithms. Leveraging its benchmarking suite and its pre-compiled algorithms, Lumen compares across a considerable subset of the related literature and under a variety of conditions. The user can scope the comparison on a subset of algorithms or datasets.

In the last step, Lumen stores all results in a query-friendly format and generates illustrations, effectively facilitating multiple tasks. First, Lumen illustrations can help an operator easily identify the most suitable algorithm to deploy considering their needs, network deployment, or attacks of interest. Second, Lumen output can help a researcher or developer to identify interesting patterns that could lead to better algorithms or just compare their algorithms with state-of-the-art alternatives with minimum overhead.

While there has been some work with similar goals [1, 12, 20, 33, 39], they are not addressing the inefficiencies we described in § 2.2. First, such works such as "netml" [39] are not expressive enough to develop complex feature-building pipelines that exist in the literature, such as Kitsune [27] IDS. Moreover, such works do not provide an evaluation framework or benchmarking suite and thus cannot be used to compare across algorithms. Finally, while some competitions on applying ML to networking data [5] have

been organized, they include very few datasets and do not provide direct access to packet captures.

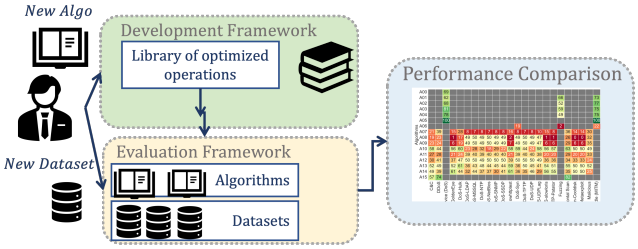


Figure 2: Lumen enables comprehensive comparison across a variety of algorithms and datasets. Lumen modular design allows (but does not require) a user to describe a new anomaly-detection algorithm and compare it with the state-of-the-art with minimum overhead.

3.2 Lumen development framework

We design Lumen to be a development framework that supports the features used by all previously-proposed algorithms and speeds up the prototyping of new algorithms. To achieve this we *(i)* identify the most commonly used operations; *(ii)* optimize these operations; and *(iii)* facilitate arbitrary connections across them. Doing so allows code reuse, reduces potential bugs, and improves overall performance.

Identifying commonly used operations We did a thorough literature search and meta-analysis of various published algorithms and publicly available codes. We observe that previously proposed algorithms vary widely in terms of *(i)* their classification granularity (e.g. packet-level or connection-level); *(ii)* the packet fields that they use (e.g. TCP sequence number, or destination IP address); *(iii)* how they build useful features (e.g. mean packet interarrival time, median packet size); and *(iv)* how they are fed to an ML model for classification(e.g. Random Forest, Autoencoder followed by a deep neural network).

Despite the variety, we recognized an emerging pattern that led us to identify a set of operations that are repeatedly used in various algorithms. These include around 30 unique operations such as extracting fields, time slicing, grouping, computing aggregates, feature normalization *etc.*

Our operations are configurable; thus, each operation can, in practice, support multiple functions. Having configurable implementation of these operations allows us to have fewer efficient implementations, effectively reducing debugging and testing efforts. Moreover, fewer implementations allow us to do further optimizations. Upon implementation, we realize that most of our operations are amenable to a map-reduce type of framework. To make our framework scalable, we have optimized many of these operations to use parallel and distributed python frameworks.

We used these modules to implement 16 algorithms as we describe in §5. As an illustration, Figure 3 shows the logical pipeline of the Kitsune classification algorithm [27]. To implement this algorithm, we need operations to extract packet fields (size, time), group data (by srcIp) , apply aggregate functions (bandwidth, number of dstIP), etc. The extraction of size and time is served by the same module and requires a single pass over the dataset.

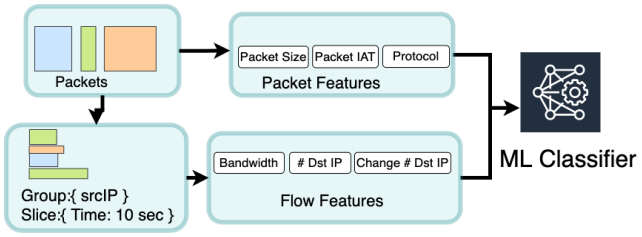


Figure 3: Kitsune classification algorithm that does packet-level classification. The proposed features are developed by applying aggregate functions (mean packet size, bandwidth, mean packet inter-arrival time) to packets, packets grouped by srcIP, and based on a sliding time window.

Flexible connections across operations While the configurable already gives us enough flexibility, we need a way to define the order in which the operations are connected to compose a logical pipeline.

As an intuition, in our previous example in Fig,3 we not only need to define the operations but the connections across them: e.g. output from a groupby operation (based on srcIP) needs to be fed to a time slicer operation (10 seconds) that needs to be fed into an aggregator operation to create a feature vector that is ultimately fed to an ML classifier operation.

To enable arbitrary connections across operations, we introduce a simple template-based language, which allows the programmer to create a configuration by only filling in the gaps on a template pipeline to file. An example of such a file is illustrated in Figure 4.

To achieve this, we extend the Lumen operations to include their input and output. Thus each operation in the template is a configurable operation and has an input, output, and algorithm-specific parameter. The input and output of each operation can either be packets or packets grouped by a particular attribute.

After the user configures a new algorithm using the template file, the file is passed to an execution engine. The execution engine verifies the file's syntax (e.g. type checks) before executing it. To further help the user, Lumen identifies the operations that need further optimizations. To this end, the execution engine generates plots of memory and time spent in each operation. The execution engine also does some basic memory optimizations, such as removing variables/data that are not used in future operations to conserve memory.

Algorithm configuration example Let's go through an example to understand how an algorithm can be written using our development framework. Our template file (Figure 4) starts with a "Field Extract" operation, and we specify what packet fields we want to extract. The set of extracted packet fields is then grouped based on source IP by operation "Groupby". For each group of packets, we can then compute aggregate functions using the "ApplyAggregates" operation. We then specify that we want to use a Random Forest

classifier using our "model" operation, and finally, the training is performed by the "train" operation.

To validate the generality and usability of our development framework, we have ported over 16 existing algorithms. Observe that each of these algorithms has very distinct characteristics. For instance, they have different classification granularities, different ways of building features (nprint, zeek, wireshark pdml [6]), etc. Importantly, our framework is also extensible, meaning that it can be easily modified to support new modules with minimal changes.

3.3 Lumen evaluation framework

We design Lumen to be a framework that can *faithfully* run a variety of algorithms on publicly available datasets, meaning the framework should honor the design decisions of the designer (benchmarking suite). Such a framework would facilitate fair comparisons among algorithms and provide actionable insights.

We are not aware of any publicly available evaluation framework addressing this requirement. There have been some competitions on applying ML to networking data [5] but they *(i)* provide a set of already extracted packet fields or features, and *(ii)* does not provide direct access to packet captures, effectively constraining the possible algorithms.

Evaluating various algorithms on a variety of publicly available datasets is challenging due to the mismatch in classification granularities of algorithms and the granularities of the dataset. We cannot faithfully run a packet classification algorithm on a flow classification dataset and vise-versa. Further, interpreting the precision difference of algorithms is challenging due to multiple possible choices of metrics (e.g. accuracy vs. precision vs. recall), multiple datasets, multiple attacks *etc.*.

Lumen addresses these challenges in two ways. First, Lumen distinguishes datasets according to their classification granularity and the included attack types. Doing so allows Lumen to only evaluate algorithms faithfully and to analyze results considering the differences across datasets, effectively allowing better interpretation. Second, Lumen stores all results in a query-friendly format; and displays the most useful results in a compact manner (using a heatmap). Doing so Lumen does give not only actionable guidance to a non-expert but also provides adequate data for further analysis and refining.

4 IMPLEMENTATION

In this section, we describe Lumen's implementation details, including the collection of datasets.

4.1 Implementation Details

Framework We implement Lumen in Python (3.8) and evaluate it on a 4-machine cluster, with each machine having 64GB of memory. We use pypacker [7] to parse pcap files and memory-profiler [4] to profile the memory consumption of each module. We use various machine-learning libraries such as Pandas [25], Tensorflow [9], Sklearn [31], Ray [29], and Modin [32] for distributed Python processing.

Benchmarking suite: We selected 15 datasets from the 16 algorithms we surveyed. Each of these datasets is used at least once in a peer-reviewed network ML-based anomaly detection algorithm. Our benchmarking suite contains both packet-level classification datasets: Kitsune, AWID3, and IEEE IoT network intrusion dataset,

```
algorithm = [
        {
        "func": "Field Extract"
        "input": None,
        "output":"Packets"
        "param":['srcIP', 'dstIP',
                'TCPFlags', 'packetLength'],
        }
        {
        "func":"Groupby"
        "input":["Packets"],
        "output":"Grouped_packets",
        "flowid": ["srcIp"]
        }
        {
        "func":"TimeSlice"
        "input":["Grouped_packets"],
        "output":"Sliced_packets",
        "flowid": ["srcIp"]
        }
        {
        "func":"ApplyAggregates"
        "input": ["Sliced_packets"],
        "output": "Features"
        "list": ag_list,
        },
        {
        "func":"model"
        "model_type":"RandomForest"
        "input": None,
        "output":"clf1"
        },
        {
        "func":"train"
        "input":["clf1","Features"],
        "output":"save_path",
        }
]
```

Figure 4: For prototyping, a Lumen user only needs to fill a template that describes the corresponding logical pipeline. Then, Lumen take over the heavy lifting of compiling the algorithm to actual code, evaluating it on various datasets, and comparing it with state-of-the-art.

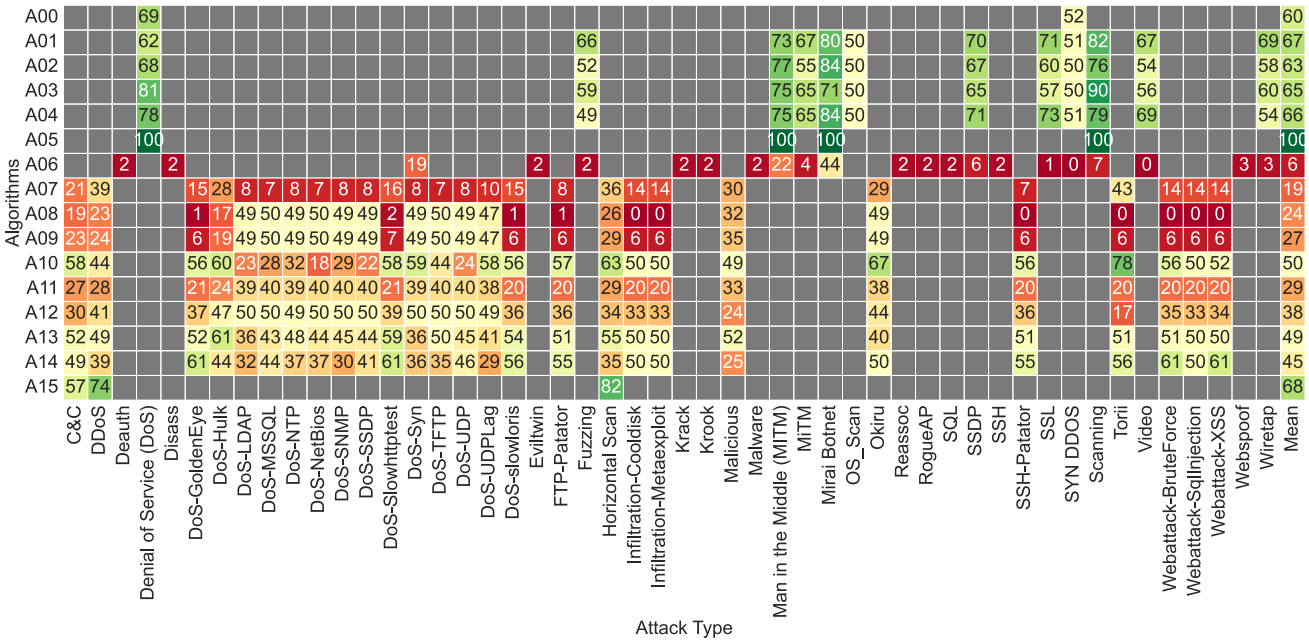


Figure 5: Heatmap illustrating the precision achieved by running the different algorithms on detecting particular attacks. To calculate the precision score of algorithm Y and an attack X, we use results from running algorithm Y on the subset of datasets that contain the attack X. Gray squares correspond to cases for which we did not have a dataset that contained the attack and on which we could faithfully run the algorithm. Certain algorithms (greener squares) are particularly good (higher score) at a subset of the attacks but are not accurate in others.

and connection-level classification data sets: CICIDS 2017, CICIDS 2019, and the CTU IoT dataset. Next, we explain how we preprocessed the different datasets before using them in Lumen.

- CICIDS 2017 [34]: This dataset includes two forms *(i)* packet captures (pcaps) with flow labels; and *(ii)* a CSV file with pre-extracted features. We found that the flow labels generated using CICFlowmeter (v3) tool in the dataset were incomplete and/or misleading (e.g., no AM/PM information with 12-hour times). Thus, we combine the pcaps and textual descriptions to recreate an accurate CSV format. We believe that our methodology is more accurate and fair to other algorithms. We then use Zeek to split large packet capture into corresponding flows and label them following the provided flow labeling schema.
- CICIDS 2019 [35]: We use the labeled ground-truth data to understand flow labeling schema and use Zeek to split the packet capture into corresponding flows.
- CTU [19]: We run our Zeek-flow extraction and matched our Zeek-flows with the labeled Zeek-flows provided in the dataset based on flow timestamps.
- Kitsune [27]: We have the ground-truth label that specifies each packet as malicious or benign.
- IEEE IoT [22]: We apply the provided Wireshark filters to split packets into malicious and benign packets.
- AWID3 [17]: We decrypt provided 802.11 packets using the provided keys and then apply the provided Wireshark filters to split packets into malicious and benign.

4.2 Implementation Challenges

We summarize some of the implementation challenges of Lumen and how we address them.

Large volume of traffic captures The captured traffic dataset can be huge, containing more than 100 million packets. Training an ML model in finite time with such a large number of packets is not trivial. Even open-source frameworks such as "nprint" fail with large pcap files. For instance, Segfault fails on a pcap file with 500,000 packets on a server-grade machine with 64Gb memory. Fortunately, we observe that many of the operations are inherently parallelizable. For example, extracting packet headers could be parallelized by splitting a pcap into smaller chunks, and building features could be parallelized by building each feature separately. To scale up our framework and speed up the pipeline, we integrated our framework with an open-source distributed parallel Python framework, "Ray" [29].

Unclear hyperparameters The lack of documentation of some of the implementation details in many of the algorithms made it difficult to replicate them. Indeed, the performance of an algorithm can be heavily influenced by the choice of hyperparameters, such as the number of leaf nodes in a random forest classifier or the loss

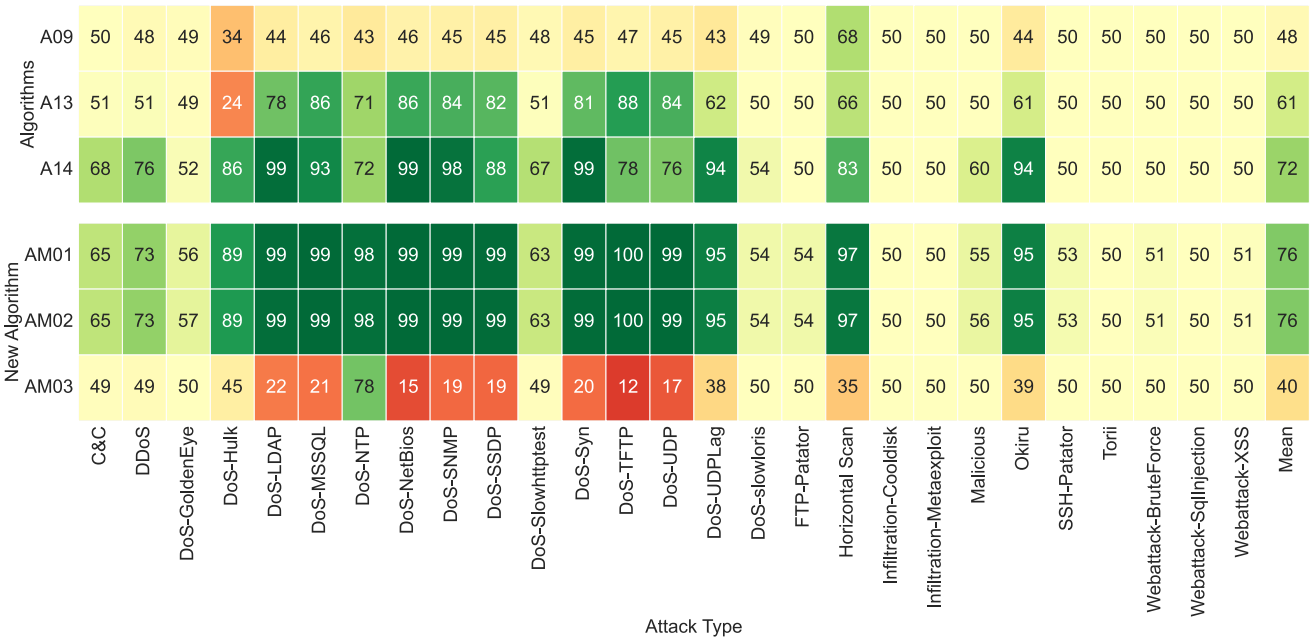


Figure 6: Lumen facilitates rapid prototyping and advancements. Straightforward heuristics such as merging datasets (rows A08, A09, A13, A14) and exploring the performance improvement from combining previously-used features and models (rows AM01, AM02, AM03) already improves the precision we measure across different attack types compared with the precision of existing algorithms evaluated in various datasets, shown in Fig.5. For this experiment, we only plot the connection classification algorithms and evaluate only on connection level datasets

function used to train a deep neural network. For those algorithms in which the hyperparameters were not specified, we use default parameters.

We have open-sourced our framework at https://github.com/rahul-anand/Lumen .

5 LUMEN IN PRACTICE

Our framework allows us to investigate the performance of various proposed algorithms across multiple datasets (§5.3). Concretely, we investigate the existence of an optimal algorithm for anomaly detection, the extent to which algorithms generalize across deployments, and the dependence of an algorithm's performance on the training dataset. Further, we demonstrate the potential of our framework in facilitating advancements in the field. Concretely, we leverage Lumen to improve the training of existing algorithms and even synthesize a new algorithm by combining pieces of previous work. We find that these heuristics can increase the precision of the algorithms by 4-27% (5.4).

5.1 Methodology

In this subsection, we summarize our evaluation dimensions:

- Algorithms (Table 2):We implement 16 algorithms using Lumen.
- Datasets (Table 3): Many of the datasets e.g. CICIDS include packet captures from multiple days containing different types of attacks. We treat each day's trace as a new dataset. We have ten connection-level classification datasets and five packet-level classification datasets.
- Training method: We use two methods: *(i)* same dataset, in which training and testing data are from the same dataset; and *(ii)* cross dataset, in which training and testing data are from distinct datasets.
- Granularity: We run and compare algorithms faithfully, i.e., connection-level classification algorithms are trained/tested against connection-level datasets and packet-level classification algorithms on packet-level datasets.
- Metrics: For each algorithm and training-testing combination pair, we compute its precision and recall score. The precision score denotes the number of times an algorithm classified the connection/packet as anomalous, and it was indeed anomalous. The recall score denotes the number of anomalous connection/packets identified correctly out of the total anomalous connection/packets.

5.2 Validating the correctness of Lumen

For most of the algorithms, we don't have access to the original source code, and many are not evaluated on the datasets that Lumen includes. The performance of an algorithm can vary drastically depending on the choice of hyperparameters, training/testing splits, etc. Moreover, no paper reports performance when training and testing data are from the same dataset. Thus, it is impossible to have a direct one-to-one comparison between Lumen and the original

Algorithm	Description
A00 [18]	ML DDoS
A01 [20]	nprint1: All
A02 [20]	nprint2: tcp + udp + ipv4
A03 [20]	nprint3: tcp + udp + ipv4 + payload
A04 [20]	nprint4: tcp +icmp + ipv4
A05 [11]	IDS smart_home
A06 [27]	Kitsune
A07 [40]	OCSVM
A08 [40]	Nystrom+ GMM
A09 [40]	Nystrom + OCSVM
A10 [24]	smartdet
A11 [15]	nokia
A12 [21]	early detection
A13 [28]	Bayesian
A14 [13]	Zeek
A15 [41]	IIoT
AM*	Modified Algorithms

Table 2: Algorithms

Data	Description
F0	CICIDS 2017, Tuesday
F1	CICIDS 2017, Wednesday
F2	CICIDS 2017, Thursday
F3	CICIDS 2019, 01-11
F4	CTU, 1-1
F5	CTU, 20-1
F6	CTU, 3-1
F7	CTU, 7-1
F8	CTU, 34-1
F9	CTU, 8-1
P0	IEEE IoT dataset
P1	Kitsune
P2	AWID3

Table 3: Datasets

Table 4: Algorithms and datasets used for evaluation

implementations for all tests. To validate the correctness of Lumen we follow two steps.

First, we validate that the features that Lumen calculates against match those that the original implementation or an alternative one calculates for a subset of our algorithms. For algorithms A01-A04 [20], we use the "nprint" (version 1.2.1) tool to generate features corresponding to each packet. We compare the features extracted by Lumen for algorithms A01-A04 with the "nprint" tool, and the features match exactly. Next, we directly feed these features into a random forest model. We find that our implementation of A06 (Kitsune) matches exactly with the author's provided implementation[1]. For A10 (smartdet) the authors have provided a script[2] to extract features. We find that Lumen's features match exactly, also in this case.

Second, we compare the Lumen-calculated accuracy scores against those reported, for some subset of the algorithms and datasets. We find that Lumen is very close to the reported in many of the tested cases, but not in all. We believe the variance is largely due to the selection of hyperparameters. For algorithm A10 [24], which is evaluated on dataset F1 (CICIDS 2017 DoS), authors report 99% precision; Lumen also achieves 99% precision for the same algorithm and dataset. For A14 [13], which is evaluated on a combined dataset of F4-F9 (CTU IoT dataset), authors report mean precision score of 99.9%; Lumen also achieves mean precision scores of 99.6%. For algorithm A07 [40] and datasets F0-F2 (CICIDS 2017), the authors report 78.6% AUC score; while Lumen achieves 66% AUC score. For algorithm A07 [40] and datasets F4-F9 the authors report AUC of 75% ; while Lumen achieves an AUC score of 49.2%.

5.3 Observing state-of-the-art

Q1: Which algorithm has the best performance across all training and testing scenarios?

To answer this question, we first find the maximum precision/recall score that was achieved by any algorithm for each pair of training-testing datasets. Next, for each training-testing pair, we calculate the difference between the maximum precision/recall (*i.e.,* by the best algorithm for the particular pair) and the precision/recall score achieved by each algorithm.

We plot the precision and recall differences grouped by the algorithm in Figure 7a. An optimal algorithm would have been a line at "Y==0" as it would have achieved the best precision and recall on all training-testing datasets. Algorithms A05 and A06 may seem like good candidates for the best algorithm, but they can run only on a small number of datasets due to their classification granularity. We observe that there is no single best algorithm that can achieve the highest precision or recall across the board. We see that the algorithms A1-A4 are generally good for packet classification as their precision difference from optimal algorithm is close to zero. We see a massive variation in the accuracy of connection-level-classification algorithms, partially due to more connection classification datasets in our Framework.

> **Observation 1** There isn't a single algorithm with the highest precision or highest recall score for all training/testing scenarios (Figure 7).

Q2: How robust are proposed algorithms under different datasets?

To answer this question, we run each algorithm under two cases. In the first case, we split a single dataset to generate the training and testing datasets. We repeat the process with multiple datasets. In the second case, we use distinct datasets for training and testing. Again we run for multiple pairs of datasets. In both cases, we faithfully evaluate algorithms: we only run algorithms with datasets of the same classification granularity to avoid affecting their precision.

Figure 8 illustrates the precision and recall score across algorithms for the first case (*i.e.,* test and train on single datasets) while Figure 9 for the second case (*i.e.,* test and train on the different datasets). As expected, the precision and recall scores in the former case are much higher overall. Yet, even in this case, we observe that for 8 out of 16 algorithms, there is at least one dataset where the algorithm precision score is lower than 20%. Clearly, those algorithms do not generalize well.

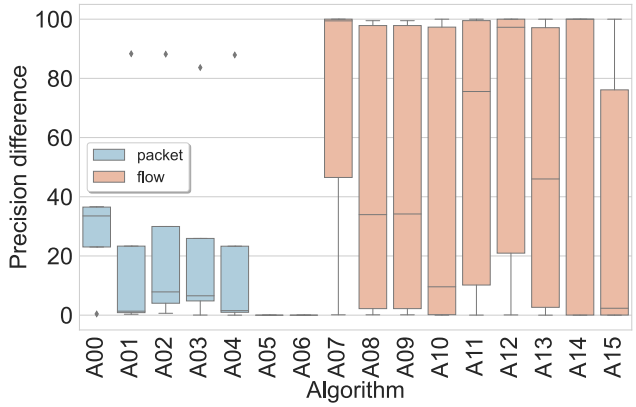
> **Observation 2** When trained and tested on data from *same* source: the precision of 8/16 algorithms and recall of 4/16 algorithms drops below "20"% for at least one data set. When trained and tested on data from *different* sources: the precision and recall of 16 of the 16 algorithms drops below "20"% for at least one data set.

Q3: Does the selection of the training dataset affect the precision of a given algorithm?

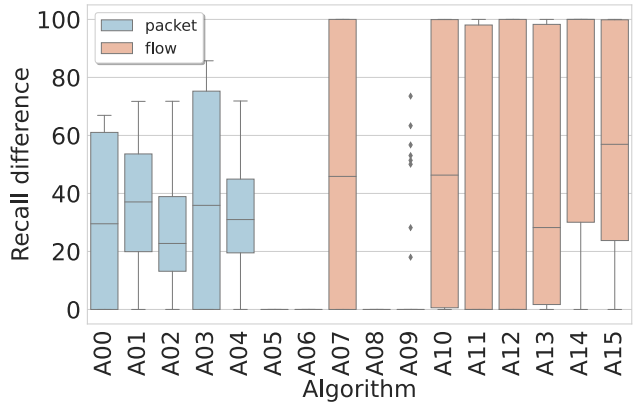
To answer this question, we plot the median precision and recall achieved by all algorithms on a particular combination of a training dataset (X-axis) on a test dataset (Y-axis) in Fig.10. Naturally, the diagonal, corresponding to training and testing on data from the same dataset, results in better median precision and recall. We also observe that the precision and recall scores are asymmetric for

[1] https://github.com/ymirsky/Kitsune-pyg
[2] https://github.com/irini90/pcap_preprocessing

[3] Algorithm A05 can only run on a single dataset, so we cannot do the evaluations with distinct training and testing datasets.



(a) Precision Scores

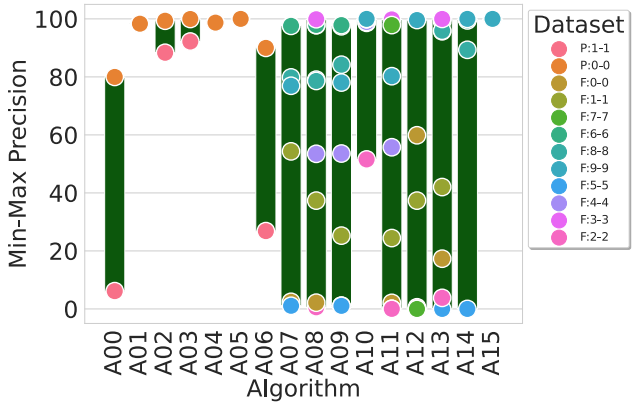


(b) Recall Scores

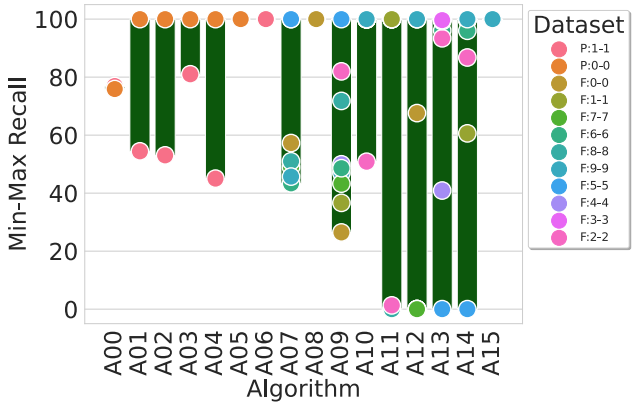
Figure 7: We plot the absolute difference between the precision and recall of the best-performing algorithm on a given combination of training and testing datasets and each algorithm on the same combination. Lower values indicate precision closer to the best algorithm. Since there is no single algorithm with always zero difference, there is no algorithm that is better than all others across training and testing dataset pairs. We group algorithms by their classification granularity (per-packet vs. per-flow).

a particular training-testing pair, indicating that certain datasets are better for training than others (greener columns), and some datasets are more challenging (more red rows). Finally, we also see some particularly interesting datasets, such as dataset "F5" (Torii attack from CTU). We observe that none of the training datasets are able to generalize to dataset F5 (in terms of precision scores), but a model trained on dataset F5 is able to generalize to others.

Observation 3 Strategically selecting the training dataset leads to a more accurate anomaly detection model.



(a) The precision scores of some (8 out of 16) algorithms drop by more than 20%, at least for one dataset, even when they are trained and tested on the same dataset. This denotes that the pseudo-code designs themselves don't generalize for some algorithms on particular datasets.



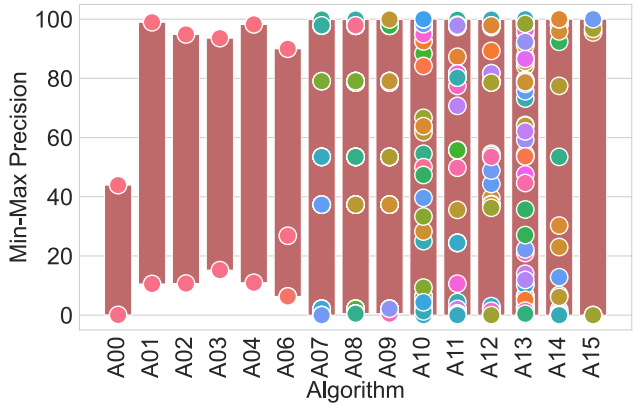
(b) Recall scores of some (4 out of 16) algorithms drop by more than 20%, at least for one dataset, even when they are trained and tested on the same dataset.

Figure 8: We plot each algorithm's precision and recall score when they are trained and tested on the same dataset.

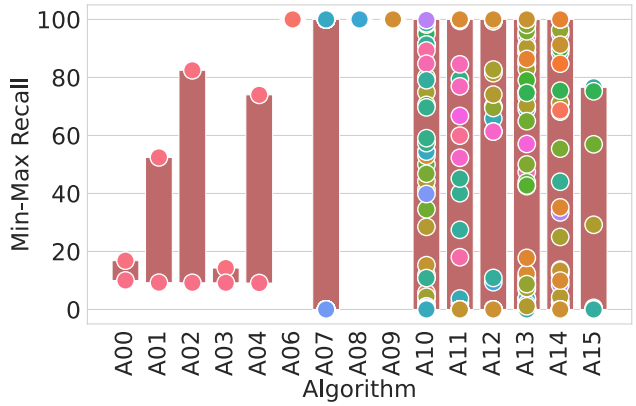
Q4: Is there an optimal algorithm for each attack?

While we have observed precision differences among algorithms and pairs of training and testing, we have, so far, little understanding of the reason. One hypothesis is that certain models are especially good for certain attacks. To test this hypothesis, we focus on the per-attack precision of various algorithms. Figure 5 illustrates the precision of each algorithm in identifying the packets of a particular attack only. Thus, to calculate the value of a square corresponding to an algorithm Y and an attack X, we use results from running algorithm Y on the subset of datasets that contain the attack X.

We observe that certain attacks are more effectively identified by a few algorithms. For example, DoS attacks are best identified by

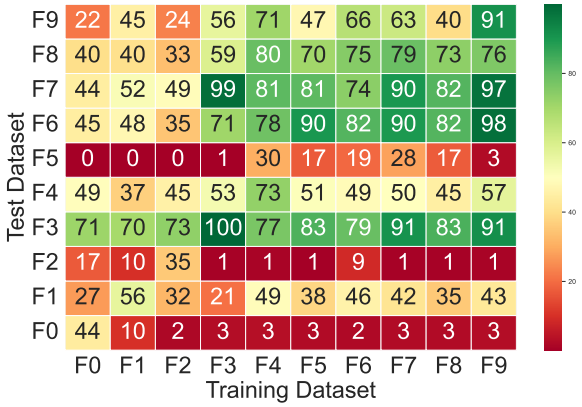


(a) Precision Scores

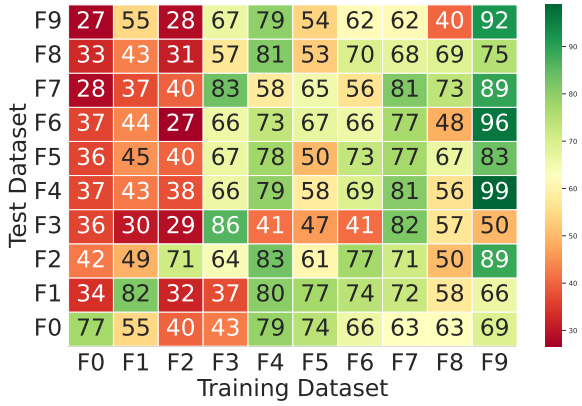


(b) Recall Scores

Figure 9: We plot each algorithm's[3] precision and recall score when they are trained and tested on different datasets. For all algorithms, the precision and recall score drops by more than 80% when trained on one and tested on other datasets.



(a) Precision Scores



(b) Recall Scores

Figure 10: We plot median precision and recall across algorithms per combination of training (X-axis) and testing (Y-axis) datasets. Precision and recall scores are asymmetric for a particular training-testing pair. For instance, training on F5 and testing on F6 results in a 90% precision score, while training on F6 and testing on F5 results in only a 19% precision score.

A15 (smartdet [24]) because the algorithm selects features such as rate of change of TCP flags, entropy of source ports, and standard deviation of IP length, which are naturally expected to change during a DoS attack. We also observe that 802.11 Wireless attacks (Death, Eviltwin, etc.) from the AWID3 dataset are hard to detect for any algorithm as the 802.11 packets do not contain IP headers, and as such only A06 can run on it, and that too with very low precision scores.

Observation 4 The precision of a given algorithm is highly affected by the attack contained in the training/testing datasets.

5.4 Improving state-of-the-art

Having a clearer understanding of the state-of-the-art algorithms, we test the ability of our framework to guide improvements. To this end, we experiment with two intuitive techniques. First, we aim to improve the training of different models by merging datasets. Second, we aim to improve the models themselves by combining ideas from existing approaches. We illustrate our results in Fig.6.

First, we experiment with the idea of training algorithms on a combination of all training datasets without increasing the size of the training set. Thus, for each classification granularity, we generate a new dataset by concatenating 10% of data from each dataset. We use the same approach to generate a testing dataset. Finally, we compute the algorithm's precision score on the common testing

dataset. This intuitive approach already improves the precision scores by 12-27%, as we observe in the first three rows of Fig.6

Second, we experiment with the idea of mixing features from existing algorithms. Concretely, we do a greedy brute-force search over the space of used features and ML models. We then evaluate each of the candidate algorithms using our benchmarking suite. To reduce the search space, we complement existing pipelines with ML techniques that typically improve the performance of classifiers, such as data normalization, removing correlated features, and autoML. This simple brute-force search already reveals algorithms with higher precision scores than those of the previous work. Concretely, we are able to find an algorithm with a 4% better average precision score than the proposed prior works.

Observation 5 Simple heuristics such as training on merged datasets and combining algorithms, improve the precision score of the algorithms by 4-27%.

6 DISCUSSION AND FUTURE WORK

Better new algorithm generation: A new algorithm can be automatically constructed by combining the operations that we have implemented in our framework. Black-box optimization techniques such as Bayesian optimization could be used to find a new algorithm in a more systematic way.

Automatic hyper-parameter tuning with Lumen: Techniques from grid-search or bayesian optimization could be used to automatically find the best hyper-parameters for an ML model. Lumen could be easily integrated with various automatic hyper-parameter tuning frameworks such as Ray-tune [29], Optuna [10], HyperOpt [14], etc.

Extending the framework to other ML tasks outside of IoT anomaly detection: Although the focus of this paper is on anomaly detection algorithms for IoT devices, our proposed framework is, in fact, more general. Our framework can be used to develop and evaluate any ML algorithm on network data. For example, if we were to extend our framework to do ML-based device classification, we would only need to add a new dataset to our framework, and the rest of the functions/modules would be used directly.

Understanding relevant features for each attack type: Lumen can also be used to understand the relevant features for each attack type or deployment.

7 CONCLUSION

In this paper, we design and implement a framework that allows for rapid prototyping, efficient evaluation, and fair comparison among multiple previously proposed algorithms for ML-based network-layer IoT anomaly detection. Paired with a benchmarking suite, our framework allows us to gain insights regarding the performance and generality of various algorithms. Moreover, our framework allows us to invent new training strategies and algorithms that draw from previously suggested ones and can already improve the precision in various testing scenarios. We have open-sourced Lumen to help operators and also inspire further innovation.

8 ACKNOWLEDGEMENT

We thank our shepherd, Zahaib Akhtar, for his help with the final version of this paper, as well as the anonymous reviewers for their detailed comments. This work was supported in part by NSF award CNS-1564009 and C3.ai DTI research award; the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA; and by the U.S. Army Research Office and the U.S. Army Futures Command under Contract No. W911NF-20-D-0002. The content of the information does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred. We acknowledge the support of C3.ai and Microsoft for our research.

REFERENCES

[1] [n.d.]. Device Functional Role ID via Machine Learning and Network Traffic Analysis. https://github.com/IQTLabs/NetworkML.
[2] [n.d.]. HOW TO STOP YOUR SMART TV FROM SPYING ON YOU. https://www.mckinsey.com/~/media/mckinsey/business%20functions/mckinsey%20digital/our%20insights/iot%20value%20set%20to%20accelerate%20through%202030%20where%20and%20how%20to%20capture%20it/the-internet-of-things-catching-up-to-an-accelerating-opportunity-final.pdf.
[3] [n.d.]. IoT vendors ignore basic security best practices, CITL research finds. https://www.csoonline.com/article/3436877/iot-vendors-ignore-basic-security-best-practices-citl-research-finds.html.
[4] [n.d.]. Memory Profiler. https://pypi.org/project/memory-profiler/.
[5] [n.d.]. NetML-Competition2020. https://github.com/ACANETS/NetML-Competition2020.
[6] [n.d.]. PDML - Packet Description Markup Language. https://wiki.wireshark.org/PDML.
[7] [n.d.]. pypacker. https://gitlab.com/mike01/pypacker.
[8] [n.d.]. Why Third-Party Vendors Are Responsible for the IoT Security Problem. https://www.securicon.com/why-third-party-vendors-are-responsible-for-the-iot-security-problem/.
[9] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 265–283.
[10] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2623–2631.
[11] Eirini Anthi, Lowri Williams, Małgorzata Słowińska, George Theodorakopoulos, and Pete Burnap. 2019. A supervised intrusion detection system for smart home IoT devices. *IEEE Internet of Things Journal* 6, 5 (2019), 9042–9053.
[12] Zied Aouini and Adrian Pekar. 2022. NFStream: A flexible network data analysis framework. *Computer Networks* (2022), 108719.
[13] Michael Austin. 2021. *IoT Malicious Traffic Classification Using Machine Learning*. West Virginia University.
[14] James Bergstra, Dan Yamins, David D Cox, et al. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, Vol. 13. Citeseer, 20.
[15] Randeep Bhatia, Steven Benno, Jairo Esteban, TV Lakshman, and John Grogan. 2019. Unsupervised machine learning for network-centric anomaly detection in iot. In *Proceedings of the 3rd acm conext workshop on big data, machine learning and artificial intelligence for data communication networks*. 42–48.
[16] Anil Chacko and Thaier Hayajneh. 2018. Security and privacy issues with IoT in healthcare. *EAI Endorsed Transactions on Pervasive Health and Technology* 4, 14 (2018).
[17] Efstratios Chatzoglou, Georgios Kambourakis, and Constantinos Kolias. 2021. Empirical evaluation of attacks against IEEE 802.11 enterprise networks: The AWID3 dataset. *IEEE Access* 9 (2021), 34188–34205.
[18] Rohan Doshi, Noah Apthorpe, and Nick Feamster. 2018. Machine learning ddos detection for consumer internet of things devices. In *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 29–35.
[19] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. 2014. An empirical comparison of botnet detection methods. *computers & security* 45 (2014), 100–123.
[20] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. 2021. New directions in automated traffic analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3366–3383.

[21] Ren-Hung Hwang, Min-Chun Peng, Chien-Wei Huang, Po-Ching Lin, and Van-Linh Nguyen. 2020. An unsupervised deep learning model for early network traffic anomaly detection. *IEEE Access* 8 (2020), 30387–30399.

[22] K Hyunjae, Dong Hyun Ahn, Gyung Min Lee, Jeong Do Yoo, Kyung Ho Park, and HK Kim. 2019. IoT network intrusion dataset. *IEEE Dataport* (2019).

[23] Kenneth Kimani, Vitalice Oduol, and Kibet Langat. 2019. Cyber security challenges for IoT-based smart grid networks. *International Journal of Critical Infrastructure Protection* 25 (2019), 36–49.

[24] Francisco Sales de Lima Filho, Frederico AF Silveira, Agostinho de Medeiros Brito Junior, Genoveva Vargas-Solar, and Luiz F Silveira. 2019. Smart detection: an online approach for DoS/DDoS attack detection using machine learning. *Security and Communication Networks* 2019 (2019).

[25] Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Vol. 445. Austin, TX, 51–56.

[26] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. 2018. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing* 17, 3 (2018), 12–22.

[27] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089* (2018).

[28] Andrew W Moore and Denis Zuev. 2005. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. 50–60.

[29] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 561–577.

[30] Nour Moustafa, Benjamin Turnbull, and Kim-Kwang Raymond Choo. 2018. An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things. *IEEE Internet of Things Journal* 6, 3 (2018), 4815–4830.

[31] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[32] Devin Petersohn, Stephen Macke, Doris Xin, William Ma, Doris Lee, Xiangxi Mo, Joseph E Gonzalez, Joseph M Hellerstein, Anthony D Joseph, and Aditya Parameswaran. 2020. Towards scalable dataframe systems. *arXiv preprint arXiv:2001.00888* (2020).

[33] Paul Schmitt, Francesco Bronzino, Renata Teixeira, Tithi Chattopadhyay, and Nick Feamster. 2018. Enhancing transparency: Internet video quality inference from network traffic. TPRC.

[34] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* 1 (2018), 108–116.

[35] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani. 2019. Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 1–8.

[36] Saleh Soltan, Prateek Mittal, and H Vincent Poor. 2018. {BlackIoT}:{IoT} Botnet of High Wattage Devices Can Disrupt the Power Grid. In *27th USENIX Security Symposium (USENIX Security 18)*. 15–32.

[37] Natalija Vlajic and Daiwei Zhou. 2018. IoT as a land of opportunity for DDoS hackers. *Computer* 51, 7 (2018), 26–34.

[38] Christos Xenofontos, Ioannis Zografopoulos, Charalambos Konstantinou, Alireza Jolfaei, Muhammad Khurram Khan, and Kim-Kwang Raymond Choo. 2021. Consumer, commercial and industrial iot (in) security: attack taxonomy and case studies. *IEEE Internet of Things Journal* (2021).

[39] Kun Yang, Samory Kpotufe, and Nick Feamster. 2020. A Comparative Study of Network Traffic Representations for Novelty Detection. *arXiv preprint arXiv:2006.16993* (2020).

[40] Kun Yang, Samory Kpotufe, and Nick Feamster. 2021. An Efficient One-Class SVM for Anomaly Detection in the Internet of Things. *arXiv preprint arXiv:2104.11146* (2021).

[41] Maede Zolanvari, Marcio A Teixeira, Lav Gupta, Khaled M Khan, and Raj Jain. 2019. Machine learning-based network vulnerability analysis of industrial Internet of Things. *IEEE Internet of Things Journal* 6, 4 (2019), 6822–6834.