Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 1185–1189
DOI: 10.15439/2023F3594
ISSN 2300-5963 ACSIS, Vol. 35

Comparing Performance of Machine Learning Tools across Computing Platforms

Pedro Vicente[*†], Pedro M. Santos[*†], Barikisu Asulba[§†], Nuno Martins[‡], Joana Sousa[‡], Luis Almeida[§†]
[*] *Instituto Superior de Engenharia do Porto (ISEP)*
[§] *Universidade do Porto – Faculdade de Engenharia*
[†] *CISTER Research Center in Real-Time & Embedded Computing Systems, Portugal*
[‡] *NOS Inovação, Portugal*
{1180558, pss}@isep.ipp.pt, up202103270@fe.up.pt, {nuno.mmartins, joana.sousa}@nos.pt, lda@fe.up.pt

***Abstract*—Embedded systems (ES) are wide-spread in our world and responsible for many critical systems. More recently, machine learning (ML) tools have become a well-established solution for data-intensive tasks, but their application in embedded systems is still gaining traction and their real-time performance is often unclear. We provide a (non-extensive) review of the ML tools that may be suited for deployment in ES, from which we selected two representative tools – the well-established Python-based Scikit-Learn, and the interoperability-oriented ONNX Runtime – to compare their response time. Using archetypal datasets and four pre-trained ML models, we measure the prediction time for each sample, for each model, in Scikit-Learn and ONNX Runtime in a standard desktop (to compare performance of the tools in the same platform), and for ONNX Runtime in a representative ES, a Raspberry Pi v4 (to compare performance of the same tool across platforms). We report that ONNX considerably improves over Scikit-Learn, and experiences a negligible performance degradation when ported to the RPi.**

***Index Terms*—Machine Learning, Embedded Systems, Prediction Time, Scikit-Learn, ONNX Runtime**

I. Introduction

Artificial intelligence (AI) and machine learning (ML) have grown dramatically in recent years, to the point where AI & ML is becoming a core technological component in many modern systems. In turn, embedded systems (ES) are a well-established technology that has enjoyed widespread use in our world for decades now, inconspicuously ensuring the efficient execution of a plethora of everyday operations. Many applications of ES are critical and time-sensitive ones [1]; for example, the timely detection and mitigation of cyberattacks, that is crucial for the integrity and dependability of many modern-world digital systems (e.g., banking sector).

The use of ML in embedded systems has garnered substantial interest, with the topic often being referred to as *TinyML*. The challenge is that embedded systems are typically resource-constrained platforms (ranging from micro-controllers to ARM or small-scale x86 platforms) and, while there is a plethora of ML libraries, not all provide the small memory footprint and stand-alone operation (i.e., sufficiently stripped-down from external dependencies) necessary for operation in embedded systems. Furthermore, a common strategy is to carry out training at the cloud (due to the higher processing capabilities available), whereas the embedded device only performs prediction. This raises the need for interoperability, as possibly the ML tool used for training can be different than the one available at the embedded device. Finally, characterization of response time is important to design real-time systems. Reports of execution time and/or speed-up against baselines can be found (e.g., [2], [3]), but typically for single models and not considering potential response time variability.

A noteworthy category of solutions are intermediate description languages, such as *Open Neural Network Exchange* (ONNX), and associated runtime environments (RTE), notably *ONNX Runtime* and *Tensorflow Lite*. Intermediate description languages describe a (trained) ML model using a (small) set of operators that the RTE is able to execute. This reduces computational requirements as it suffices that the RTE implements that set of operators to produce predictions from a given model. Downsides are that training may not be available and that the set of models at disposal may be limited.

In this work we report the performance of two selected libraries, *Scikit-Learn* and *ONNX Runtime*, in two platforms: a standard desktop and an archetypal embedded system, a Raspberry Pi v4. We deploy four one-class ML models – Isolation Forest (iForest), Local Outlier Factor (LOF), One Class Support Vector Machine (OC-SVM) and Stochastic Gradient Descent OC-SVM (SGD-SVM) –, that were pre-trained with network traffic data sets (legitimate and malicious) to detect cyberattack-related traffic. We show that ONNX Runtime can offer a speed-up of at least $\approx$ 14x with respect to Scikit-Learn for most models when both are executed in the desktop, and that ONNX Runtime running in the Raspberry Pi produces speed-ups of at least $\approx$ 8x against Scikit-Learn running in the desktop.

The structure of this paper is as follows. Section II portrays a motivating use-case and relevant ML models. An overview on ML for ES is provided in Section III. Section IV reports response times for selected ML libraries and computing platforms. Section V draws final remarks.

II. Motivating Use-Case and Selected ML Tools

A. Cybersecurity Use-Case

Cybersecurity is a domain of notable technological and societal impact in the modern world. The exposure surface for cyberattacks, and for recruiting devices that can be commandeered to participate in those attacks, increases everyday

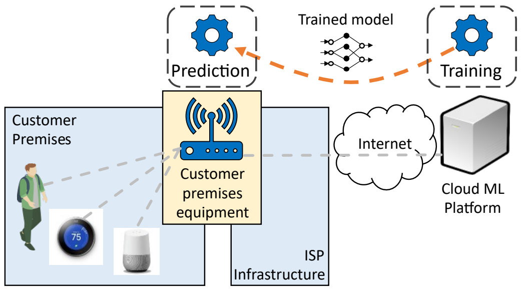

Fig. 1: Architecture of a cloud-edge system.

as the number of low-security IoT devices grows. This has been a driver for the increase of Distributed Denial of Service (DDoS) attacks, that aim at disrupting the servers of high-profile online services (e.g., Amazon, Google or Netflix) by having a very large number of infected devices (typically vulnerable IoT devices) issuing dummy requests to those servers. Internet Service Providers (ISP), that enable Internet service at customer premises through a Customer-Premises Equipment (CPE), are interested in mitigating the involvement of their customers' devices in cyberattacks through the use of Intrusion Detection Systems (IDS). The fastest response time is attained by deploying the IDS the closest possible to the targeted (or involved) nodes; for the ISP, this is the CPE.

Machine learning, whose successful application to cybersecurity is well documented [4]–[6], can help addressing this issue by learning network traffic patterns that are legitimate, and apply that knowledge to identify anomalous patterns that may concern malicious traffic. However, the CPE is often an embedded system with relatively few computational resources; while it can perform model prediction, it lacks the power to perform training, that ends up taking place in the cloud. The question arises of how to transfer models trained in the cloud, often with a state-of-the-art ML library, to the embedded system, that often will support only a limited set of ML libraries. Figure 1 presents the architecture of a cloud/edge system, and how can an ML-based IDS be deployed by leveraging the resource-rich cloud for training and transferring trained models to the resource-constrained CPE. Additional details on this use-case can be found in [7].

B. *Datasets & ML Tools*

To enable the presented use-case, we prepared datasets of network traffic (both legitimate and malicious) from publicly available sources, and trained four models to produce a ML mechanism that detect anomalous (potentially malicious) traffic. The details are described in [6]. We focus on One-Class (OC) models, i.e., models trained with samples of a single class to create a boundary around these, against which outliers can be detected. This semi-supervised approach allows the models to learn the regular (legitimate) traffic at a customer's network, and report anomalies that can potentially reveal themselves to be malicious traffic. All models were trained using Scikit-Learn [8], a free Python library that enjoys widespread use in the ML community.

The four selected models are reviewed next for convenience:

Isolation forest [9]: an unsupervised mechanism based on decision trees. It leverages the assumption that an anomalous sample requires less partitioning steps to be isolated. Thus, an isolation forest work by recursively generating partitions, by randomly splitting an attribute's value between the minimum and maximum values allowed for that attribute, until a target sample is contained in its own partition. Anomalies will require less partitions to be isolated.

Local Outlier Factor (LOF) [10]: LOF identifies local outliers by measuring the deviation of the density of a data point to its neighbors. The k-Nearest Neighbors is used to compute the reachability distance and local reachability density of each data point. The associated LOF score is calculated as the ratio of its local reachability density to the densities of its k-nearest neighbors. Points with high LOF scores are considered outliers. The value k (number of nearest neighbors) must be chosen carefully to avoid overfitting or underfitting.

One-Class Support Vector Machines [11]: traditional Support Vector Machines (SVM) select a decision boundary for which the margin between data points of different classes is maximized. Other interpretation is that SVMs maximize the distance between the convex hulls of points belonging to each class. One-Class SVM (OC-SVM) applies the same boundary-based mechanism for semi-supervised learning. It uses a hypervolume to encompass all of the instances; points outside the hypervolume are classified as anomalies.

Stochastic Gradient Descent [One Class] SVM (SGD-SVM) [12]: an online linear version of One-Class SVM, using a stochastic gradient descent (SDG). SDG algorithms are suited for applications where the number of data points and the problem dimensionality are both very large.

III. Overview of ML Tools for Embedded Systems

We review (not exhaustively) ML libraries targetting embedded systems and tools for interoperability and transpilation.

A. *ML Libraries for Embedded Systems*

TensorFlow (TF) [1] is an open-source library for AI/ML, composed of datasets and pre-trained models developed and released by the Tensorflow Community. Colaboratory (Colab) for instance, is a free Jupyter notebook environment and runs in the cloud so the user doesn't need to setup anything in his local machine. This library is supported in Haskell, C#, Julia, R, Ruby, Scala and Javascript.

Armadillo [2] is a library in C++ for linear algebra and scientific computing. It can use Open Multi-Processing (OpenMP), a free easy-to-use library for parallel computing.

mlpack [3] is a C++ ML library focused in providing fast and extensible implementations of ML models. This library is the combination of *Armadillo*, *ensmallen*, a library for numerical optimization and *cereal*, a serialization library.

[1]https://www.tensorflow.org/ (Note: all links last accessed on 2023-07-31)
[2]https://arma.sourceforge.net/
[3]https://mlpack.org/

Shogun [4] is an open-source library in C++ for machine learning development. It provides interfaces for C++, Python, Octave, R, Java, Lua, C#, Ruby and implements all the standard ML algorithms and some advanced as well. It is available for most operating systems.

SHARK [5] is an open-source machine learning library implemented in C++. It provides neural networks, kernel-based learning algorithms, linear and nonlinear optimization methods and is available for the most common operating systems.

A notable mention also goes to **CAFFE** [6], that focus on deep learning, thus supporting mostly neural networks (e.g., CNN, RCNN, LSTM).

There are also efforts focusing on deploying specific ML models in resource-scarce devices. The authors of [13] present *ProtoNN*, an algorithm that replicates k-Nearest Neighbor (k-NN) but has several orders lower storage and prediction complexity, and *ProtoNN* models can be deployed in very scarce plaforms (e.g. an Arduino UNO with 2kB RAM). The authors of [3] presents *SeeDot*, a domain-specific language to express ML inference algorithms and a compiler that compiles SeeDot programs to fixed-point code that can efficiently run on constrained IoT devices. In [2] *CMSIS-NN* is presented, which is essentially efficient kernels to maximize the performance and minimize memory footprint of neural network applications on Arm Cortex-M processors.

B. Interoperability of ML models

The following options, rather than tools, are standards to provide a common description of ML models, therefore enabling porting between libraries.

Open Neural Network Exchange (ONNX) [7] is an open specification with the following components: a definition of an extensible computation graph model; definitions of standard data types; and definitions of built-in operators. The first two make up the ONNX Intermediate Representation (or IR). In ONNX IR, each computation dataflow graph is structured as a list of nodes that form an acyclic graph. Each node is a call to an operator, and they have one or more inputs and outputs. Built-in operators are divided into a set of primitive operators and functions (the latter being, essentially, sub-graphs using primitive operators and/or other functions). Operators are implemented externally to the graph, but the set of built-in operators is portable across frameworks. Every framework supporting ONNX will provide implementations of these operators on the applicable data types. ONNX is compatible with at least 29 frameworks and converters and 30 inference runtimes.

Predictive Model Markup Language (PMML) [8] is a document format based on the Extensible Markup Language (XML) that can be used to described machine learning algorithms. It enables ML model porting between existing supporting libraries; these exist for C++, such as *cPMML* [9], and for Python, notably with the Scikit-Learn library *sklearn2pmml* [10], among others.

C. Transpilers

Transpilers translate a source code into a language different than the original one. The resulting code is described natively in the target language.

Sklearn-porter [11] is a Python library specifically developed to transpile ML models built with Scikit-Learn to other programming languages such as C, GO and JavaScript.

Model 2 Code Generator (m2cgen) [12] is a free, open-source library mainly developed in Python, that transpiles trained statistical models (trained, e.g., with Scikit-Learn or *lightning* libraries) into a native code for at least 16 different programming languages (R, Visual Basic, Haskell, C#, etc.).

D. Runtime Environments

A third dimension discussed here are tools that offer runtime environments (or simply runtime). Some of the aforementioned ML libraries leverage mechanisms for intermediate model representation that can be compiled or interpreted by a runtime environment. This solution avoids the need to deploy the entire library at the target device, thus resulting in a lightweight version of the initial library.

ONNX Runtime [13] is a cross-platform machine-learning model accelerator, used to deploy ONNX format models into production. It is meant to enable acceleration of machine learning inferencing across a variety of target hardware.

Tensorflow Lite [14] is a TF-variant tailored for resource-constrained systems that also uses a runtime. Using Tensorflow Lite, the target devices do not require the full TF library installation, but solely the *tflite_runtime* to perform inference. This tool eases the computational requirements of the target system, but its accuracy can be compromised if it uses operations not supported by the Tensorflow Lite. A recent paper reports TensorFlow Lite Micro [14], that adopts an interpreter-based approach to address ML efficiency and fragmentation in ES.

IV. Prediction Time Comparison of Selected Tools

A. Selected Tools & Experimental Setup

We have picked ONNX Runtime as the target ML tool to evaluate, and Scikit-Learn as the baseline reference. The option for Scikit-Learn was straightforward, as it is one of the most widely-used ML tools. It is also the tool used to train the models used in these measurements. As for the tools for deployment in embedded systems, we opted for ONNX Runtime based on a mix of our own requirements (that, when crossed against the available documentation, lead us to eliminate the remaining candidate tools), and impressions

[4]https://github.com/Kolkir/mlcpp/tree/master/classification_shogun
[5]https://www.shark-ml.org/
[6]https://caffe.berkeleyvision.org/
[7]https://onnx.ai/about.html
[8]https://dmg.org/pmml/v4-1/GeneralStructure.html
[9]https://amadeusitgroup.github.io/cPMML/
[10]https://github.com/jpmml/sklearn2pmml
[11]https://github.com/nok/sklearn-porter
[12]https://github.com/BayesWitnesses/m2cgen
[13]https://onnxruntime.ai/
[14]https://www.tensorflow.org/lite

TABLE I: Dataset descriptions.

Dataset	Traffic type	# samples	# Features
IOT23 [15]	IoT devices	487	26
Botnet [16]	Data theft	196	26

TABLE II: Selected platforms.

	Desktop	Raspberry Pi
Number of cores	4	4
Frequency utilized	2.00 GHz	600.00 MHz
RAM memory	9.64 GB	1.91 GB
Operating System	Ubuntu 20.04.6 LTS	Debian GNU/Linux 11
Python version	3.8.10	3.9.2
ONNX version	1.13.1	N/A
ONNXRuntime	1.14.1	1.14.1

acquired from experimenting with the other high-potential candidates. We lay down next the authors' impressions of the reviewed tools; this should not be interpreted, in any way, as a methodical and criterious analysis of these tools.

ML libraries: *Tensorflow* proved to be a collection of disperse, pre-trained models, making it hard to train new models from scratch. *Armadillo*, *mlpack*, *Shogun*, *SHARK* and *Caffe*, despite being described in C/C++, do not seem tailored for deployment in resource-constrained devices.

Interoperability: ONNX provides a clear and well document specification of how to convert models between tools, with extensive software support. PMML enables model porting between supporting libraries but, as aforementioned, we found no library to be a suitable candidate.

Transpilers: *sklearn-porter* is still under development and the range of models that can be transpiled to C is small (SVM and Decision Trees/Random Forest). Regarding *m2cgen*, even though transpiled models were able to perform closely to the original model, the tool offers very little documentation, making it hard to interpret the tool's output or even understand how the transpilation process actually occurs.

Runtimes: *ONNX Runtime* showed up as the best option. *TensorFlow Lite* was not explored, as usage of standard *TensorFlow* was also not straightforward.

Table I describes the data sets used in this performance analysis; more details in [6]. Table II presents the characteristics of the selected computing platforms. The models were converted to the ONNX specification using the *sklearn-onnx* library. A variant named *ONNX Runtime Optimized*, that optimizes the ONNX graphs describing the models, was also evaluated. Model accuracy obtained with ONNX Runtime and its Optimized variant was similar to that of Scikit-Learn.

B. Results

Figure 2 presents the average prediction time (over all input samples) of the four ML models across the three tools in the desktop equipment. Presented values are the average time of prediction for each new sample. We observe that ONNX produces an acceleration for most models, notably of $\approx$ 16x for Isolation Forest, $\approx$ 14x for OC-SVM, and $\approx$ 49x for SGD-SVM. In all this cases, the performance of the ONNX Runtime and its Optimized version do not differ substantially from each other. The same is not true, however, for the Local Outlier Factor (LOF), as shown in Figure 2 (top-right). We observed

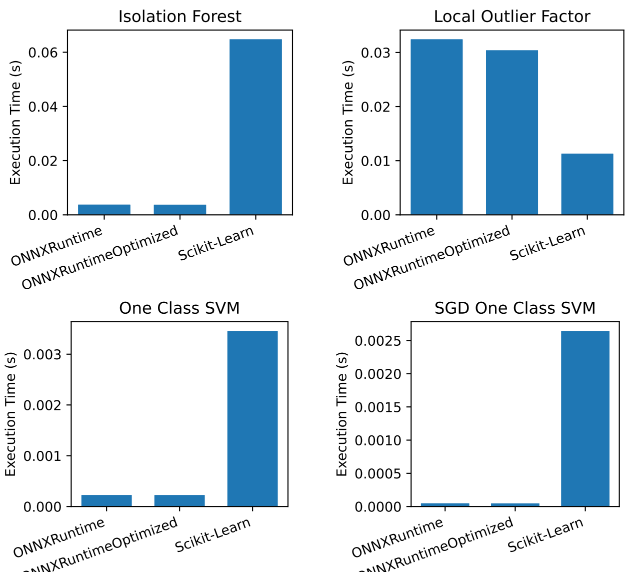

Fig. 2: Average prediction time per ML model on Desktop.

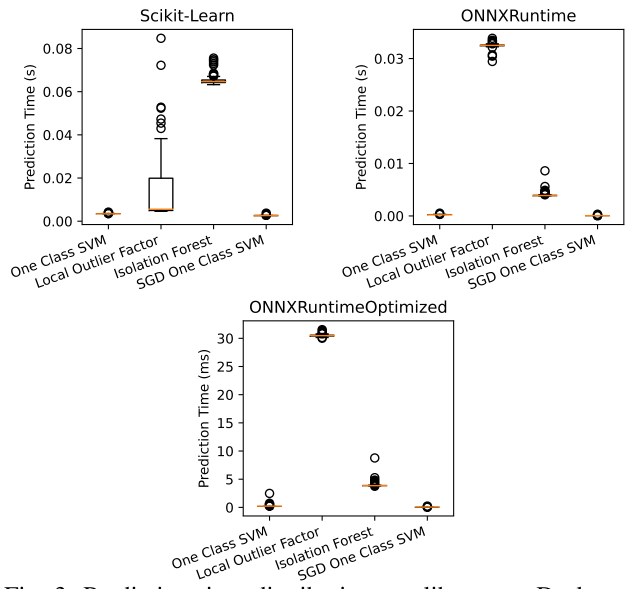

Fig. 3: Prediction time distribution per library on Desktop.

that ONNX underperforms in this model, taking longer than the Scikit-Learn. This leaves the door open for a more efficient implementation of LOF using the ONNX operators.

Figure 3 depicts the distribution of the prediction time of the various models per tool when executed in the Desktop. It is noteworthy for that, for ONNX Runtime (vanilla and Optimized), LOF presents the highest prediction time whereas, for Scikit-Learn, it is iForest that takes up the most time. Regarding the distribution of the samples, this is limited in the case of ONNX Runtime and Optimized to a few occasional outliers of additional time. For Scikit-Learn, LOF experiences considerable variability in prediction time. This may be a trade-off of the Scikit-Learn LOF implementation to achieve a lower average time for this concrete model.

Figure 4 exhibits the same analysis as Figure 2 for the

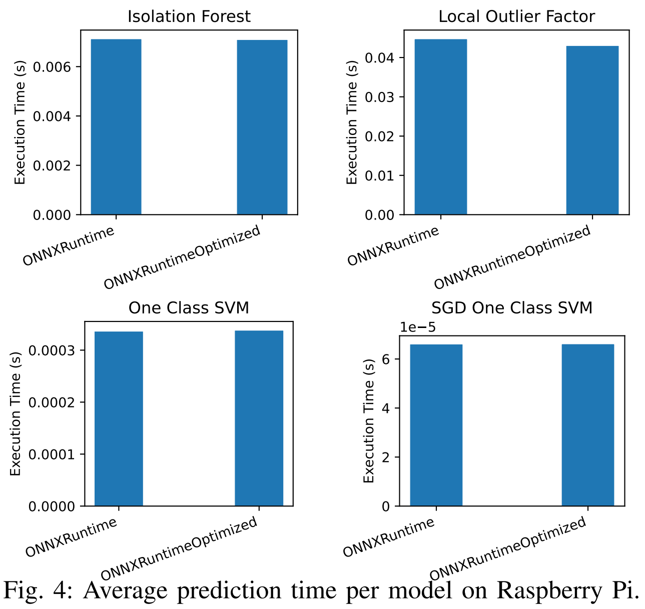

Fig. 4: Average prediction time per model on Raspberry Pi.

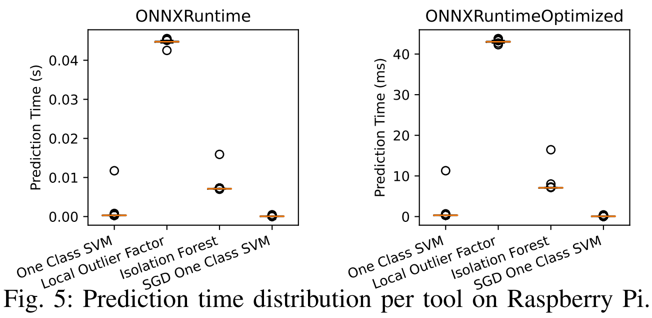

Fig. 5: Prediction time distribution per tool on Raspberry Pi.

second platform. The average of prediction time for the four ML models in the Raspberry Pi is superior to that of the Desktop response time; in detail, for Isolation Forest by $\approx$ 81%, for LOF by $\approx$ 37%; and for OC-SVM by $\approx$ 43%. However, when comparing with the Scikit-Learn running in the desktop, we obtain speedups of $\approx$ 8x for Isolation Forest, $\approx$ 9x for OC-SVM, and $\approx$ 39x for SGD-SVM. Results in Figure 5 presents little differences to Figure 3 (right and bottom) where it applies, apart from the generally higher median values in the Raspberry Pi.

V. Conclusion

We reviewed Machine Learning (ML) tools according to their potential for embedded system. We selected a particular tool, ONNX Runtime, for comparing prediction time against the well-established Python-based Scikit-Learn. ONNX Runtime is capable of running models described in the ONNX format; the models were trained in Scikit-Learn and exported to ONNX. The prediction time was measured in two platforms – a standard desktop and a target embedded system, a Raspberry Pi v4 – for four pre-trained ML models and datasets. We observe that ONNX Runtime considerably improves over the prediction time of Scikit-Learn, and experiences a negligible performance degradation when ported to the RPi. Future work will evaluate performance on more ML tools and platforms and investigate trade-offs with model target accuracy.

Acknowledgements

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDB/04234/2020), and by the Portuguese National Innovation Agency (ANI) through the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement, through the European Regional Development Fund (ERDF), within project(s) grant POCI-01-0247-FEDER-069522 (MIRAI).

References

[1] A. Hristoskova, N. González-Deleito, S. Klein, J. Sousa, N. Martins, J. Tagaio, J. Serra, C. Silva, J. Ferreira, P. M. Santos, R. Morla, L. Almeida, B. Bulut, and S. Sultanoğlu, "An Initial Analysis of the Shortcomings of Conventional AI and the Benefits of Distributed AI Approaches in Industrial Use Cases," in *IFIP AIAI 2021 Workshops*. Springer International Publishing, 2021, vol. 628, pp. 281–292.
[2] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs," *arXiv:1801.06601 [cs]*, Jan. 2018.
[3] S. Gopinath, N. Ghanathe, V. Seshadri, and R. Sharma, "Compiling KB-sized machine learning models to tiny IoT devices," in *40th ACM SIGPLAN Conference*. Phoenix AZ USA: ACM, Jun. 2019, pp. 79–95.
[4] R.Vinayakumar, M.Alazab, K.Soman, P.Poornachandran, A.Al-Nemrat, and S.Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019.
[5] M. Eskandari, Z. H. Janjua, M. Vecchio, and F. Antonelli, "Passban IDS: An intelligent anomaly-based intrusion detection system for IoT edge devices," *IEEE IoT Journal*, vol. 7, no. 8, pp. 6882–6897, Aug. 2020.
[6] N. Schumacher, P. M. Santos, P. F. Souto, N. Martins, J. Sousa, J. M. Ferreira, and L. Almeida, "One-Class Models for Intrusion Detection at ISP Customer Networks," in *IFIP AIAI 2023*, León, Spain, Jun. 2023.
[7] P. M. Santos, J. Sousa, R. Morla, N. Martins, J. Tagaio, J. Serra, C. Silva, M. Sousa, P. Souto, L. L. Ferreira, J. Ferreira, and L. Almeida, "Towards a Distributed Learning Architecture for Securing ISP Home Customers," in *IFIP AIAI 2021 Workshops*. Springer International Publishing, 2021, vol. 628, pp. 311–322.
[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[9] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *2008 IEEE Int'l Conference on Data Mining*, Pisa, Italy, Dec. 2008, pp. 413–422.
[10] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying Density-Based Local Outliers."
[11] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support Vector Method for Novelty Detection," p. 7.
[12] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *21st Int'l Conference on Machine Learning*, 2004, p. 116.
[13] C. Gupta, A. S. Suggala, A. Goyal, H. V. Simhadri, B. Paranjape, A. Kumar, S. Goyal, R. Udupa, M. Varma, and P. Jain, "ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices," in *34th Int'l Conference on Machine Learning*, Sydney, Australia, 2017, p. 16.
[14] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden, "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems," *arXiv:2010.08678 [cs]*, Mar. 2021.
[15] S. Garcia, A. Parmisano, and M. J. Erquiaga, "IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0) [Data set]," *Stratosphere Lab., Praha, Czech Republic, Tech. Rep*, 2020.
[16] N. Koroniotis, "Designing an effective network forensic framework for the investigation of botnets in the Internet of Things," Ph.D. dissertation, UNSW Sydney, 2020.