ACETA: Accelerating Encrypted Traffic Analytics on Network Edge

Derek Manning*, Peilong Li*
Department of Computer Science
Elizabethtown College
*{*ManningD*1, *LiP*}@etown.edu

Xiaoban Wu†, Yan Luo‡
Electrical and Computer Engineering
University of Massachusetts Lowell
†*Xiaoban_Wu*@student.uml.edu,
‡*Yan_Luo*@uml.edu

Tong Zhang§, Weigang Li§
Intel Corporation
§{*Tong*2.*Zhang*, *Weigang.Li*}@intel.com

***Abstract*—Applying machine learning techniques to detect malicious encrypted network traffic has become a challenging research topic. Traditional approaches based on studying network patterns fail to operate on encrypted data, especially without compromising the integrity of encryption. In addition, the requirement of rendering network-wide intelligent protection in a timely manner further exacerbates the problem. In this paper, we propose to leverage x86 multicore platforms provisioned at enterprises' network edge with the software accelerators to design an encrypted traffic analytics (ETA) system with accelerated speed. Specifically, we explore a suite of data features and machine learning models with an open dataset. Then we show that by using Intel DAAL and OpenVINO libraries in model training and inference, we are able to boost the performance by a maximum order of 32x and 46x respectively.**

***Index Terms*—Encrypted Traffic Analysis; DAAL; OpenVINO; Edge Computing; uCPE;**

I. Introduction

Encrypted network traffic is the major form of the data transmitting over the Internet today. According to the statistics, 94% of all Google web traffic is encrypted [1] as of July 2019. In 2019 alone, the encryption certificates issued per day is over 1 million [2]. While protecting the privacy and providing security for enterprises, encryption technology also benefits evaders to secure their malicious activities.

Traditional solutions to identify network threats fall into two major categories: 1) deep packet inspection and signatures; and 2) offline network pattern modeling. However, the first category is not applicable on encrypted traffic. Furthermore, solutions that work on decrypted network data will compromise the user privacy and data integrity, and are computationally intensive. By passively monitoring malicious behaviors on a session-based level, the second category of traditional solutions [3], [4] demonstrate high accuracy and low false positives. However, training machine learning models, and inferencing with effective models in a timely fashion are challenging due to the high computational power and low response latency requirements.

With the recent adoption of network function virtualization (NFV) technology, enterprises start to deploy universal customer premises equipment (uCPE) on their network edge to perform complex network functionality including encrypted traffic analytics (ETA) [5], [6], [7]. Such uCPE equipments rely on commodity x86 platforms to enable virtual network functions provisioned without pre-configuration or manual operations. Meanwhile, recent innovation on x86 multicore systems, such as the Intel Data Analytics Acceleration Library (DAAL) [8] for machine learning and the Intel OpenVINO toolkit [9] for computer vision and neural networks, have enabled ETA to render faster training and inference and analyze larger data sets with available compute resources on x86 uCPE equipments. The library also exploits the next-generation processors such as Intel Xeon D [10] to render optimal performance. By exploiting DAAL and OpenVINO, we can greatly accelerate the ETA speed without compromising performance.

In this paper, we propose to leverage Intel x86 multicore platforms with the aforementioned software to design an ETA system named `ACETA` with accelerated model training and inference. Specifically, we contribute in the three following aspects. 1) We seek to study the Cisco's open-sourced ETA software named `Joy` [11] as the reference design, and extend Joy with three new machine learning models and test them with a new open-source dataset. 2) We optimize the ETA pipeline in the proposed software package `ACETA` with multithreading and Intel software packages to render faster training and inference speed without sacrificing performance. 3) We conduct thorough performance evaluation and comparison between `Joy` and `ACETA` for future reference.

The rest of the paper is organized as follows. Section §II introduces the prior arts. Section §III depicts the overall design of `ACETA` and its advantages. We evaluate the performance of `ACETA` in Section §IV. Finally, we conclude the paper in Section §V.

II. Motivation and Related Works

Anderson et al. [3], [4] present a collection of supervised machine learning algorithms for ETA. The authors

highlighted two primary reasons why traditional pattern matching approaches cannot be applied to ETA, namely, inaccurate ground truth and a highly non-stationary data distribution. To address the aforementioned problems, the authors develop supervised machine learning models that leverage a unique and diverse set of network flow data features, including 1) TLS handshake meta-features, 2) DNS contextual flows linked to the encrypted flow, 3) HTTP headers of HTTP contextual flows from the same source IP address within a 5-minute window, and 4) META feature. Experiment results show that by incorporating the contextual information into a supervised learning system, the ETA system can achieve 0.00% false discovery rate and high detection accuracy.

`Joy` [11] is an open-sourced software initiated by the same authors Anderson et al. [3], [4]. Though lacking of the network traffic data for reference, the published version of Joy is capable of generating the aforementioned data features. In this paper, we leverage Joy as the feature extraction stage of the `ACETA` pipeline, and extend the only model published in Joy, namely logistic regression, to a suite of machine learning/deep learning models. More importantly, we implemented an accelerated version of the model suite with Intel DAAL and OpenVINO, and conduct extensive performance study over a public dataset `CICAndMal2017` [12].

III. Design

In this section, we present the design of `ACETA` by firstly describing the ETA pipeline and feature generation of Cisco `Joy`. Then we explain each of the pipeline components of `ACETA` in detail.

A. Reference Design - *Joy*

As shown in Figure 1, the pipeline processing of `Joy` starts with having the labelled training data collected in packet capture (.pcap) format. Then, using the feature extractor from `Joy`, the raw data is processed to generate flow level statistics, which is then used to form the features in a structured JSON format. At this point the user can solicit which features to use on the model for training and classification of samples. Once the data features for each of the samples are prepared, a model is trained, tested, and saved for further inference. It is worth noting that logistic regression is the only model provided by `Joy`, which is written with Scikit-learn package. To compare with the four machine learning models in `ACETA` designed with DAAL and OpenVINO libraries, we have to implement the corresponding Scikit-learn versions for `Joy`.

B. Features

The feature extractor of `Joy` is capable of extracting TLS, DNS, HTTP and metadata features from network flows. The TLS client-based features are the list of offered ciphersuites, advertised extensions, and the client's public key length, while the server-side TLS features are

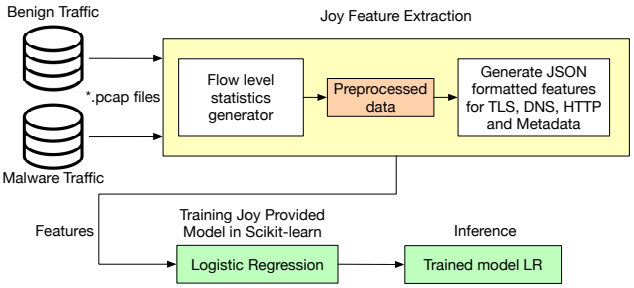

Fig. 1. ETA Pipeline of Cisco `Joy`

the selected ciphersuite, supported extensions, number of certificates, number of SAN names, validity in days, and the existence of a self-signed certificate. The DNS flow data is gathered from the corresponding DNS response to the TLS flow. From this flow the gathered features are the lengths of both the domain name and FQDN, the 32 most common TTL values, and binary features representing whether the domain name was in the top-100, top-1,000, top-10,000, top-100,000, top-1,000,000, or not contained in the Alexa list. HTTP data is collected from the source address of every TLS flow. Seven main types of features are used out of the flow data, which are the presence of outbound and inbound HTTP fields, Content-Type, User-Agent, Accept-Language, Server, and code. For metadata, features include the sequence of packet lengths, inter-arrival times, and byte distribution.

A comprehensive list of features used in `ACETA` can be found from the list below.

- The TLS features includes: 1) Client advertised ciphersuites; 2) Client advertised extensions; 3) Server supported ciphersuites; 4) Server supported extensions; 5) Client public key length; 6) Number of server certificates; 7) Number of subject alternative names (SAN) of server; 8) Validity of server certificates; and 9) Whether the certificate is self signed.
- The DNS features includes: 1) Length; 2) Suffix; 3) TTL; 4) Number of numerical character; 5) Number of Non-alphanumerical character; 6) Number of IPs returned per request; and 7) Alexa rank.
- The HTTP features includes: 1) Presence of HTTP fields; 2) Content-type; 3) User-agent; 4) Server; and 5) Return code.
- The metadata features includes: 1) Inter-Packet Times Markov Matrix; 2) Inter-Packet Lengths Markov Matrix; and 3) Byte Distribution.

C. *ACETA* Overview

As shown in the Figure 2, `ACETA` improves upon the baseline `Joy` by accelerating the processing steps at different levels. The initial processing with the `Joy` tool is accelerated by taking advantage of multicore processors, using multithreading. Likewise, each of the feature processing steps are accelerated by using multiple processes as well.

The data processing steps of ACETA use JSON files as the structured storage for our processed data. With the large number of reads and writes our tool needs to complete, the standard JSON library was a substantial bottleneck in the data processing pipeline. Using the C optimized Ultra JSON (ujson) library as a drop in replacement, we gain faster processing with identically structured results. With the data prepared, using the Intel Data Analytics Acceleration Library [8], we can accelerate the training and testing of models significantly faster than the standard machine learning libraries.

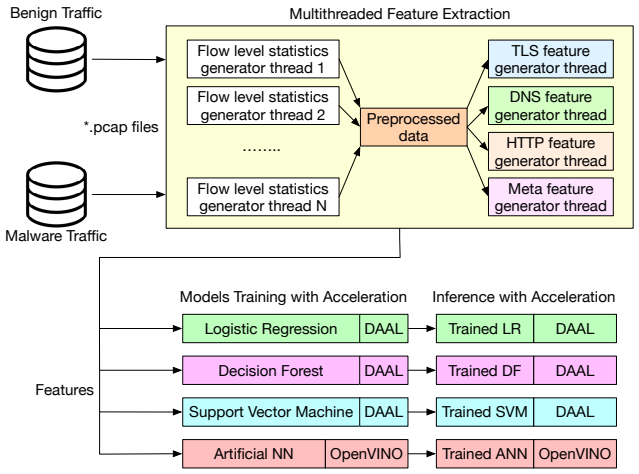

Fig. 2. ETA Pipeline of ACETA

D. Acceleration

Our initial optimizations target the first few data processing steps in ACETA. The performance is enhanced by using several different strategies. First, the code is simplified by implementing faster coding alternatives, such as list comprehensions and conditional exectution using try-catch. Next, the performance libraries Numpy and UltraJSON are implemented for further acceleration. With raw data samples being separated into their unique malware/benign family folders, the organizational structure of the raw data is established effectively to be processed in parallel. Using the multiprocessing library, the parallel capability of multicore machines are leveraged for additional performance gains.

After processing the raw data, the primary focus is on accelerating the model training and inference. The Intel Data Analytics Acceleration Library (DAAL) [8] provides optimized machine learning routines and algorithms targeted for Intel processors. DAAL functions maximize processing speed by knowing the instruction set, vector width, core counts, and memory architecture for each processor they run on. Similar to the popular Scikit-learn library, DAAL offers support for many of the popular machine learning algorithms. For our purposes, we compare logistic regression, decision forest, support vector machine, and neural network, by training on identical datasets with both non-DAAL and DAAL version code.

The DAAL machine learning pipeline was designed closely based off of the current popular libraries' structure, such as Scikit-learn. A code comparison of DAAL vs. Scikit-learn is shown in Algorithms 1 and 2. Both algorithms implement the same general pipeline of creating a model, training the model, and then predicting using the trained model. This similarity makes implementing DAAL into code that uses one of the standard libraries as simple as possible.

Algorithm 1 Scikit-learn code

```
1: import sklearn
2: logreg = sklearn.linear_model.LogisticRegression(parameters)
3: logreg.fit(trainData, trainLabels)
4: predictions = logreg.predict(testData)
```

Algorithm 2 DAAL code

```
1: import daal4py
2: logreg = daal4py.logistic_regression_training(parameters)
3: trainedModel = logreg.compute(trainData, trainLabels)
4: predictAlg = daal4py.logistic_regression_prediction(parameters)
5: predictResult = predictAlg.compute(testData, trainedModel.model)
6: predictions = predictResult.prediction
```

For deep learning, Intel also offers a dedicated framework, OpenVINO [9], that optimizes deep learning models in a similar manner to DAAL. OpenVINO has two components, its Model Optimizer and Inference Engine. The optimization process first requires a pre-trained model from one of its supported deep learning frameworks, in our case TensorFlow. First, the TensorFlow must be frozen into a format the Model Optimizer can analyze. The Model Optimizer converts the TensorFlow model to make an optimized intermediate representation based on the trained network topology, weights, and bias values. The newly optimized model is then used by the Inference Engine to classify new data, at a faster rate and with identical accuracy compared to the original TensorFlow model.

IV. Evaluation

In this section we evaluate the performance of the proposed ETA system ACETA. In particular we focus on the training time, inference time, accuracy, and performance breakdown of the different steps. First, we cover the experiment background and methodology for executing the comparisons. Then we discuss general performance improvements made in ACETA relative to the Joy baseline. After addressing general improvements, we cover each of the aforementioned performance metrics in detail for each of the models tested.

A. Experiment Platform and Methodology

We perform all testing for ACETA on a commodity workstation with Intel x86 processors to resemble the real-world uCPE setting. The Intel Distribution for Python is used

in an Anaconda environment for all programming, with all applicable additional packages coming via the Intel channel. The detailed hardware and software specifications are listed in Table I. We use the `CICAndMal2017` dataset [12], provided by the University of New Brunswick, for training and testing in all comparisons . In addition to benign flows, this dataset contains four categories of malware: adware, ransomware, scareware, and SMS malware, with samples coming from 42 unique malware families. Comparing the baseline with `ACETA` first involves randomly splitting subsets of the processed data for training and testing, providing one sample from each of the malware families per subset, along with many more benign samples. The subsets are partitioned, and then saved to separate files to be shared by the baseline and `ACETA` for level comparison results. We use the standard Python time library for the timing of model training, and we measure the accuracy by using the labels of 0 and 1 for benign and malicious samples respectively. Performance statistics are gathered using the generic cProfiling Python utility, along with the kcachegrind tool [13] to process and present the data.

TABLE I
EXPERIMENT PLATFORM SPECIFICATION

Item	Specification
CPU	Intel Core i5-8600k 6 cores @ 3.60GHz
Memory	16 GB DDR4 @ 2667MHz
HDD	1TB @ 7200RPM
Software	Intel DAAL v0.2019.3
	Intel OpenVINO v2019.1.144
	Anaconda v5.3.0
Host OS	Ubuntu 16.04 Desktop

B. Performance Results

The baseline raw to processed data pipeline takes on average 33.7 minutes for completion. Given this result, we begin to implement strategies to accelerate the pipeline. Initial data processing optimizations such as list comprehensions, string formatting, accelerated conditional processing using try-catch, and using Numpy arrays instead of Python lists improve the average performance to 30.1 minutes, a speedup of 8.3%. Multiprocessing the initial data processing with `Joy`, and also the feature analysis result in an average performance of 23.2 minutes, a speedup of 31.2% relative to the baseline. Next, we exploit the ujson library as a drop-in replacement for the standard JSON library, resulting the process to take only 17.8 minutes, a cumulative speedup of 47.2%.

1) Training Time: In the model creation phase of `ACETA`, the user has the option to select which type of model to train and save. Currently, the model options are logistic regression, decision forest, support vector machine, and an artificial neural network. The baseline version of the code uses Scikit-learn library for the models, while `ACETA` uses DAAL, with both versions' models using identical parameter settings. The log scale training time of each of the models for the two versions are shown in figure 3. For all of the models except the neural network, the `ACETA` versions outperform the original by several factors: 10x faster for logistic regression, 3.25x faster for decision forest, and 31x faster for SVM. A 100-epoch neural network comparison results in the `ACETA` version being around twice as slow as the original. One possible reason for this inconsistency is that the original model uses a deprecated, Scikit-learn neural network API, whereas `ACETA` uses an Intel Python distribution Keras [14] model. DAAL offers low level APIs, similar to standard TensorFlow, to design neural networks piece-by-piece, but for the sake of simplicity Keras is used in our implementation. However, it is also worth noting that optimizing this TensorFlow based model, using Intel's OpenVINO deep learning framework [9], results in a faster inference time relative to the baseline neural network.

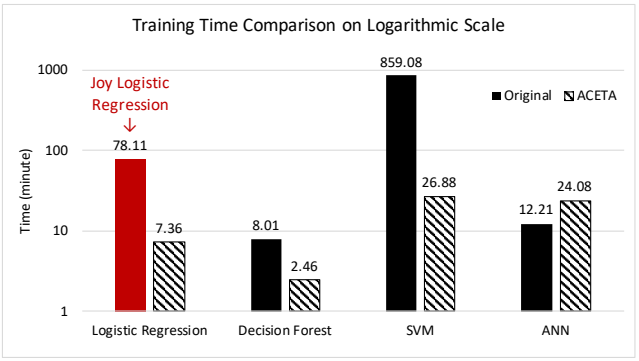

Fig. 3. Training Time Comparison on Logarithmic Scale

2) Inference Time: In addition to improving training performance, `ACETA` also performs significantly faster for inference. Figure 4 presents the time comparisons for each model on an identical dataset. The baseline models all complete in roughly 925 milliseconds, compared to the majority of `ACETA` models taking only 20 milliseconds, a 46x improvement. The only model to underperform is the `ACETA` ANN, taking about 1.49 seconds to complete its inference (not shown in Figure 4). However, by optimizing the `ACETA` Keras/TensorFlow ANN model with Intel OpenVINO, the optimized model inference takes only 30 milliseconds, on the order of the other `ACETA` models, while retaining the same accuracy as its unoptimized counterpart.

3) Accuracy: Figure 5 shows accuracy comparisons for model training while Figure 6 demonstrates inference accuracy on an identical test dataset immediately after model training. With each of the models, the performance shows very little difference between the baseline and `ACETA`. At most, `ACETA`'s accuracy is 2.5% lower for logistic regression training, but on average the difference is around a single percent. In the case of the decision forest classifier, the `ACETA` version even outperforms the slower, baseline model. This data, in addition to the training and

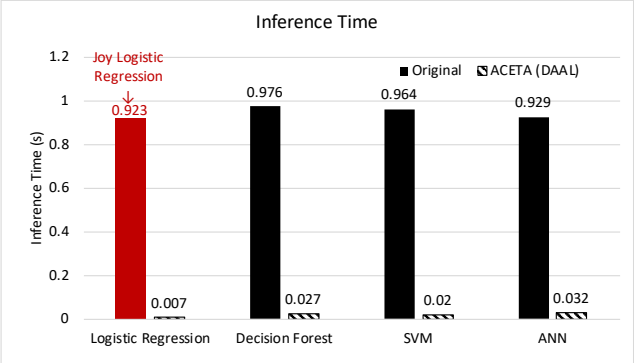

Fig. 4. Inference Time Comparison

inference time results, provides significant support for the use of DAAL as an alternative to the standard machine learning libraries.

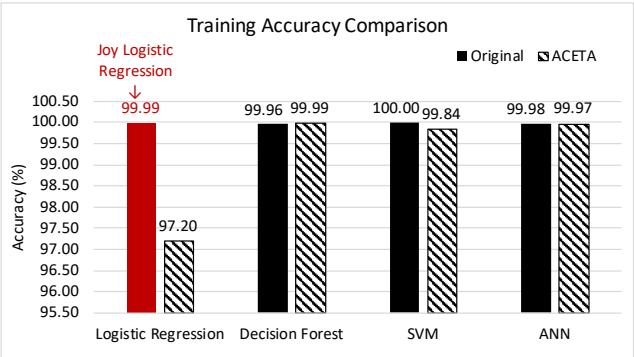

Fig. 5. Training Accuracy Comparison

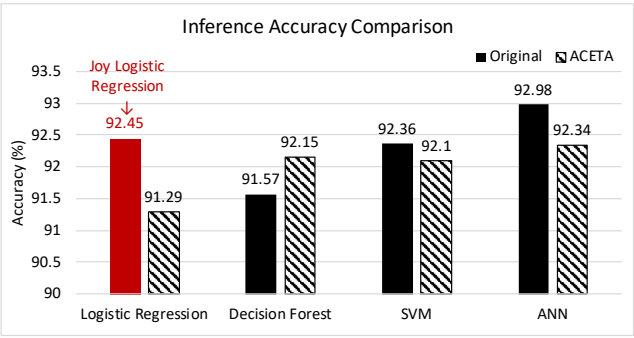

Fig. 6. Inference Accuracy Comparison

4) Performance Breakdown: We leverage the cProfiling utility to track and save the cumulative time the program spends in each of its functions. Using kcachegrind [13], we visualize the data to locate the major hotspots within the training of each of the models. Figure 7 displays this breakdown separated into three categories: training, data preparation, and other. For the majority of the models, the training of the models occupies the biggest percentage of time, which is to be expected. With the acceleration of DAAL and OpenVINO, the `ACETA` versions receive a great decrease on the overall training time percentage. One interesting observation is the training time percentage of the `ACETA` decision forest: it is not the most time consuming part of the algorithm. This can be explained by this model being by far the fastest to complete training, so the data processing and overhead inevitably occupy larger percentages of time. With both logistic regression and SVM showing a similar breakdown, the other interesting comparison is between the original and `ACETA` ANN. As mentioned earlier, the `ACETA` ANN version uses Keras [14], which provides a high-level API to create TensorFlow models. This supports the measured `ACETA` ANN breakdown that shows a larger percentage of time, 32.3%, being consumed by other tasks, other being the lump sum of any additional overhead. This irregularity explains at least in part the performance disparity between the baseline and `ACETA` ANN versions.

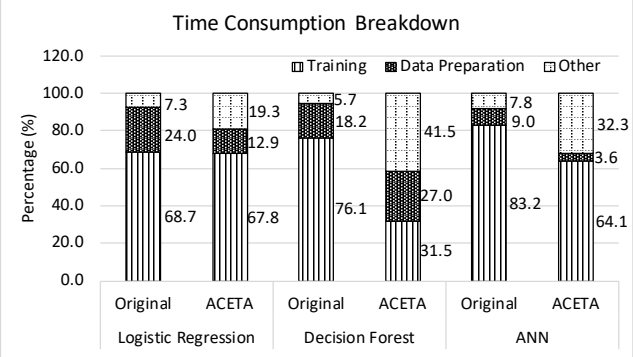

Fig. 7. Time Consumption Breakdown

V. Conclusion

In this paper, we aim to address the challenges in designing a high performance ETA system with the x86 processors provisioned at enterprises' network edge. In the proposed `ACETA` system, we study machine learning techniques for the identification of encrypted malware network flows. Specifically, we first explore and design multiple sets of features and four machine learning models over a public dataset. We then implement accelerated versions of the four machine learning models without sacrificing classification accuracy using Intel DAAL and OpenVINO. Our experimental results have demonstrated both the feasibility as well as the boosted training and inference performance of such a framework. Going beyond the experiments, our future work will focus on implementing an real-time ETA system for uCPE devices at network edge that comprises of online feature extraction, speeded model training and real-time inference.

Acknowledgment

This work is supported in part by the National Science Foundation (No. 1547428, No. 1541434, No. 1738965 and

No. 1450996), a grant from Intel Corporation, and a grant from Summer Scholarship, Creative Arts and Research Projects (SCARP) Program of Elizabethtown College.

References

[1] Google, LLC, "Https encryption on the web," 2019. [Online]. Available: https://transparencyreport.google.com/https/overview

[2] Let's Encrypt, "Let's encrypt stats," 2019. [Online]. Available: https://letsencrypt.org/stats/

[3] B. Anderson and D. McGrew, "Identifying encrypted malware traffic with contextual flow data," in *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, ser. AISec '16. New York, NY, USA: ACM, 2016, pp. 35–46. [Online]. Available: http://doi.acm.org/10.1145/2996758.2996768

[4] ——, "Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: ACM, 2017, pp. 1723–1732. [Online]. Available: http://doi.acm.org/10.1145/3097983.3098163

[5] Cisco Public, "Cisco encrypted traffic analytics (white paper)," 2019. [Online]. Available: https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf

[6] Intel Corporation, "Intel select solutions for ucpe solution brief," 2019. [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/select-solutions/select-solutions-for-ucpe-brief.html

[7] ADVA Optical Networking, "Universal customer premises equipment: Fast and agile service creation at the network edge," 2019. [Online]. Available: https://www.advaoptical.com/-/media/adva-main-site/resources/application-notes/pdfs/universal-customer-premises-equipment.ashx?la=en&hash=8C960A70F2168615A70ED353570785F2795C6C35

[8] Intel Corporation, "Intel® Data Analytics Acceleration Library (Intel® DAAL)," 2019. [Online]. Available: https://software.intel.com/en-us/intel-daal

[9] ——, "OpenVINO Toolkit: Develop Multiplatform Computer Vision Solutions. Explore the Intel® Distribution of OpenVINO™ toolkit," 2019. [Online]. Available: https://software.intel.com/en-us/openvino-toolkit

[10] ——, "Intel XEON D Processors," 2019. [Online]. Available: https://www.intel.com/content/www/us/en/products/processors/xeon/d-processors.html

[11] Cisco, "Joy: A package for capturing and analyzing network flow data and intraflow data, for network research, forensics, and security monitoring," 2019. [Online]. Available: https://github.com/cisco/joy

[12] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in *2018 International Carnahan Conference on Security Technology (ICCST)*, Oct 2018, pp. 1–7.

[13] "kcachegrind." [Online]. Available: https://kcachegrind.github.io/html/Home.html

[14] "Keras: The python deep learning library." [Online]. Available: https://keras.io/