FPGA-Accelerated Neural Networks for Real-Time Anomaly Detection using High-Level Synthesis

Anila Hoxha[1,*], Shekoufeh Neisarian[1,2,*], Talaya Farasat[1], Joachim Posegga[1] and Elif Bilge Kavun[2,3]

[1]Faculty of Computer Science and Mathematics, University of Passau, Passau, Germany
[2]Barkhausen Institut, Dresden, Germany
[3]Institute of Systems Architecture, Dresden University of Technology, Dresden, Germany

Abstract

Anomaly-based network intrusion detection is a critical and challenging task. Some anomaly detection techniques apply neural networks, benefiting from their ability to recognize patterns associated with malicious activity. The increasing complexity of anomaly detection models requires hardware platforms that offer high performance and real-time processing capabilities. Field-Programmable Gate Arrays (FPGAs) prove to be an effective solution for real-time inference acceleration, outperforming conventional choices such as Central Processing Units (CPUs) and Graphics Processing Units (GPUs). In this paper, we accelerate a Fully-Connected Neural Network (FCNN) and a one-dimensional Convolutional Neural Network (CNN). Both models are trained on the CIDDS-001 dataset and deployed on the PYNQ-Z2 FPGA. To streamline hardware design, high-level synthesis with hls4ml is utilized. We evaluate and compare the classification and deployment performance of both models with those deployed on the Intel Core i7-9750H CPU and the NVIDIA GeForce RTX 2080 Ti GPU. The results indicate that both FPGA deployments outperform the CPU and GPU. The FCNN FPGA deployment achieves 193.50× faster inference and 170.56× higher throughput than the CPU, and 898.50× faster inference and 792.07× higher throughput than the GPU. Compared to the CPU, the CNN FPGA deployment achieves 64.00× faster inference and 72.67× higher throughput, and 574.00× faster inference and 650.58× higher throughput compared to the GPU.

Keywords

Neural networks, FPGA, Hls4ml, Latency, Throughput

1. Introduction

Anomaly-based Network Intrusion Detection Systems (NIDSs) are essential for identifying and mitigating cybersecurity anomalies in network traffic. The evolving and complex nature of cyber threats makes real-time anomaly detection a crucial and challenging task. Some anomaly detection techniques integrate concepts from machine learning, including neural networks, as these models can efficiently extract complex features and recognize patterns associated with malicious behavior [1, 2, 3, 4]. While traditionally deployed on Central Processing Units (CPUs) and Graphics Processing Units (GPUs), the increasing complexity of anomaly detection models requires the target platforms to meet higher performance demands. To mitigate these challenges, Field-Programmable Gate Arrays (FPGAs) have become increasingly popular over the years. Their key characteristics, such as data parallelism, energy efficiency, and the ability to achieve low inference latency and high throughput simultaneously, provide a significant performance advantage over conventional hardware platforms [5, 6].

To validate this claim, this research evaluates the classification and deployment performance of neural networks on FPGAs compared to CPU and GPU applications. We train a Fully-Connected Neural Network (FCNN) and a one-dimensional Convolutional Neural Network (CNN) on the Coburg Intrusion Detection Data Set (CIDDS-001), created for the evaluation of anomaly-based NIDSs [7, 8]. To facilitate the hardware design process, we use High-Level Synthesis (HLS) with hls4ml, an open-source framework that efficiently translates neural networks into FPGA implementations [9, 10, 11, 12, 13, 14]. Both models are deployed on the PYNQ-Z2 FPGA development board featuring Xilinx Zynq-7020 SoC,

CPSWS'25: CPS Summer School PhD Workshop, September 22, 2025, Alghero, Sardinia, Italy
*Corresponding authors.
hoxha02@ads.uni-passau.de (A. Hoxha) & neisar01@ads.uni-passau.de (S. Neisarian)

the Intel Core i7-9750H CPU, and the NVIDIA GeForce RTX 2080 Ti GPU. Our analysis is mainly focused on the inference latency and throughput metrics.

1.1. Related Work

Ioannou et al. introduce a flexible FPGA-based network intrusion detection approach using neural networks on the NSL-KDD dataset. Their deployment on the Xilinx Zynq Z-7020 FPGA achieves approximately 26.6× acceleration over previous research on intrusion detection systems while identifying not only Denial-of-Service (DoS) attacks but also new threats, including Probe, U2R, and R2L [15]. In a more recent study, Pham-Quoc et al. propose an Artificial Intelligence (AI)-powered framework for anomaly-based NIDSs, suitable for real-time deployment. Two models, Anomaly-Detection Autoencoder (ADA) and Artificial Neural Classification (ANC), are trained and evaluated on the NSL-KDD, UNSW-NB15, and CIC-IDS2017 datasets with respect to latency, throughput, and accuracy metrics. Results show that the proposed architecture efficiently utilizes the hardware resources of the NetFPGA-SUME platform while achieving higher throughput than the GPU and CPU deployments. On the NSL-KDD dataset, the ADA model achieves a slightly lower training accuracy of 90.87% compared to Ioannou et al.'s approach, which achieves an accuracy of 96.02% [16].

Ngadiuba et al. discuss the application of Binary Neural Networks (BNNs) and Ternary Neural Networks (TNNs) as possible quantization approaches in integration with hls4ml. They deploy these models, trained on MNIST and Jet Tagging datasets, on the Xilinx Virtex Ultrascale 9+ FPGA. Results indicate that both BNNs and TNNs retain good accuracy (with TNNs being slightly more accurate) and latency (~100 ns) while achieving a significant reduction in hardware resource utilization [17]. In their study, Aarrestad et al. extend prototype models trained and evaluated on the SVHN dataset to explore the acceleration of convolutional networks on FPGAs using hls4ml. They apply model optimization techniques such as quantization and pruning, and discuss how FPGA resource utilization can be minimized by at least 97%, if compromising accuracy is justified. They successfully deploy the model on the PYNQ-Z2 FPGA with low latency and power demand, defining a benchmark in the integration of the convolutional network architecture with hls4ml [18].

While several studies have analyzed the acceleration of intrusion detection systems on FPGAs, and separately, the acceleration of neural networks on FPGAs using HLS with hls4ml, to the best of our knowledge, no previous research has applied HLS with hls4ml specifically in the anomaly detection domain. What sets this research apart is the integration of HLS with hls4ml for real-time anomaly-based intrusion detection in network traffic, specifically targeting port scan, ping scan, DoS, and brute force attacks.

2. Methodology

The first part of this research is conducted in a virtual environment running Ubuntu 18.04.2, hosted on Oracle VirtualBox 7.0.18. The virtual machine is allocated 12 GB of RAM, 4 CPU cores, and approximately 300 GB of disk space.

2.1. Dataset

The dataset chosen for this research is CIDDS-001, published by Hochschule Coburg, Germany, in 2017. The dataset is created for the evaluation of anomaly-based NIDSs and contains approximately 32 million network flows and 92 recorded attack instances categorized as port scan, ping scan, DoS, and brute force, captured over four weeks. To gather the data, the team developed a simple cloud environment using OpenStack, as well as deployed an external server on the internet, and monitored the network flow on these two server points. 60% of the attacks targeted at the OpenStack environment occurred during week 1, while the remaining attacks were launched in week 2. The flows captured during week 3 and week 4 would worsen the dataset imbalance, if considered for our research, and are therefore ignored. At the external server, only port scan and brute force attacks occurred [7, 8]. Given the distribution of attacks and due to resource constraints, only the traffic recorded in the OpenStack environment during week 1 is considered for this research. Additional preprocessing, dataset balancing, and normalization techniques are applied to a partially cleaned and formatted version of this dataset [19]. 75% of the dataset is allocated for training the models, while the remaining 25% is allocated for evaluation.

2.2. Keras-Based Neural Network Implementations

The first proposed model architecture, described in Figure 1 (left), is an FCNN consisting of four layers. The input layer receives 14 inputs corresponding to the number of features in the dataset. The fully-connected dense layers contain 32 and 16 units, respectively. A ReLU activation function is applied to both hidden layers. Lastly, the output dense layer consists of three units for multi-class classification, using a softmax activation function [20]. The second proposed model architecture, visualized in Figure 1 (right), is a one-dimensional CNN consisting of five layers. The input layer receives a sequence of 14 features. The `Conv1D` layer applies four filters to sliding windows of three input features to learn patterns from potentially meaningful, local relationships between features. This layer outputs a set of four 12-unit vectors. The next layer flattens the output of the convolutional layer into one 48-unit vector, followed by a fully-connected layer with eight units and a ReLU activation function. The output layer consists of three units with a softmax activation function, similar to the FCNN architecture [20].

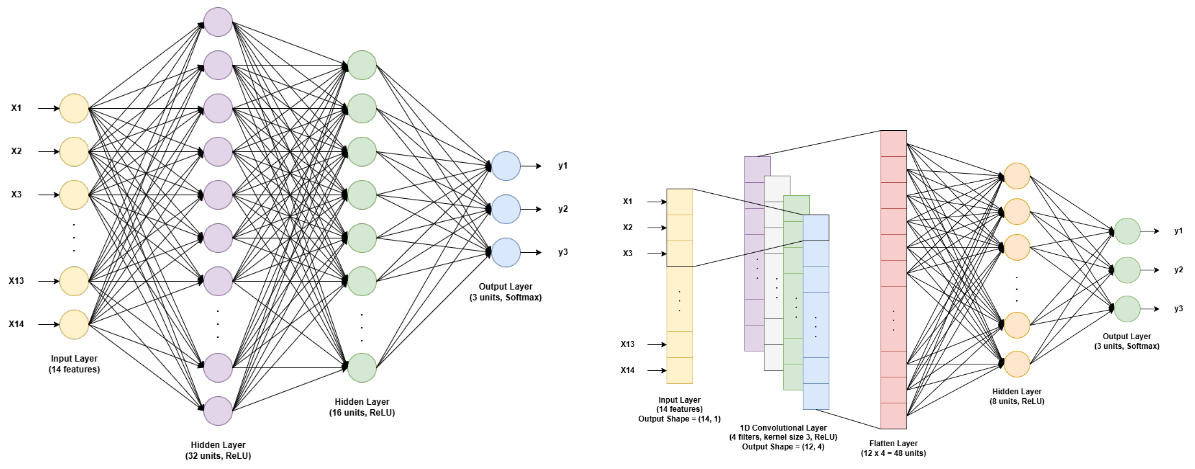
X1
X2
X3
X13
X14
Input Layer (14 features)
Hidden Layer (32 units, ReLU)
Hidden Layer (16 units, ReLU)
Output Layer (3 units, Softmax)
y1
y2
y3
X1
X2
X3
X13
X14
Input Layer (14 features) Output Shape = (14, 1)
1D Convolutional Layer (4 filters, kernel size 3, ReLU) Output Shape = (12, 4)
Flatten Layer (12 x 4 = 48 units)
Hidden Layer (8 units, ReLU)
Output Layer (3 units, Softmax)
y1
y2
y3

Figure 1: Proposed FCNN (left) and 1D CNN (right) architectures.

Both architectures are kept simple with small layer sizes to ensure that the computational complexity during HLS is reduced before applying model optimization techniques and adjusting the reuse factor per layer. The model weights and activations are quantized using quantization-aware training with QKeras with a precision of 6 bits and pruned to a target sparsity of 80% [21, 22, 23]. The models are trained using the Adam optimizer with the default learning rate of 0.001 [24]. The categorical cross-entropy loss function is utilized during training. Each model is trained over 10 epochs with a batch size of 32. To promote generalization on new, unseen data, the `L1`, `EarlyStopping`, and `ReduceLROnPlateau` regularization and callback techniques are applied [20].

2.3. From Keras to hls4ml

Each Keras model is loaded to generate its corresponding HLS configurations. For the FCNN HLS configuration, the default fixed-point precision of $< 16, 6 >$ is applied, and the reuse factor is set to 16 for all model layers. For the CNN HLS configuration, the fixed-point precision is reduced to $< 12, 4 >$, and the reuse factor is reset to its default value of one, except for the dense layer `hdl1`, which is set to a reuse factor of 64. For the softmax layers, the fixed-point precision formats for the exponential and inverse LUTs are set to $< 18, 8 >$ and $< 18, 4 >$, respectively, in the FCNN HLS configuration, and to

$< 14, 6 >$ and $< 14, 8 >$ in the CNN HLS configuration. The FPGA optimization strategy is set to `Latency` for both configurations to help achieve the lowest possible inference latency. For the `hdl1` layer, the strategy is set to `Resource` to minimize the utilization of the FPGA resources. This sacrifices latency within tolerable limits [11]. Both models are converted into hls4ml models and compiled successfully in preparation for synthesis. We could successfully synthesize both neural networks and generate each custom intellectual property core and bitstream file necessary for programming the FPGA.

2.4. Hardware Deployments

The same virtual setup is utilized to deploy the models on the Intel Core i7-9750H CPU. For GPU deployment, a JupyterHub server instance provided by the University of Passau's InnKube Infrastructure Group is utilized. It is configured with 2 CPU cores, 16 GB of RAM, 100 GB of storage, and one NVIDIA GeForce RTX 2080 Ti GPU with 12 GB of memory [25]. To measure the inference latency and throughput on hardware, we first load the pre-trained models and the respective test files, then measure the time it takes each model to make predictions on new, unseen data, and lastly calculate the results for both metrics accordingly [5]. For the GPU deployment, we enforce CPU utilization on a few operations before loading the pre-trained models to avoid GPU just-in-time compilation failures. As a result, only about 70% of the GPU is utilized for this deployment. The code snippet executed for deploying the models on the PYNQ-Z2 FPGA is based on the tutorial notebooks for hls4ml [26]. An instance of the `NeuralNetworkOverlay` class is created, including the bitstream and test files for temporary memory allocation before running the inference.

3. Results and Evaluation

Our results demonstrate that the non-quantized FCNN and CNN models achieve accuracies of 98.16% and 98.21% on the CPU, respectively. The precision, recall, and F1-score results for each class - *normal*, *attacker*, and *victim* - are presented in Table 1. From an anomaly detection point of view, since 93.17% and 90.57% of the attacks in the test set are detected, these results are considered acceptable.

Table 1
Evaluation metrics (%) for both neural networks on the CIDDS-001 dataset.

Category	FCNN			CNN		
	Precision	Recall	F1-score	Precision	Recall	F1-score
Normal	100.00	100.00	100.00	100.00	100.00	100.00
Attacker	89.25	93.17	91.17	91.76	90.57	91.16
Victim	89.18	83.37	86.18	86.29	87.95	87.11

The accuracy of the QKeras-quantized models decreases to 93.10% and 91.41% relative to their non-quantized baselines as a result of applying 6-bit quantization. We analyze the accuracy and classification performance of the QKeras-quantized models deployed on the CPU against (1) the QKeras-quantized models deployed on the GPU and (2) the corresponding hls4ml implementations deployed on the FPGA. When deployed on the GPU, the accuracy of the QKeras-quantized FCNN model decreases by 0.24% to 92.88%, while the QKeras-quantized CNN model performs 1.01× faster with an accuracy of 92.09%. The hls4ml implementations achieve accuracies of 91.64% and 86.91% on the FPGA, respectively. The accuracy drop comes from the reduced fixed-point precision of the HLS configurations before model synthesis. Furthermore, to validate the predictions run on the GPU and FPGA, the deployment outputs are saved and checked against the QKeras-quantized outputs captured on the CPU. Figures 2 and 3 show the ROC curves, comparing the performance of the three classes, denoted as *taggers* on the graphs.

We measure the inference latency and calculate the throughput of both neural networks across the three hardware platforms. The test data consists of 15000 samples. The results are shown in Table 2. Compared to the CPU, the FPGA achieves 193.50× faster inference and 170.56× higher throughput

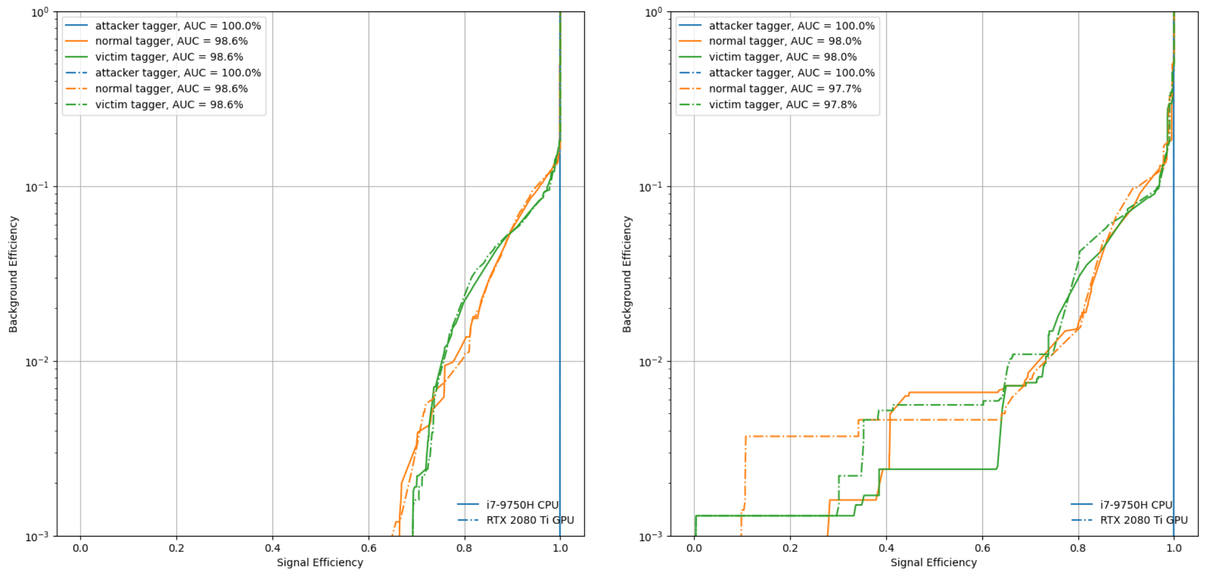

Figure 2: ROC curves for the FCNN (left) and CNN (right) models, i7-9750H CPU vs. RTX 2080 Ti GPU.

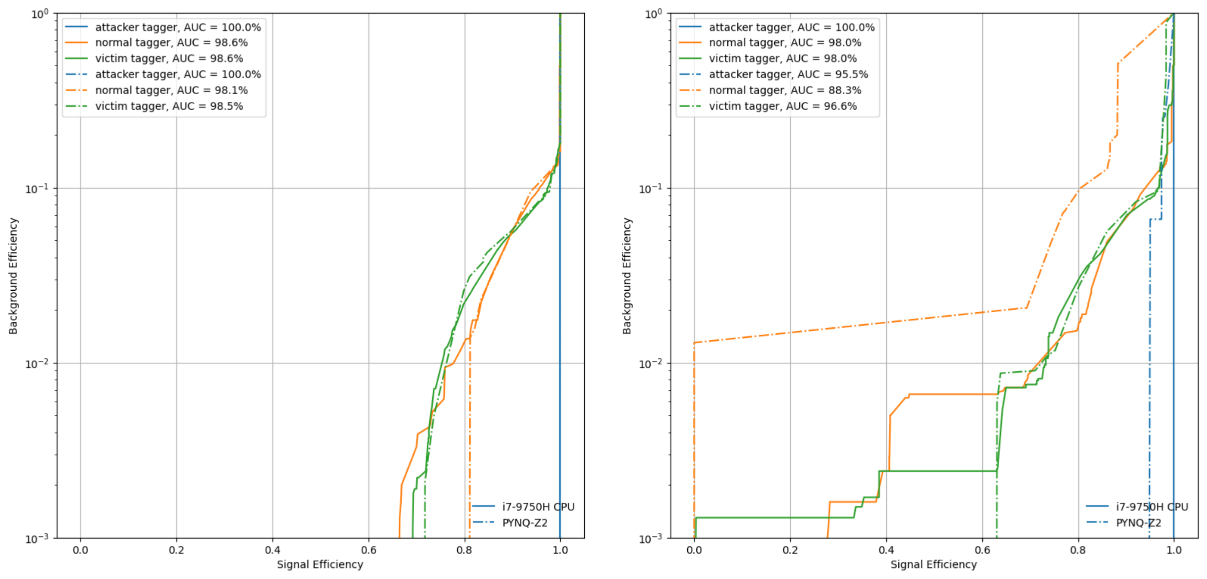

Figure 3: ROC curves for the FCNN (left) and CNN (right) models, i7-9750H CPU vs. PYNQ-Z2 FPGA.

for the FCNN model, and 64.00× faster inference and 72.67× higher throughput for the CNN model. Compared to the GPU, the FPGA achieves 898.50× faster inference and 792.07× higher throughput for the FCNN model, and 574.00× faster inference and 650.58× higher throughput for the CNN model. Contrary to our expectation, both neural network deployments on the CPU outperform the GPU. For the FCNN model deployment, the CPU achieves 4.64× faster inference and 4.65× higher throughput, while for the CNN model, it achieves 8.97× faster inference and 8.95× higher throughput. The research by Lind et al. concludes that while complex neural networks take advantage of the parallel computing capabilities of GPUs, this advantage is insignificant when working with simpler neural networks [27]. This observation seems to support our results. It should be noted that a fair comparison of our results with similar research on anomaly detection is not possible due to the different datasets used.

Table 2
Inference latency (s) and throughput (inferences/second).

Target Deployment Platform	FCNN		CNN	
	Latency	Throughput	Latency	Throughput
i7-9750H CPU	3.87	3879.09	1.92	7797.37
RTX 2080 Ti GPU	17.97	834.90	17.22	870.95
PYNQ-Z2 FPGA	0.02	661404.82	0.03	566615.04

4. Conclusion

In this paper, the acceleration of an FCNN and a one-dimensional CNN is presented for anomaly-based network intrusion detection on FPGAs. We integrate HLS with hls4ml to translate the neural networks into FPGA implementations. The research is conducted across a virtual machine running Ubuntu, a JupyterHub server instance, and the PYNQ-Z2 FPGA development board. These working environments support dataset preprocessing, model training and synthesis, and hardware deployments. The CIDDS-001 dataset is used to train and evaluate the neural networks. We measure and evaluate the classification accuracy and deployment performance in terms of inference latency and throughput on the PYNQ-Z2 FPGA, comparing it with the Intel Core i7-9750H CPU and the NVIDIA GeForce RTX 2080 Ti GPU. The results indicate that the FPGA deployment outperforms both the CPU and GPU deployments.

4.1. Limitations & Future Work

Restricted access to appropriate hardware equipment limits our work capacity. One challenge is the limited RAM allocation on the virtual machine, which slows down synthesis for complex neural networks. As a result, simpler neural architectures need to be used to avoid costly computations and fit within the available resources. Another issue is that resource-constrained FPGA boards, such as the PYNQ-Z2, limit the capability to work with larger datasets like NSL-KDD or CIC-IDS2017. Our future work includes exploring methods to improve the classification accuracy of the neural networks on the PYNQ-Z2 FPGA and possibly reimplementing this research on less resource-constrained hardware to enable extended research with more complex models and larger datasets for anomaly-based intrusion detection, as well as vulnerability detection. A comparison of the power consumption of the neural network deployments across all three hardware platforms could be a potential extension of this research.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly and Chat-GPT-4 for grammar and spelling checks. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] J. Ryan, M.-J. Lin, R. Miikkulainen, Intrusion detection with neural networks, Advances in neural information processing systems 10 (1997).

[2] F. Meghdouri, M. Bachl, T. Zseby, Eagernet: Early predictions of neural networks for computationally efficient intrusion detection, in: 2020 4th Cyber Security in Networking Conference (CSNet), IEEE, 2020, pp. 1–7.

[3] A. Awajan, A novel deep learning-based intrusion detection system for IOT networks, Computers 12 (2023) 34.

[4] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM computing surveys (CSUR) 41 (2009) 1–58.

[5] F. Al-Ali, T. D. Gamage, H. W. Nanayakkara, F. Mehdipour, S. K. Ray, Novel casestudy and benchmarking of AlexNet for edge AI: From CPU and GPU to FPGA, in: 2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), IEEE, 2020, pp. 1–4.

[6] M. Dhouibi, A. K. Ben Salem, A. Saidi, S. Ben Saoud, Accelerating deep neural networks implementation: A survey, IET Computers & Digital Techniques 15 (2021) 79–96.

[7] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, A. Hotho, Flow-based benchmark data sets for intrusion detection, in: Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS), ACPI, 2017, p. 361–369.

[8] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, A. Hotho, Creation of flow-based data sets for intrusion detection, Journal of Information Warfare 16 (2017) 40–53.

[9] P. Coussy, A. Morawiec, High-level synthesis, volume 1, Springer, 2010.

[10] A. Takach, High-level synthesis: Status, trends, and future directions, IEEE Design & Test 33 (2016) 116–124.

[11] FastML Team, fastmachinelearning/hls4ml, 2025. URL: https://github.com/fastmachinelearning/hls4ml. doi:10.5281/zenodo.1201549.

[12] J. Duarte, et al., Fast inference of deep neural networks in FPGAs for particle physics, JINST 13 (2018) P07027. doi:10.1088/1748-0221/13/07/P07027. arXiv:1804.06913.

[13] T. Aarrestad, et al., Fast convolutional neural networks on FPGAs with hls4ml, Mach. Learn. Sci. Tech. 2 (2021) 045015. doi:10.1088/2632-2153/ac0ea1. arXiv:2101.05108.

[14] N. Ghielmetti, et al., Real-time semantic segmentation on FPGAs for autonomous vehicles with hls4ml, Mach. Learn. Sci. Tech. (2022). doi:10.1088/2632-2153/ac9cb5. arXiv:2205.07690.

[15] L. Ioannou, S. A. Fahmy, Network intrusion detection using neural networks on FPGA SoCs, in: 2019 29th International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2019, pp. 232–238.

[16] C. Pham-Quoc, T. H. Q. Bao, T. N. Thinh, FPGA/AI-powered architecture for anomaly network intrusion detection systems, Electronics 12 (2023) 668.

[17] J. Ngadiuba, V. Loncar, M. Pierini, S. Summers, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, M. Liu, et al., Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml, Machine Learning: Science and Technology 2 (2020) 015001.

[18] T. Aarrestad, V. Loncar, N. Ghielmetti, M. Pierini, S. Summers, J. Ngadiuba, C. Petersson, H. Linander, Y. Iiyama, G. Di Guglielmo, et al., Fast convolutional neural networks on FPGAs with hls4ml, Machine Learning: Science and Technology 2 (2021) 045015.

[19] Dhoogla, CIDDS-001 dataset, 2025. URL: https://www.kaggle.com/datasets/dhoogla/cidds001, accessed: 2025-04-10.

[20] F. Chollet, Deep learning with Python, Simon and Schuster, 2021.

[21] O. Weng, Neural network quantization for efficient inference: A survey, arXiv preprint arXiv:2112.06126 (2021).

[22] C. N. Coelho, A. Kuusela, H. Zhuang, T. Aarrestad, V. Loncar, J. Ngadiuba, M. Pierini, S. Summers, Ultra low-latency, low-area inference accelerators using heterogeneous deep quantization with qkeras and hls4ml, arXiv preprint arXiv:2006.10159 108 (2020).

[23] B. Ramhorst, V. Lončar, G. A. Constantinides, FPGA resource-aware structured pruning for real-time neural networks, in: 2023 International Conference on Field Programmable Technology (ICFPT), IEEE, 2023, pp. 282–283.

[24] K. D. B. J. Adam, et al., A method for stochastic optimization, arXiv preprint arXiv:1412.6980 1412 (2014).

[25] U. o. P. InnKube Infrastructure Group, Innkube documentation, 2025. URL: https://innkube.pages.gitlab.innkube.fim.uni-passau.de/documentation/index.html, accessed: 2025-04-10.

[26] FastML Team, fastmachinelearning/hls4ml-tutorial, 2025. URL: https://github.com/fastmachinelearning/hls4ml-tutorial, accessed: 2025-04-10.

[27] E. Lind, Ä. Pantigoso Velasquez, A performance comparison between CPU and GPU in TensorFlow, 2019.