High performance platform to detect faults in the Smart Grid by Artificial Intelligence inference

Le Sun Leire Muguira Jaime Jiménez Wang Yong Jesús Lázaro

Inferring faults throughout the power grid involves fast calculation, large scale of data, and low latency. Our heterogeneous architecture in the edge offers such high computing performance and throughput using an Artificial Intelligence (AI) core deployed in the Alveo accelerator. In addition, we have described the process of porting standard AI models to Vitis AI and discussed its limitations and possible implications. During validation, we designed and trained some AI models for fast fault detection in Smart Grids. However, the AI framework is standard, and adapting the models to Field Programmable Gate Arrays (FPGA) has demanded a series of transformation processes. Compared with the Graphics Processing Unit platform, our implementation on the FPGA accelerator consumes less energy and achieves lower latency. Finally, our system balances inference accuracy, on-chip resources consumed, computing performance, and throughput. Even with grid data sampling rates as high as 800,000 per second, our hardware architecture can simultaneously process up to 7 data streams. [1]

Keywords: AI, FPGA, Smart-Grid

1 Introduction

Electrical power systems suffer occasional sudden faults associated with damage to equipment, service suspensions, or even the death of employees. Fast and precise high-impedance fault detection is crucial, and it is still an open issue in the electric industry. To avoid these, fast and precise high-impedance fault detection, among others, is crucial, yet still an open issue in the electric industry. Fault identification can be understood as a binary classification, and AI technologies have been demonstrated to detect those faults. Nevertheless, automation, validation processes, and performance related to electric fault detection have revealed deficiencies that demand a first-rate intelligent system [1, 2]. For a long time, DNN models' inference, especially the uncompressed ones, has been predominantly deployed in CPUs and GPUs. However, approximating DNN has paved the way to FPGAs and ASICs, whereby better inferences have been reached —an accuracy decrease close to 1 % [3], and the current need for real-time and accelerating inference has driven deployments in custom hardware platforms. So, due to their flexibility and high performance, FPGAs have become efficient accelerators for CNN inference; for instance, Microsoft Brainwave and Xilinx DPU.

Designing and optimizing FPGA accelerators for CNN inference using hardware description languages are complicated and require a deep knowledge of the hardware. They can be managed using CNN acceleration platforms based on DPU —a comparison between different accelerator development flows was presented in [3]. The DPU is an IP core that enhances the performance of CNN inference on the FPGA [4]. It allows various CNN architectures, efficient and versatile regarding energy consumption, latency, and design time. Nevertheless, the selected CNN model has to be adapted to the DPU engine, which means to be compiled for the instructions supported by the DPU.

The method presented in this paper is versatile and can be applied to various scenarios. Although we have used it to detect faults in overhead lines, it can be used in other applications as well. Our approach is capable of processing data from different points in the grid simultaneously and in real-time.

2 Background

Big data facilitates access to massive historical information required by data-driven procedures. These approaches can be divided into statistical, signal processing, and machine learning methods. Machine learning is a notable research topic applied in multiple engineering fields [5]. Binary classification tasks can learn from a set of information, effectively distinguishing two groups. These are gradually implemented using machine learning techniques in order to be automatized. An electrical fault is detected by estimating the system's state with a mathematical model and analyzing the produced deviation from the actual output.

[1]This work has been supported, within the fund for research groups of the Basque university system IT1440-22, by the Department of Education and, within PILAR ZE-2020/00022 and COMMUTE ZE-2021/00931 projects, by the Hazitek program, both of the Basque Government; the latter also by the Ministerio de Ciencia e Innovación of Spain through the Centro para el Desarrollo Tecnológico Industrial (CDTI) within the project IDI-20201264 and IDI-20220543, and through the Fondo Europeo de Desarrollo Regional 2014-2020 (FEDER funds).

There are different machine learning families. Shallow methods, such as ANN and SVM models, are ineffective in learning about complex systems under uncertainty and non-linearity. Nevertheless, deep learning techniques have fixed shallow learning models' shortcomings. Deep learning models better solve high-dimensional input classification problems [1]. In contrast with shallow ones, deep learning methods obtain high-level characteristics from the neurons located in numerous layers, which learn from unstructured and structured information, emulating the human brain for decision-making.

Deep learning techniques have considerable interest in the power industry and are proliferating. Pattern recognition algorithms based on deep learning have been deployed in image and speech recognition. However, in Medium Voltage, there is a lot of dispersal literature in the deep learning-based analysis [6]. Performance differences between seven shallow models (ANN, SVM, NB, BDT, DF, DJ, and QSVM) and four deep learning models (DNN, LSTM-RNN, DNN, and DBN) were investigated in [1]. DNN, LSTM-RNN, DNN, and DBN were chosen for electrical fault detection. These three deep learning models were chosen because of their superiority for static classification tasks.

2.1 Deep learning in the electrical fault detection task

Due to the progress in AI, multiple fault-sensing techniques have been implemented with machine learning technologies.

A realistic and modern dataset for power line fault detection provided by ENET Centre at VSB can be used to measure the systems' performance [1, 6].

It is the most extensive known medium voltage overhead data in open source.

Electrical fault detection was automated in [1] with an Ensemble Deep Learning Approach. The computational complexity in terms of time and resources was high because three deep learning techniques were used to deploy the ensemble system: DNN, LSTM-RNN, and DBN. They suggested implementing parallel computing capabilities with a cluster of machines to solve the high computational overhead problem.

Table 1 depicts different implementations of machine learning methods used for electrical fault detection tasks.

Table 1: Machine learning methods used for electrical fault detection tasks.

Aim	Method	Ref.
Electrical fault detection	DNN, LSTM-RNN and DBN	[1]
Fault localization in the power grid	CNN	[7]
Detection of fault power line in a medium voltage overhead covered conductor	LSTM	[6]
Identification of faults in high-voltage cable	CNN	[7]
Identification of faults in high-voltage transformer	CNN	[8]
HIF detection and localization	Semi-supervised and probabilistic learning	[9]
HIF detection	Empirical model decomposition combined with a NN	[10]
HIF detection	Discrete wavelet transform combined with LSTM	[11]

2.2 FPGA implementations for Deep learning

In [5] also, LSTM-RNN was used to deal with real-time fault detection and isolation on the More Electric Aircraft. They compared the performances of the fault detection and isolation on the FPGA (Xilinx Virtex UltraScale FPGA VCU118 Evaluation Kit platform), applying different types of Long Short-Term Memory networks in terms of accuracy and time and resource consumption. CNN1D was compared with their method, concluding that it could be an alternative in terms of accuracy but wasting more hardware resources.

Deep neural networks can achieve high accuracies but entail massive computational costs. Most of the CNN-based classifiers rely on key modules. For example, ResNet in residual blocks, DeseNet in dense blocks, GoogleNet in the inception module, MobileNet v1 in depthwise convolution, and MobileNet v2 in inverted residual blocks. CNN architecture extensive review is presented in [12]. Different architectures are compared, showing that MobileNet v1, MobileNet v2, and DenseNet present the best performance on applications with restricted resources like IoT.

In [2], a compact CNN architecture engine was proposed for HIF detection and classification at the sensors, protective relays, and other intelligent electronic devices with online monitoring capabilities, decreasing computing complexity in HIF detection. In [3], an edge device was presented with a processing unit based on CNN with different mathematical operations and layers for resource-limited applications.

The DPU-IP is an FPGA-based hardware design released by Xilinx Inc. It supports multiple deep learning essential functions making possible an efficient implementation of different CNNs on FPGA. Multiple standardized CNN architectures' inference can be accelerated conveniently by using DPUs and improving schedule efficiency [4]. Multiple works are related to CNN inference acceleration, as explained in [4]. The reasons for the low schedule efficiency problem on the DPU-based platform were analyzed. The task scheduler was optimized in [4], developing a task assignment based on the Vitis AI software stack. DPU-based accelerator on FPGA was also evaluated in [13], demonstrating that DPU outperformed GPU and CPU concerning power efficiency. They proposed a CNN architecture for classification and compression. Power consumption and resource utilization were considered the most critical performance metrics. They obtained better performance than Nvidia

Jetson TX2 with 16-bit floating-point precision and ARM Cortex A57 CPU with 32-bit floating-point precision in terms of throughput being the accuracy loss because of the model compression insignificant.

In order to take advantage of both FPGA's flexibility and CPU's generality, a hybrid CPU-FPGA device was selected in [4], the Xilinx Zynq UltraScaleC MPSoC zcu104, and the DPU-based acceleration platform was compared with NullHop [14], another representative CNN accelerator. The compression engine based on a deep-learning design proposed by [3] is also implemented by a DPU on ZCU104 FPGA Evaluation Kit. They gave design recommendations to decrease the CNN accelerators' design time.

3 Algorithm

3.1 Data

The data set [15] we used in the experiment comes from "Enet Centre, VSB — T.U. of Ostrava" named "VSB Power Line Fault Detection." Faults in electric transmission lines —among others, of Smart Grids— can lead to a destructive phenomenon called partial discharge. Real-time, fast and high-throughput analysis of whether there are faults on the grid is meaningful work. The signal data comes from the real overhead power lines, not a lab, and they contain a lot of background noise [16]. These signals were measured by a device with lower sampling rate deployed on more than 20 different locations. This implies that the spectrum of noise and quality of partial dischargh's are so different from each other.

A total of 8712 measurement samples contains data for 3 phases in the dataset. Each sample contains 800,000 measurements of a power line's voltage over 20 ms. This leads to a sample rate of $50\,\text{Hz} \cdot 8e5 = 40\,\text{MHz}$. As the underlying electric grid operates at 50 Hz, each signal covers a single complete grid cycle (see Fig. 1). Another part of the dataset labels each phase at each measurement to state whether there was a failure. The method used to label the data is not specified by the author.

In order to use this data, some preprocessing must be done. Data is divided according to their fault tag.

The first step is to normalize the training and testing datasets using the `StandardScaler` method. This method is considered one of the most suitable normalization methods for deep learning [17]. It standardizes a feature by subtracting the mean and then scaling it to unit variance.

Due to the scarce nature of errors in the line, only 6.026 % of the data corresponds to a faulty signal. This quantity is relatively insufficient compared to the full standard sample size. Therefore, it is necessary to balance the number of samples of the two datasets in the data preprocessing stage. The ideal situation is that the proportion of normal and fault samples is the same. In the 8712 sets of measurement data, 525 sets are labeled as faulty. That is relatively rare fault data. These failure data accounted for 6.026 % of all data. 80 % of the sets are selected for training, and the remaining 20 % for validation. At this point, there are 420 faulty sets and 6549 error-free sets. The proportion of faulty sets is too low. We have selected 500 sets from the error-free group and merge them with 420 sets of faulty data to make the training dataset. At this point, the data collation and balance are completed. To reduce processing time, we selected half of the total measurement points per set, which was 400,000 out of 800,000 points, as the research target.

3.2 AI model

This article focuses on a hardware implementation of the AI model; therefore, the number and type of neural nodes are constrained. All supported AI layers and operators will be mapped to the AI computing engine inside the DPU, which must be based on a CNN, a limitation of the DPU.

The proposed AI model consists of 4 `conv` based structures (see Fig. 3). Each module normalizes the values through the `BatchNorm` layer. Then it passes the variables to the `MaxPooling` layer through the `ReLu` activation function to do down-sampling calculations to reduce the computation complexity.

It should be noted that the input grid voltage data is one-dimensional sequential data. This data is converted to two-dimensional to meet the limitations of the DPU. The second dimension is meaningless, so the convolution kernel only moves laterally in the dimension of the valid data.

The frame is 25,000 long. This size means that every frame covers 0.625 ms of the signal ($T_{frame} = 25e3/8e5 \cdot 20\,\text{ms} = 0.625\,\text{ms}$).

The AI model consists of two parts, one is a convolutional layer, and the other is a fully connected layer. The input data is 1-dimensional, and after convolution, it will expand to 32-dimensional data under the scan of a 1-dimensional convolution kernel with a length of 16 and 10. The convolution kernel extracts features every two strides —Stride is the number of element shifts over the input matrix. After the convolutional layer, `BatchNorm2d` is added to normalize the data. The pool layer is an operation to extract meaningful information, which can remove unimportant information and reduce computing overhead. The possible overfitting is reduced through the dropout layer. After the 4-times convolutional structure, the data is flattened to 1 dimension and then gradually converge to 1 output result after three fully connected layers for binary classification inference. A `ReLu` activation function and a 20 % dropout layer are passed between each fully connected network. Figure 2 shows the code of the system.

Finally, the neural network's output is passed through a sigmoid function to get an inference from 0 to 1. If the final output is greater than or equal to 0.5, it is considered a fault.

The training uses an Adam optimizer and Binary Cross Entropy Loss with Sigmoid as the final activation function. The learning rate is 0.0001, and the training is done 100 times.

3.3 AI model modification

Based on the previous model, some modifications were explored. The main objective was to improve the throughput. Two different approaches were explored: (a) Maintaining the time the dataset covers but decimating the number of points by a factor (T). (b) Maintaining the dataset size but increasing the time it covers by a factor (S).

Figure 4 shows an example of both approaches. As can be seen, `T` decimates the data by a factor of 2 and `S` increases the time by a factor of 2.

Table 2 shows the impact of the different decimation factors on the accuracy of the model. The results show that decimating the data only slightly reduces the accuracy of the model. On the other hand, increasing the time of the dataset has a significant impact on the accuracy of the model.

Table 2: The reduction ratio of the accuracy of the model applying different decimation factors. `T` reduces the dataset and `S` increases the time span.

Decimation Factor	S	T
1	1.00	1.00
2	-7.41	-3.18
4	-7.06	-3.53
8	-8.11	-3.88

On Alveo U50DD, we deployed 2 DPUs, each of which has three computing engines. These 2 DPUs used almost all available on-chip resources. We divided the data set into several threads to write into/read from the accelerator card. These threads provide data to the DPUs using up to four HAL interfaces (see Fig. 5).

4 Hardware architecture design

The design is based on the Alveo U50DD FPGA accelerator board. It requires a CPU to work, and up to three DPUs can be deployed inside. The System hardware we use is an Intel I7 9700K, 80G RAM, and the Alveo U50DD accelerator plugged in a PCIe gen3 port.

There are different DPU versions on the Alveo accelerator. Some prioritize throughput over latency, and some do the opposite. In our approach, we are using the DPU for latency optimization.

The XDMA shell must be used when designing for this kind of board. It is responsible for the communication between the accelerator and the CPU, and the host CPU side controls it. The XDMA shell forms a logic bridge between the host's main memory and the HBM on the accelerator. The FPGA on the accelerator side is in charge of scheduling data from the onboard HBM memory. The overall design is shown in Fig. 6.

The next step is the migration of the AI model. In our case, from the general PyTorch [18] model to the DPU. The basic steps are: (a) Designing an appropriate AI model in the standard PyTorch design framework and introducing data set to train the model. (b) Saving in a checkpoint the trained model's subgraph, bias, and weights. (c) Performing the quantization and compilation processes. After compiling, an archive containing subgraph information, DPU type, device type, and other parameters used by Vitis AI runtime to implement the model on the DPU is obtained.

Once the model is trained, the first step is quantizing the model (see Fig. 7). The numeric representation is converted to suit the FPGA capabilities better. When training neural networks, 32-bit floating-point weights, and activation values are generally used. Converting the 32-bit floating-point weights and activations to the 8-bit integer format can reduce computing complexity without losing prediction accuracy [19]. The fixed-point network model requires less memory bandwidth and resources, thus providing faster speed and higher power efficiency than the floating-point model.

The second step is compiling the model. Previously the feasibility of the model must be ensured in that all layers, operators, and tensors of the designed AI model are supported by the compiler (see Fig. 8). Then, the designer informs the AI compiler of the specific DPU and device models.

The compiler eliminates the differences between these frameworks and provides a unified representation. When the model contains operations not supported by the DPU, some subgraphs are created and mapped to the CPU. After the above work is completed correctly, the AI model can run on the DPU-based Alveo accelerator in the Vitis AI runtime environment.

5 Results

Two DPU instances were deployed on the Alveo card. In total, 10 million batches were tested. Each batch has 25,000 samples and includes 3 phase voltage values, measured as 32 bit floating point numbers. Such a group of vectors constituted by 25,000 samples was considered as one frame —for the case of no decimation. The inference accuracy on the training set reaches 100 % and 91.2 % on the test set. Due to the translation of number types between the PyTorch and DPU, the accuracy is slightly lower than the original PyTorch model.

Back to the very beginning of the question discussed, the result of detecting grid failure is between yes and no. This is a binary classification problem. Equation (1) represents the Sigmoid function. The result of the Sigmoid function takes a value between 0 and 1, which answers the question of yes or no. When the value obtained by this activation function is greater than a certain threshold, the result will be regarded as yes, and vice versa.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

The Sigmoid function is executed in the CPU because there is no corresponding computing core in the DPU architecture. It has little impact since this activation function will only be called once when the binary classification result is finally obtained. We measured that with the sigmoid activation function applied to two DPUs, the inference speed is 432.3 FPS. However, without the Sigmoid, the results go up to 434.1 FPS, leading to an improvement of 0.4 %.

Over the basic structure, we have decimated the dataset (see Table 2). Figure 9 depicts the results. The accuracy evaluation is performed on the test dataset after the number of regular and faulty is balanced. Both datasets are randomly sampled on the overall dataset. The percentage of precision is calculated as if the inference result is equal to the label of the dataset, then it is correct. This label is set in a cycle by cycle basis.

Figure 9 shows that decimation of 8 improves accuracy and FPS compared to smaller decimation factors. A decimation of 16 significantly reduces accuracy, rendering the system unusable. In many studies [20], the MCC is used to measure the correlation between the inference data and the original data. In addition, the F1 score is used in statistical analysis of binary classification. It is calculated from the precision and recall of the test, where the precision is the number of true positive results divided by the number of all positive results, including those not identified correctly, and the recall is the number of true positive results divided by the number of all samples that should have been identified as positive. The value of F1 score ranges from 0 to 1. A higher F1 score means that inference has better accuracy for both false positives and false negatives. MCC ranges from -1 to 1. A higher MCC score indicates a higher correlation in the inference results. Negative numbers indicate negative correlation, near 0 indicates no correlation. Table 3 shows the evaluation results, as the decimation sampling window is 8 samples.

Table 3: Statistical analysis of classification results.

Precision	0.88
Recall	0.89
F1	0.88
MCC	0.77

If we calculate the throughput using Eq. (2):

$$T = FPS \cdot Fs \cdot 32\,\text{bit} \tag{2}$$

Where Fs is the number of samples per frame, FPS is the number of frames per second, and T is the throughput in Mbit s^{-1}. The maximum throughput under this AI model for 2 DPU implementation for a decimation factor of 8 and 11,733 FPS is about 1173 Mbit s^{-1}. The Alveo U50DD's power consumption is 49 W. With this speed, the system can process 7 streams from different measuring units (see Eq. (3)).

$$Streams = FPS \cdot T_{frame} = \lfloor 11{,}733\,\text{s}^{-1} \cdot 0.625\,\text{ms} \rfloor = 7 \tag{3}$$

In this experiment, two units of 3-engine DPUs were used. The utilization of LUT and on-chip RAM are close to 60 % for the Alveo application acceleration design flow. The design guideline advises utilizing less than 80 % to obtain consistent implementations. For designs between 60 % and 80 %, congestion difficulties may arise. If we deploy only one DPU, it should be possible to deploy more functional areas in the same FPGA chip. For reference, the system resource usage is summarized in Table 4.

Table 4: Resource usage of the system

		1		2	
Resource	Available	Utilization	%	Utilization	%
LUT	870,720	251,632	28.90	445,468	51.16
LUTRAM	402,720	19,921	4.95	35,096	8.71
FF	1,743,360	371,399	21.30	668,243	38.33
BRAM	1,344	276	20.50	427	31.77
URAM	640	192	30.00	384	60.00
DSP	5,952	1,566	26.31	3,128	52.55

6 Comparison

The preprocessing stage, like the ones described in [1], dramatically impacts the inference of the DNN model. It is essential to extract features from data before feeding it to an AI model. A well-designed procedure like [21], combined with a clever algorithm, can increase inference accuracy. Nevertheless, in the same way, in addition to the preprocessing stage, the computing platform and computing power are also important. Because in some scenarios, such as power grid scenarios, the requirement for inference speed is very high. This work focuses on finding the balance between speed and accuracy rather than deploying feature extraction and preprocessing algorithms on the CPU. When the sampling window is suitable, the accuracy is good in the current rough preprocessing mode mentioned above.

[6] also discusses the use of RNN or LSTM neurons for inference of sequential signals such as VSB. RNN is a basic feed-forward neural network with a feedback connection to its previous state in each hidden layer. However, the neurons of RNN are entirely different from the structure of CNN. FPGA cannot change the internal structure at any time, so the convolution kernel inside the current DPU cannot be directly applied to RNN neurons. When the DPU of the RNN neuron structure matures, inference accuracy may improve further.

Table 5 shows the accuracy of our proposal with other present in the literature. Compared with other studies, the experimental method in this paper has no obvious advantage in inference accuracy. However, the inference speed of this paper is much higher than other studies as Table 6 and Table 7.

Table 5: Accuracy results from different works.

Study	Technique	Accuracy
[22]	SAE+DNN	89 %
[23]	Training Interference+CNN	96 %
[24]	PMU placement+CNN	95 %
[1]	Double PSO+DNN	97 %
[6]	BLSTM+RNN	98 %
Our study	FPGA accelerator + DNN	91 %

AI tools and DPUs from Xilinx can be deployed on heterogeneous accelerators. FPGA PL and ARM CPUs can exist on modern SoCs like Xilinx Zynq UltraScale+ MPSoC [3]. The combination of the two can be competent for many applications. For example, AI DPU can be deployed on its PL to connect with the ARM CPU using the AXI bus. This structure is more efficient and concise than our project's DPU — HBM memory — PCIe — CPU connection. Nevertheless, the scale of their chips means that their performance cannot be compared to heterogeneous platforms.

Few studies have been done on the impact of computing performance on AI inference. And indeed, some people deployed enhanced AI models on cloud services, but their specific algorithms are not deep learning algorithms [25]. We can only use our data to simulate the performance comparison in this case. In this project, the AI model must first complete the training on CPU or optionally on GPU and will be transplanted to the DPU of the FPGA when the requirements are met. Therefore, the AI models before and after transplantation have the same structure and nodes. In this way, we can use the performance indicators of running the same AI model on the CPU, GPU, and DPUs to speculate on the actual situation.

Table 6 shows a throughput comparison. In [3], each frame is not identical to ours, so the FPS is not comparable between the two studies. The speedup percentage of the CPU can be compared.

Table 6: Throughput results from different works. The FPS is not comparable between the studies. However, the speedup percentage of the CPU can be compared.

Accelerator	DPU (FPS)	CPU (FPS)	Speedup (%)
UltraScale+ MPSoC [3]	123.46	22.20	5.56
Alveo (2 x 3E)	1031.86	12.92	79.86

Next, we use the CPU as the benchmark indicator to calculate the speedup ratio of the DPU. As a performance comparison, we used: (a) I7-9700k to test the CPU inference performance. (b) Tesla V100 GPU computing card running on the cloud to test GPU inference performance. (c) A single DPU. (d) Two DPUs. The test data scale is 76,500 frames, 7650 MB overall. 25,000 dataset was used for the comparison as a baseline. The results are summarized in Table 7.

7 Conclusions

In this paper, we used Xilinx's latest tool Vitis AI to deploy the AI inference FPGA core DPU in the Alveo accelerator. Our electronic platform quickly finds faults in the power grid by fast processing specifically designed and trained AI. This heterogeneous computing architecture features high computing performance, throughput, inference accuracy, and few on-chip resources consumed. Even with grid data sampling rates as high as 800,000 per second, the AI model in our hardware architecture can process up to 7 such data streams simultaneously.

Table 7: Comparison results between CPU, GPU, and DPU with 25,000 datasets.

Platform	FPS	Power (W)	Throughput (Mbit s^{-1})	FPS/W
CPU	12.92	95.00	10.34	0.14
GPU	381.19	300.00	304.95	1.27
DPU (1 × 3E)	519.50	49.00	415.60	10.60
DPU (2 × 3E)	1,031.86	58.90	825.49	17.52

The paper also details porting standard AI models to Vitis AI and discusses its limitations and possible implications. Although AI models are standard, adapting them to FPGAs has demanded a series of transformations.

References

[1] Wisam Elmasry and Mohammed Wadi. EDLA-EFDS: A novel ensemble deep learning approach for electrical fault detection systems. *Electric Power Systems Research*, 207:107834, jun 2022.

[2] Shiyuan Wang and Payman Dehghanian. On the use of artificial intelligence for high impedance fault detection and electrical safety. *IEEE Transactions on Industry Applications*, 56(6):7208–7216, nov 2020.

[3] Abdelrahman S. Hussein, Ahmed Anwar, Yasmine Fahmy, Hassan Mostafa, Khaled Nabil Salama, and Mai Kafafy. Implementation of a DPU-based intelligent thermal imaging hardware accelerator on FPGA. *Electronics*, 11(1):105, dec 2021.

[4] Jiang Zhu, Lizan Wang, Haolin Liu, Shujuan Tian, Qingyong Deng, and Jianqi Li. An efficient task assignment framework to accelerate DPU-based convolutional neural network inference on FPGAs. *IEEE Access*, 8:83224–83237, 2020.

[5] Qin Liu, Tian Liang, Zhen Huang, and Venkata Dinavahi. Real-time fpga-based hardware neural network for fault detection and isolation in more electric aircraft. *IEEE Access*, 7:159831–159841, 2019.

[6] Dilshad Ahmad, Shaorong Wang, and Mohd Alam. Long short term memory based deep learning method for fault power line detection in a mv overhead lines with covered conductors. In *2020 21st National Power Systems Conference (NPSC)*, pages 1–4, 2020.

[7] Wenting Li, Deepjyoti Deka, Michael Chertkov, and Meng Wang. Real-time faulted line localization and pmu placement in power systems through convolutional neural networks. *IEEE Transactions on Power Systems*, 34(6):4640–4651, 2019.

[8] Jiajun Duan, Yigang He, Bolun Du, Ruaa M. Rashad Ghandour, Wenjie Wu, and Hui Zhang. Intelligent localization of transformer internal degradations combining deep convolutional neural networks and image segmentation. *IEEE Access*, 7:62705–62720, 2019.

[9] Qiushi Cui and Yang Weng. Enhance high impedance fault detection and location accuracy via μ -PMUs. *IEEE Transactions on Smart Grid*, 11(1):797–809, jan 2020.

[10] Himadri Lala and Subrata Karmakar. Detection and experimental validation of high impedance arc fault in distribution system using empirical mode decomposition. *IEEE Systems Journal*, 14(3):3494–3505, sep 2020.

[11] K. V. Shihabudheen. Detection of high impedance fault using advanced ELM-based neuro-fuzzy inference system. In *Control and Measurement Applications for Smart Grid*, pages 397–408. Springer Nature Singapore, 2022.

[12] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artif Intell Rev*, 53(8):5455–5516, apr 2020.

[13] Murad Qasaimeh, Kristof Denolf, Alireza Khodamoradi, Michaela Blott, Jack Lo, Lisa Halder, Kees Vissers, Joseph Zambreno, and Phillip H. Jones. Benchmarking vision kernels and neural network inference accelerators on embedded platforms. *Journal of Systems Architecture*, 113:101896, feb 2021.

[14] Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Iulia-Alexandra Lungu, Moritz B. Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu, and Tobi Delbruck. NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps. *IEEE Transactions on Neural Networks and Learning Systems*, 30(3):644–656, mar 2019.

[15] VSB Enet Centre. Vsb power line fault detection, dec 2018.

[16] S. Misak, J. Fulnecek, T. Vantuch, T. Burianek, and T. Jezowicz. A complex classification approach of partial discharges from covered conductors in real environment. *IEEE Transactions on Dielectrics and Electrical Insulation*, 24(2):1097–1104, apr 2017.

[17] Yanbin Li, Yuxin Huang, Fuwei Jia, Qingsong Zhao, Ming Tang, and Shougang Ren. A gradient deconvolutional network for side-channel attacks. *Comput. Electr. Eng.*, 98:107686, March 2022. Genérico, utiliza StandardScaler y es de CompElecEng y del 2022.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[19] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 7686–7695, Red Hook, NY, USA, 2018. Curran Associates Inc.

[20] Davide Chicco, Valery Starovoitov, and Giuseppe Jurman. The benefits of the matthews correlation coefficient (MCC) over the diagnostic odds ratio (DOR) in binary classification assessment. *IEEE Access*, 9:47112–47124, 2021.

[21] Kunjin Chen, Tomas Vantuch, Yu Zhang, Jun Hu, and Jinliang He. Fault detection for covered conductors with high-frequency voltage signals: From local patterns to global features. *IEEE Transactions on Smart Grid*, pages 1–1, 2020.

[22] Yixing Wang, Meiqin Liu, and Zhejing Bao. Deep learning neural network for power system fault diagnosis. In *2016 35th Chinese Control Conference (CCC)*. IEEE, jul 2016.

[23] Wei Zhang, Chuanhao Li, Gaoliang Peng, Yuanhang Chen, and Zhujun Zhang. A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment and different working load. *Mechanical Systems and Signal Processing*, 100:439–453, feb 2018.

[24] Wenting Li, Deepjyoti Deka, Michael Chertkov, and Meng Wang. Real-time faulted line localization and PMU placement in power systems through convolutional neural networks. *IEEE Transactions on Power Systems*, 34(6):4640–4651, nov 2019.

[25] Weiliang Li, Yan Li, Jiawei Shi, Kai Chao, and Xiaoliang Zhang. Power grid fault detection method based on cloud platform and improved isolated forest. In *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. IEEE, mar 2021.

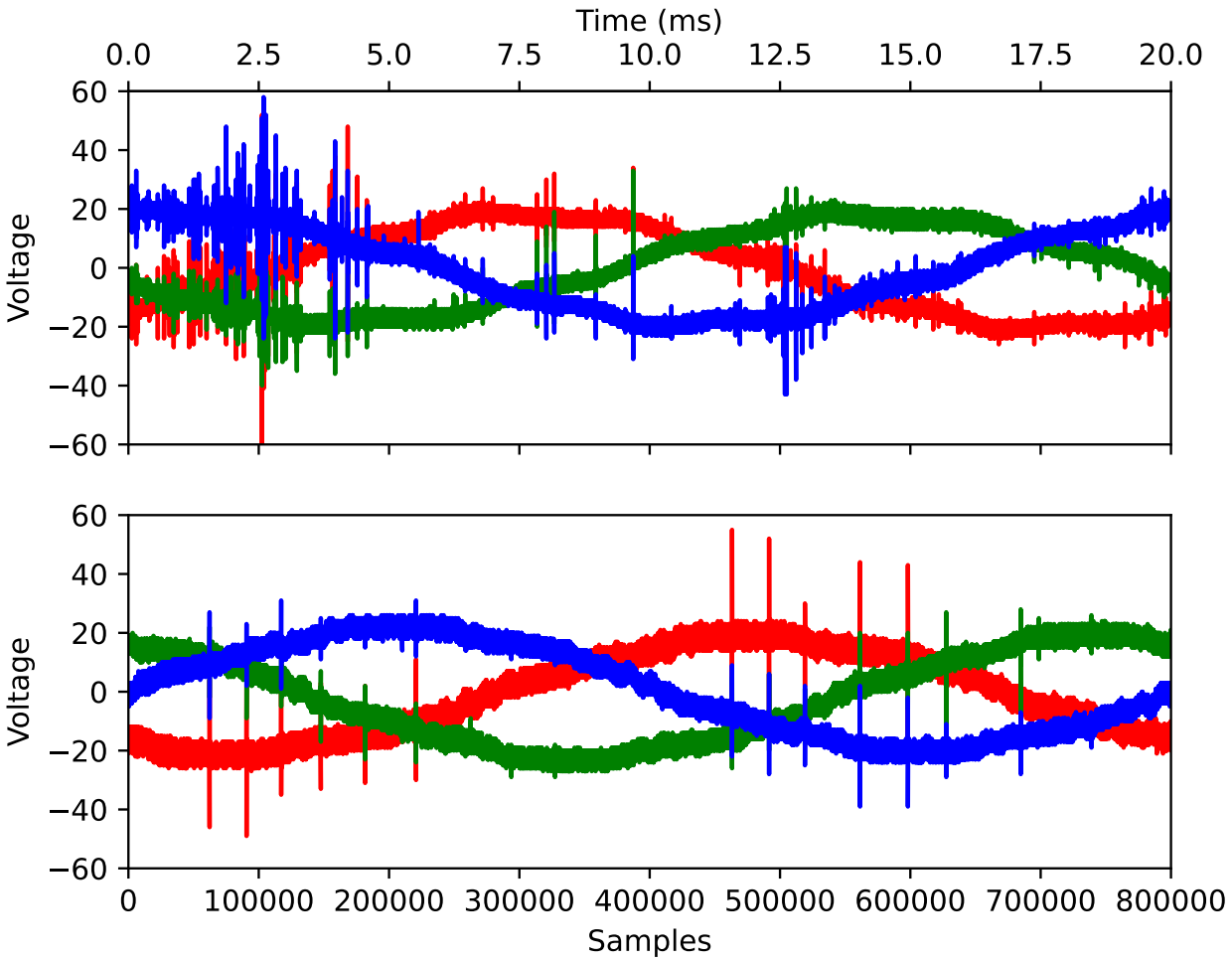

Figure 1: Examples of the dataset. The top signal shows a fault in the electric transmission line, while the bottom signal is normal. Both signals have high frequency components, but the signal with the fault has a more pronounced high frequency component. The IA algorithm is able to classify the different nature of the high frequency components and detect the fault. While we do not know the origin of the high frequency components, training the IA does not require understanding their origin. Instead, it requires a large dataset of signals with and without faults.

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=(16, 1), padding=0, stride=(2, 1))
        self.bn1 = nn.BatchNorm2d(32)
        self.rl1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d((2, 1))
        self.do1 = nn.Dropout(0.2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=(16, 1), padding=0, stride=(2, 1))
        self.bn2 = nn.BatchNorm2d(64)
        self.rl2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d((2, 1))
        self.do2 = nn.Dropout(0.2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=(10, 1), padding=0, stride=(2, 1))
        self.bn3 = nn.BatchNorm2d(128)
        self.rl3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d((2, 1))
        self.do3 = nn.Dropout(0.2)
        self.conv6 = nn.Conv2d(128, 64, kernel_size=(10, 1), padding=0, stride=(2, 1))
        self.rl6 = nn.ReLU()

        self.fc1 = nn.Linear(12096, 2048)
        self.rl7 = nn.ReLU()
        self.do7 = nn.Dropout(0.2)
        self.fc2 = nn.Linear(2048, 128)
        self.rl8 = nn.ReLU()
        self.do8 = nn.Dropout(0.2)
        self.fc3 = nn.Linear(128, 1)
```

Figure 2: This code defines the CNN with multiple convolutional, batch normalization, pooling, dropout, and fully-connected layers. The CNN uses ReLU activation and is specially designed for binary classification.

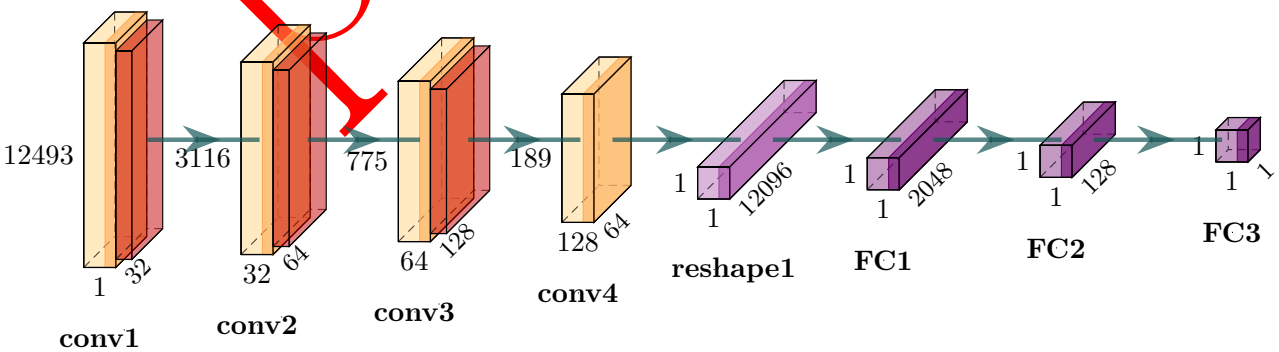

Figure 3: AI model. There are four initial convolutional steps with added MaxPool and Dropout to reduce the data size. Afterwards there are a series of transforms to end with the sigmoid function that provides a True/False output.

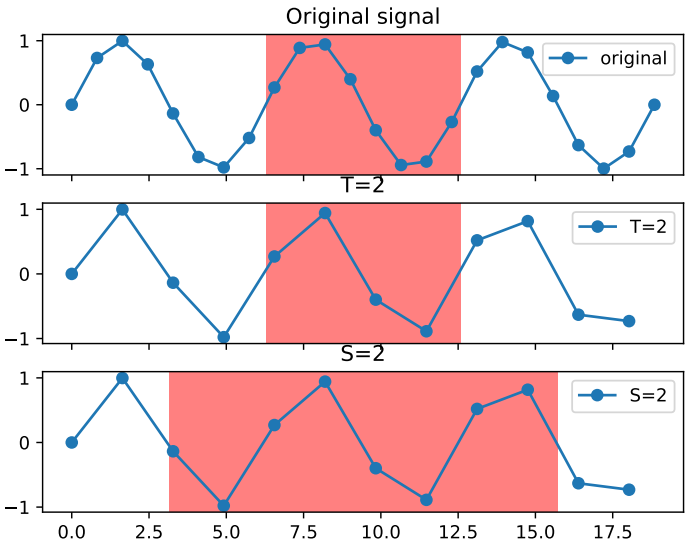

Figure 4: Example of both kinds of dataset' decimation. Top original signal, followed by both kinds of decimations: Same time (T) or same number of samples (S).

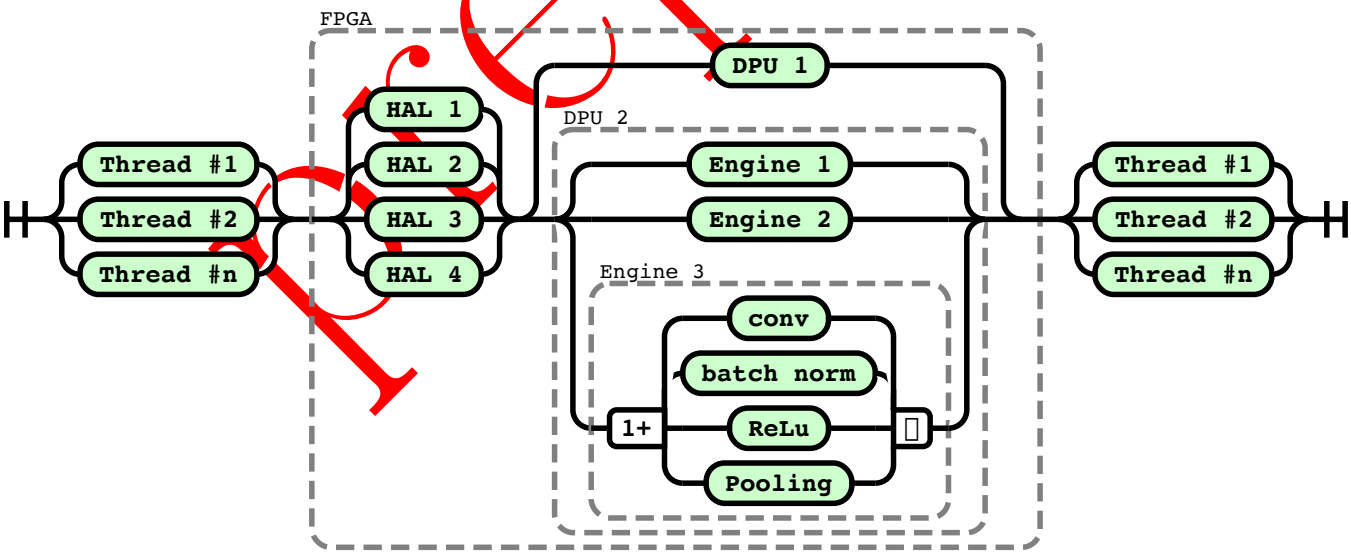

Figure 5: DPU inference engine structure. Inside the FPGA there are 4 HAL blocks that feed up to two DPUs. Each DPU is composed of 3 engines. Each engine has 4 cores for the different operations over the data.

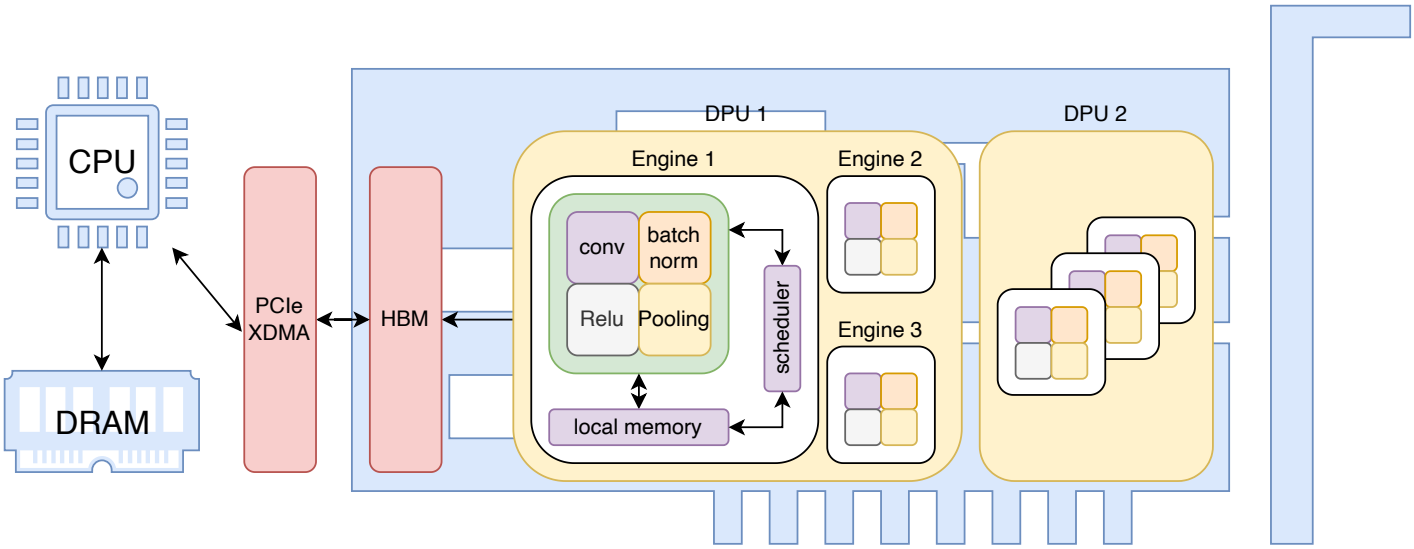

Figure 6: Overall system. The Alveo board is connected to the host PC using PCIe. Data is passed from main memory to the dedicated board memory (HBM). The two DPUs take the data and store the results into HBM.

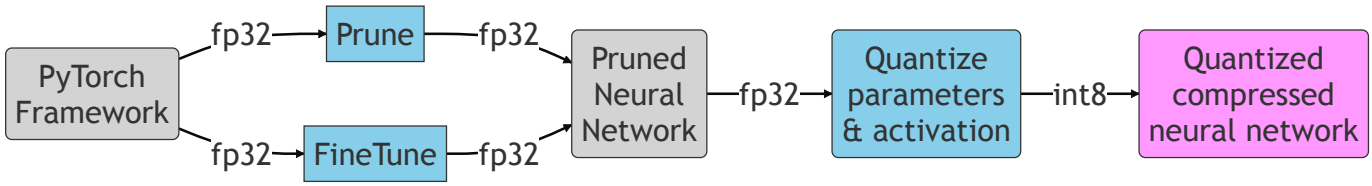

Figure 7: Quantization flow. The process using 32-bit floating point until we get a pruned neural network. Afterwards it is quantized for the final hardware implementation.

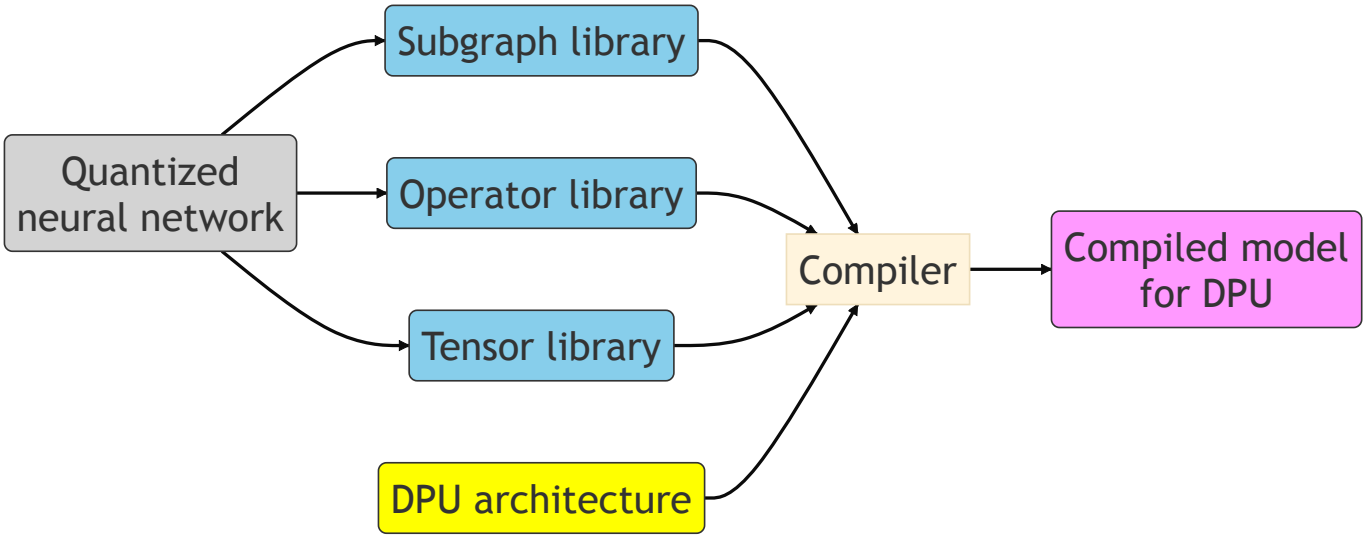

Figure 8: Compilation flow. We built the final complied model for the DPU using the quantized neural network as well as the hardware libraries and the DPU architecture.

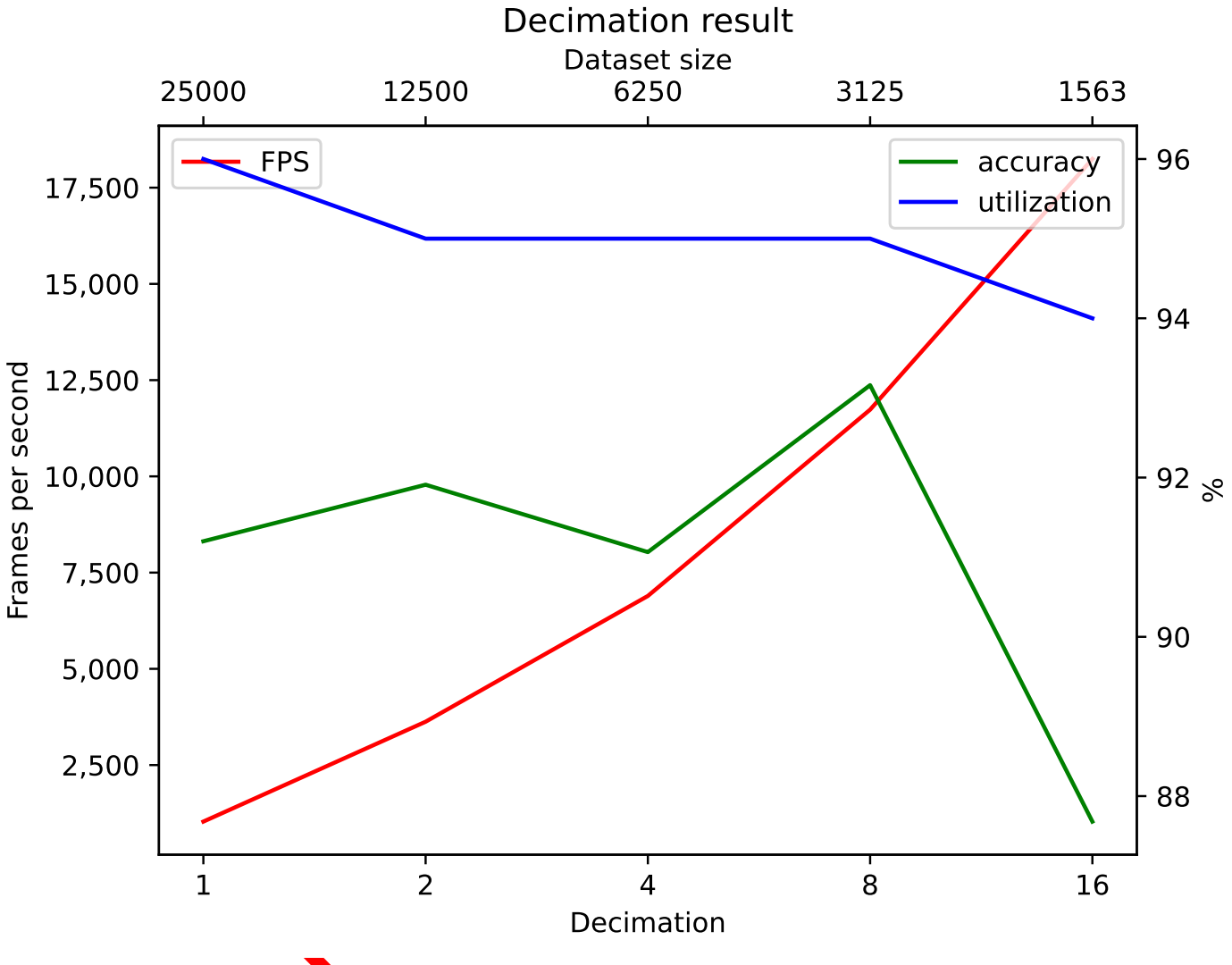

Figure 9: Decimation results. Even by reducing the dataset, the DPU is still processing data 95 % of the time (`utilization` value). Accuracy drops sharply after a decimation factor of 8.