AI Deployment Environments and Requirements

Matthew Lima, Krishu Nakarmi
Department of Electrical, Computer, and Biomedical Engineering
The University of Rhode Island
{matthew.lima, krishu.nakarmi}@uri.edu

***Abstract* — Deep learning is a rapidly expanding field of data science and is being used increasingly in everyday life. This field is unique in that the calculations required rely heavily on matrix operations. Computations have generally been optimized to be performed on Graphics Processing Units (GPUs) due to their highly parallel nature. While GPUs are still widely used during the training phase of Deep Neural Networks (DNNs), research is being done towards the best platform for deploying trained DNNs in increasingly complicated environments. We created and analyzed a custom DNN trained on the Modified National Institute of Standards and Technology (MNIST) dataset for Optical Character Recognition (OCR) on Handwritten Digits in order to establish quantitative metrics for determining the best deployment platform for inference.**

I. Introduction

Artificial Intelligence (AI) is becoming increasingly popular in mainstream use, but these complicated systems also demand intense computational ability to power them. OpenAI conducted a study and found that the amount of computational power used to train the largest AI models had doubled every 3.4 months since 2012 [1]. The current solution is to utilize cloud-based technologies to perform the heavy calculations and then forward the results of these calculations to the user on the edge. While this solves the computing problem it also introduces a slew of new issues, such as latency, network reliability, and security concerns. There is increasing interest in avoiding these issues by running AI inference on the edge by utilizing specialized hardware to perform intensive calculations. This hardware can be costly and require a large power budget [2]. Trying to determine the best solution to this problem is very difficult. This is why a set of quantitative metrics were sought after to determine what deployment environments drive target specific platforms over others.

Not all AI solutions are driven by the same requirements, but instead they are usually case-specific. For example, the computer-vision algorithm run by a self-driving car would have a higher power budget than one running on a cell phone, but it would also have a higher accuracy requirement due to its safety-critical nature. This example can be divided even further by taking into consideration the different hardware components within the cell phone itself, such as the GPU or possibly even a dedicated AI accelerator. Running the model on these sub-components can result in significant performance gain over running it on the Central Processing Unit (CPU), but it could also result in higher power draw for the battery-limited system. This example shows that both the deployment environment and target requirements can drive different hardware solutions for running different AI models. A set of criteria to analyze these platforms on can help to ease this decision-making process.

II. Related Work

Several studies have explored the performance of DNNs on various embedded platforms, including traditional CPU-based systems, GPUs, and FPGAs. Huang et. al. wrote a paper about the trade-offs of speed and accuracy associated with modern object detectors and some of the main aspects that influence the accuracy of object detectors [3].

In addition, multiple studies have been conducted that explore the use of FPGAs for accelerating DNNs. For example, Sui et al. proposed a hardware-friendly method for implementing Convolutional Neural Networks (CNNs) on FPGAs, which achieved high accuracy with low power consumption [4].

In a study by Velasco-Montero et al., they investigated the performance of popular open-source deep learning tools and real-time DNN inference on a Raspberry Pi [5]. They compared four different software frameworks and four popular DNN models for image classification and evaluated the results on accuracy, throughput, and power consumption.

With this paper we expand on these previous works and present an experimental study of implementing a DNN model trained on the MNIST dataset on five different platforms: an x86-64 CPU, an x86-64

CPU with a GPU accelerator, a Raspberry Pi, and a Xilinx FPGA development board (both with and without utilizing a custom logic-based accelerator). We evaluate the model on each platform in terms of accuracy, performance, and power efficiency, and compare the results to identify the strengths and limitations of each platform.

III. Deep Neural Network

A DNN is a class of machine learning that was designed to operate in a similar fashion to the behavior of the human brain [6]. It consists of a sequence of "neurons" that are "activated" and "deactivated" depending on their inputs. DNNs have been proven to solve a variety of complex computing problems that were previously very difficult, such as image detection and natural language processing. Many of them use a training method known as back-propagation, which relies on feeding the network an input, comparing the result to a known output, and making incremental adjustments to decrease the error of the model's output.

To evaluate the different platforms, a simple DNN architecture was chosen to implement the OCR model. The model has three layers: An Input Layer, a Hidden Layer, and an Output Layer. The Input Layer is a flattened 28x28 (768) pixel greyscale image with the datatype of float32. The Hidden Layer is comprised of 64 neurons and uses the ReLU activation function due to its simplicity to implement in an integer format. The Output Layer is a 1x10 vector that is passed through the SoftMax activation function to represent the final outputs as a confidence score. For training and evaluation data, the MNIST hand-written digit dataset was used because of its simplicity and volume [7]. 60K of the images were used as training data and the final 10K images were used to evaluate the model. A simple topographical image of the model can be seen in Figure 1.

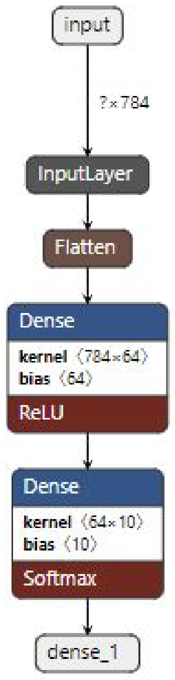


Figure 1: DNN Model Layout

IV. Platform Implementation

Each platform tested during this study presented opportunities for optimizing the performance and power consumption of the DNN model. In this section, each platform will be discussed, highlighting the different approaches considered for the specific hardware characteristics and constraints of each platform.

A. Windows Implementation

To implement the DNN model on Windows, the Keras library was used with a TensorFlow backend, which provides high-level Application Programming Interfaces (APIs) for building and training DNN models- [8]. The hyper-parameters of the model were set to a default learning rate, a batch size of 256, and 25 epochs for training. The Adam optimizer, a popular stochastic gradient descent optimization algorithm, was used to minimize the loss during training [9]. The model was trained and evaluated on a Windows machine with a Ryzen 7 5800X3D Processor and 64GB of DDR4 RAM, which took approximately 1 second per epoch. After training on the Windows platform, the model was evaluated by utilizing Windows Subsystem for Linux (WSL2) and achieved an accuracy of 97.25%.

The Windows implementation was also used to quantize the model for the FPGA portion of testing. This was done to both reduce the model size and improve the inference speed on the FPGA implementation. The post-training quantization was done using TFLite. An online tool called Netron was used to export the quantized weights and biases of the model to 8-bit and 32-bit integers, respectively [10] [11]. The ReLU activation function did not need any modifications since it easily ports to integer operations, but the SoftMax activation function (which requires floating-point operations) was discarded since it was not necessary for inference.

With a suitably accurate model trained, testing began first on the Windows platform. The other platforms used the Windows implementation as a baseline. To increase the size of the dataset for evaluation, the input image array was modified such that it repeats the 70K MNIST images 10 times (for a total size of 700K images). This was done to attempt to alleviate any memory bottlenecks that may be present in each system by performing the inference on a very large batch of data at once. The DNN model was first evaluated using only the CPU. Afterwards it was tested with a GeForce RTX 3060 GPU with 12GB of GDDR6 memory as a hardware based DNN accelerator. This was done in order to compare the differences between a CPU only implementation and one with GPU acceleration. In order to aid with power data collection for the Windows implementation, Open Hardware Monitor (OHM) and the NVidia System Management Interface (SMI) command-line utility were used to gather power draw data for the CPU and GPU, respectively [12] [13]. The other quantitative metrics gathered did not require external software to collect. All tests were run multiple times in order to get as close to the true geometric mean as possible.

The biggest challenge with the Windows implementation was installing and setting up the required software components (e.g. Python, TensorFlow, PyCharm, CUDA toolkit, etc.). There were many conflicts between software and system requirements, and their dependencies. Additionally, ensuring that the model was able to take advantage of all the system's available resources, such as the GPU, proved to be a challenge because of the numerous versions and backwards-incompatibility with cuDNN and TensorFlow.

B. Raspberry Pi Implementation

To evaluate the performance of the DNN model on a Raspberry Pi, a Raspberry Pi 4 Model B with 8GB of LPDDR4 running the 64-bit Raspberry Pi OS was used. The same version of TensorFlow and Keras used on the Windows system was installed on the Raspberry Pi to ensure no performance differences could be attributed to newer library versions. It achieved an accuracy of 97.25% on the MNIST dataset, the same as the accuracy achieved on the Windows platform. This indicates that the trained DNN model can be generalized across different platforms.

Implementing the DNN model on a Raspberry Pi presented different challenges compared to implementing it on a more powerful system, such as a Windows desktop. With limited processing power and limited memory, the model needed to be optimized to run efficiently on the Pi's CPU. The modified image array of 700K MNIST images exceeded the available memory on the Pi, causing the model to crash. Several iterations of code optimization were done to alleviate this issue, such as importing new modules and libraries, taking advantage of the various built-in functions of NumPy, and rewriting the code to load the dataset in batches of 100 images rather than loading the entire 700K array at once.

C. FPGA Implementation

To evaluate the performance of the DNN model on an FPGA the Xilinx PYNQ-Z2 FPGA development board was selected, which includes a Zynq 7000 System-on-a-Chip (SoC) and 512MB of DDR3 memory. The Zynq 7000 SoC contains dual-ARM cores capable of running a modified version of Ubuntu Linux, as well as an FPGA with 85K logic cells. Evaluating the DNN on the development board required porting the model in a way that maintained accuracy while reducing the size of the parameters. This porting process was broken into three stages: High-Level Synthesis (HLS) development, logic system development, and software development.

To develop the logic that performs the actual DNN calculations, Vitis-HLS was used to convert a C-based accelerator to a Register Transfer Level (RTL) implementation. The first step of this process was creating a Python script to convert the quantized weights and bi-

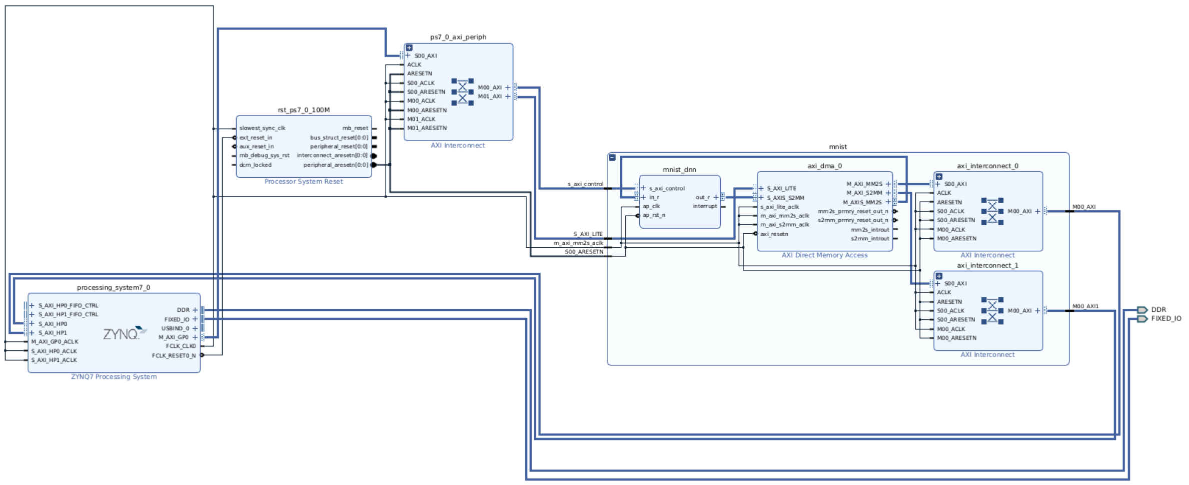


Figure 2: Vivado Block Diagram

ases to a set of constant C-arrays that could be accessed from a standard C header file. With the weights and biases imported, a Direct Memory Access (DMA) based transfer system was implemented to both transfer image files into the local FPGA SRAM and transfer the DNN results out of the accelerator. With the image data stored locally and the quantized weights and biases incorporated, performing the matrix and vector operations required by the DNN was fairly simple to implement. The most difficult challenge introduced during this development stage was trying to fit the logic on the relatively small FPGA. Various HLS pragmas were utilized to reduce the utilization of the component at the cost of limited throughput.

With the DNN accelerator component synthesized, the RTL was imported into Vivado and implemented onto a larger system. The overall logical design of the system can be broken into three major components: The Zynq Processor, the DMA, and the DNN accelerator. The Zynq processor provides an interface for the operating system running on the ARM processors to communicate with and control the other components. The DMA is used to ferry data between the Zynq processor and the DNN accelerator without hogging memory resources on either component. The DNN accelerator is used to accelerate the matrix and vector operations required to run the DNN. Figure 2 displays the block diagram used to define all component connections.

Software development for the FPGA system consisted of the creation of two Jupyter Notebooks and a Python-based software driver. The driver is used to provide a high-level interface for running inference with the DNN accelerator. It consists of two primary functions, inference and batch inference, that are used to run inference on either a single or batch of MNIST images. The first Jupyter Notebook is used to evaluate the accelerator on individual images with the benefit of plotting whatever image is being inferenced. This was used primarily during the development stage and was replaced by the evaluation Jupyter Notebook. This newer notebook was used to evaluate the performance of both the DNN accelerator and the performance of running the DNN on only the ARM cores. One key difference to note with the FPGA implementation is that since the model is quantized, the inputs need to remain as uint8 rather than being converted to float32 like the other platforms. The 70K images and results of the MNIST dataset were saved as NumPy arrays of uint8 to simplify loading the data onto the DDR3 memory.

V. Data Collection Methodology

The simplest metric to evaluate for each implementation was accuracy. This consisted of running inference on the 10K MNIST images not used to train the model and comparing the model's result with the expected result. Because each platform's implementation was completely deterministic, running inference on each platform was only required once. To thoroughly vet the system however, accuracy was run 100x on the x86-64 and Raspberry Pi based systems, but after seeing the results of those systems, it was only run on the FPGA implementations once.

Performance was captured by running the model numerous times and timing how long each batch of inference took to complete. Using TensorFlow, the number of operations required to run inference on an image was found to be 101.63K. Using this number and the time taken to run inference, the total number of operations per second can be calculated. For the x86-64 (both with and without GPU acceleration) and Raspberry Pi implementations, this calculation was made 100 times to increase confidence in finding the true mean performance for each system. The FPGA systems only ran this test 25 times due to the length of time each inference batch took to calculate.

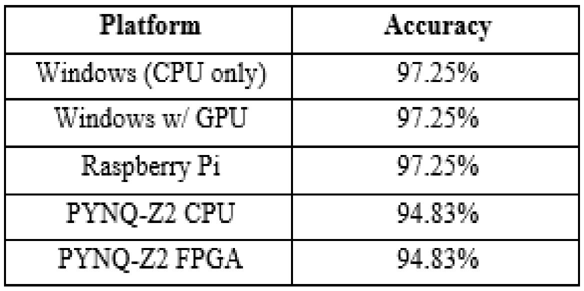
Power efficiency was calculated by determining how much power draw was required to run the DNN on each platform. To calculate this metric two things were required: the nominal power draw of the system and the power draw during inference. For the x86-64 implementations this was accomplished utilizing different software tools (OHM and NVidia-SMI) to record power draw and sampling over a 60 second interval both with and without running inference. After the data was collected, further processing was required. This included taking the mean of each group and subtracting the nominal power draw from the inference power draw. The Raspberry Pi and FPGA board did not have similar software tools to record power draw, so instead a power meter was placed in-line with each board's power supply and recordings were taken manually. Each implementations DNN power draw was then divided by their respective performance to obtain the operations per second per Watt each system achieved.

VI. Data Analysis

A. Accuracy

The accuracy of the DNN model on each platform is presented in Table 1. For each device the standard deviation of the sample set was zero, meaning that the accuracy for each platform was consistent with no deviation. The FPGA based systems have a notably lower accuracy than the other implementations, and this is due to running a quantized model with lower precision weights. The relatively small drop in accuracy shows that the chosen DNN is able to generalize well across different platforms even with the reduced precision inherent with model quantization.

Table 1: Accuracy Metrics

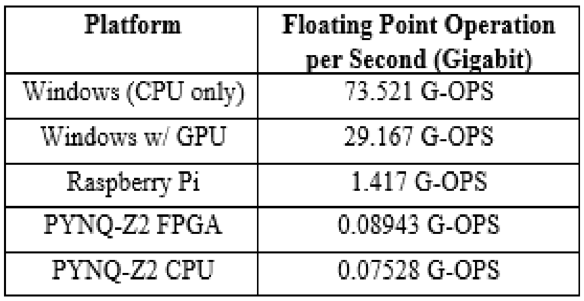


Platform	Accuracy
Windows (CPU only)	97.25%
Windows w/ GPU	97.25%
Raspberry Pi	97.25%
PYNQ-Z2 CPU	94.83%
PYNQ-Z2 FPGA	94.83%

B. Performance

Table 2 shows the performance results of each platform as billions of operations performed per second. As shown in the table, the performance of the DNN model on the Windows system was significantly faster than on the Raspberry Pi and Xilinx FPGA development board. This is likely due to the higher processing power and memory of the Windows system. The performance of the GPU accelerator compared to the CPU on the Windows system is significantly lower, which was not the expected behavior. This is most likely attributed to the very small size of the DNN model, since the overhead of transferring data to the GPU is probably significantly longer than the time required to perform the actual DNN operations. To test this theory larger batch sizes were used during inference, but due to limitations with TensorFlow this could not be tested as thoroughly as desired. The rest of the performance metrics look as expected, with the Raspberry Pi having decent performance, followed by the FPGA and finally the Zynq ARM processors.

Table 2: Performance Metrics



Platform	Floating Point Operation per Second (Gigabit)
Windows (CPU only)	73.521 G-OPS
Windows w/ GPU	29.167 G-OPS
Raspberry Pi	1.417 G-OPS
PYNQ-Z2 FPGA	0.08943 G-OPS
PYNQ-Z2 CPU	0.07528 G-OPS

C. Power Efficiency

Table 3 shows the results of the Power Draw test that was done to calculate the Power Efficiency metric. As shown in the table, the Raspberry Pi and Xilinx FPGA consumed significantly less power than the Windows system during inference, which makes them ideal options for low-power applications.

Table 3: Power Efficiency Metrics - Windows w/ GPU section does account for the CPU power consumption

Platform	Nominal Power Consumption	Power Consumption during Power Test	DNN Inference Power Consumption	Measuring Tool(s)
Windows (CPU only)	56.96 W	77.08 W	20.12 W	Open Hardware Monitor (software)
Windows w/ GPU	23.70 W	28.07 W	4.37 W	NVidia SMI (software)
Raspberry Pi Model 4B	3.70 W	4.70 W	1.00 W	Power Meter
PYNQ-Z2 CPU	2.10 W	2.30 W	0.20 W	Power Meter
PYNQ-Z2 FPGA	2.10 W	2.30 W	0.20 W	Power Meter

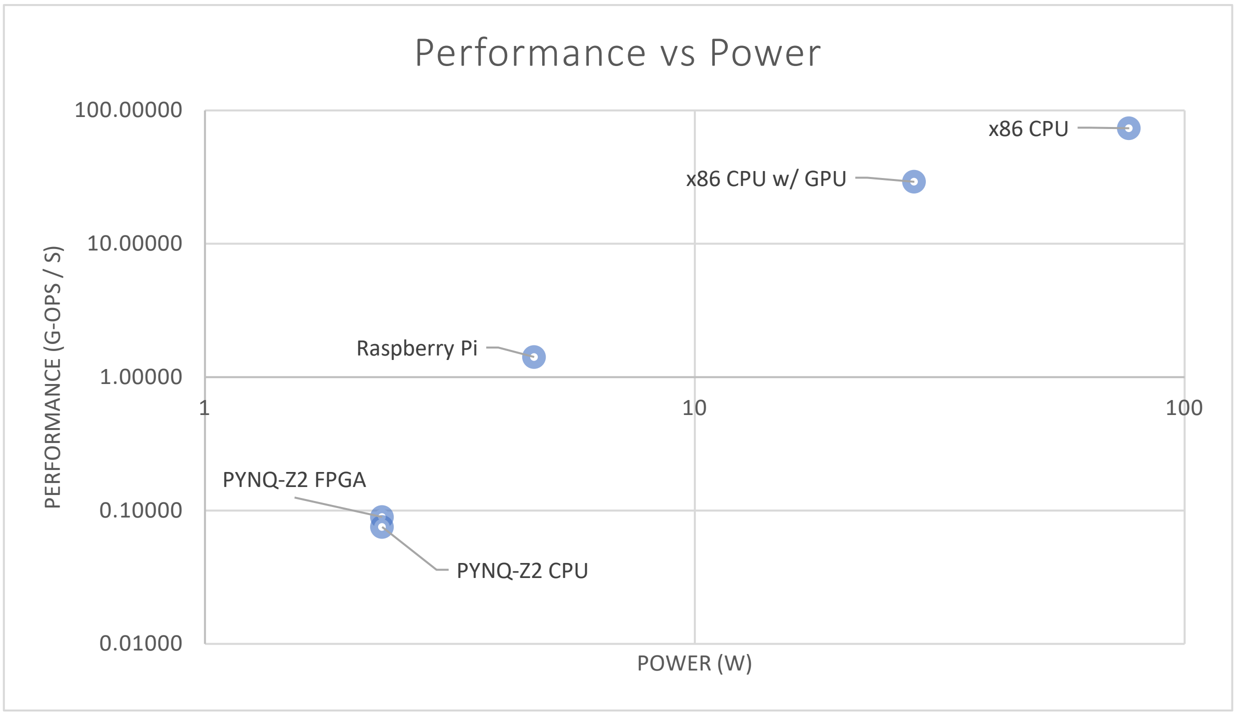


Figure 3: Performance vs Power for each platform

Figure 3 shows a graph that plots each platform on a grid of Performance vs Power. The vertical axis represents the performance metric of each system captured during the performance test, and the horizontal axis represents the power drawn during inference. This graph is useful for helping to decide the best deployment platform based on the project requirements for a specific DNN. By graphing a range of allowable power draw and minimum performance, an engineer could use this graph to determine what the most appropriate deployment platform could be.

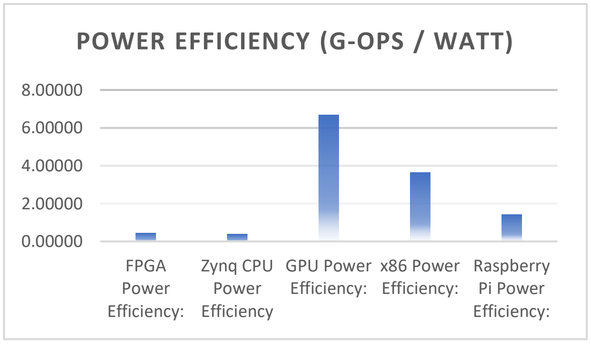


Figure 4: Power Efficiency Metric

Figure 4 presents the power efficiency for each implementation as a bar graph. The most efficient platform was the GPU, but this revelation comes with some important caveats. Although the GPU only used 4.37W to run the DNN, it still had a nominal system draw of approximately 20W. Compare this to the FPGA which had a much lower power efficiency due to its limited performance, but it also had both the lowest power draw and lowest nominal power of each tested platform. Depending on the deployment requirement, either the GPU or FPGA could make excellent DNN accelerators if the appropriate power budget is allocated.

VII. Limitations and Future Directions

While this study provided valuable insights into the performance and power efficiency of a DNN implementation on various platforms, there are some limitations that need to be acknowledged. Firstly, the performance of any DNN implementation is highly dependent on the network structure, and future studies could investigate the performance of more complicated DNNs with more complex datasets. Secondly, only one set of hardware was used for each platform which may limit the generalizability of the results to other hardware configurations. Future research could investigate the accuracy, performance and power efficiency of a DNN implementation on different hardware configurations, including different CPU, GPU, and FPGA components.

Additionally, this study did not consider the impact of other factors, such as device temperature and memory transfer speeds, which could affect the performance and power efficiency of a DNN implementation. Future studies can investigate the impact of these factors on DNN implementations across different platforms. Overall the data-collection techniques for each platform could be further optimized to improve the DNN's performance and standardize all external factors. Future research could investigate more sophisticated optimization techniques or investigate the effectiveness of the proposed techniques in this paper on different datasets and platforms.

VIII. Conclusion

In this study, we developed a modified DNN model for handwritten digit recognition using the MNIST dataset. We evaluated the performance of our model on five different platforms: an x86-64 CPU, an x86-64 CPU with GPU acceleration, a Raspberry Pi, and a Xilinx FPGA development board.

On the implementation using only an x86-64 CPU our DNN model achieved a high accuracy of 97.25% on the MNIST dataset. The average calculation rate was about 73 billion operations per second, which is fast enough for real-time applications. The power consumption was about 77 Watts, which is typical for a desktop computer.

On the x86-64 CPU implementation with a GPU accelerator our DNN model also achieved an accuracy of 97.25% on the MNIST dataset. The average calculation rate was about 30 billion operations per second, which is also fast enough for real-time applications. The power consumption was about 27 Watts when only accounting for the GPU, but due to the requirement of having a CPU to command this piece of hardware the result should be approached cautiously when trying to decide on a deployment platform.

On the Raspberry Pi our DNN model achieved the same accuracy of 97.25% on the MNIST dataset. The average calculation rate was about 1.4 billion operations per second, which is slower than both x86-64 implementations due to the lower processing power and memory of the Raspberry Pi. However, the power consumption was only 4.7 Watts, making it suitable for low-power applications.

On the Xilinx FPGA development board we successfully implemented our quantized DNN model using Vitis-HLS and Vivado, and were able to achieve an accuracy of 94.83%. The CPU implementation running on the SoC was able to achieve a performance of 0.075 billion operations per second, while the FPGA accelerated implementation was able to reach a performance of 0.089 billion operations per second. The power consumption for both implementations was 2.3 watts, making it suitable for applications that require very low power consumption.

In conclusion, although the inference time was slower on the Raspberry Pi and the FPGA development board than on the x86-64 based implementations, the power consumption was much lower, making them suitable for low-power applications. There is also great potential in FPGAs for accelerating DNN model inference in applications that require very little power consumption. Future work could involve optimizing the model architecture and hyper-parameters for better performance on the Raspberry Pi and FPGAs, or exploring other embedded platforms for DNN model inference. Additionally, our implementation can serve as a starting point for developing more complex DNN models on x86-64 based systems for a variety of applications. The results of this study can inform the selection of hardware platforms for DNN model inference based on the specific requirements of the application, such as performance, power consumption, and cost.

IX. Appendix

A. ***References***

[1] K. Hao. “The computing power needed to train AI is now rising seven times faster than ever before.” https://www.technolo-gyreview.com/2019/11/11/132004/the-computing-power-needed-to-train-ai-is-now-rising-seven-times-faster-than-ever-before/ (accessed May 1, 2023).

[2] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo and J. Zhang, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," in Proceedings of the IEEE, vol. 107, no. 8, pp. 1738-1762, Aug. 2019, doi: 10.1109/JPROC.2019.2918951.

[3] Huang, Jonathan, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara Balan, Alireza Fathi, Ian S. Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama and Kevin P. Murphy. “Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors.” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 3296-3297.

[4] X. Sui et al., “A Hardware-Friendly High-Precision CNN Pruning Method and Its FPGA Implementation,” Sensors, vol. 23, no. 2, p. 824, Jan. 2023, doi: 10.3390/s23020824.

[5] Velasco-Montero, Delia, Jorge Fernández-Berni, Ricardo Carmona-Galán and Ángel Rodríguez-

Vázquez. “Performance analysis of real-time DNN inference on Raspberry Pi.” Commercial + Scientific Sensing and Imaging (2018).

[6] “What is deep learning?,” IBM. https://www.ibm.com/topics/deep-learning#:~:text=Deep%20learning%20is%20a%20subset,from%20large%20amounts%20of%20data (accessed: May 1, 2023).

[7] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," in *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141-142, Nov. 2012, doi:10.1109/MSP.2012.2211477.

[8] M. Abadi et al., TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. [Online]. Available: https://www.tensorflow.org/

[9] Kingma, D.P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. CoRR, abs/1412.6980.

[10] G. Boesch. “TensorFlow Lite – Real-Time Computer Vision on Edge Devices (2022).” Viso.ai. https://viso.ai/edge-ai/tensorflow-lite/#:~:text=TensorFlow%20Lite%20(TFLite)%20is%20a,on%20mobile%20and%20edge%20devices. (accessed May 1, 2023)

[11] “*Netron”*. Netron.ai. https://netron.app/ (accessed: May 01, 2023).

[12] *Open Hardware Monitor v0.9.6*, Open Hardware Monitor, 2023.

[13] “NVidia System Management Interface,”NVIDIA Developer, https://developer.nvidia.com/nvidia-system-management-interface (accessed: 01-May 2023).

B. ***Statement of Work***

Table 4 below provides information about the allocation of work for this project based on the overall percentage each team member contributed to a specific milestone.

Table 1: Statement of Work Allocation

Milestone	**Matthew Lima Percentage**	**Krishu Nakarmi Percentage**
DNN Model Training	50%	50%
x86-64 DNN Model Evaluation	50%	50%
x86-64 with GPU DNN Model Port	50%	50%
x86-64 with GPU DNN Model Evaluation	50%	50%
Raspberry Pi DNN Model Port	0%	100%
Raspberry Pi DNN Model Evaluation	0%	100%
Zynq CPU DNN Model Port	100%	0%
Zynq CPU DNN Model Evaluation	100%	0%
Zynq FPGA DNN Model Port	100%	0%
Zynq FPGA DNN Model Evaluation	100%	0%