Final Report

Project #43: Reconfigurable Neural Processing Unit (NPU) for Energy-Efficient AI at the Edge

Kristelle Sampang

Project Report

Project Partner:	Pratham Chhabra
Supervisor:	Dr Morteza Biglari-Abhari
Co-Supervisor:	Dr Maryam Hemmati

16th October 2025

Waipapa Taumata Rau University of Auckland

COMPSYS043-2025

RECONFIGURABLE NEURAL PROCESSING UNIT (NPU) FOR ENERGY-EFFICIENT AI AT THE EDGE

Kristelle Sampang

ABSTRACT

Abstract goes here.

DECLARATION

Student I hereby declare that:

1. This report is the result of the final year project work carried out by my project partner (see cover page) and I under the guidance of our supervisor (see cover page) in the 2025 academic year at the Department of Electrical, Computer and Software Engineering, Faculty of Engineering, University of Auckland.
2. This report is not the outcome of work done previously.
3. This report is not the outcome of work done in collaboration, except that with a potential project sponsor (if any) as stated in the text.
4. This report is not the same as any report, thesis, conference article or journal paper, or any other publication or unpublished work in any format.

In the case of a continuing project, please state clearly what has been developed during the project and what was available from previous year(s):

Signature: [illegible]

Date: 19/10/2025

Table of Contents

Acknowledgements

Thank important people here.

Glossary of Terms

Term	Definition

Abbreviations

AOA	**Angle of attack**

1. Introduction

1.1 Motivation

- As the demand for artificial intelligence grows to become prominent in today's society, its energy consumption has become a key issue.
- The processing required to run machine learning models is large, with millions of computations needed to be executed.
- This means that low-power processing devices at edge are limited to its use.
- The processing power to run machine learning models is large, limiting its use for low-power processing devices at the edge.
- With high computational power required, energy-efficiency is a key concern, especially nowadays when energy is an essential resource to not waste.
- In the past decade, common types of processors for running AI are CPU, GPU, and FPGA.
- However, as technology continues to improve, Neural Processing Units (NPU) are developed. These are processors that specialise in processing AI computations.
- By integrating NPU alongside other processors, the performance and energy-efficiency are improved compared to a standalone processor.

1.2 Problem Statement

1.3 Report Structure

The remainder of this report goes as follows: Section X covers Y....

2. Background

The rapid growth of Artificial Intelligence (AI) in the past decade has driven significant advancements across numerous field. Machine Learning (ML) models, particularly Deep Neural Networks (DNNs), are popular for its applications in image classification to autonomous driving [1]. There has been significant shift from AI inferencing occur at a cloud-level to resource-constrained edge devices. This move motivates the "AI at the Edge" in the research, where lower latency, enhanced data privacy, and real-time processing capabilities must are requirements that must be met without relying on constant network connection [2].

However, this shift presents an alarming issue where DNNs are computationally intensive and power-hungry, whilst edge devices operate under strict power and resource limitations. To bridge this gap, hardware accelerators, such as Neural Processing Units (NPUs) are introduced to execute AI algorithms at faster rates than general-purpose CPUs [3]. This section provides necessary background on the core technologies that support this project, starting with the most popular model for image-based tasks, the Convolutional Neural Network.

2.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) is a prominent type of DNN model, mainly utilised image processing tasks, such as object recognition, classification, and detection. The network is compromised of four main layers: convolutional, activation layer, pooling layers, and fully connected layers, where this research focuses on the convolutional and activation layers. The convolutional layer is where majority of the computations occur [4], as it extracts features from an image and converts it into numerical values. In a convolutional layer, there are several filters that slide through the image, searching for a specific pattern. These filters are typically in the size of 3x3 or 5x5, and is applied to the image by multiplying the filter by the 2D pixel representation of the image. Mathematically, this operation can be represented as

$$O[h][w][c] = \sum_{i=1}^{f_h} \sum_{j=1}^{f_w} \sum_{k=1}^{i_c} I[h + i \cdot s_h][w + j \cdot s_w][k] \times F[i][j][k][c]$$

where I, O, F are input activation, output activation, and filter weights respectively [4]. This can be represented as an enormous number of Multiply-Accumulate (MAC) operations, making CNNs computationally intensive.
An activation layer determines whether a neuron should be activated based on its input. Its primary role is to introduce non-linearity into the network. Without this, a neural network can only learn simple, linear patterns. Non-linearity allows the network to execute complex tasks, such as learning complicated patterns. A commonly used function is the Rectified Linear Unit (ReLU). The main functionality is to allow for positive inputs to remain unchanged whist setting any negative input to zero. A critical consequence of this is that it introduces significant sparsity as approximately half of the elements are zero [5]. This sparsity is a key property that this project exploits to improve energy-efficiency, and will be discussed further in Section 3.1.1.

2.2 Hardware Acceleration for Machine Learning

General-purpose CPUs alone are not powerful enough to handle the computational requirements of modern deep learning models. Typically, specialised hardware accelerators, such as GPUs and FPGAs, are integerated with the CPU as a heterogenous system to enhance the performance of AI applications [3]. A study discovered that GPUs highly accelerate computationally intensive tasks with parallelism at ease but with a trade-off of high power consumption [6]. Furthermore, GPUs are not typically found in edge devices, such as smartphones and IoT devices, due to its high power consumption and thermal output. Neural Procesing Units (NPUs) is a type of specialised hardware accelerator, best a executing AI-based algorithms. Due to its dedicated use, it has extreme performance and low-energy effiency, however, it is limited in its flexibility. On the other hand, FPGAs can achieve energy-efficiency by consuming half the power of a GPU[7] by tailoring the hardware architecture to a specific task. This customisability allows for optimised dataflow and memory access patterns, crucial for improving the performance of deep learning models. Therefore, this project chooses to implement the NPU on an FPGA platform to create a reconfigurable hardware accelerator.

2.3 Systolic Array Architecture

A systolic array is a structured network of processing elements (PEs) where computations are carried out in a systematic approach, similar to pipelining. Each PE will receive, process, and pass on data to its neighbour concurrently in one clock cycle. Due to this, it provides characteristics like parallel computing, pipelining, synchronicity, and spatial and temporal locality. These are advantageous as processes are completed simultaneously at higher speeds, computations are timed by a global clock for predictable data movement, and faster memory accessing time. Additionally, systolic arrays are highly compact together and allow simple data control flow. As its architecture is an array, it can easily be scaled for larger data sets, perfect for ML, and the predictable structure makes it easier to design and optimise algorithms. Furthermore, since the PEs are densely packed together, there are less data transmission, causing lower energy consumption. However, there are some disadvantages to systolic arrays. These include difficult and costly to build and specialised and inflexible in the problems it can solve, limiting its versatility. Because systolic arrayss offer high throughput and effiency, it is commonly used in NPUs to accelerate matrix multiplication, the core operation in a CNN's convolutional layer. However, its rigid and structured dataflow is inefficient for sparse matrices, leading to imbalanced workloads and low PE utilisation. This limitation motivates the need for a reconfigurable systolic array that can adapt to the sparsity in neural networks, becoming the focus of this project.

3. Literature Review

This section delves deep into the novelties and gaps in the current research on handling sparsity in neural networks, particularly in the context of systolic arrays. It explroes the challenges posed by sparsity and reviews existing hardware architectures deisgned to mitigate these issues, leading to the motivation for this project.

3.1 Sparsity in Neural Networks

As discussed in Section 2.1, sparsity is a property of matrices that arise from many sources, such as activation functions and model compression techniques. This section explores these in detail to find the root cause of sparsity in neural networks.

3.1.1 Rectified Linear Unit (ReLU)

To optimise the performance of a neural network, gradients are calculated to update the network's weights. However, this introduces the Vanishing Gradient Problem (VGP) where gradients become increasingly miniscule when backpropagated from the output to earlier layers. The consequence of VGP presents slow convergence and impaired learning. This issue becomes even more problematic when structures have many layers and a solution to this is applying ReLU. As previously mentioned in Section 2.1, the outcome of ReLU is that all negative values are set to zero, introducing significant sparsity in the activation matrices. However, since ReLU is most commonly applied [5], it is difficult to avoid using a model that does not produce sparse matrices due to ReLU in real-case scenarios.

3.1.2 Pruning

Model Pruning is a technique used to remove redundant data that may not significantly contribute to the final prediction. Kim et al. [8] found that more than half of weights in convolutional layers can be set to zero without impacting the overall accuracy. To decide what weights to prune, the sensitivity of the weight is considered and how it influences the output. Gorvadiya et al. [9] discovered that their Weight Pruning technique resulted in minimal accuracy loss of 1-2% whilst saving 25% in power consumption. These results suggests that pruning is an effective method to reduce model size and computation without sacrificing performance.

Whilst both ReLU and pruning work together to help accelerate the efficiency of the neural network by introducing sparsity in the system, this presents a challenge for systolic array based hardware architecture. As the architecture of systolic arrays is rigid and structured, it is advantageous for dense matrices with regular data access patterns. However, with the introduction of irregularity due to sparsity, this leads to inefficient memory access patterns and imbalanced workloads, causing low PE utilisation [10]. Additionally, with the high volume of computations required to run a CNN, the impact of the inefficiency is magnified. This means that handling sparsity is a critical issue that must be solved to fully exploit the benefits of sparsity in neural networks. Balancing the benefits and drawbacks of how sparsity is handled in systolic arrays is explored in the following section.

3.2 Hardware Architectures for Sparsity

3.2.1 Power Gating and Zero-Skipping

As multiplying and accumulating with zero does not change the overall result, it is redundant to execute operations with zero. Thus, a method like power-gating is introduced to skip these unnecessary operations. The most straight forward approach to handling sparsity in hardware is power-gating. This method skips the MAC operation if one of the operands is zero, dynamically saving power [1]. However, this method requires additional hardware to check if an operand is zero, adding complexity to the design. Additionally, this does not reduce the latency as the PE has to wait for the data to be fetched from memory for the systolic pipeline to advance at the same rate.

A more advanced approach that reduces latency is zero-skipping. This technique involves only processing non-zero values and skipping over the zeros entirely. On a smaller scale, this can be applied to individiual weights to reduce the number of clock cycles [11]. On a larger scale, in the architecture by Kim et al. [8], the accelerator was able to skip 128x128 square blocks if structured sparsity existed, leading to a significant reduction in latency. However, this requires high levels of structured sparsity, that may not always occur in real-case scenarios.

3.2.2 Compressed Data Formats

To further improve performance, many architectures use compressed data formats to reduce the significant energy and latency costs of memory access. This is often implemented as a software-hardware co-design where a pre-processing stage reorganises the sparse matrix before it is sent to the accelerator. For example, He et al. [10] proposes a packing technique to confense the matrix offline then loads it into the systolic array. Similarly, Seo and Kong [12] uses a pre-processing column-wise condensation of the weights to remove zero values in advanced. Additionally, Kim et al. [8] and Palacios et al. [13] found that with structured pruning, where entire rows or columns of weights are pruned, the hardware can be optimised to skip entire rows or columns of computations. This is more efficient than unstructured pruning, where individual weights are pruned, as it maintains the regularity of the data access patterns.

Although these methods significantly reduce memory access costs and improve performance, there are trade-offs. As these methods change the structure of the input matrices, it requires complex hardware and control logic to manage the new dataflow. Additionally, post-processing is required to rearrange the result matrix is required to ensure that the indices match up the original input Seo and Kong [12]. This added complexity may lead to increased power consumption and use of hardware resources, potentially nullifying the benefits of handling sparsity. Therefore, whilst compressed data formats are effective, there are significant design challenges that must be carefully considered to fully exploit its advantages.

3.2.3 Specialised Dataflows and Architectures

Advanced architectures redesign the entire dataflow and control logic to efficiently handle sparsity in systolic array based hardware. These architecture offer higher performance but at the cost of significant hardware complexity. This section focuses on two key architectures, Sparse-TPU [10] and SCNN [1], to illustrate the current state-of-the-art and identify the research gap that this project aims to fill.

- **Sparse-TPU**: This architecture implements a column-packing algorithm to reorganise the sparse weight matrix into a denser format. It does this by examining each column and pushing non-zero elements to the left, creating a new row if a collision occurs. A collision is considered when two non-zero elements exist in the same row after pushing. This effectively reduces the number of columns, especially with non-structured sparsity, leading to lower memory access costs. However, this method requires complex pre-processing and alters the input data structures, complicating the hardware dataflow. It is critical that the rearrangement is tracked to ensure that the MAC operations are mathematically correct [10].

- **SCNN**: This architecture uses a different approach to handle sparsity. It operates on a compressed-sparse dataflow, where only non-zero weights and activations are fetched from memory and delivered to the multiplier array. This dataflow tracks the output coordinate for each multiplication and sends the resulting product to a specialised scatter accumulator, ensuring the value is summed at the correct location in the output feature map. This approach maximises data reuse but diverges significantly from standard systolic array designs [1].

The examples of Sparse-TPU and SCNN demonstrate a complex yet powerful solutions for unstructured, fine-grained sparsity. However, they require significant redesign of the core dataflow and control logic, leading to increased hardware complexity. As Palacios et al. [13] suggests that fine-grained sparsity requires a fair amount of control logic overhead. In contrast, structured sparsity, where entire rows or columns of weights are zero, is easier to exploit in hardware. This project aims to fill the gap for a simpler that can exploit structured sparsity without overhauling the classic systolic array pipeline.

4. Design and Methodology

4.1 System Architecture Overview

4.2 Data Generation and Pre-processing

This is section covers the software side of the co-design. It utilises AlexNet to generate and extract realistic weight and activation matcies, then pre-processes it into tiles suitable for hardware testing. The processed tiles are then saved as a mif file and stored into the FPGA's BRAM for static testing. This section is crucial as it ensures that the hardware is tested with real-world data, validating its effectiveness in handling sparsity. Additionally, this section was implemented in a straight forward manner, meaning there were no major iterations or challenges faced.

4.2.1 AlexNet Model and Data Extraction

The AlexNet model was chosen for its popularity in image classification task and balance between complexity and size. It was commonly found in literature as the model of choice for edge devices due to its efficiency [1], [5]. A pre-trained model from the PyTorch library was used, as training is out of the scope of this project. The weights and activations were extracted from the first convolutional layer, as it has the largest matrices and highest sparsity due to ReLU. This was quantised to INT8 as edge devices are resource-constrained and require low memory usage. In comparison to other data types such as FP32 and INT16, INT8 provides a good balance between accuracy, memory usage, and energy-efficieny [9]. The model was tested with a sample image to extract the activation matrix. The image used for testing is a photo of a cat (shown in Figure 1), as it is a common test image for image classification tasks and has a good variety of features for the convolutional layer to extract.

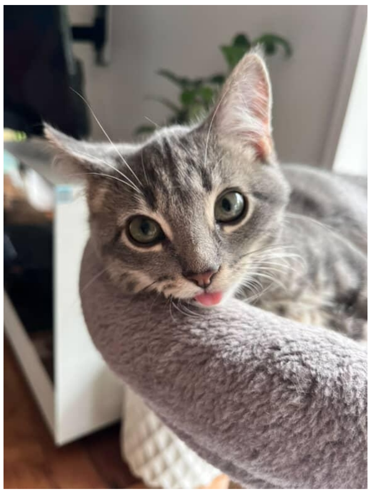

Figure 1 Image of a cat used for testing the AlexNet model

4.2.2 Tiling and .mif File Generation

Once the weight and activation matrices were extracted from the model, it was found that its size were (64, 363) and (3025, 576) respectively. These sizes are much larger than what the systolic array can handle, making it necessary to tile the matrices into smaller sub-matrices. Literature from Palacios et al. [13] and Parashar et al. [1] support this approach, as it improves data locality and reduces memory access costs. Additionally, sizes were not perfectly divisible by 8, leading to incomplete tiles. To address this, zero-padding was applied to the matrices to ensure that they could be evenly divided into 8x8 tiles. This involved adding rows and columns of zeros to the bottom and right of the matrices until their dimensions were multiples of 8. This ensured that all tiles were complete and could be processed by the systolic array without any issues. Furthermore, by tiling the matrices, it allows for easier identification of features that look like structured sparsity that may not be apparent when looking at the entire matrix. This means that when looking at the layer as a whole, it may not be obvious that there are entire rows or columns of zeros. However, when the layer is broken down into smaller tiles, it becomes easier to see these patterns. This is important as it allows for more effective pre-processing and optimisation of the data for the hardware design.

These are saved as .mif files, which can be directly loaded into the FPGA's BRAM for static testing. This allows for easy verification of the hardware design with real-world data. It is important to note that only one tile is currently loaded into the BRAM at a time, meaning that the testing is static and not real-time. However, a master data and master weight file does exist to load different tiles into the BRAM, but this is not currently implemented in the hardware design.

Overall, by pre-processing the data in this manner, it ensures that the hardware tested is relevant and compatible with the hardware design detailed in the following sections. This validates the effectiveness of the approach in handling sparsity present in CNN models. The code for this software component can be found in the compendium with the filename `pipelined_v2.py`.

4.3 Baseline Systolic Array Architecture

The core of the hardware accelerator is an 8x8 systolic array, designed to perform matrix multiplication for convolutional layers. The fundamental building block of this array is the Processing Element (PE), a custom-designed Multiply-Accumulate (MAC) unit. The entire hardware accelerator was designed in VHDL.

4.3.1 Processing Elements and Systolic Array Formation

The fundamental process that the PE executes is the MAC operation, the core computation in convolutional layers. This can be represented in a Register-Transfer Level (RTL) diagram in Figure 2. The PE is designed to handle INT8 data types, as discussed in Section 4.2.1, to balance accuracy and resource usage. The PE takes two inputs: a weight from the filter matrix and an activation from the input feature map. These inputs are multiplied together and adds the result to an accumulate that holds the partial sum for the output feature map. As each PE is independent from one another and operates concurrently, it allows for parallel processing of multiple MAC operations. Together, 64 PEs are arranged in an 8x8 grid to form the systolic array, as shown in Figure 3.

Each PE communicates with its neighbours to pass data along the array, enabling a continuous flow of data through the system. The dataflow chosen for this design is output-stationary, where the output partial sums remain in the PE until the final result is computed. This approach minimises data movement and maximises data locality, reducing memory access costs. The PE is designed to operate in a pipelined manner, allowing it to accept new inputs every clock cycle while still processing previous inputs. This pipelining is crucial for maintaining high throughput in the systolic array.

The bus communication between each PE uses the "generate" statement in VHDL to create a scalable and modular design. This allows for easy expansion or modification of the array size in the future. The systolic array is integrated into a top-level module, which manages the overall operation of the array, including data input and output, as well as interfacing with the Control Unit and BRAMs. For the purpose of this project, the systolic array is designed to handle 8x8 tiles, as discussed in Section 4.2.2. This size was chosen as it provides a good balance between hardware resource usage and computational capability, making it suitable for edge devices with limited resources.

4.3.2 Control Unit

The Control Unit is the mastermind of the Systolic Array operation, orchestrating the flow of data and ensuring that each PE operaetes synchronously. It generates the necessary control signals to manage the timing and coordination of data movement through the array. The Control Unit is responsible for initiating the loading of weights and activations into the array, as well as signalling when to start and stop computations. It also manages the readout of the final output feature map once all computations are complete. The Control Unit is designed to be flexible, allowing for adjustments in the operation of the systolic array based on the specific requirements

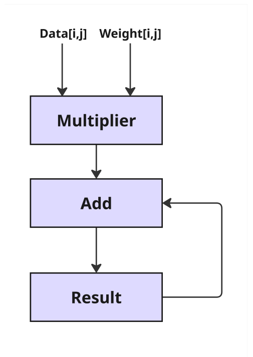

Figure 2 Image of the MAC unit in the Processing Element (PE)

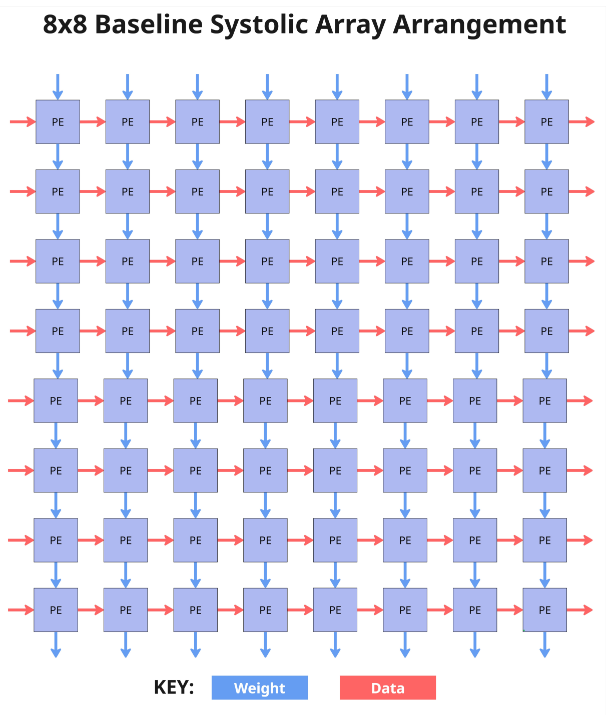

Figure 3 Image of the Baseline Systolic Array Arragement

of the input data. This flexibility is crucial for adapting to different matrix sizes and sparsity patterns, which will be further explored in Section 4.4.

The dataflow of the systolic array is designed to be highly efficient. Weights are fed into the array from the left, moving horizontally across each row of PEs. Activations are fed in from the top, moving vertically down each column. As weights and activations move through the array, each PE performs its MAC operation and passes the activation to the PE below it and the weight to the PE to its right. This creates a wave-like motion of data flowing through the array, ensuring that all PEs are utilised effectively. The output partial sums are stored in local registers within each PE until the entire matrix multiplication is complete. Once all inputs have been processed, the final output feature map is read out from the bottom-right corner of the array.

As stated previously, the hardware accelerator is designed to take in 8x8 tiles of weights and activation matrices. The systolic array is capable of handling different sized inputs less than 8x8. This is achieved by the cotrol unit checking the input matrix size and adjusting the operation of the array accordingly. For example, if a 6x6 tile is inputted, the control unit will ensure that the systolic array will synthesise the correct amount of PEs and busses required. This effectively "shrinks" the systolic array to only use the necessary resources, improving efficiency. This feature is crucial for handling sparsity, as it allows the array to adapt to the effective size of the input matrices after pre-processing, which will be discussed in Section 4.5.

4.4 Dynamic Control Unit for Variable Matrix Sizes

4.5 Sparsity Handling Algorithm

4.5.1 Software Pre-Processing

5. Verification and Results

5.1 Testbench and Simulation Environment

5.2 Baseline Performance (Dense Matrices)

5.3 Optimised Performance (Stripped Matrices)

5.4 Performance Analysis

6. Discussion

6.1 Analysis of Results

6.2 Design Trade-offs

6.3 Limitations

7. Conclusion

8. Future Work

References

[1] A. Parashar et al., *SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks*, en, arXiv:1708.04485 [cs], May 2017. DOI: 10.48550/arXiv.1708.04485. Accessed: Sep. 18, 2025. [Online]. Available: http://arxiv.org/abs/1708.04485.

[2] B. Kim, S. Lee, A. R. Trivedi, and W. J. Song, "Energy-Efficient Acceleration of Deep Neural Networks on Realtime-Constrained Embedded Edge Devices," en, *IEEE Access*, vol. 8, pp. 216 259–216 270, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3038908. Accessed: Mar. 31, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/9262933/.

[3] E. Manor and S. Greenberg, "Custom Hardware Inference Accelerator for TensorFlow Lite for Microcontrollers," en, *IEEE Access*, vol. 10, pp. 73 484–73 493, 2022, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3189776. Accessed: Mar. 31, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/9825651/.

[4] J. Choi et al., "Enabling Fine-Grained Spatial Multitasking on Systolic-Array NPUs Using Dataflow Mirroring," en, *IEEE Transactions on Computers*, vol. 72, no. 12, pp. 3383–3398, Dec. 2023, ISSN: 0018-9340, 1557-9956, 2326-3814. DOI: 10.1109/TC.2023.3299030. Accessed: Apr. 2, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10198513/.

[5] W. Sun, D. Liu, Z. Zou, W. Sun, S. Chen, and Y. Kang, "Sense: Model-Hardware Codesign for Accelerating Sparse CNNs on Systolic Arrays," en, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 4, pp. 470–483, Apr. 2023, Publisher: Institute of Electrical and Electronics Engineers (IEEE), ISSN: 1063-8210, 1557-9999. DOI: 10.1109/tvlsi.2023.3241933. Accessed: Jul. 22, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10043636/.

[6] S. Oh, M. Kim, D. Kim, M. Jeong, and M. Lee, "Investigation on performance and energy efficiency of CNN-based object detection on embedded device," en, in *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, Kuta Bali: IEEE, Aug. 2017, pp. 1–4, ISBN: 978-1-5386-0600-1. DOI: 10.1109/CAIPT.2017.8320657. Accessed: Mar. 31, 2025. [Online]. Available: http://ieeexplore.ieee.org/document/8320657/.

[7] X. Liu, W. Xu, Q. Wang, and M. Zhang, "Energy-Efficient Computing Acceleration of Unmanned Aerial Vehicles Based on a CPU/FPGA/NPU Heterogeneous System," en, *IEEE Internet of Things Journal*, vol. 11, no. 16, pp. 27 126–27 138, Aug. 2024, ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2024.3397649. Accessed: Mar. 31, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10525057/.

[8] S. Kim, S. Cho, E. Park, and S. Yoo, "FPGA Prototyping of Systolic Array-based Accelerator for Low-Precision Inference of Deep Neural Networks," en, in *2021 IEEE International Workshop on Rapid System Prototyping (RSP)*, Paris, France: IEEE, Oct. 2021, pp. 1–7, ISBN: 978-1-6654-6956-2. DOI: 10.1109/RSP53691.2021.9806200. Accessed: Mar. 30, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/9806200/.

[9] J. Gorvadiya, A. Chagela, and M. Roy, “Energy Efficient Pruning and Quantization Methods for Deep Learning Models,” en, in *2025 International Conference on Sustainable Energy Technologies and Computational Intelligence (SETCOM)*, Gandhinagar, India: IEEE, Feb. 2025, pp. 1–6, ISBN: 979-8-3315-2054-0. DOI: 10.1109/SETCOM64758.2025.10932458. Accessed: Mar. 28, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10932458/.

[10] X. He et al., “Sparse-TPU: Adapting systolic arrays for sparse matrices,” en, in *Proceedings of the 34th ACM International Conference on Supercomputing*, Barcelona Spain: ACM, Jun. 2020, pp. 1–12, ISBN: 978-1-4503-7983-0. DOI: 10.1145/3392717.3392751. Accessed: Apr. 2, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3392717.3392751.

[11] Y. Duk Kim et al., “2.4 A 7nm High-Performance and Energy-Efficient Mobile Application Processor with Tri-Cluster CPUs and a Sparsity-Aware NPU,” en, in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, San Francisco, CA, USA: IEEE, Feb. 2020, pp. 48–50, ISBN: 978-1-7281-3205-1. DOI: 10.1109/ISSCC19947.2020.9062907. Accessed: Oct. 15, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/9062907/.

[12] J. Seo and J. Kong, “VerSA: Versatile Systolic Array Architecture for Sparse and Dense Matrix Multiplications,” en, *Electronics*, vol. 13, no. 8, p. 1500, Apr. 2024, ISSN: 2079-9292. DOI: 10.3390/electronics13081500. Accessed: Oct. 15, 2025. [Online]. Available: https://www.mdpi.com/2079-9292/13/8/1500.

[13] P. Palacios, R. Medina, J.-L. Rouas, G. Ansaloni, and D. Atienza, “Systolic Arrays and Structured Pruning Co-design for Efficient Transformers in Edge Systems,” en, arXiv:2411.10285 [cs], Jun. 2025, pp. 320–327. DOI: 10.1145/3716368.3735158. Accessed: Jul. 16, 2025. [Online]. Available: http://arxiv.org/abs/2411.10285.

Appendix A The First Appendix

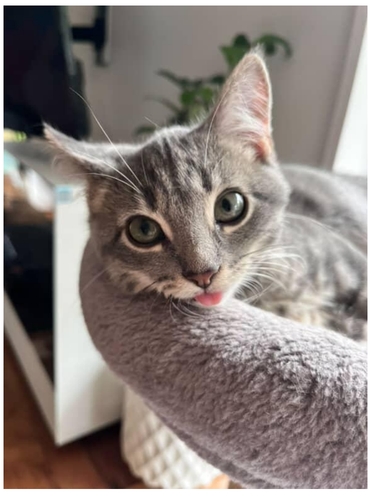

Figure 1 Image of a cat used for testing the AlexNet model

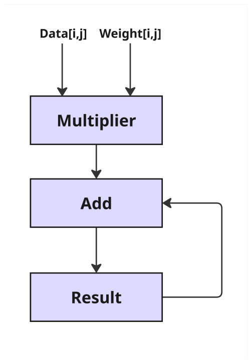

Figure 2 Image of the MAC unit in the Processing Element (PE)

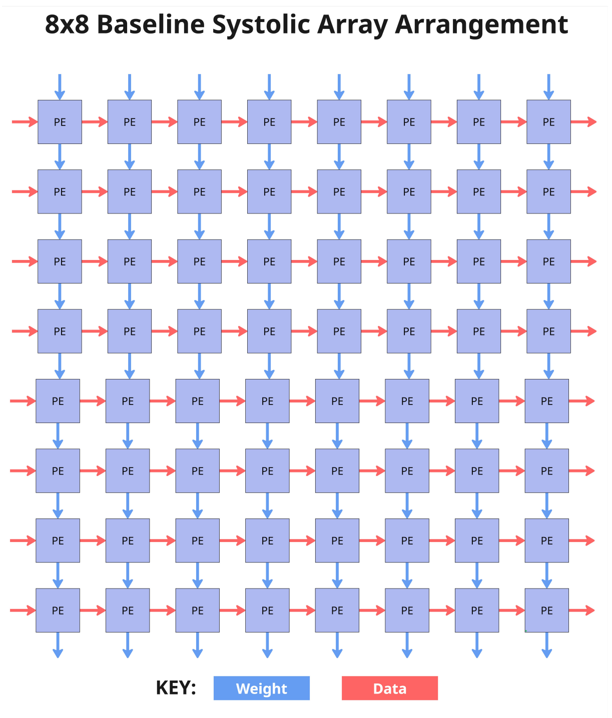

Figure 3 Image of the Baseline Systolic Array Arragement