Degree Project in Computer Science and Engineering

First Cycle, 15 Credits

Comparison of Akida Neuromorphic Processor and NVIDIA Graphics Processor Unit for Spiking Neural Networks

CARL CHEMNITZ

MALIK ERMIS

Comparison of Akida Neuromorphic Processor and NVIDIA Graphics Processor Unit for Spiking Neural Networks

CARL CHEMNITZ
MALIK ERMIS

Degree Project in Computer Science and Engineering, First Cycle 15 credits
Date: June 9, 2025
Supervisor: Jörg Conradt
Examiner: Pawel Herman
Swedish title: Jämförelse av neuromorfisk processor Akida och NVIDIA grafikkort för Spiking Neural Networks
School of Electrical Engineering and Computer Science

Abstract

This thesis investigates the latency, throughput and energy efficiency of the BrainChip Akida AKD1000 neuromorphic processor compared to a NVIDIA GeForce GTX 1080 when running two different spiking neural network models on both hardwares. Spiking neural networks is a subset of neural networks that are specialized for neuromorphic processor. The first model is a simple image classification model (GXNOR on MNIST), and the second is a more complex object detection model (YOLOv2 on Pascal VOC). The models were trained and quantized to 2-bit and 4-bit weight precision, respectively, enabling spiking execution both on Akida AKD1000 and on GTX 1080, for the GPU CUDA was used.

Results show that Akida achieved significant reductions in energy consumption and clock cycles for both models, consistent with prior findings within the field. Specifically, for the simple classification model the AKD1000 achieved 99.5 % energy reduction with 76.7 % faster inference times, despite having a clock rate 91.5 % slower than the GPU. However, for the more complex object detection model, the Akida took 118.1 % longer per inference, while reducing the energy expenditure by 96.0 %.

For the MNIST model the AKD1000 showed no correlation in both cycles & time and cycle & energy. While for the YOLOv2 model it had a 0.2 correlation for both previous mentioned ratios. Suggesting that as model complexity increases, the Akida's behaviour converges toward the GPU's linear correlation patterns.

In conclusion, the AKD1000 processor demonstrates clear advantages for low-power, edge-oriented applications where latency and efficiency are critical. However, these benefits diminish with increasing model complexity, where GPUs maintain superior scalability and performance. Due to limited documentation of the chosen models, a 1-to-1 comparison was not possible. Future work should focus on fully customized models to further explore the dynamics.

Sammanfattning

Denna rapport studerar latens, beräkningstid och energieffektivitet hos den neuromorfa processorn, BrainChip Akida AKD1000, jämfört med NVIDIA GeForce GTX 1080-grafikkort genom körandet av två olika spiking neural networks på båda hårvaror. Spiking neural networks är en underkategori av neural networks som neuromorfisk hårdvara är specifikt konstruerat för. Den första modellen är en simplare bildklassificeringsmodell (GXNOR på MNIST) och den andra är en mer komplex objekt detekteringsmodell (YOLOv2 på Pascal VOC). Modellerna tränas och kvantiseras till 2-bitars respektive 4-bitars viktprecision, vilket möjliggör aktivitetsbaserad exekvering både på Akida AKD1000 och på GTX 1080 via CUDA.

Resultaten visar att Akida uppnår en minskning av energiförbrukning och antal klockcykler vid körning av båda modellerna, som stämmer överens med tidigare resultat inom området. AKD1000 för den enkla klassificeringsmodellen uppnår en energireduktion på 99.5 % och 76.7 % snabbare beräkningstider, trots en klockfrekvens som var 91.5 % långsammare än GPU:n. För den mer komplexa objekt detekteringsmodellen tog dock Akidan 118.1 % längre tid per beräkning, men energiförbrukningen reduceras fortfarande med 96.0 %.

För MNIST-modellen uppvisar AKD1000 ingen korrelation mellan cykler & tid eller mellan cykler & energi. För YOLOv2-modellen visas det en korrelation på 0.2 för båda tidigare nämnda relationer, vilket antyder att när modell komplexiteten ökar så konvergeras Akidas beteende mot GPU:ns linjära mönster.

Sammanfattningsvis demonstrerar AKD1000-processorn tydliga fördelar för låg energi, realtids orienterade tillämpningar, där latens och beräkningstid är ett krav. Dessa fördelar avtar dock med ökande modellkomplexitet, där GPU:n behåller överlägsen skalbarhet och prestanda. På grund av begränsad dokumentation av de valda modellerna var en strikt en-till-en jämförelse inte möjlig. Framtida arbeten bör därmed fokusera på fullt anpassade modeller för att ytterligare utforska dynamiken.

Contents

Abbreviations

AI	Artificial Intelligence
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
FC layer	Fully Connected Layer
GPU	Graphics Processing Unit
LIF	Leaky Integrate-and-Fire
MAD	Median Absolute Deviation
mAP	Mean Average Precision
NPU	Neural Processing Unit
SNN	Spiking Neural Network
STDP	Spike-Timing Dependent Plasticity
YOLO	You Only Look Once

Glossary

AKD1000	The AKD1000 is a neuromorphic AI processor designed by BrainChip for energy-efficient, low-latency inference on edge devices.
CUDA	CUDA is a parallel computing platform and programming model developed by NVIDIA that allows developers to use GPUs for general-purpose computing beyond graphics rendering.
Dense data	Dense data contains mostly non-zero or active values, with information distributed uniformly across the dataset.
Edge AI	Running neural networks on edge devices instead of running it an external server.
Edge Computing	Deploying SNNs on NPUs or GPUs for real-time processing in devices with limited resources, such as Internet of Things devices or autonomous vehicles.
GPU	A hardware accelerator originally designed for rendering graphics but widely used for parallel processing tasks, including training and inference of neural networks.
Label	The target output or correct answer provided in supervised learning.
LIF neuron	Mathematical model to describe the behaviour of a neuron, where the membrane potential leaks over time and fires a spike when a threshold is reached.
MNIST	A prominent dataset of handwritten digits commonly used for training and testing machine learning models.
Neuromorphic Computing	A computing paradigm inspired by the structure and function of the human brain, often using SNNs and specialized hardware like NPUs.

NPU	A specialized hardware accelerator designed for efficient execution of neural network operations, particularly for AI and machine learning tasks. NPUs are optimized for low-power, high-performance inference.
PASCAL VOC	A benchmark in object category recognition and detection, providing standardized datasets and evaluation protocols.
Quantization	Quantization is a model optimization technique that lowers the numerical precision of neural network parameters such as weights and activations, reducing computational power and memory footprint.
Sparse data	Sparse data refers to datasets where a significant proportion of the values are zero, missing, or inactive.
Spike	A discrete event in an SNN, representing neuron firing, which is used in SNN as the primary means of communications between neurons.

Chapter 1

Introduction

1.1 Thesis Outline

Artificial intelligence has witnessed rapid growth in recent years, with deep learning models delivering cutting-edge results in areas like computer vision, language processing, and autonomous systems. However, these advances come at a high computational cost, relying heavily on energy-intensive GPU infrastructures. As AI moves toward deployment in mobile and edge devices, the need for more energy-efficient AI computing becomes increasingly desired [1].

Neuromorphic computing offers a compelling solution to this problem. By mimicking the event-driven and asynchronous nature of the human brain, SNNs and neuromorphic hardware, such as Brainchip's Akida AKD1000 processor, aim to deliver high computational performance at low power. Unlike traditional neural networks, SNNs transmit information via discrete spikes, enabling sparse and temporal computation, particularly suited for event-based data [2].

This thesis explores the conversion of a simple MNIST classification model and a complex Tiny YOLOv2 object detection model into simplified SNNs, and benchmarks its performance on both GPU and neuromorphic hardware. The goal is to evaluate latency, throughput, and energy efficiency, offering insight into converting CNNs to SNNs as well as deploying SNNs in neuromorphic hardware.

1.2 Problem Statement

The majority of contemporary SNN research is conducted via emulation on conventional GPU hardware [3]. This is primarily due to the accessibility and maturity of GPU infrastructures, in contrast to the relative scarcity and lack of standardized development tools of neuromorphic hardware. However, this emulation introduces discrepancies in performance, as GPUs are not optimized for the event-driven, asynchronous computation paradigm native to SNNs. Neuromorphic processors, such as the AKD1000, are architecturally tailored to exploit the temporal sparsity and energy-efficient characteristics of SNNs, potentially offering improvements in power-performance.

This thesis seeks to quantify the performance gap between GPU-based SNN emulation and deployment on dedicated neuromorphic hardware. Specifically, we investigate how latency, throughput and energy consumption differ between these two execution environments when running two similar SNN models.

1.3 Scope

The simpler object classification model is trained on the MNIST dataset, which consists of 28x28 greyscale pixel images. On the contrary the object detection model is trained on Pascal VOC2007 dataset which consists of coloured images with ranging pixel sizes of up to 500x500. The first mentioned model is assumed to be simple due to the sparse dataset and the latter model is refered to as complex due to training on complex data, relative to MNIST.

Chapter 2

Background

2.1 Neural Networks

Neural networks are computational models inspired by the structure and function of the biological brain. They consist of interconnected nodes, or neurons, organized in layers. Each neuron receives input signals, processes them, and transmits the output to neurons in the subsequent layer. The strength of each connection is represented by a trainable parameter called a weight [4], [5].

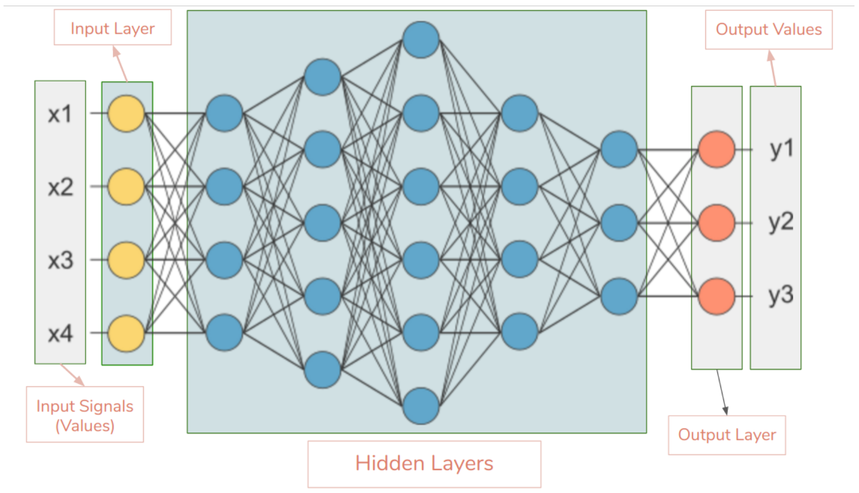

Figure 2.1.1: Visualization of a simple neural network [6].

The fundamental operation within a neuron is a weighted summation of inputs followed by the application of a non-linear activation function, such as ReLU (see figure below). Training a neural network involves adjusting these weights to minimize a loss function, typically using gradient-based optimization algorithms like backpropagation [5].

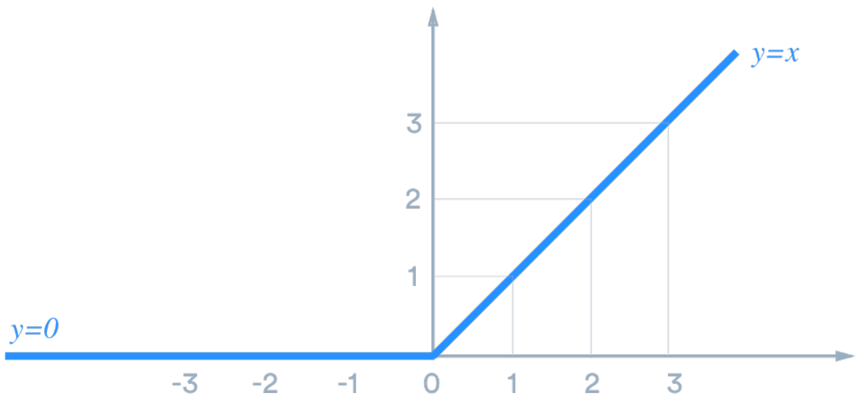

Figure 2.1.2: ReLU activation function [7].

2.1.1 Fully Connected Layers

A FC layer connects each neuron in one layer to each neuron in the next. FC layers are responsible for high-level reasoning and decision-making in a network. They are typically used in the final stages of a neural network to aggregate extracted features and produce classification outputs [8]. See visualization below Figure 2.1.3.

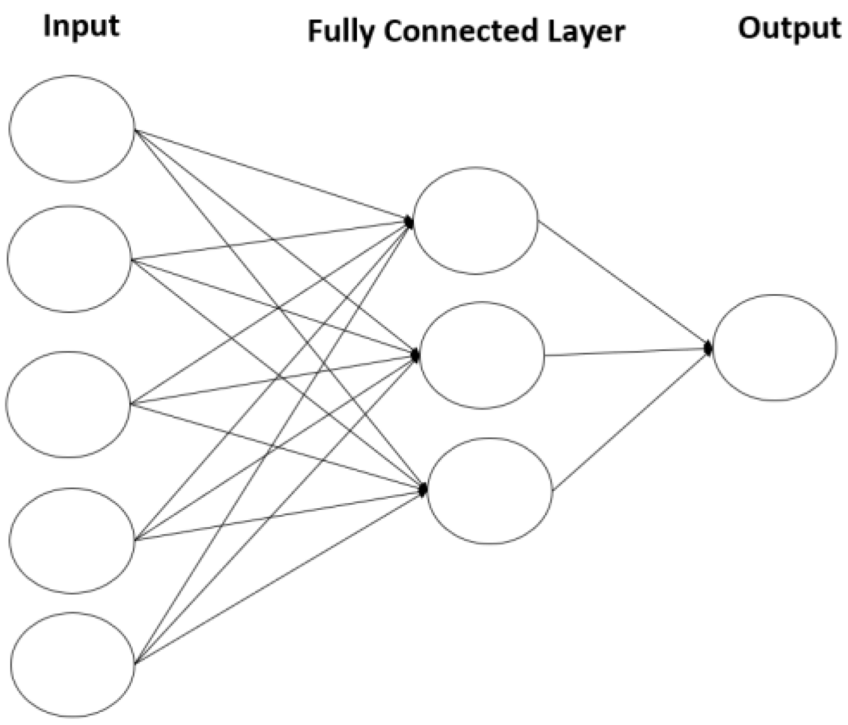

Figure 2.1.3: Example of a FC layer [9].

The output y of an FC layer can be expressed mathematically as:

$$y = f(Wx + b),$$

where x is the input vector, W is the weight matrix, b is the bias vector (acts as an offset), and f is a non-linear activation function [5].

2.1.2 Convolutional Neural Networks

CNNs are a class of neural networks specifically designed to process data with a grid-like topology, such as images. Unlike a FC layered neural network, where each neuron is connected to each neuron in the next layer, CNNs exploit the local spatial structure of input data by introducing convolutional layers [10].

A convolutional layer applies small, learnable filters, called kernels, which slide across the input feature map. Feature map is a 2D representation of features extracted from the input data. Each kernel is responsible for detecting specific local patterns, such as edges, corners, or textures, and generates a corresponding feature map that highlights the presence and spatial location of the detected feature [10]. As the depth of the network increases, subsequent convolutional layers are able to capture not only low-level patterns but increasingly abstract representations of high-level semantic features.

Mathematically, the convolution operation between an input x and a kernel k at a position (i, j) can be described as:

$$y(i,j) = (x * k)(i,j) = \sum_m \sum_n x(i+m, j+n) \cdot k(m,n),$$

where the input region is locally multiplied and summed with the kernel weights, producing a new value in the output feature map [10].

To further manage the spatial dimensions and computational complexity, CNN architectures often integrate pooling layers. A pooling layer performs a downsampling operation on feature maps, typically by summarizing local regions using operations such as max-pooling or average-pooling. In max-pooling, for instance, the maximum value within a small window is selected, effectively reducing the size of the feature map while retaining the bigges piece of information. Pooling not only decreases the number of parameters and computational load but also introduces a degree of invariance, making the network more robust to small shifts and distortions in the input [10].

CNNs progressively transform raw input data into compact data by alternating convolutional and pooling layers, making them suitable for classification or detection tasks [10]. See example below.

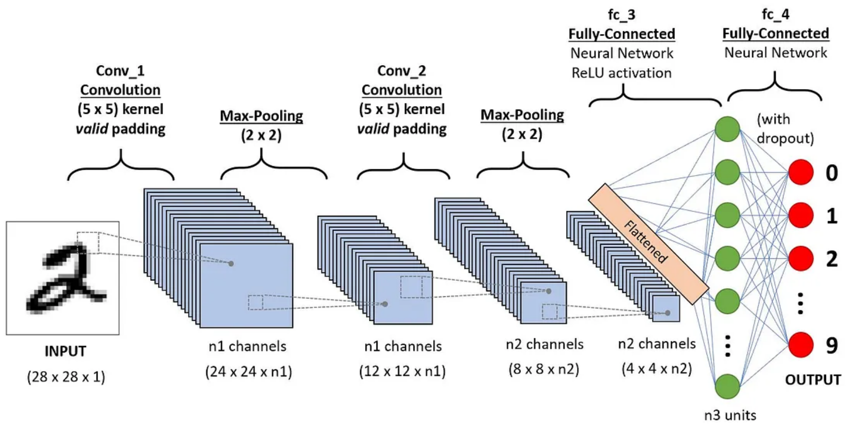

Figure 2.1.4: A CNN sequence to classify handwritten digits [11].

2.2 Spiking Neural Networks

2.2.1 History of SNNs

The origins of SNNs are rooted in the pursuit of mimicking the behaviors of biological neurons more faithfully than traditional CNNs. Inspired by the neural activity in the brain, early explorations into neuromorphic computing began in the late 1980s and early 1990s, notably with Mahowald and Mead's development of the first "silicon retina" [12]. This pioneering work set the stage for hardware implementations of neuron-like processing units.

Contrary to first-generation neural networks, which used simple threshold-based units, and second-generation networks, which introduced continuous activation functions and gradient-based learning, SNNs emerged as a third-generation neural network model [13]. Introducing temporal dynamics and event-driven computation. Unlike CNNs, which process information in a continuous and dense fashion.

As summarized in Gallego et al.'s comprehensive survey [14], the field saw significant growth with the development of event-based sensors and the realization that traditional frame-based vision systems were inefficient in dynamic or power-constrained environments. This gave rise to event-driven learning algorithms, such as STDP, and frameworks for converting trained CNNs into equivalent SNNs [15].

While early SNN implementations faced challenges including poor training algorithms, hardware limitations, and a lack of standardized benchmarks, recent advances in neuromorphic hardware (e.g., Intel's Loihi, Brainchip's Akida) and simulation tools like `snnTorch` have accelerated development [16]. Modern strategies increasingly combine CNN pretraining followed by conversion into SNNs, balancing learning efficiency with deployment benefits on low-power, event-driven platforms.

2.2.2 How do SNNs work?

SNNs are a biologically inspired class of neural networks that attempt to replicate how neurons operate in the human brain. Unlike traditional CNNs, which rely on continuous-valued activations and synchronous processing, SNNs transmit information via discrete spikes. A spike is generated when multiple inputs to a neuron sums up to a spike strong enough to send a signal. This summation is an integrator over time, which introduces a temporal dimension into computation, enabling SNNs to naturally process spatial-temporal data [17].

As illustrated in Figure 2.2.1, input data such as pixel intensities are first encoded into spike trains by dedicated encoding neurons [18]. These are shown as gray-coloured nodes in the figure. The spike train is then propagated through the network to learning neurons via synapses, where the synaptic weight modulates the effect of each incoming spike. The weighted spikes are accumulated at the membrane of the post-synaptic neuron. Once the membrane potential crosses a defined threshold, the neuron fires a spike, continuing the process.

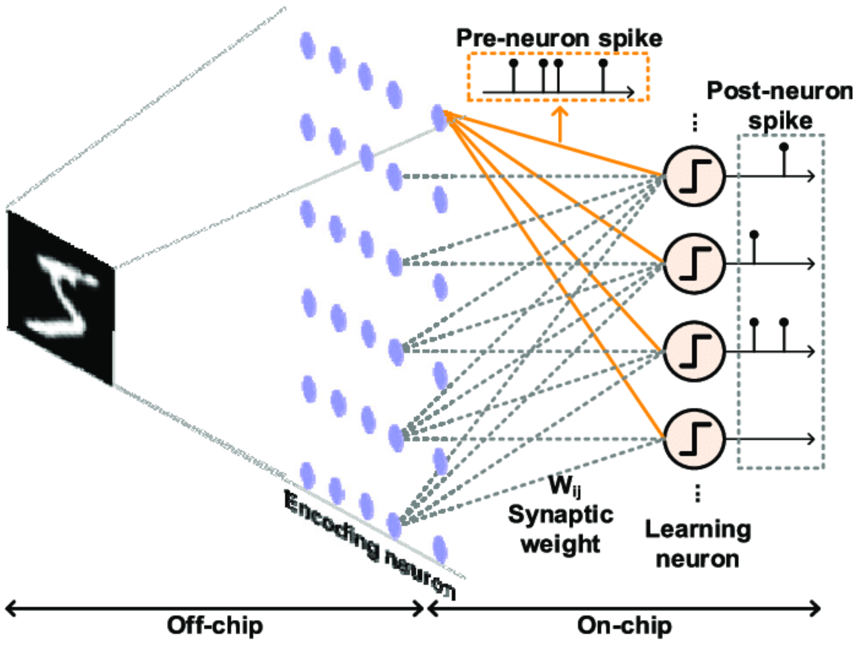

Figure 2.2.1: Overview of SNN spike propagation and neuron firing process [18].

The fundamental design of SNNs supports event-driven computation, where neurons remain inactive unless relevant input stimuli are present. This leads to sparse activity and substantially lower power consumption compared to dense

CNNs [18], [19]. Moreover, since spike events are one-bit signals, memory and bandwidth requirements are drastically reduced.

Despite these advantages, SNNs are primarily simulated on GPU platforms due to the limited availability of neuromorphic hardware. GPUs are widely accessible and offer high parallelism, making them a practical platform for SNN emulation [20]. However, as GPUs are based on the von Neumann architecture, where processing and memory are physically separated, they suffer from memory bottlenecks and elevated power consumption when running event-based SNN models [21].

2.2.3 Training SNNs

SNNs present unique training challenges due to their event-driven nature and non-differentiable activation functions. Unlike traditional CNNs that utilize continuous activation functions amenable to gradient-based optimization, SNNs rely on discrete spike-based computations. Two primary training methodologies have emerged, CNN-to-SNN conversion and direct training of SNNs [22].

The first mentioned approach involves training a conventional CNN using standard backpropagation techniques and subsequently converting it to an SNN. The conversion process typically aligns CNN activations with SNN firing rates, facilitating the approximation of spike behaviour through rate encoding. Notably, Kim et al. introduced Spiking-YOLO, the first spike-based object detection model, achieving performance comparable to its CNN counterpart while significantly reducing energy consumption [23].

While CNN-to-SNN conversion can yield models with performance close to their ANN equivalents, it often necessitates approximations for components like batch normalization and pooling layers, which are not directly transferable to SNNs [22]. Additionally, deeper networks may experience increased latency due to the higher spike counts required to emulate CNN behaviour accurately.

While direct training aims to leverage their inherent temporal dynamics and sparsity. The primary challenge here is the non-differentiability of spike generation. Solutions are categorized as follows:

- Unsupervised Learning: Methods such as STDP adjust synaptic weights based on the timing of spikes. Diehl and Cook demonstrated unsupervised digit recognition using STDP, highlighting the potential for energy-efficient learning [24].
- Supervised Learning: Surrogate gradient techniques approximate the gradient of the spiking function, enabling backpropagation. For instance, Zenke and Ganguli proposed the SuperSpike algorithm, facilitating supervised learning in multi-layer SNNs [25]. Additionally, Su et al. introduced EMS-YOLO, a directly trained SNN for object detection, achieving performance surpassing CNN-SNN conversion methods with reduced time steps [26].

Direct training methods allow SNNs to fully exploit their temporal coding capabilities, potentially leading to more efficient models. However, these approaches often require meticulous tuning and may not yet achieve the accuracy levels of converted models on complex datasets [22].

Hybrid training strategies have evolved to address the challenges of CNN-to-SNN conversion by minimizing quantization and sequential errors. In particular, the Fast-SNN framework [27] proposes a two-stage approach designed to produce high-performing SNNs with low inference latency. Instead of applying unsupervised learning or direct spike-based backpropagation, Fast-SNN begins by training a quantized CNN using supervised learning. During this stage, both the clipping range and the distribution of weights and activations are optimized to minimize quantization error. This effectively aligns the CNN's output activations with the spike rates expected in the converted SNN.

After quantization-aware training, the model undergoes conversion into a spiking format, where a novel signed integrate-and-fire neuron model is introduced to address errors caused by misfired spikes. To further correct discrepancies that accumulate across layers during conversion, a layer-wise fine-tuning mechanism is applied [27]. This second stage reduces sequential error by adjusting the SNN's firing rates to better match the CNN's activations on a per-layer basis. Through this hybrid conversion and fine-tuning process, Fast-SNN achieves state-of-the-art results on complex tasks like object detection while maintaining a low number of simulation time steps.

2.2.4 AKD1000

The Akida AKD1000 is a system-on-chip developed by BrainChip, designed specifically as an AI hardware accelerator, with a maximal clock rate of 300 MHz [28]. Similar to other neuromorphic processors, Akida does not use traditional neuron models. Instead, it is designed to operate similarly to event-based vision sensors, where information is transmitted as discrete spike events rather than continuous values. Akida networks also only operates using integers to reduce power consumption and memory [29].

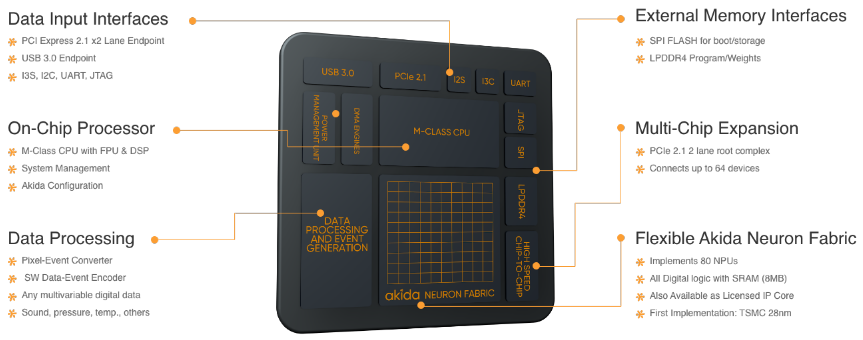
Data Input Interfaces
PCI Express 2.1 x2 Lane Endpoint
USB 3.0 Endpoint
I3S, I2C, UART, JTAG
On-Chip Processor
M-Class CPU with FPU & DSP
System Management
Akida Configuration
Data Processing
Pixel-Event Converter
SW Data-Event Encoder
Any multivariable digital data
Sound, pressure, temp., others
USB 3.0
PCIe 2.1
I2S
I3C
UART
POWER MANAGEMENT UNIT
DMA ENGINES
M-CLASS CPU
JTAG
SPI
LPDDR4
DATA PROCESSING AND EVENT GENERATION
HIGH SPEED CHIP-TO-CHIP
akida NEURON FABRIC
External Memory Interfaces
SPI FLASH for boot/storage
LPDDR4 Program/Weights
Multi-Chip Expansion
PCIe 2.1 2 lane root complex
Connects up to 64 devices
Flexible Akida Neuron Fabric
Implements 80 NPUs
All Digital logic with SRAM (8MB)
Also Available as Licensed IP Core
First Implementation: TSMC 28nm

Figure 2.2.2: Overview of AKD1000 chip, from Brainchip [30].

To encode input data, Akida employs Rank Order Coding [30], [31]. This encoding method captures the relative order of spike events, prioritizing the rank or sequence in which inputs are received rather than relying on precise spike timing. This allows the processor to interpret data efficiently, using a sparse and asynchronous communication model inspired by biological systems.

Unlike conventional GPUs which operate at a state fixed clock frequency, the Akida AKD1000 features dynamic clocking, adjusting its operational frequency based on workload demands and spike activity. This adaptive range is 10 to 300 MHz. It enables the processor to save additional energy during sparse or less demanding inference tasks [28]. This behaviour is a direct result of Akida's event-driven architecture, which eliminates idle computations and dynamically scales processing effort according to the volume and distribution of input spikes. Such variability supports efficient deployment in edge scenarios, where power savings often outweigh the need for consistent high-speed performance. The AKD1000 also utilizes in-memory processing, significantly limiting the need for data movement, increasing both energy efficiency and inference latency [32].

2.2.5 CNN2SNN Toolkit

Deploying a SNN on neuromorphic hardware like Brainchip's Akida processor requires a tailored model conversion process. To streamline this transition, Brainchip provides the CNN2SNN toolkit, which forms the final stage of their model deployment pipeline, as illustrated in Figure 2.2.3 [29].

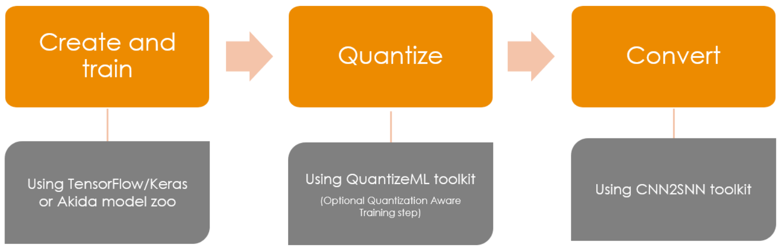
Create and train
Quantize
Convert
Using TensorFlow/Keras or Akida model zoo
Using QuantizeML toolkit
(Optional Quantization Aware Training step)
Using CNN2SNN toolkit

Figure 2.2.3: Depiction of Akida workflow. From Brainchip documentation [29].

The workflow begins by creating and training a CNN using either TensorFlow/Keras framework or a pre-trained model from the Akida Model Zoo. These CNNs are built using standard deep learning practices, allowing researchers to leverage familiar tooling and transfer learning techniques to develop performant models. However, in order to meet the constraints of neuromorphic deployment, particularly power efficiency and sparse event-driven computation, the model must be both quantized and converted to a spiking format.

The second step involves model quantization. Using Brainchip's QuantizeML toolkit, weights and activations are reduced to a lower bit fixed-point precision. This significantly reduces memory and computational overhead, enabling energy-efficient execution on hardware while maintaining acceptable accuracy. The toolkit optionally supports quantization-aware training, which allows the model to adjust to low-precision constraints during training, improving final performance on the Akida chip [33], [34].

In the final step, the quantized CNN is passed through the CNN2SNN toolkit. This component is responsible for mapping the CNN architecture to an SNN format that is compatible with Akida's architecture. Unlike conventional conversion workflows that require manual spike encoding or major reengineering, CNN2SNN automates the transformation by internally handling temporal encoding, spike routing, and architectural constraints unique to the Akida hardware. The resulting model leverages Akida's rank-order coding scheme and is structured for sparse, event-driven inference, enabling real-time deployment with minimal energy usage.

2.3 Graphics Processing Unit

GPUs are specialized hardware accelerators originally designed for rendering graphics in personal computers. Over the past two decades, GPUs have become dominant in fields beyond graphics, including scientific computing, cryptography, and most notably, deep learning, due to their ability to perform massive parallel computations efficiently [35].

At their core, GPUs are composed of thousands of small cores capable of executing operations in parallel [35]. This architecture is particularly well-suited for tasks like matrix multiplications, convolution operations, and element-wise arithmetic, fundamental building blocks of deep neural networks.

Key architectural features of GPUs include:

- Clock Cycles: GPUs operate at high clock speeds (typically in the range of 1–2 GHz), executing many instructions per second [35]. However, performance is determined less by raw speed, but rather by the parallelism and memory throughput.
- Memory Hierarchy: GPUs include global memory (high latency, large capacity), shared memory (medium latency, per block), and registers (low latency, per thread) [35]. Efficient memory access patterns are critical for high performance.
- Parallelism: GPUs use a Single Instruction, Multiple Threads execution

model, where thousands of threads execute the same instruction concurrently [35]. This architecture is ideal for batched matrix operations typical in convolutional neural networks.

GPUs are widely accessible through consumer-grade hardware, gaming laptops, and cloud computing platforms (e.g., Google Colab, AWS, and Azure). Their ease of access and mature software ecosystem, such as CUDA, make them the default compute platform for training and deploying modern neural networks [35].

2.3.1 GPU vs. Neuromorphic Processors

Neuromorphic processors represent a fundamental departure from the von Neumann architecture found in GPUs. Inspired by the structure and function of biological neural systems, neuromorphic processors aim to replicate the event-driven, asynchronous nature of the brain.

Key differences between GPUs and neuromorphic processors include:

- **Clocking and Timing**: GPUs rely on a global clock to synchronize computation, executing operations in discrete steps. In contrast, neuromorphic processors are asynchronous and event-driven. Computation occurs only when spikes (events) are received, eliminating idle cycles and reducing unnecessary energy consumption.
- **Memory and Processing Integration**: GPUs separate memory and processing units, which leads to a von Neumann bottleneck. Neuromorphic architectures co-locate memory and computation within small units, often called cores, which operate independently. This mimics the biological principle of neurons storing their own state and updating based on local input, reducing data transfers.
- **Data Representation**: GPUs process high precision, floating point numbers unless explicitly quantized. Neuromorphic processors typically work with binary or low-precision spiking signals, using temporal information to increase signal information.
- **Processing Paradigm**: While GPUs are optimized for dense, continuous

computation, neuromorphic processors are optimized for sparse, discontinuous activity. This results in power savings, particularly for applications with low spike rates.

2.3.2 Clock cycles as a Performance Metric

A clock cycle represents the smallest unit of time in a digital system, defined by the period of the processor's internal clock signal. It dictates how often operations are executed. While the architectural paradigms of GPUs and neuromorphic processors differ significantly, the concept of a clock cycle remains fundamentally the same; a discrete unit tied to the system's clock frequency, typically measured in hertz [36]. In both systems, elapsed time can be measured by counting the number of clock cycles consumed during a specific task. This shared temporal unit enables meaningful comparisons between the two hardware platforms.

However, the way clock cycles manifest differs due to their computational models. GPUs execute operations in dense, synchronized batches tied closely to clock ticks. Neuromorphic processors, in contrast, are event-driven and operate asynchronously, but still rely on underlying clocked digital circuits to process and propagate spikes. Thus, although spike-driven activity appears continuous and unsynchronized at a higher abstraction level, each internal logic operation, such as synaptic accumulation or threshold, is still governed by an internal clock. To compare performance across platforms, the number of clock cycles can be profiled using low-level hardware counters or software instrumentation tools. This enables a fair analysis of latency, throughput, and energy efficiency, using clock normalized execution time as a common ground [36].

2.4 Image Classification

Image classification is a fundamental task in computer vision that involves assigning a label to an input image from a predefined set of categories. Conventional image classification pipelines rely on preprocessing the input, passing it through deep CNNs, and utilizing fully connected layers to predict class probabilities. CNNs have demonstrated outstanding performance on benchmarks such as

ImageNet, benefiting from hierarchical feature extraction and large-scale supervised training [37].

However, traditional CNNs require substantial computational resources, relying heavily on floating-point operations, leading to high power consumption and making them less suitable for deployment in resource-constrained environments. To address these challenges, researchers have investigated efficient network architectures, including binary and ternary neural networks, where activations and weights are restricted to a limited set of discrete values [38].

SNNs represent another energy-efficient alternative, operating with discrete spike-based computations instead of continuous activations. This event-driven paradigm naturally aligns with neuromorphic hardware design principles, offering significant reductions in both power consumption and memory usage.

2.4.1 GXNOR-Net

GXNOR-Net is a neural network architecture designed to operate with both ternary synaptic weights and ternary neuronal activations, taking values in the set {-1, 0, 1}. This ternarization enables significant reductions in memory usage and computational complexity, making GXNOR-Net highly suited for low-power and limited memory implementations. The "GXNOR" name stands for Gated XNOR, highlighting its event-driven computation mechanism where operations are only executed when both the weight and activation are non-zero [39].

Unlike conventional neural networks that rely on floating-point operations, GXNOR-Net replaces them with lightweight logical operations [39]. The presence of a zero state introduces computational sparsity, meaning many operations can be skipped entirely. This "gating" behavior allows inactive computation units to remain in a resting state, reducing energy consumption significantly.

During the forward pass, a ternary quantization function discretizes neuron activations. Only when the weighted sum of inputs surpasses a certain threshold does the neuron activate to either +1 or -1. In the backward pass, a differentiable surrogate gradient is used to approximate the derivative of the non-differentiable activation function, allowing effective learning via backpropagation [39]. Weights are updated using a probabilistic Discrete State Transition method, which avoids storing hidden full-precision weights by projecting gradients directly into the discrete space.

2.5 Object Detection Models

Object detection extends beyond simple image classification by simultaneously localizing and classifying multiple objects within a single image [40]. While image classification predicts a single label for an entire image, object detection requires predicting multiple bounding boxes and their corresponding class labels, making it inherently more complex.

Early object detection methods relied on sliding window techniques with hand-crafted features such as Histogram of Oriented Gradients and Support Vector Machines [41]. Although foundational, these approaches were computationally expensive and struggled with detecting objects at varying scales and locations.

The introduction of deep learning significantly advanced object detection. Region proposal-based methods like R-CNN [42] and its successors Fast R-CNN [43] and Faster R-CNN [44] improved detection accuracy by first proposing candidate regions and then classifying them. However, these two-stage models often suffered from slower inference speeds, limiting their use in real-time applications.

To address the speed limitations, one-stage detectors were developed. Models such as Single Shot MultiBox Detector [45] and the YOLO family [46] reframed detection as a direct regression task, predicting bounding boxes and classes in a single pass through the network. This approach greatly enhanced inference speed while maintaining competitive accuracy.

Modern object detectors generally fall into two paradigms:

- Two-Stage Detectors: High accuracy models like Faster R-CNN that separate proposal generation and classification.

- One-Stage Detectors: Faster models like YOLO and SSD that predict detections directly without intermediate proposals.

2.5.1 YOLOv2

YOLOv2, also known as YOLO9000, is a real-time object detection model that reformulates object detection as a single regression task. Unlike traditional two-stage approaches that first generate proposals and then classify them, YOLOv2 passes the input image through a CNN only once to simultaneously predict bounding boxes and class probabilities across the entire image. This unified architecture allows for highly efficient inference, making the model well-suited for real-time applications [46].

At its core, YOLOv2 processes an input image through a deep CNN backbone, commonly based on the Darknet-19 architecture, which acts as a feature extractor. The input image is resized, typically to 416×416 pixels, and passed through multiple convolutional layers that progressively abstract spatial information. Toward the end of the network, YOLOv2 outputs a grid of feature maps, where each grid cell is responsible for predicting multiple bounding boxes, associated confidence scores (objectness), and class probabilities for objects whose centers fall within that cell [46].

Each predicted bounding box consists of:

- Bounding box coordinates (x, y, w, h), representing the centre and dimensions of the box relative to the grid cell.
- Objectness score, the predicted probability that the bounding box contains an object, regardless of its class.
- Class probabilities, the predicted likelihood of the object belonging to each of the predefined categories.

To manage the dense set of predictions, YOLOv2 introduces an objectness threshold during inference. Predictions with objectness scores below this threshold are discarded early, reducing the computational burden of post-processing by focusing only on the most confident detections.

However, since multiple overlapping bounding boxes may still remain for the same object, YOLOv2 applies non-maximum suppression as a post-processing step [46]. This selects the bounding box with the highest objectness score and suppresses all other boxes with significant overlap (measured by Intersection over Union, IoU), ensuring that only the most relevant detections are retained in the final output.

In addition to this pipeline, YOLOv2 incorporates several key innovations to improve detection accuracy and flexibility [46]:

- Anchor Boxes: Multiple anchor boxes per grid cell allow the model to predict objects of various sizes and aspect ratios, with anchor dimensions optimized using k-means clustering on the training set.
- Passthrough Layer: Low-level spatial features are concatenated with deeper features to enhance detection of smaller objects.
- Batch Normalization: Applied throughout the network to stabilize training and improve generalization.
- Multi-Scale Training: The model is trained on images of varying resolutions, enabling robustness across different input sizes.

All outputs are consolidated into a tensor of shape $S \times S \times (B \cdot 5 + C)$, where S is the grid size, B is the number of anchor boxes per cell, and C is the number of classes. This tensor, after applying the objectness threshold and non-maximum suppresion, yields the final set of bounding boxes and class labels [46].

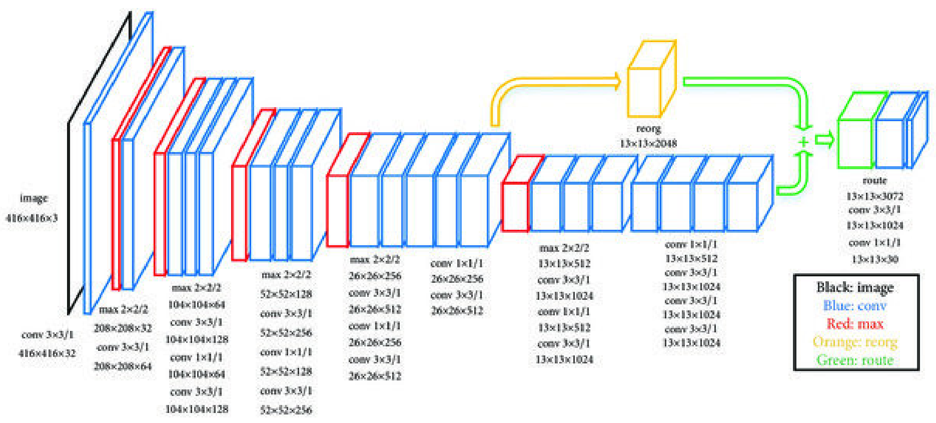

Figure 2.5.1: Overview of the YOLOv2 architecture. From Liu 2018 [47].

Figure 2.5.1 illustrates the overall YOLOv2 detection pipeline, as adapted from Liu et al. [47]. The network employs a fully convolutional architecture, eschewing fully connected layers to improve computational efficiency and maintain flexibility in input size. By integrating objectness thresholding and non-maximum supression, YOLOv2 achieves a balance between speed and accuracy, enabling efficient, real-time object detection across diverse scenarios.

Chapter 3

Method

3.1 Akida Neural Network Modelling

We employed two different SNN models; a classification model, and an object detection model. The classification model was constructed based on the GXNOR model, while the object detection used a YOLOv2 architecture.

For both implementations, the in-built pre-trained Akida Keras models were used. Both models were built on the AkidaNet architecture, yielding AKD1000 compatible Keras CNN models. Due to time constrictions, pre-trained Akida models were used. The GXNOR model was trained the MNIST dataset, while the YOLO model was trained to detect the person and car classes using the PASCAL VOC2007 dataset [48].

3.1.1 Datasets

The MNIST dataset is a standardized, and widely popular dataset used for image classification tasks. The dataset contains greyscale images of handwritten digits from 0 to 9. Each image is 28x28 pixels large. Due to the small size and the single colour channel, the data is considered sparse.

The PASCAL VOC2007 dataset is a popular benchmark dataset for object detection tasks. It contains images of varied size of up to 500x500 pixels with 3 colour channels; red, green, and blue. The dataset is had objects labelled across 20 classes, however, this study only utilizes 2 classes: person and car.

3.1.2 SNN Conversion

Following training, the MNIST classification and YOLO model were both converted into SNN to enable inference on neuromorphic hardware, by leveraging the CNN2SNN toolbox to quantize the CNN model to ensure compatibility and then convert the model to an Akida SNN model, following the Akida zoo examples [49], [50]. The pre-trained GXNOR model had been quantized to 2-bit weights , and the pre-trained YOLOv2 had been quantized to 4-bit weights.

3.2 GPU Simulation

While the mapping of the AkidaNet SNN models to the Akida hardware was trivial, the mapping of the SNN models to the GPU was non-trivial. To run SNN models using CUDA directly on the GPU (NVIDIA GeForce 1080 8 GB, Gainward), snnTorch was used to extend PyTorch to allow the GPU to utilize spiking neurons. For the MNIST classification model, the AkidaNet model architecture was replicated from the converted SNN model in snnTorch using the LIF neurons from snnTorch to simulate and approximate the hardware neuron processors and the correlated CNN layers used in the Akida models, as seen in Figures 3.2.1 and 3.2.2.

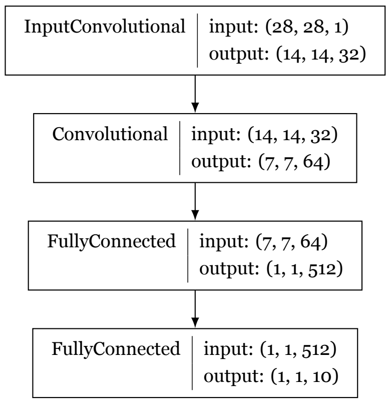

Figure 3.2.1: The converted MNIST classification SNN model on the Akida AKD1000

To simulate the neuromorphic hardware neurons present in the AKD1000, the snnTorch's `Leaky` layer was used, which simulates a first-order LIF neuron. This layer does not have any learnable parameters, and only performs transformations.

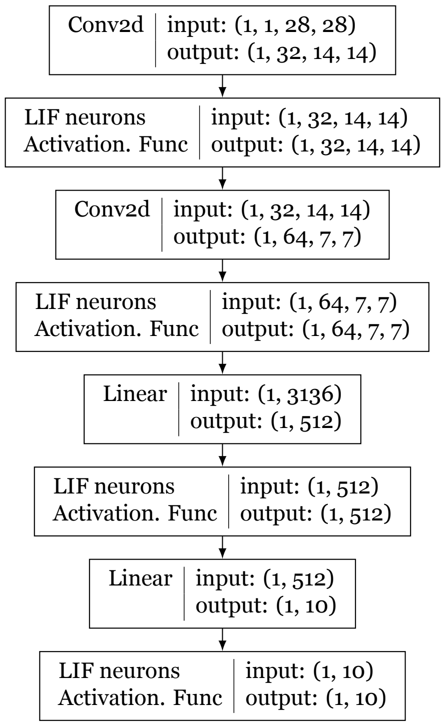

Figure 3.2.2: The MNIST classification SNN model translated to snnTorch layers

Due to the more complex nature of the YOLOv2 architecture, and the various backbone architectures that can be used, an existing SNN PyTorch implementation of the model was chosen. As the pre-trained Akida implementation was based on a Keras implementation using a Darknet backbone architecture, the chosen GPU model was a Tiny spiking YOLOv2 implementation using Darknet-19 architecture. Since Akida did not specify the exact Darknet backbone version, nor the exact dataset, the Akida model could not be identically replicated. As the Akida YOLOv2 model was quantized to 4-bit widths, it was decided to quantize the GPU spiking YOLOv2 to both 8 and 4-bit. The 4-bit model was used to have an equivalent model to the Akida implementation for a fair comparison, while the 8-bit implementation was used to study how the quantization may effect the model, and what trade-offs are made. It was not possible to do the same comparison between quantizations on the Akida, as we used a pre-trained model, already quantized.

3.2.1 Training of snnTorch Models

The distinct architectural differences of the Akida SNN models and the simulated snnTorch models, made transferring the pre-trained weights non-trivial. Thus, the models had to be re-trained. For the MNIST model, we leveraged PyTorch which made the training of the snnTorch model trivial as PyTorch uses heaviside function for spikes, which is differentiable, instead of a non-differentiable heaviside function. This allows for back-propagation through smooth surrogate gradient (in these models fast-sigmoid was used). The model used 2-bit weight quantization using Brevitas python library. The model was trained for 50 epochs, and the tuned for another 50 epochs after quantization.

For the YOLOv2 PyTorch model, we used the Fast-SNN framework to train and then quantize the CNN model to an 8-bit SNN YOLOv2 model and a 4-bit SNN YOLOv2 model [27]. The classification model were trained on the MNIST dataset while the YOLOv2 model were first trained to detect the person and car classes in the PASCAL VOC2007 dataset, similarly to the original SNN models. The YOLO models were trained for 250 epochs before quantization, and then post-quantization trained for another 250 epochs on the same dataset.

3.3 Evaluation

To assess the computational efficiency of the SNN model across the AKD1000 and GTX 1080, we adopted a benchmarking methodology focused on measuring inference energy consumption (mJ per inference), throughput (inference per clock cycle) and latency (time per inference). The core objective was to compare the performance of the same SNN model when executed on a conventional GPU and on the Akida NPU. This was done by measuring the throughput, latency and energy consumption of each inference. To ensure meaningful performance comparison, the throughput was measured as clock cycles, as clock rate locking was not supported by the GTX 1080. The AKD1000 ran on performance mode without any further limitations, while the GTX 1080 was power limited to 200 W and clock rate limited to 2050 MHz.

Inference latency was measured using built-in CUDA functions in PyTorch for the GPU and with built-in AKD1000 measurements. This was chosen to isolate the specific hardware's performance and remove the additional time that the CPU would add if using the Python's `time` module, for example. The throughput and the energy consumption was measured using Akida's statistics module and NVIDIA's system management interface. Thus, only the energy drawn by the models processing the inference were measured.

The prediction performance of the YOLO models were measured with non-maximum suppression threshold of 0.5 and object confidence threshold of 0.1.

3.3.1 Statistical Analysis

Outliers are data points that is statistically outside of the general patterns in a dataset and can be caused by measurement errors or by natural variances. The Z-score is a statistic to identify outliers in a dataset, which can cause skewed results and lead to poor generalizations. It identifies the outliers based on the number of standard deviations a point is from the mean:

$$Z = \frac{x_i - \overline{X}}{\sigma}, \tag{3.1}$$

where x_i is the data point, $\overline{X}$ is the dataset mean and σ is the standard deviation. Often a data point with a Z-score greater than ± 3 is considered an outlier. However, to be able to evaluate the performances of the different machine learning models and compare them, the modified Z-score was used instead as it reduces outliers in complex and is more suitable for non-normally distributed data.

$$Z_{\text{mod}} = \frac{x_i - \tilde{X}}{\text{MAD}}, \tag{3.2}$$

$$\text{MAD} = \text{median}(|x_i - \tilde{X}|),$$

where $\tilde{X}$ is the median of the dataset X. A data point would be considered an outlier if the Z-score was greater than ± 3-3.5.

The modified Z-score was used to pre-process the datasets, as it would improve the comparability of the resulting data points, by removing influential outliers that would otherwise distort or skew the dataset. It was chosen over the regular Z-score method as it does not assume normal distributed data, which may not hold in our measured datasets.

To determine if the differences are of any statistical significance between the two hardware mappings of the same model, the Student's *t*-test is used, with a significance level of $\alpha = 0.05$ was chosen:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}, \tag{3.3}$$

where $\overline{X}_1$ and $\overline{X}_2$ are the sample means, s_1^2 and s_2^2 are the sample variances and $n_1 = n_2$ is the sample size.

To study the relationships within different measured metrics, the Pearson correlation method was used. Pearson correlation is a commonly used method to measure linear statistical correlation between a pair of continuos variables.

$$r_{xy} = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}}, \tag{3.4}$$

where r_{xy} is the Pearson coefficient between x and y, n is the number of observations and x_i and y_i is the value of x and y for the i-th observation.

Spearman's correlation was used to study the relationship between discrete and continuos variables. It ranks the data points in each variable in descending order, establishing a basis for rank assignment. If multiple data points are identical, the rank becomes the average rank it would have otherwise taken. Then the correlation values is calculated using:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}, \tag{3.5}$$

where d_i is the difference in paired ranks and n is number of data points.

Chapter 4

Results

In this section, the performance metrics of the AKD1000 and the GPU across two tasks: MNIST classification and YOLO detection, are presented. Key metrics include accuracy, power consumption, processing time, and clock cycles. The results highlight the trade-offs between energy efficiency and processing speed, providing insights into the differences of each hardware.

The data was pre-processed before presented by using the modified Z-score with a threshold of 3 to remove outliers to get a more accurate and reliable representation of the average performance,

$$|Z_{\text{mod}}| \geq 3.$$

The non-processed data can be found in Appendix .1 and .2.

4.1 Classification Performance

Table 4.1.1: Relative differences in performance metrics between snnTorch and Akida AKD1000 for MNIST classification. Relative differences use the snnTorch as the base.

	Clock Cycles (10^3)	**Clock Speed (MHz)**	**Energy (mJ)**	**Accuracy (%)**	**Time (ms)**
–	12 037.173	1597.144	311.195	6.96	5.382
(%)	−98.287	−91.474	−99.520	7.71	−76.733

[a] Not possible to measure the correlation as the clock rate was constant.

For the MNIST classification model, the AKD1000 showed a reduction in energy consumption of 99.520 %, while being 7.71 % more accurate than the GPU. As shown in Table 4.1.1, the Akida platform also reduced the throughput by 98.287 % and the latency with 76.733 %, despite the clock speed being 91.474 % slower than its GPU counterpart.

Table 4.1.2: Summary Statistics for performance of SNN Classification model over 2000 samples on different hardware.

Metric	Clock Cycles (10^3)	Clock Speed (MHz)	Energy (mJ)	Accuracy (%)	Time (ms)
NVIDIA GeForce GTX 1080 (2-bit)[a]					
Mean	12 246.923	1746.000	312.697	90.25	7.014
Median	12 480.464	1746.000	316.848	100.00	7.148
Std	1584.513	0.000	41.238	29.68	0.908
CV %	12.938	0.000	13.188	32.89	12.938
Min	8210.670	1746.000	203.012	0.00	4.703
Max	20 461.388	1746.000	536.807	100.00	11.719
Akida AKD1000					
Mean	209.750	148.856	1.502	97.20	1.632
Median	209.941	162.000	1.186	100.00	1.288
Std	4.631	35.949	0.901	16.50	0.979
CV %	2.206	24.187	59.966	16.91	59.973
Min	198.644	35.000	1.072	0.00	1.164
Max	223.680	179.000	5.483	100.00	5.953

[a] The GPU was limited to a max clock rate of 2050 MHz and power limit of 200 W and could not be changed as the device did not support locked frame rates.

While both the GPU and the AKD1000 showed high predictive accuracy (90.25 % and 97.20 %, respectively), the difference was shown to be statistically significant ($p < 0.05$). As seen in Table 4.2.2, the coefficient of variation (CV) for accuracy was higher on the GPU platform than Akida. Furthermore, though the difference in energy consumption CV had no statistically significance ($p < 0.05$), the range of the 1080 (333.8 mJ) was 75 times larger than the range of the AKD1000 (4.411 mJ), seen in Table .1.1.

As seen in Figure 4.2.2, the throughput of both platforms were relatively consistent (CV of 12.938 % and 2.206 %, seen in Table .1.1). Though due to the different platforms clock speed management systems, the latency on AKD1000 was highly inconsistent (59.973 %) compared to the GTX 1080 (12.938 %).

Table 4.1.3: Correlation coefficients and p-values for the relationships between different variable pairs measured on MNIST model on GTX 1080 and AKD1000 GPUs. Pearson and Spearman correlation was used.

	GTX 1080		AKD1000	
Variables	**Correlation**	**p-value**	**Correlation**	**p-value**
Cycles & Energy	0.9677	**0.000**	0.0153	0.497
Cycles & Time	1.0000	**0.000**	0.0153	0.497
Clock rate & Time	nan[a]	nan[a]	-0.9667	**0.000**
Cycles & Accuracy	0.0187	0.403	0.0557	**0.014**
Energy & Accuracy	0.0224	0.317	0.0336	0.136

[a] Not possible to measure the correlation as the clock rate was constant.

As seen in Table 4.1.3, clock cycles and energy consumption had a very strong linear relationship on the GPU (0.9677), while clock cycles and elapsed time had a perfect linear relationship (1.0000), due to its fixed clock rate. On the AKD1000, there was a weak correlation between clock cycles and the predictive accuracy, see Table 4.1.3. There was also a nearly perfect, negative, linear relationship between the clock rate of the AKD1000 and the latency (-0.9667).

4.2 Object Detection Performance

The performance comparisons between the 4-bit snnTorch model and the AKD1000 show promising numbers for the Akida; 65.161 % reduction in clock cycles, 84.028 % reduction in clock speed, and 95.956 % reduction in energy, as seen in Figure 4.2.1. However, the AKD1000 also displayed an inference latency increase of 118.076 % compared to the 4-bit snnTorch, as well as an accuracy loss of 10.87 %, as seen in Table 4.2.1 and Figure 4.2.1.

Although the AKD1000 implementation had a lower mean accuracy, it had a substantially higher median accuracy while being almost twice as reliable, with a CV of 61.24 % compared to 114.13 %, seen in Table 4.2.1. This reliability difference was also observed between the the 8-bit and 4-bit snnTorch implementations, as seen in Table 4.2.1, indicating that the AKD1000 is able to store more information through temporal coding than the GPU. In fact, the 4-bit snnTorch implementation displayed the largest variance in all metrics except clock rate, where it had a constant 1746 MHz. With a cycles per inference CV of 6.558 %, energy consumption CV of 13.193 %, accuracy CV of 114.13 %, and latency of 6.558 %, the 4-bit

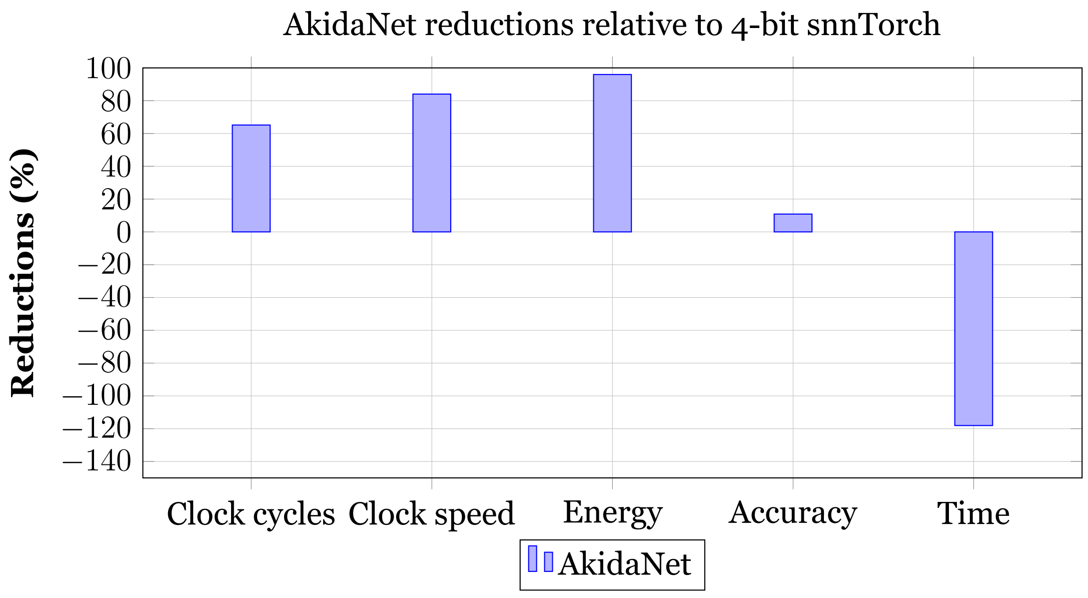

Figure 4.2.1: Comparison of the relative average reduction in key metrics: clock cycles, clock speed, energy, accuracy, and time, between the AkidaNet architecture, relative to the 4-bit snnTorch model. The blues represent the reduction in each metric relative to 4-bit snnTorch.

model proved to be the least reliable implementation of the three tested. However, it did prove a balanced mix compromise between the 8-bit GPU implementation and the AKD1000 implementation.

Table 4.2.1: Summary Statistics for performance of SNN YOLO detection model over 2000 samples on different hardware.

Metric	Clock Cycles (10^3)	Clock Speed (MHz)	Energy (mJ)	Accuracy (%)	Time (ms)
NVIDIA GeForce GTX 1080 (4-bit)[a]					
Mean	128 800.151	1746.000	3881.066	37.52	73.769
Median	130 484.393	1746.000	3889.811	0.00	74.733
Std	8447.142	0.000	512.023	42.82	4.838
CV %	6.558	0.000	13.193	114.13	6.558
Min	102 670.166	1746.000	2507.044	0.00	58.803
Max	154 631.350	1746.000	5080.029	100.00	88.563
Akida AKD1000					
Mean	44 872.340	278.870	156.956	33.44	160.872
Median	44 859.585	282.000	155.617	50.00	159.372
Std	457.148	10.953	6.438	20.48	6.715
CV %	1.019	3.927	4.102	61.24	4.174
Min	43 476.725	241.000	143.892	0.00	149.355
Max	46 269.710	292.000	179.420	100.00	184.880
NVIDIA GeForce GTX 1080 (8-bit)[a]					
Mean	2 027 748.176	1779.580	52 138.142	55.25	1139.508
Median	2 022 216.901	1746.000	51 852.227	50.00	1140.233
Std	61 585.474	43.119	2695.058	35.54	23.775
CV %	3.037	2.423	5.169	64.32	2.086
Min	1 872 626.348	1746.000	45 167.945	0.00	1070.454
Max	2 234 738.371	1847.000	61 308.953	100.00	1210.804

[a] The GPU was limited to a max clock rate of 2050 MHz and power limit of 200 W and could not be changed as the device did not support locked frame rates.

The transition from 8-bit to 4-bit precision in the snnTorch model resulted in substantial computational efficiency gains. Specifically, clock cycles decreased by 93.648 %, with corresponding reductions in power consumption (92.556 %) and inference time (93.544 %), as seen in Table 4.2.1. However, the extra quantization step introduced a statistically significant accuracy loss of 32.10 % ($p < 0.05$). It was observed that not only did the accuracy decrease after the quantization (reduction of 17.736 percentage points), the variation increased (42.82 vs 35.54 percentage points), with a substantial median drop from 50.00 % to 0.00 %.

Notably, there was a small, significant difference between the mean clock rate of the two snnTorch models (Δ = 33.580 MHz). As seen in Table 4.2.1, the 4-bit snnTorch model had a constant clock speed (CV of 0.000 %), similar to the MNIST snnTorch model, while the 8-bit version had to increase the clock rate for a subset of the inferences (CV of 2.423 %).

The AKD1000 implementation also demonstrated a substantial performance advantage when compared to the 8-bit snnTorch model. Specifically, Akida

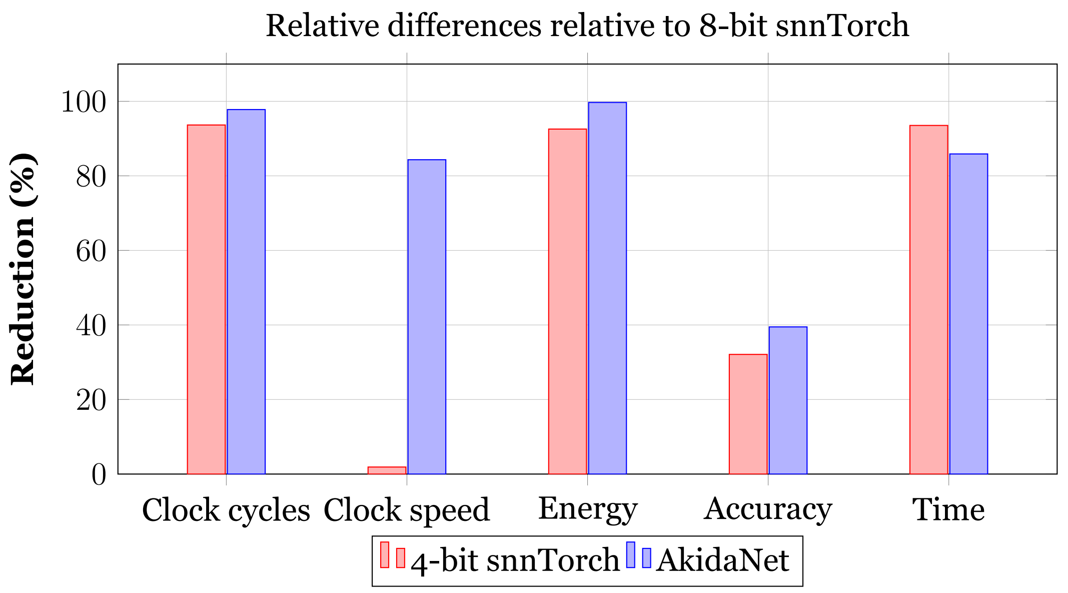

Figure 4.2.2: Comparison of the relative average reduction in key metrics: clock cycles, clock speed, energy, accuracy, and time, between the 4-bit snnTorch model and the AkidaNet architecture, relative to the 8-bit snnTorch model.

achieves a 97.787 % reduction in clock cycles, alongside a 99.699 % decrease in energy consumption (equivalent to 52 J) and an 85.882 % improvement in inference time (0.979 seconds), as shown in Table 4.2.1. In addition, compared to the 8-bit snnTorch model, Akida exhibits an 84.329 % reduction in clock frequency. However, the performance gain came at the cost of a statistically significant 39.48 % drop in classification accuracy ($p < 0.05$), as seen in Table 4.2.1. It has to be noted that while the mean prediction accuracy was higher for the 4-bit implementation (55.25 % vs 33.44 %), the median was the same for both models (50 %), even though the CV was similar (64.32 % vs 61.24 %).

Table 4.2.2: Correlation coefficients and p-values for the relationships between different variable pairs measured on YOLOv2 model on GTX 1080 (4- and 8-bit) and AKD1000 GPUs. Correlation and p-values was calcualted using Pearson and Spearman correlation method.

	GTX 1080 (8-bit)		GTX 1080 (4-bit)		AKD1000	
Variables	**Correlation**	**p-value**	**Correlation**	**p-value**	**Correlation**	**p-value**
Cycles & Energy	0.6425	**0.000**	0.5540	**0.000**	0.2119	**0.000**
Cycles & Time	0.6106	**0.000**	1.0000	**0.000**	0.2022	**0.000**
Clock rate & Time	-0.0793	**0.000**	nan[a]	nan[a]	-0.9689	**0.000**
Cycles & Accuracy	-0.0256	0.258	0.0618	**0.006**	-0.0828	**0.001**
Energy & Accuracy	-0.0324	0.153	0.0246	0.272	-0.0172	0.196

[a] Not possible to measure the correlation as the clock rate was constant.

All three implementations had a positive correlation between cycles and energy consumption, as seen in Table 4.2.2. The two GTX models had a moderate relationship (0.6425 and 0.5540), while the AKD1000 had a weak relationship (0.2119). Similarly, all three implementations had weak to perfect linear relationship between throughput and latency (0.6106, 1.0000 and 0.2022).

For the snnTorch 8-bit and the AKD1000 implementation, as time clock rate increases, latency decreases. The 8-bit implementation had very weak relationship (-0.0793), while the AKD1000 implementation had a very strong relationship (-0.9689).

For the snnTorch 4-bit and AKD1000 implementation, cycles and accuracy has a very weak correlation (0.0618 and -0.0828 respectively), as observed from Table 4.2.2. In the 8-bit implementation had no correlation, suggesting that the accuracy is dependant on other factors, such model architecture or training parameters.

4.3 Key Observations

- Neuromorphic Akida demonstrates 99.520 % (MNIST) and 95.956-99.699 % (YOLO) energy reduction compared to GPU for the same or similar networks.
- For simpler networks, Akida processes 76.733 % faster (1.622 ms vs 7.014 ms), proving its suitability for latency critical real-time processing tasks. However, for more complex models, the AKD1000 is outperformed by the GTX 1080, 73.769 ms to 160.872 ms.
- Akida's adaptive clocking (35-179 MHz) reduces clock speeds by an average of 86.610 % versus GPUs' relatively fixed 1746 MHz operation, reflecting its dynamic power engagement through sparse computing.
- The sparse input patterns, utilizing Akida's neuromorphic architecture, achieves up to 58 times fewer clock cycles through spike-based, asynchronous processing, demonstrating significant improvements in computational efficiency and suitability for edge AI systems.
- Akida correlation for MNIST model show that between clock cycles and both

energy and time were essentially zero (0.0153, with a p-value of 0.497). In contrast to the YOLOv2 model where it shows a small but definite correlation between clock cycles and both energy consumption (0.2119) and inference time (0.2022), with p-value below 0.05 (making the correlation plausible).

- Quantization has a major effect on all key metrics except for the clock speed on the GPU, reducing energy consumption, throughput, and latency by 85.9-99.9 %. This demonstrates the suitability and importance of quantization for deploying neural networks on resource limited hardware. However, for more complex neural networks, this comes at the cost of reduced accuracy, highlighting critical trade-off between computational efficiency and predictive performance.

Chapter 5

Discussions

In this section, we discuss the results to compare how our SNN models performed on the two different hardware platforms: GTX 1080 and AKD1000. While we developed both a 4-bit and 8-bit YOLOv2 model on the GPU, the 4-bit model serves as the benchmark for comparison against the Akida model.

This discussion aims to highlight the performance differences observed between the two platforms, emphasizing the implications of hardware architecture on model latency, throughput, efficiency and accuracy. Additionally, we will explore the limitations inherent in our methodologies and the hardware used, providing insights into the potential for future advancements in neuromorphic computing and SNN applications.

5.1 Understanding The Findings

Across both models, AKD1000 displayed an average energy reduction of 98.4 %. This was expected, as the AKD1000 architecture leverages event-driven computation and in-memory processing to reduce redundant computations, as outlined in sections 2.2.4 and 2.3.1.

Interestingly, energy consumption and accuracy had no correlation for either models or hardware platforms. A potential reason for this behaviour is the presence of redundant computations or the redundant on high-precision operations, resulting in greater energy usage without necessarily yielding higher accuracy. At the same time, small amounts of energy could be consumed to compute highly accurate predictions. This explains the uncorrelated relationship.

Additionally, due to the AKD1000's use of optimized temporal coding and sparse spike-based activation, discussed in sections 2.2.3 and 2.4.1, the AKD1000 performed between 65.2 % to 98.3 % less cycles than the GPU, depending on the models complexity. Although, the AKD1000's asynchronous, spike-based processing reduces the required clock cycles, the limited clock rate sets a limit to the potential gains in latency. For a simpler model such as the MNIST classification model, this had no practical effect on latency as the AKD1000 outperformed the GPU implementation by 76.7 %.

However, with a maximum clock rate of 300 MHz, the AKD1000 displayed an average latency increase of 118.1 % (or 87.1 ms), despite the substantial reduction of cycles. The Akida AKD1000 implementation of YOLOv2 achieved a latency of 160.9 ms, which is comparable to the human visual reaction time of approximately 180–200 ms [51]. While not a direct benchmark, this accentuates the potential for neuromorphic hardware to approach biologically plausible response times, aligning with its goal of brain-inspired energy-efficient computation. Combined with significant energy reduction, this highlights AKD1000's suitability for visual on-edge applications where near, or slightly better than human performance is acceptable.

The substantial increase in inference latency between the GPU and AKD1000 implementations also reveals some limitations in the AkidaNet architecture. The AkidaNet architecture lacks support for complex CNN features such as skip connections and floating-point operations as noted in sections 2.5.1 and 2.3.1. It may also be a result of the YOLOv2's architecture not effectively utilizing AKD1000's event-driven sparse processing as it is made up by multiple densely connected convolutional layers. Since GPUs are optimized for massively parallel computations, it is able to process and compute each inference faster, at an energy cost, as noted in 2.3.

Quantization drastically improved energy efficiency, throughput and latency, highlighting its importance for resource constrained hardware. These gains comes at an measurable accuracy cost, however, even with post-quantization calibration. This is likely a result of the model's sensitivity to reduced numerical precision. Thus, it appears that for high dimensional data with high ratio of noise, feature extraction becomes less reliable when decreasing the precision. While not observed, the high accuracy of the highly quantized image classification AKD1000's implementation may indicate that feature extraction on low dimensional data with low ratio of noise is more robust, making them more suitable for on-edge applications.

5.2 Hardware Evaluation

While the AKD1000 processor demonstrates strong potential in enabling energy efficient, event-driven computation, suitable for edge application, several limitations emerged during its evaluation that constrains its broader applicability.

The AKD1000 processor excelled at handling lightweight classification tasks such as low dimensional input data, and minimal architectural complexity, being superior to the GPU in all measured key metrics. When introducing higher model complexity and higher dimensional data, the AKD1000 remained superior in some key metrics, though its advantages diminished rapidly. However, the added complexity also introduced a trade-off in regard of accuracy. Due to the reduced precision, the model is unable to reliable extract features from the noisy and high dimensional images. This makes them unsuitable for edge application with dense continuous data and where accuracy is highly critical.

Latency

As seen by the results, the AKD1000 is able to outperform the GTX in terms of energy efficiency and throughput on both the simple and complex model, but the advantage of low latency seems to diminish with model complexity. For YOLOv2, Akida's inference time was approximately 55% slower than the GPU baseline, despite operating at a lower power. This highlights that while Akida excels in low energy, simple scenarios, it becomes increasingly inefficient for real-time processing of more demanding architectures. The fixed routing and lack of dynamic memory allocation further constrain Akida's flexibility in high-throughput contexts.

5.2.1 Memory constraints

Despite the simple classification model being architecturally identical on both hardware platforms, with quantized weights of 2-bits, the AkidaNet implementation is significantly superior in terms of accuracy. Both implementation have a high accuracy of 90.25 % and 97.20 %, which is made possible by how SNN uses temporal coding to add another dimension of information. However, as the AKD1000 processes the spikes on hardware, it is able to get higher definition temporal data than the GTX as it uses discrete simulation steps. This results in AKD1000 being more robust to low-bit precision and quantization than the GTX.

5.2.2 Correlation Differences in Akida

The correlation analysis conducted on the Akida AKD1000 platform for the MNIST and YOLOv2 models (Table 4.1.3) revealed an interesting divergence in the relationship between clock cycles, energy consumption, and inference time. For the simpler MNIST model, the correlations between clock cycles and both energy and time were essentially zero (0.0153, with a p-value of 0.497). This indicates that, in the case of MNIST, Akida's inherent event-driven processing and dynamic clocking capabilities effectively decouple the number of clock cycles from both energy usage and inference time. This can be attributed to the low computational complexity and high data sparsity of MNIST, which allows Akida to fully exploit its asynchronous and sparse event-driven architecture, where power consumption and latency are more influenced by input activity and neuron spiking patterns rather than raw clock cycles.

In contrast, the results for the more complex YOLOv2 model (Table 4.2.2) showed a, albeit weak, correlation between clock cycles and both energy consumption (0.2119) and inference time (0.2022), with p-values below 0.05 (making the correlation plausible). This indicates that as model complexity increases, the Akida processor demonstrates a partial convergence toward a more clock cycle-dependent behaviour, similar to conventional GPUs. A plausible explanation for this is that with higher model complexity and denser feature maps, as present in object detection tasks like YOLOv2, the inherent sparsity advantage of the Akida diminishes. More neurons are active simultaneously, leading to more uniform workloads and more predictable power and latency scaling with clock cycles.

5.3 Ethics & Privacy

The low energy consumption of the Akida allows it to be deployed directly on edge devices, allowing edge AI. This may enhance data privacy, a critical aspect of the digital world, both for individuals and corporations.

By enabling data processing directly on your own hardware, instead of using a cloud server, so that data can remain on your own device. This hides all data from public networks, reducing risks of interception by malicious parties. This means they are also able to be operated offline, having no connections to a public network. If device needs to communicate with other devices or servers on a network, they can transmit descriptive or statistical metadata only, abstracting the recorded data, while maintaining important information. This is especially critical for computer vision where data can be used to track, monitor or create fake imagery.

However, this edge deployment can cause security issues. By deploying AI directly on the edge device, they become vulnerable to direct physical attacks such as tempering or disabling. This privacy vulnerability is further increased if the device uses multi-stage processing, storing data to analyse patterns over time, as the attacker could access the stored data if left unencrypted.

5.4 Sustainability

The Akida can both directly and indirectly benefit environmental sustainability. An apparent direct benefit is its lower energy consumption. As shown, it is able to reduce energy consumption of a model with an average of 98.4 %, solving one of the major limitation of traditional AI, its high computational cost. However, as also proven in this study, Akida would not be able to completely replace AI due to its performance limitation for complex models. Complex models such as natural language processing models and generative AI would still required the usage of GPU.

Another direct benefit would be the reduced space required. As an Akida processor can be embedded in an edge device and used for edge AI, it removes the need for cloud processing. This would also eliminate the need for large cloud centres, freeing up both energy consumption and land. Indirectly, this also reduces energy consumption associated with data transmission and cooling of the centres.

5.5 Method Limitations

A key limitation of this study is the reliance on pre-existing machine learning models, rather than developing custom models. Due to constraints in time and expertise, we were unable to develop, design, and train models from scratch in multiple distinct frameworks. This limits the extent to which performance differences can be attributed only to the hardware, and introduces potential bias and trends based on the design and architectural differences between the models. However, it still enables valuable insights into the practical performance of similar models across frameworks and hardware.

Specifically, the Akida YOLOv2 model is pre-trained and based on the AkidaNet architecture instead of the Darknet-19 architecture used in the GPU. This means that they may not use or support the same types of layers or use the same amount of layers, thus, the architecture is not directly comparable. However, they are functionally equivalent with the same input and output dimensions. As our thesis compares how the different hardware platforms performs for the same task, the exact architecture of the models is not the primary focus; rather, the emphasis is on evaluating practical performance metrics such as latency, clock cycles, and energy consumption when executing equivalent workloads.

Another limitation is that the snnTorch MNIST classification implementation was not gradually quantized. This was due to how PyTorch and Brevitas functions. This means that, while the model architecture is 1-to-1, the trained models cannot be exactly comparable in terms of accuracy, but should have no significant effect on latency, clock cycles or energy based on the measured correlations.

Another limitation is that SNN models can be run on GPU using multiple different frameworks, which may all result in different energy efficiency, latency, and throughput, for the same model and architecture. Similarly, there are multiple different frameworks and neuromorphic processors that also will have different characteristics from each other. Thus, it was not possible to get a direct comparison between the hardware only, but rather a comparison between two different frameworks suited for two different hardware, GPU and Akida.

Lastly, for the YOLOv2 models, it was not possible to determine if the quantized trade-off are a result from the model complexity or the dense data input. A future improvement would be to run both model architectures (GXNOR and YOLOv2) trained on two different datasets, a sparse and a dense dataset. This would make it possible to separate these 2 variables, removing ambiguity in the results.

5.5.1 Future Work

Future work could mitigate these limitations by implementing a fully custom and quantized YOLOv2 model that is explicitly converted using both Akida's CNN2SNN pipeline and a GPU-based SNN simulator. This would enable a controlled comparison of spiking inference across heterogeneous platforms with better consistency in model structure and training data. It could be further improved by evaluating the same model on different GPU and neuromorphic frameworks to lessen the impact of framework specific trends.

Chapter 6

Conclusions

Overall, the Akida processor is well-suited for edge applications where low latency, power efficiency, and privacy are critical, such as real-time processing on embedded or edge devices. Due to its temporal coding, it is also highly suited for application where memory is a critical constraint, that GPU is unable to emulate. However, its limitations in scaling to more a complex deep learning architectures like object detection, and its reliance on quantized, simplified models, restrict its applicability to tasks where raw computational throughput or flexibility is required. These findings emphasize the importance of selecting the appropriate hardware platform based on task complexity and energy constraints.

Bibliography

[1] Shankar, S. and Reuther, A., "Trends in energy estimates for computing in AI/machine learning accelerators, supercomputers, and compute-intensive applications," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, ISSN: 2643-1971, Sep. 2022, pp. 1–8. DOI: 10.1109/HPEC55821.2022.9926296. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9926296 (visited on 05/28/2025).

[2] Malviya, R. K., Danda, R. R., Kumar, K., and Kumar, B. V., "Neuromorphic computing: Advancing energy-efficient AI systems through brain-inspired architectures," vol. 20, 2024.

[3] Bhuiyan, M. A., Pallipuram, V. K., Smith, M. C., Taha, T., and Jalasutram, R., "Acceleration of spiking neural networks in emerging multi-core and GPU architectures," in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, Apr. 2010, pp. 1–8. DOI: 10.1109/IPDPSW.2010.5470899. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5470899 (visited on 05/28/2025).

[4] Picton, P., "What is a neural network?" In *Introduction to Neural Networks*, P. Picton, Ed., London: Macmillan Education UK, 1994, pp. 1–12, ISBN: 978-1-349-13530-1. DOI: 10.1007/978-1-349-13530-1_1. [Online]. Available: https://doi.org/10.1007/978-1-349-13530-1_1 (visited on 05/28/2025).

[5] Jansson, P. A., "Neural networks: An overview," *Analytical Chemistry*, vol. 63, no. 6, 357A–362A, Mar. 15, 1991, ISSN: 0003-2700, 1520-6882.

DOI: 10.1021/ac00006a739. [Online]. Available: https://pubs.acs.org/doi/abs/10.1021/ac00006a739 (visited on 04/28/2025).

[6] Protocol, R. "Everything you need to know about neural networks," Raven-Protocol. (Jun. 25, 2018), [Online]. Available: https://medium.com/ravenprotocol/everything-you-need-to-know-about-neural-networks-6fcc7a15cb4 (visited on 04/28/2025).

[7] Liu, D. "A practical guide to ReLU," Medium. (Nov. 30, 2017), [Online]. Available: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7 (visited on 05/01/2025).

[8] Ma, W. and Lu, J., *An equivalence of fully connected layer and convolutional layer*, Dec. 4, 2017. DOI: 10.48550/arXiv.1712.01252. arXiv: 1712.01252[cs]. [Online]. Available: http://arxiv.org/abs/1712.01252 (visited on 05/28/2025).

[9] Kalaycı, T. A. and Asan, U., "Improving classification performance of fully connected layers by fuzzy clustering in transformed feature space," *Symmetry*, vol. 14, no. 4, p. 658, Apr. 2022, Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2073-8994. DOI: 10.3390/sym14040658. [Online]. Available: https://www.mdpi.com/2073-8994/14/4/658 (visited on 05/01/2025).

[10] Wu, J., "Introduction to convolutional neural networks," May 1, 2017. [Online]. Available: https://cs.nju.edu.cn/wujx/paper/CNN.pdf (visited on 04/04/2025).

[11] Saha, S. "A comprehensive guide to convolutional neural networks — the ELI5 way," TDS Archive. (Nov. 16, 2022), [Online]. Available: https://medium.com/data-science/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visited on 04/28/2025).

[12] Mahowald, M. and Douglas, R., "A silicon neuron," *Nature*, vol. 354, no. 6354, pp. 515–518, Dec. 1991, Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/354515a0. [Online]. Available: https://www.nature.com/articles/354515a0 (visited on 04/04/2025).

[13] Maass, W., “Networks of spiking neurons: The third generation of neural network models,” *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997, ISSN: 08936080. DOI: 10.1016/S0893-6080(97)00011-7. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0893608097000117 (visited on 04/04/2025).

[14] Gallego, G., Delbrück, T., Orchard, G., *et al.*, “Event-based vision: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 154–180, Jan. 2022, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2020.3008413. [Online]. Available: https://ieeexplore.ieee.org/document/9138762?denied= (visited on 04/04/2025).

[15] Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M., “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing,” in *2015 International Joint Conference on Neural Networks (IJCNN)*, ISSN: 2161-4407, Jul. 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280696. [Online]. Available: https://ieeexplore.ieee.org/document/7280696 (visited on 04/04/2025).

[16] “snnTorch documentation — snntorch 0.9.4 documentation.” (), [Online]. Available: https://snntorch.readthedocs.io/en/latest/ (visited on 04/15/2025).

[17] Xiao, C., Chen, J., and Wang, L., “Optimal mapping of spiking neural network to neuromorphic hardware for edge-AI,” *Sensors*, vol. 22, no. 19, p. 7248, Sep. 24, 2022, ISSN: 1424-8220. DOI: 10.3390/s22197248. [Online]. Available: https://www.mdpi.com/1424-8220/22/19/7248 (visited on 04/04/2025).

[18] Kim, G., Kim, K., Choi, S., Jang, H. J., and Jung, S.-O., “Area- and energy-efficient STDP learning algorithm for spiking neural network SoC,” *IEEE Access*, vol. 8, pp. 216 922–216 932, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3041946. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9276400 (visited on 04/16/2025).

[19] Huynh, P. K., Varshika, M. L., Paul, A., Isik, M., Balaji, A., and Das, A., *Implementing spiking neural networks on neuromorphic architectures:*

A review, Feb. 17, 2022. DOI: 10.48550/arXiv.2202.08897. arXiv: 2202.08897[cs]. [Online]. Available: http://arxiv.org/abs/2202.08897 (visited on 04/04/2025).

[20] Jung, B., Kalcher, M., Marinova, M., Powell, P., and Sakalli, E., "Neuromorphic computing - an overview," *Cognitive Science Student Journal*, no. 7, 2023. [Online]. Available: https://vm821.rz.uos.de/wp-content/uploads/2023/05/CognitiveScienceStudentJournal_007.pdf (visited on 04/04/2025).

[21] Sandamirskaya, Y., Kaboli, M., Conradt, J., and Celikel, T., "Neuromorphic computing hardware and neural architectures for robotics," *Science robotics*, vol. 7, eabl8419, Jun. 29, 2022. DOI: 10.1126/scirobotics.abl8419.

[22] Nunes, J. D., Carvalho, M., Carneiro, D., and Cardoso, J. S., "Spiking neural networks: A survey," *IEEE Access*, vol. 10, pp. 60 738–60 764, 2022, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3179968. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9787485 (visited on 04/04/2025).

[23] Kim, S., Park, S., Na, B., and Yoon, S., "Spiking-YOLO: Spiking neural network for energy-efficient object detection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 7, pp. 11 270–11 277, Apr. 3, 2020, Number: 07, ISSN: 2374-3468. DOI: 10.1609/aaai.v34i07.6787. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6787 (visited on 04/04/2025).

[24] Diehl, P. U. and Cook, M., "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, Aug. 3, 2015, Publisher: Frontiers, ISSN: 1662-5188. DOI: 10.3389/fncom.2015.00099. [Online]. Available: https://www.frontiersin.org/journals/computational-neuroscience/articles/10.3389/fncom.2015.00099/full (visited on 04/04/2025).

[25] Zenke, F. and Ganguli, S., "SuperSpike: Supervised learning in multi-layer spiking neural networks," *Neural Computation*, vol. 30, no. 6, pp. 1514–

1541, Jun. 2018, ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco_a_01086. arXiv: 1705.11146[q-bio]. [Online]. Available: http://arxiv.org/abs/1705.11146 (visited on 04/04/2025).

[26] Su, Q., Chou, Y., Hu, Y., *et al.*, *Deep directly-trained spiking neural networks for object detection*, Jul. 27, 2023. DOI: 10.48550/arXiv.2307.11411. arXiv: 2307.11411[cs]. [Online]. Available: http://arxiv.org/abs/2307.11411 (visited on 04/04/2025).

[27] Hu, Y., Zheng, Q., Jiang, X., and Pan, G., *Fast-SNN: Fast spiking neural network by converting quantized ANN*, May 31, 2023. DOI: 10.48550/arXiv.2305.19868. arXiv: 2305.19868[cs]. [Online]. Available: http://arxiv.org/abs/2305.19868 (visited on 05/08/2025).

[28] admin. "Products – akida neural processor SoC," BrainChip. (), [Online]. Available: https://brainchip.com/akida-neural-processor-soc/ (visited on 04/04/2025).

[29] "Akida user guide — akida examples documentation." (), [Online]. Available: https://doc.brainchipinc.com/user_guide/akida.html (visited on 05/09/2025).

[30] Neuromorphic, O. "Akida - BrainChip." (), [Online]. Available: https://open-neuromorphic.org/neuromorphic-computing/hardware/akida-brainchip/ (visited on 04/04/2025).

[31] Delorme, A., Perrinet, L., and Thorpe, S. J., "Networks of integrate-and-fire neurons using rank order coding b: Spike timing dependent plasticity and emergence of orientation selectivity," *Neurocomputing*, Computational Neuroscience: Trends in Research 2001, vol. 38-40, pp. 539–545, Jun. 1, 2001, ISSN: 0925-2312. DOI: 10.1016/S0925-2312(01)00403-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231201004039 (visited on 04/04/2025).

[32] Inc, B. "Akida™ PCIe board," BrainChip Inc. (), [Online]. Available: https://shop.brainchipinc.com/products/akida%e2%84%a2-development-kit-pcie-board (visited on 04/25/2025).

[33] Wei, L., Ma, Z., Yang, C., and Yao, Q., “Advances in the neural network quantization: A comprehensive review,” *Applied Sciences*, vol. 14, no. 17, p. 7445, Jan. 2024, Number: 17 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2076-3417. DOI: 10.3390/app14177445. [Online]. Available: https://www.mdpi.com/2076-3417/14/17/7445 (visited on 04/05/2025).

[34] Nagel, M., Fournarakis, M., Amjad, R. A., Bondarenko, Y., Baalen, M. v., and Blankevoort, T., *A white paper on neural network quantization*, Jun. 15, 2021. DOI: 10.48550/arXiv.2106.08295. arXiv: 2106.08295[cs]. [Online]. Available: http://arxiv.org/abs/2106.08295 (visited on 04/05/2025).

[35] Dally, W. J., Keckler, S. W., and Kirk, D. B., “Evolution of the graphics processing unit (GPU),” *IEEE Micro*, vol. 41, no. 6, pp. 42–51, Nov. 2021, ISSN: 1937-4143. DOI: 10.1109/MM.2021.3113475. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9623445 (visited on 05/28/2025).

[36] “Benchmarking neuromorphic computing for inference,” in *Industrial Artificial Intelligence Technologies and Applications*, 1st ed. New York: River Publishers, Aug. 29, 2023, pp. 1–19, ISBN: 978-1-003-37738-2. DOI: 10.1201/9781003377382-1. [Online]. Available: https://www.taylorfrancis.com/books/9781003377382/chapters/10.1201/9781003377382-1 (visited on 05/01/2025).

[37] Krizhevsky, A., Sutskever, I., and Hinton, G. E., “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (visited on 04/28/2025).

[38] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y., *Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1*, Mar. 17, 2016. DOI: 10.48550/

arXiv.1602.02830. arXiv: 1602.02830[cs]. [Online]. Available: http://arxiv.org/abs/1602.02830 (visited on 04/28/2025).

[39] Deng, L., Jiao, P., Pei, J., Wu, Z., and Li, G., *GXNOR-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework*, May 2, 2018. DOI: 10.48550/arXiv.1705.09283. arXiv: 1705.09283[cs]. [Online]. Available: http://arxiv.org/abs/1705.09283 (visited on 04/24/2025).

[40] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A., "The pascal visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010, ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-009-0275-4. [Online]. Available: http://link.springer.com/10.1007/s11263-009-0275-4 (visited on 04/28/2025).

[41] Dalal, N. and Triggs, B., "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, ISSN: 1063-6919, vol. 1, Jun. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177. [Online]. Available: https://ieeexplore.ieee.org/document/1467360 (visited on 04/28/2025).

[42] Girshick, R., Donahue, J., Darrell, T., and Malik, J., *Rich feature hierarchies for accurate object detection and semantic segmentation*, Oct. 22, 2014. DOI: 10.48550/arXiv.1311.2524. arXiv: 1311.2524[cs]. [Online]. Available: http://arxiv.org/abs/1311.2524 (visited on 04/28/2025).

[43] Girshick, R., "Fast r-CNN," in *2015 IEEE International Conference on Computer Vision (ICCV)*, ISSN: 2380-7504, Dec. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169. [Online]. Available: https://ieeexplore.ieee.org/document/7410526 (visited on 04/28/2025).

[44] Ren, S., He, K., Girshick, R., and Sun, J., *Faster r-CNN: Towards real-time object detection with region proposal networks*, Jan. 6, 2016. DOI: 10.48550/arXiv.1506.01497. arXiv: 1506.01497[cs]. [Online]. Available: http://arxiv.org/abs/1506.01497 (visited on 04/28/2025).

[45] Liu, W., Anguelov, D., Erhan, D., *et al.*, “SSD: Single shot MultiBox detector,” in vol. 9905, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2. arXiv: 1512.02325[cs]. [Online]. Available: http://arxiv.org/abs/1512.02325 (visited on 04/28/2025).

[46] Redmon, J. and Farhadi, A., *YOLO9000: Better, faster, stronger*, version: 1, Dec. 25, 2016. DOI: 10.48550/arXiv.1612.08242. arXiv: 1612.08242[cs]. [Online]. Available: http://arxiv.org/abs/1612.08242 (visited on 04/15/2025).

[47] Liu, Z., Chen, Z., Li, Z., and Hu, W., “An efficient pedestrian detection method based on YOLOv2,” *Mathematical Problems in Engineering*, vol. 2018, no. 1, p. 3 518 959, 2018, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2018/3518959, ISSN: 1563-5147. DOI: 10.1155/2018/3518959. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2018/3518959 (visited on 04/15/2025).

[48] “The PASCAL visual object classes homepage.” (), [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/ (visited on 04/25/2025).

[49] “GXNOR/MNIST inference — akida examples documentation.” (), [Online]. Available: https://brainchip-inc.github.io/akida_examples_2.3.0-doc-1/examples/general/plot_0_gxnor_mnist.html (visited on 05/15/2025).

[50] “YOLO/PASCAL-VOC detection tutorial — akida examples documentation.” (), [Online]. Available: https://brainchip-inc.github.io/akida_examples_2.3.0-doc-1/examples/general/plot_5_voc_yolo_detection.html#sphx-glr-examples-general-plot-5-voc-yolo-detection-py (visited on 05/15/2025).

[51] Thompson, P. D., Colebatch, J. G., Brown, P., *et al.*, “Voluntary stimulus-sensitive jerks and jumps mimicking myoclonus or pathological startle syndromes,” *Movement Disorders*, vol. 7, no. 3, pp. 257–262, 1992, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/mds.870070312, ISSN: 1531-8257. DOI: 10.1002/mds.870070312. [Online]. Available: https:

//onlinelibrary.wiley.com/doi/abs/10.1002/mds.870070312 (visited on 05/16/2025).

.1 Original MNIST data

Table .1.1: Summary Statistics of raw for performance of SNN Classification model over 2000 samples on different hardware.

Metric	Clock Cycles (10^3)	Clock Speed (MHz)	Energy (mJ)	Accuracy (%)	Time (ms)
NVIDIA GeForce GTX 1080 (4-bit)[a]					
Mean	12 246.923	1746.000	312.697	90.25	7.014
Median	12 480.464	1746.000	316.848	100.00	7.148
Std	1584.513	0.000	41.238	29.68	0.908
CV %	12.938	0.000	13.188	32.89	12.938
Min	8210.670	1746.000	203.012	0.00	4.703
Max	20 461.388	1746.000	536.807	100.00	11.719
Akida AKD1000					
Mean	209 751.774	146.796	1.621	97.20	1.761
Median	209 935.500	162.000	1.186	100.00	1.289
Std	4646.452	39.092	1.318	16.50	1.432
CV %	2.215	26.630	81.326	16.98	0.000
Min	198 644.000	10.000	1.072	0.00	1.164
Max	224 372.000	179.000	18.327	100.00	19.899

[a] The GPU was limited to a max clock rate of 2050 MHz and power limit of 200 W and could not be changed as the device did not support locked frame rates.

.2 Original YOLOv2 data

Metric	Clock Cycles (10^3)	Clock Speed (MHz)	Energy (mJ)	Accuracy (%)	Time (ms)
NVIDIA GeForce GTX 1080 (8-bit)[a]					
Mean	2 031 877.488	1783.276	52 319.915	55.40	1139.449
Median	2 023 213.616	1746.000	51 911.322	50.00	1140.429
Std	70 202.522	51.031	3027.523	35.54	24.114
CV %	3.455	2.862	5.787	64.15	0.000
Min	1 855 465.376	1746.000	45 167.945	0.00	1059.157
Max	2 404 808.871	2025.000	67 369.828	100.00	1210.804
NVIDIA GeForce GTX 1080 (4-bit)[a]					
Mean	128 622.427	1746.000	3875.544	37.64	73.667
Median	130 401.084	1746.000	3887.246	0.00	74.686
Std	8728.822	0.000	516.307	42.86	4.999
CV %	6.786	0.000	13.322	113.87	0.000
Min	94 236.117	1746.000	2307.466	0.00	53.973
Max	154 631.350	1746.000	5080.029	100.00	88.563
Akida AKD1000					
Mean	44 873.943	278.007	157.506	33.43	161.466
Median	44 860.466	281.000	155.775	50.00	159.519
Std	471.975	12.481	7.511	20.57	7.914
CV %	1.052	4.490	4.769	61.54	0.000
Min	42 799.697	211.000	143.892	0.00	148.276
Max	46 585.944	292.000	200.783	100.00	213.146

[a] The GPU was limited to a max clock rate of 2050 MHz and power limit of 200 W and could not be changed as the device did not support locked frame rates.

TRITA-EECS-EX-2025:329
Stockholm, Sweden 2025

www.kth.se