Energy and Temperature Analysis of AI/Machine Learning Algorithms on Different Hardware Systems

Vedant Karia †§ *, Murat Isik†§, Anurag Daram*, Ceyhun Efe Kayan‡,
Baris Taskin‡, Dhireesha Kudithipudi*, Sadasivan Shankar§¶C

§ SLAC National Accelerator Laboratory, Menlo Park, California, USA
*Neuromorphic AI Lab, University of Texas at San Antonio, San Antonio, Texas, USA
‡VLSI and Architecture Lab, Drexel University, Philadelphia, Pennsylvania, USA
¶Stanford University, Stanford, California, USA

***Abstract*—Neural Networks, especially architectures such as Graph Neural Networks, Spiking Neural Networks, Multilayer Perceptron Networks, Transformers, and methods such as Support Vector Machines, have shown significant influence in areas such as natural language processing, computer vision and time series analysis. This paper presents a systematic and comprehensive analysis of energy consumption across these architectures, executed on hardware platforms such as the CPU, GPU, and FPGA. Preliminary findings show that deploying ML algorithms on FPGAs, especially Transformer models in natural language processing tasks, consumes less energy than CPUs and GPUs. Furthermore, spiking neural networks exhibit distinct energy consumption patterns as a result of recurrent weight updates. This research introduces a refined energy estimation approach, applicable to custom accelerators and specialized designs. It presents a comprehensive energy footprint analysis across three domains: ML algorithms, hardware systems, and applications, providing insights for optimizing energy use in diverse ML methods across heterogeneous hardware and applications.**

***Index Terms*—machine learning, CPU, GPU, FPGA, natural language processing, computer vision, time series data, energy estimation**

†Authors contributed equally to this research. CCorresponding Author.

1. Introduction

In recent years, artificial intelligence (AI) has witnessed remarkable advancements, particularly in the fields of image recognition, natural language processing (NLP), and autonomous systems. Recent advances can be attributed to the availability of significant data, access to compute-intensive hardware driven by technology scaling, the development of tools and algorithms that are accurate and scale with the size of the problem, and tools that reduce the latency of the development cycle (1). However, as these algorithms become increasingly complex and computationally demanding, it is crucial to consider the energy implications associated with their execution on different architectures implemented on hardware. The energy consumption of computing systems has garnered significant attention due to its environmental and economic implications. As demand for AI applications continues to increase, the energy required in hardware platforms becomes a critical factor to consider(2). Understanding the energy consumption patterns of these platforms that run various algorithms is essential for designing efficient and sustainable large-scale AI systems. This analysis aims to investigate the energy consumption characteristics of various machine learning algorithms during inference when deployed on various Central Processing Unit (CPU), Graphics Processing Unit (GPU), and Field-Programmable Gate Arrays (FPGAs). Although training is reported to be more energy-intensive than inference, given the large search space and real-world applications, we first study inference. By quantifying energy consumption and comparing results between different platforms, we can gain insights into their respective efficiencies and identify potential areas for improvement. Furthermore, this analysis can guide developers, researchers, and system designers in selecting the most appropriate hardware platform based on energy considerations for their specific algorithm applications. Our reason for including FPGAs in this analysis is due to their inherent advantages of configurability, adaptability, and parallelism, which are essential for achieving the desired throughput rates in specific applications (3; 4). Concurrently, there has been a notable advancement in tool chains designed for FPGA-based application development, expanding their accessibility and utility within the developer community (5). To achieve the best performance and energy efficiency, many researchers have focused on building custom Applied Specific Integrated Circuits (ASICs) for accelerating network inference workloads. Despite being an attractive solution, ASICs cannot offer sufficient flexibility to accommodate the rapid evolution of machine learning algorithms (6). The main contributions of this paper are:

- Analysis of inference energy efficiency of various machine learning algorithms for three different types of applications on CPU, GPU, FPGA.
- Evaluation of the thermal impact of different ML topologies on CPU, GPU and FPGA.

The rest of the paper is organized as follows: **Section 2** discusses the methodology to estimate energy, performance, and temperature across applications and hardware. **Section 3** outlines the datasets used. **Section 4** describes the machine learning algorithms evaluated. **Section 5** focuses on hardware architectures and benchmarks. **Section 6** evaluates the energy and thermal performance of the algorithms on the hardware. **Section 7** summarizes key findings, emphasizing the importance of co-design in AI/ML for energy efficiency and suggesting future research directions.

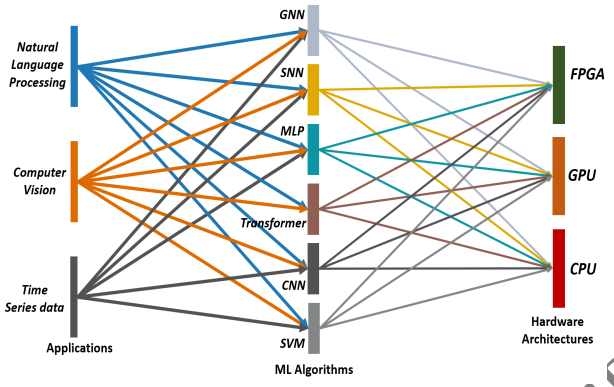


Figure 1: A flowchart of the analysis performed across three dimensions *i.e.* three types of applications simulated with six machine learning algorithms realized on three types of hardware architectures. Note: Graph Neural Networks (GNN), Spiking Neural Networks (SNN), Multi-layer Perceptron (MLP), Convolutional Neural Networks (CNN), and Support Vector Machines (SVM).

2. Related Work

The pursuit of energy-efficient computing has become a paramount concern in the field of ML and AI. As computational demands escalate, the energy consumption of ML models, particularly those involving deep learning, has come under scrutiny. This section provides an overview of the current state of the art in energy estimation tools and methodologies, underscoring the limitations of existing approaches and the necessity for versatile tools that span across various computing layers.

Energy estimation in computing systems is a critical area of research, and several tools have been developed to measure and analyze power consumption. These tools vary in their approach, accuracy, and the level of detail they provide. Marcher and Wattch, for instance, are architectural-level power analysis tools that provide insights into the energy profiles of processor architectures (7; 8). Power API offers a standardized approach to power measurement, enabling the integration of power-aware metrics in performance analysis (9). Accelergy (10) is a more recent tool that focuses on the energy estimation of accelerator designs, while ScaleSim (11) is tailored for the analysis of systolic arrays commonly used in deep learning accelerators.

Despite their contributions, these tools exhibit limitations, particularly in their flexibility and applicability across different computing layers. Many are confined to specific hardware architectures or lack the granularity required for a comprehensive analysis of complex ML models. Furthermore, they often do not account for the dynamic nature of ML workloads, leading to less accurate energy estimates.

The potential of top-down and bottom-up analysis in energy estimation is significant. Top-down approaches, which start at the application layer and work downwards, provide a macroscopic view of energy consumption, often relying on performance counters and software-level indicators. Bottom-up methods, conversely, begin at the hardware layer, offering a microscopic perspective that can capture the nuances of energy usage at the component level.

The literature presents a wealth of research that addresses the energy efficiency of ML models. (12) provides a comprehensive estimation of energy consumption in ML, offering a foundational understanding of the factors that influence energy efficiency in ML computations. Li, (13) evaluates the energy efficiency of deep convolutional neural networks on CPUs and GPUs, highlighting the disparities in energy usage across different hardware platforms (14).

3. Methodology

To evaluate the energy, performance and temperature estimations of ML algorithms, the approach is categorized into three dimensions, namely i) applications, ii) algorithms, and iii) hardware architectures as shown in Figure 1. Application dimension is categorized further into computer vision, natural language processing, and time-series data which are simulated using six different types of ML algorithms (graph neural network (GNN), spiking neural network(SNN), multilayer perceptron (MLP), transformers, convolutional neural network (CNN) and support vector machine (SVM)) on CPU, GPU and FPGA. Energy estimation methodologies generally fall into either top-down or bottom-up categories. While the bottom-up method relies on a statistical model that integrates workload analysis and power consumption, the former approach zeros in on power dissipation during algorithm simulations.

- **Top-down approach:** Top-down approach estimates energy by executing the algorithm/program on the system which tends to produce accurate results. This approach is limited by the system's scalability and its ability to perform various tasks.
- **Bottom-up approach:** Bottom-up approach is based on a statistical model that integrates workload analysis and power consumption metrics based on an estimated number of instructions and operations. The essential advantage of this approach allows the developer to es-

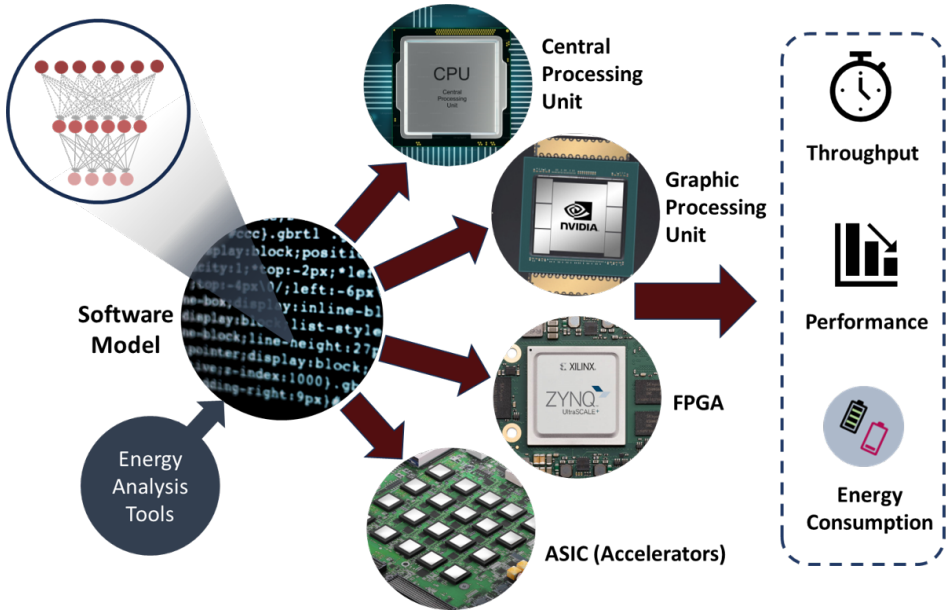


Figure 2: Illustration of the comprehensive methodology employed to evaluate the energy efficiency of machine learning algorithms. This figure delineates the process of simulating a software-based ML model across diverse hardware architectures. The analysis is conducted using a top-down approach, which begins with the examination of the overall system performance and energy consumption, and then delves into the finer details of specific components. This method allows for a comprehensive understanding of how the system operates as a whole, and how its various parts contribute to the total energy usage. The figure highlights key metrics and benchmarks used in the evaluation, demonstrating how different hardware configurations impact the energy consumption of the ML algorithm. This comparative analysis is crucial for identifying energy bottlenecks and optimizing ML deployments for sustainable and energy-conscious computing.

timate the energy usage of a program without executing it (15).

To conduct our evaluation, we utilized two distinct methods. Method 1, the Direct Measurement Method, employs hardware tools like NVIDIA SMI and Intel's Power Gadget for real-time monitoring of GPUs and CPUs, respectively. This method is aligned with the top-down approach, capturing peak power to estimate the system's overall energy during algorithm execution.

Method 2, the Estimation Method, uses the software tool PyJoules, which utilizes Intel's Running Average Power Limit (RAPL) interface for energy estimation. This method, resonating with the bottom-up approach, calculates average power to determine the total energy expenditure, enabling energy consumption estimation based on the system's behavior and instantaneous power.

For FPGAs, power consumption assessment was conducted using proprietary software tools provided by FPGA vendors. These tools offer precise power analysis by simulating the FPGA's power consumption under varied operational conditions, accounting for design specifics, resource utilization, clock frequencies, and I/O activities to generate detailed power reports.

4. Applications

In the following section, we summarize the key aspects of the different applications that were used to estimate energy requirements. The first application is based on Natural Language Processing (NLP), given the recent reported successes of AI platforms such as ChatGPT, Bard, and Llama. The second application is based on Computer Vision (CV), given its wide ranging applications from automated vehicles to medical applications. The third application is based on time-series (TS), as it enables potential linking to domains ranging from particle detectors to analogue sensing. Each of the applications and the datasets used are briefly discussed in the following sections.

4.1. Natural Language Processing

Our study used the NLP Question/Answer dataset, a comprehensive assembly of question-answer pairs designed for the purpose of exploring Question-to-Answer and Answer-to-Question tasks in hardware. This dataset set the scope for assessing the performance and accuracy of natural language processing techniques, with the answers being generated from the text of Wikipedia articles. The dataset is distributed across three files, each pertaining to a particular academic year: S08, S09, and S10. Together, they encompassed a vast textual expanse of 690,000 cleaned

words extracted from Wikipedia, which served as the raw material for the formulation of the questions (16). The organization and breadth of the NLP Question/Answer dataset thus provide us with a suitable test suite for analyzing NLP techniques, particularly in different hardware architectures.

4.2. Computer Vision

In the field of computer vision, CIFAR-10 (17) and MNIST (18) are widely recognized benchmarking datasets in image classification tasks. The first dataset consists of a collection of 60,000 color images, evenly distributed across 10 different classes. The classes encompass a diverse range of objects, including airplanes, automobiles, birds, cats, deer, and horses. Each image in the dataset measures 32x32 pixels, and each color image is represented in the standard Red-Green-Blue (RGB) format. The dataset is split into a training set of 50,000 images and a test set of 10,000 images, each set containing an equal number of images from every class. This balanced distribution ensures the absence of any bias towards a particular class during model training or testing.(19). Compact yet diverse collection of CIFAR-10 images help in comprehensive evaluation and comparison of machine learning algorithms, particularly those that are specifically applicable to image classification tasks. The low resolution and relatively small size of this dataset make it feasible for rapid prototyping, especially in evaluating multiple complex computer vision algorithms.

4.3. Time Series

For time-series, we utilized a dataset derived from X-ray radiography of the laser powder bed fusion process, specifically for aluminum. This dataset, chronologically ordered, captures the dynamic evolution of a fusion process over time. Procured from the SSRL and originating from a beamline (20), this dataset's temporal nature is evident in its structure, with each frame representing a specific timestamp in the fusion process. The implementation focuses on discerning and documenting pivotal characteristics of the data: total number of time-stamped frames, height, and width of each image, and computations related to data size.

5. Machine Learning Algorithms

In this section, we briefly discuss the six different types of machine learning algorithms ("software architectures" that were used to estimate energy costs on different hardware platforms. In addition to the five of them, which are based on neural network (NN) frameworks, the last one (Support Vector Machines) are from a kernel-based formalism. The ML algorithms were chosen based on computational complexity and their relevance to current applications and are depicted in Figure 3. A brief description of these algorithms and their use in our analysis are summarized below.

5.1. Graph Neural Network (GNN)

GNNs are used for a variety of application domains given their unstructured architecture and adaptable topologies. The GNN architecture consists of four graph convolution layers that use a ReLU activation function. Within each application, entities are graphically represented, wherein nodes and edges are contextualized according to the inherent data structure. During inference, entities are subjected to requisite pre-processing and graph conversion, followed by predictions and comparisons. Traditionally, words or sentences are processed with tokenization and embedding in NLP, while images are treated as 2D or 3D tensors in computer vision, and time-series data are analyzed as an array of sequential data points (21; 22; 23).

5.2. Spiking Neural Network (SNN)

SNNs have become very popular given their sparseness (leading to lower memory requirements) and the flexibility to use temporal dimensions to encode information. Typically the SNNs use a particular type of neuron model called Leaky Integrate and Fire(LIF). This LIF model effectively captures the transient nature of neuronal activity, offering a more biologically plausible representation of neural processing compared to traditional neural network models. It embodies the essence of SNNs, where the temporal aspect of information processing is as critical as the spatial one. To train the SNN, event-driven back propagation (eRBP) learning mechanism is used (24; 25). The eRBP algorithm uses an error-modulated synaptic plasticity to learn deep representations, which are shown to be promising in classifying images.

5.3. Multi-layer perceptron (MLP)

A Multi-Layer Perceptron (MLP) is a class of feedforward artificial neural networks, characterized by the presence of multiple layers of nodes, including an input layer, one or more hidden layers, and an output layer. The inclusion of non-linear activation functions, such as the Sigmoid or Rectified Linear Unit (ReLU), within hidden layers enables MLP to model complex, non-linear relationships within data, rendering it suitable for tasks like classification, regression, and pattern recognition. In this context, we evaluate an Extreme Learning Machine (ELM) model, which is a hardware-friendly MLP algorithm for supervised learning tasks (26; 27). ELM is a single-layer feed forward neural network that randomly initializes the input-to-hidden layer weights and analytically calculates the output weights, bypassing the need for an iterative optimization process like backpropagation. This reduces the scope of applications for ELM models, especially for complex problems (28; 29).

5.4. Transformer

Transformers are based on long-range association of words used in languages (30). Their applications have been

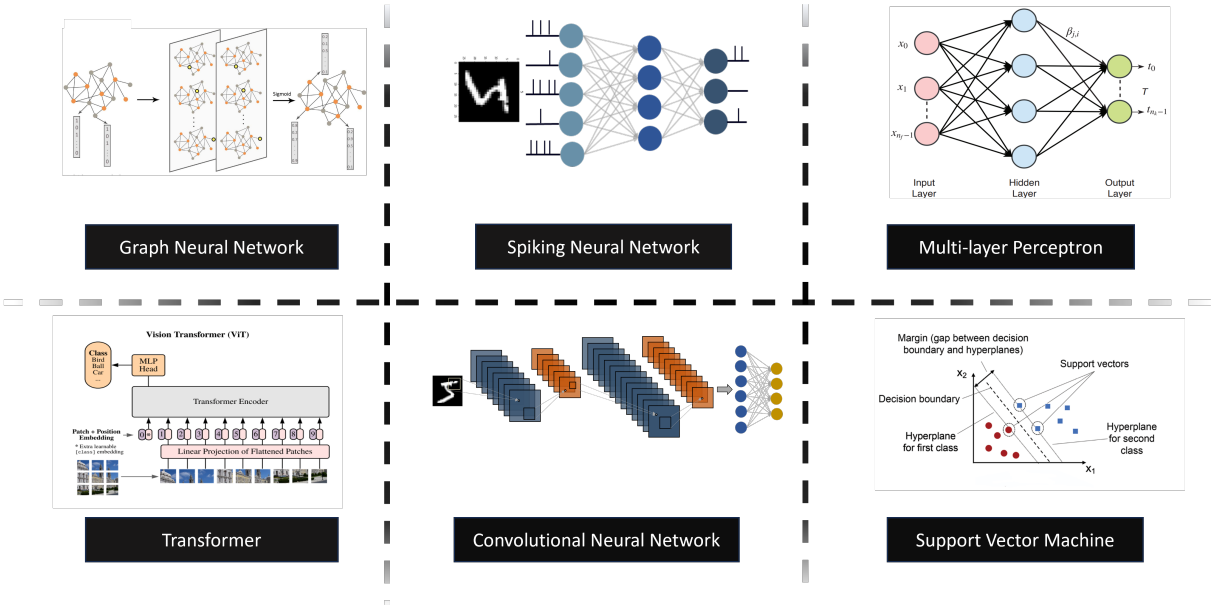


Figure 3: Comparative analysis of six distinct machine learning algorithms to assess their energy consumption when simulated on various hardware platforms. This figure presents a side-by-side comparison of classification, regression, clustering, dimensionality reduction, ensemble methods, and neural networks, highlighting the energy costs associated with each algorithm type. The comparison takes into account the computational complexity, memory requirements, and processing power needed for each algorithm, providing insights into the energy efficiency of different ML approaches when deployed in hardware. This evaluation is critical for understanding the trade-offs between algorithmic performance and energy expenditure, guiding the selection of the most energy-efficient algorithms for practical applications in energy-constrained environments.

demonstrated to be effective in language-to-language translation and to form contextual texts without the need for recurrence and convolution. We apply transformer-based architectures, which have been developed for specific domains: Vision Transformer (ViT) (31) for image processing, DistilBERT (32) for textual analysis, and we use the standard transformer (30) for time-series data interpretation, all developed using a specialized TensorFlow framework. For image processing tasks, the CIFAR-10 dataset undergoes standard pre-processing techniques, including normalization and resizing. The ViT architecture is delineated by a series of layers: a patch extraction mechanism, an embedding projection, multi-head attention modules, and finally a fully connected layer. In particular, the sequence of image patches is analogous to a sequence of tokens in textual data. In the textual analysis domain, we employ the pre-trained DistilBERT architecture, a streamlined variant of the original BERT model, renowned for its computational efficiency without compromising performance significantly. For time-series data analysis, a canonical transformer model is utilized, adept at extracting features from temporal patterns and dependencies.

5.5. Convolutional Neural Network (CNN)

CNNs were first introduced 1990, inspired by neocognitron proposed by (33; 34). CNNs models incorporate convolution and backpropagation and were developed to recognize images efficiently without overfitting. We used ResNet-18 in this study, which is a type of deep CNN architecture. In this model, residual learning introduces “skip connections” in the architecture, where the output of one layer is connected to the input of another layer that is not adjacent to it. The information transfer to non-adjacent layer by bypassing several layers allows amplification of gradients to address the problem of vanishing gradients caused by multiplication of small values. This makes it easier for the network to learn identity mappings (functions that closely resemble the identity function) and capture more complex features.

5.6. Support Vector Machine (SVM)

Support Vector Machine (SVM) algorithm is used for classification and regression tasks, in fields such as image classification, text classification, and informatics (35; 36; 37). The advantage of this algorithm is the intuitive and physical basis for the classification of the data. They work by finding the hyperplane that maximally separates different classes in a dataset. Two parallel hyperplanes are constructed on each side of the main hyperplane, and the algorithms try to find the best-separating hyperplane that maximizes the distance between secondary hyperplanes. The primary objective of SVM is to find the optimal hyperplane that maximally separates the data into different classes. Separation is

determined by a linear kernel function, which computes the dot product between input vectors. The decision function then weighs these computations using Lagrange multipliers, considers the labels of the data points, and adjusts for bias to make the final classification decision.

Table 1: Considerations of six machine learning algorithms for hardware deployment.

Algorithm	Advantages	Drawbacks
GNN	Non-Euclidean data, relationships in graphs	Computationally expensive, preprocessing needed
SNN	Computationally less intensive	asynchronous issues
MLP	Simplicity, broad applicability	No temporal behavior, limited to simple tasks
CNN	Image processing, learns spatial features	For grid-like data, adaptation needed for non-image tasks
SVM	High-dimensional spaces, memory efficient, versatile	Kernel choice, noise sensitive, can be inefficient for complex tasks.
Transformer	Parallelizable, long-range dependencies, powerful for translation	Computationally intensive, memory-hungry

Table 3 presents a comparative analysis of six prevalent machine learning algorithms, delineating their respective advantages and limitations with regard to deployment on hardware platforms. The table serves as a succinct guide for researchers and practitioners in selecting an appropriate algorithm based on the specific requirements and constraints of their computational tasks.

GNNs are lauded for their adeptness in handling non-Euclidean data and elucidating relationships inherent in graph structures. However, they are encumbered by their computational demands and the prerequisite of extensive preprocessing. SNNs are recognized for their computational efficiency, particularly in scenarios that do not necessitate synchronous processing. Nonetheless, they grapple with challenges related to asynchronous computation which can complicate their implementation. MLPs, with their simplicity and wide-ranging applicability, offer a straightforward solution for a multitude of tasks. Despite this, they are impeded by their inability to model temporal dynamics and are generally confined to simpler, non-sequential tasks. CNNs are the cornerstone of image processing tasks, with an innate capacity to learn spatial hierarchies of features. However, their efficacy is predominantly limited to grid-structured data, necessitating significant adaptation for non-image related applications. SVMs excel in high-dimensional spaces and are memory efficient, making them a versatile choice for various applications. Their performance, however, is contingent on the appropriate selection of kernel functions and they exhibit sensitivity to noise. Additionally, SVMs may falter in efficiency when tasked with more complex, large-scale problems. Lastly, Transformers demonstrate exceptional performance in tasks involving long-range dependencies, such as language translation, and are inherently parallelizable. Their prowess, however, comes at the cost of substantial computational and memory resources, which may limit their practicality for resource-constrained environments. the table underscores the trade-offs between the computational efficiency and the applicability of each algorithm. It highlights the necessity for a careful consideration of the specific characteristics of the task at hand, the available computational resources, and the desired outcome when selecting an algorithm for hardware deployment.

6. Hardware Architectures and Benchmarks

This section presents the physical implementations of the architectures, highlighting the different approaches adopted for the CPU, GPU, and FPGA platforms. As the underlying hardware uses different process technologies, we specify the attributes based on published data.

6.1. Central Processing Unit (CPU) and Graphical Processing Unit (GPU) Systems

Our computational experiments leveraged the NVIDIA GeForce RTX 3060/3090 GPU, an integral component of NVIDIA's Ampere architecture, renowned for its computational capabilities tailored for machine learning tasks. In tandem, we utilized the Intel Core i9 12900H and AMD threadripper 5995wx CPU, a product from Intel's Alder Lake series, ensuring optimal performance for tasks not optimized for GPUs. The specifications of the CPU and GPU systems are summarized below.

CPU:

- **AMD threadripper 5995wx**: Crafted on TSMC's 7nm technology node, it operates at 2.7GHz with 64 cores.
- **Intel i9 12900HA**: Based on an 8nm technology node, it runs at 3.7GHz, featuring 14 cores.

GPU:

- **NVIDIA RTX 3090**: Operating on Samsung's 8nm Technology node at 1.7GHz, equipped with 10,496 CUDA cores.
- **NVIDIA RTX 3060**: Also from Samsung's 8nm Technology node, operating at 900 MHz and houses 3,584 CUDA cores.

6.2. Field Programmable Gate Array (FPGA)

Our studies also extended to FPGA implementations. The models were instantiated on the XCVC1902 and Zynq®-7000 FPGA platform. Vivado Design Suite (version 2021.1) is used for the design and synthesis of digital logic. The SNN model simulations were built on dedicated Leaky Integrate-and-Fire (LIF) neuron models from previous work (38). The GNN model's representation on the FPGA involved nodes and edges as modules, interacting through predefined logic. The Transformer model, with its

Model	Topology	Input	Output	Hidden	Others
SNN	784-200-10	784	10	200	-
CNN	128-[100-100-100]-300-10	128	10	[100-100-100]	300 (Dense), 128 (Convolutional), 300 (Sequential)
MLP	784-1000-10	784	10	1000	-
Transformer	3072-64-2048-1024-10	3072	10	-	64 (Attention), 2048, 1024 (Intermediate)
SVM	3072-10	3072	10	-	-
GNN	3072-2048-1024-128-10	3072	10	-	2048 (Successive), 1024 (Successive), 128 (Successive)

Table 2: Detailed Machine Learning Model Topologies

intricate architecture, required multiple encoder and decoder modules, processing input sequences concurrently. For the SVM, specific hardware accelerators were designed to handle high-dimensional vector computations efficiently. The CNN implementation took advantage of the FPGA's parallel processing capabilities, allowing for simultaneous convolution operations across different layers. The MLP, which is a basic neural network model, was slightly optimized (39). The inherent capabilities of the FPGA ensured efficient, real-time simulations of all these models, underscoring potential avenues for scalability. The specification of the FPGA system is summarized below.

FPGA:

- **Zynq®-7000 series FPGA**: Produced by AMD, it incorporates an ARM A9 processor and 13,300 LUTs.
- **AMD VCK190 board**: Featuring the Versal AI Core VC1902 device, it is equipped with 1.2 million LUTs.

Application-Specific Integrated Circuit:

- **Loihi-2**: Intel Labs' Loihi 2, the advanced successor to the original Loihi, represents a significant leap in neuromorphic research technology. Unveiled in 2022, this state-of-the-art research test chip is predicated on an asynchronous SNN architecture. This design facilitates adaptive, self-modifying, and event-driven computations with fine-grained parallelism, thereby enhancing efficiency in both learning and inference tasks. Constructed using Intel's 4 process technology, Loihi 2 incorporates 128 neuromorphic cores, categorizing it as a multi-core Integrated Circuit (IC). A notable innovation in Loihi 2 is its bespoke programmable microcode learning engine, which permits on-chip SNN training. We intend to adhere to the protocols delineated in the documentation for setting up and operating the Lava extension for Loihi (lava-loihi) on the Intel Neuromorphic Research Cloud (vLab) systems (40; 41). Intel's research team, in their quest to leverage the capabilities of the Loihi 2 system, employed standard modules for network configuration within the Lava framework. This study also aims to demystify the functionality of the profiling tool integrated within Lava for Loihi 2. The Loihi-2 chip, fabricated using ASIC technology, utilizes a 7nm process. Intel, as the manufacturer of this chip, has implemented the model using a fixed-precision 32-bit format. Lava, an open-source software library, is dedicated to the development of algorithms tailored for neuromorphic computing. It significantly eases the creation of neuromorphic algorithms by offering an intuitive Python interface, allowing for the assembly of essential components. Lava not only facilitates the testing and execution of these algorithms on conventional von Neumann architectures, such as CPUs, but also ensures their seamless deployment on neuromorphic processors like Intel Loihi 1/2, capitalizing on their speed and energy efficiency. Furthermore, Lava is designed with versatility at its core, accommodating bespoke neuromorphic behavioral implementations and supporting novel hardware backends. The application of Lava is twofold: it enables users to construct intricate algorithms using existing resources without necessitating an in-depth understanding of neuromorphic principles, thereby democratizing access to this advanced computational paradigm (42).

7. Measurements and Analysis

This section delves into the methodologies employed for measuring variables across the three application types, six ML algorithms, and three hardware platforms. We present the results of simulations, scaling of instruction numbers, trends in energy consumption, and temperature profiles for ML algorithms running on CPU, GPU, and FPGA systems. The forthcoming analysis will elucidate the energy dynamics and thermal behavior of these systems under computational stress from various ML workloads.

Both the Direct Measurement and Estimation Methods have their respective advantages and limitations. The former provides precise, direct measurements, while the latter offers flexibility and ease of implementation, particularly in systems lacking specific hardware tools. These methods can also be applied in emulators for scenario analysis. Researchers may select the method that best aligns with their study's needs and constraints. These methods are integral to a tool currently under development and testing, promising to enhance our understanding of energy efficiency in ML applications.

7.1. Model and Computational Complexity

The efficiency of an algorithm on a platform is evaluated by determining the model complexity and computational complexity of an algorithm for a specific size of each of the applications. The model complexity defines the total number of parameters in the algorithm and the computational complexity defines the total number of operations required to perform one single inference where the total number of operations in an algorithm is generalized to total number of multiply and accumulate(MAC) operations which are considered to be the fundamental operations required while executing a machine learning algorithm. The total number of MAC operations for six algorithms is calculated based on the equations mentioned in Table 3. MAC operations in an algorithm depend on the connection topology (the connection network in the case of neural network algorithms) and size of the algorithm since the total number of MAC operations in the fully connected layer is equivalent to the product of the number of input neurons(Nin) and number of output neurons($Nout$). However, CNN uses a kernel matrix that is convoluted over the input neurons to produce the output features. The resulting number of MAC operations is indicated in the table where Kw, Kh are kernel width and height, Cin is the number of input channels, $Hout, Wout$ are the height and width of output feature, and $Cout$ is the total number of output channels. Although the SNNs share a structure similar to that of MLPs and CNNs, the data is fed in the format of spike sequences, which adds an additional term $tsim$ (simulation time) to the computational complexity. The number of MAC operations for SVM is measured primarily on the basis of Nsv (number of support vectors) and the dimensionality of the data. In the Transformer architecture, multiple layers encompass both multi-head self-attention mechanisms and feed-forward networks. Here, d represents the dimensionality of the model, while $sequencelength$ denotes the count of tokens in a given input sequence. The term $d \times sequencelength$ captures the linear transformations applied to both the input embeddings and the output from the multi-head attention. Conversely, the expression $2 \times d \times d$ corresponds to the weight matrices utilized within the multi-head attention mechanism.

Table 3: Number of MAC operations in the neural algorithms.

ML Algorithm	# of MAC Ops
MLP	$Nin \times Nout$
Convolutional NN	$Kw \times Kh \times Cin \times Hout \times Wout \times Cout$
Spiking NN	$Nin \times Nout \times tsim$
Graph NN	$Nin \times Nout$
SVM	Nsv $\times$ Dimensionality
Transformers	$2 \times d \times d + d \times$ sequence length

Figure 4 illustrates the trends in the complexity of the algorithms chosen for comparison. The computational complexity and the model complexity are conditioned by the network topologies which were chosen by normalizing the algorithm performances over an application. Transformers require a larger number of operations to perform a task compared to their other counterparts followed by graph neural networks and SVMs. Though the computational complexity of spiking neural networks is moderate, the model complexity is significantly lower than the other algorithm due to the simulation time parameter which plays a critical role in computing. Moreover, convolutional neural networks show a similar trend as SNNs due to the weight-sharing mechanisms where the kernel is connected to multiple neurons which corresponds to the reduction in model complexity(43).

$$\text{Energy/Op} = \frac{\text{Energy per inference (J)}}{\text{\# of MAC operations}} \quad (1)$$

$$\text{Throughput} = \frac{\text{\# of MACs}}{\text{Latency (Simulation Time)}} \quad (2)$$

$$\text{Simulation Time} = \frac{\text{Total Inference Time}}{\text{\# of inference samples}} \quad (3)$$

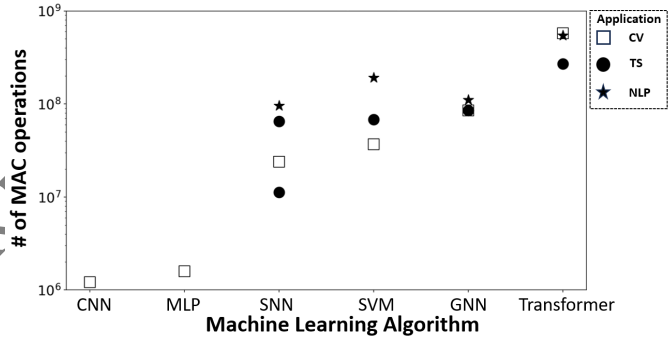


Figure 4: The computational complexity of six ML algorithms is defined by the total number of MAC operations required to perform one single task.

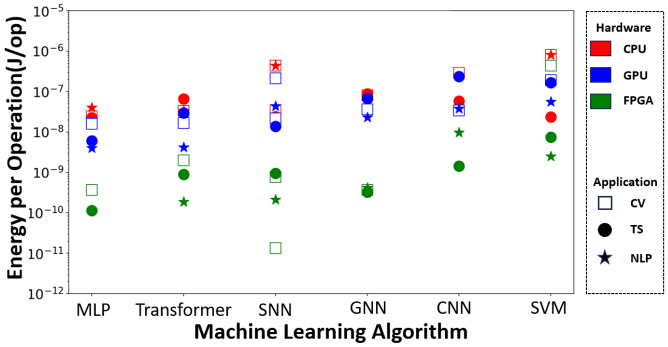


Figure 5: Energy consumption to compute one MAC operation in an ML algorithm, on three different types of hardware.

In our calculation of the Multiply-Accumulate (MAC) count for a neural network, we employed two libraries, as referenced in (44; 45). These libraries make use of the equations provided in Table 3 to determine computational complexity. This methodology enables us to gauge

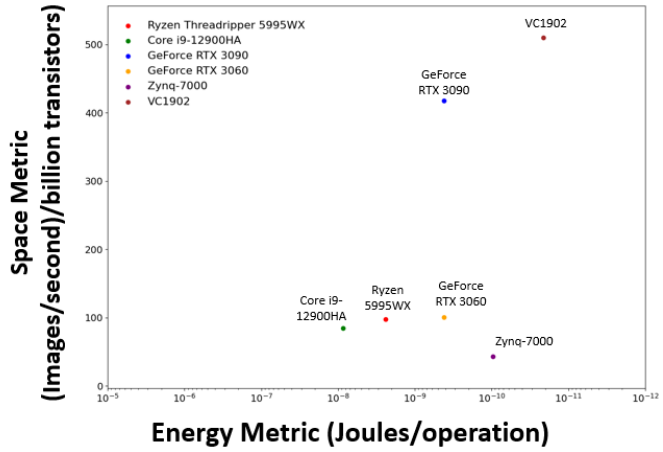


Figure 6: Comparison of energy efficiency, measured as images processed per second per billion transistors, against the energy required for one operation (Joules per operation) across various computing hardware. This plot illustrates the efficiency trade-offs between CPUs, GPUs, and FPGAs in processing machine learning algorithms.

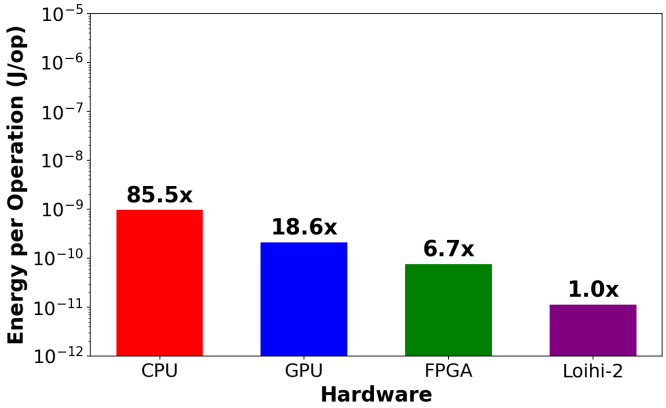


Figure 7: Comparison of energy consumption for computing one MAC operation in a SNN algorithm for Computer Vision application across four different hardware types: CPU, GPU, FPGA, and Loihi-2. The plot includes annotations indicating the relative efficiency of each hardware compared to Loihi-2.

the hardware platform's performance through Equation 2, derived from Equation 3. We have identified metrics such as operations per second and energy per operation as potent criteria for hardware evaluation. In this study, we exclusively employ these two metrics for our evaluation.

7.2. Energy Estimates

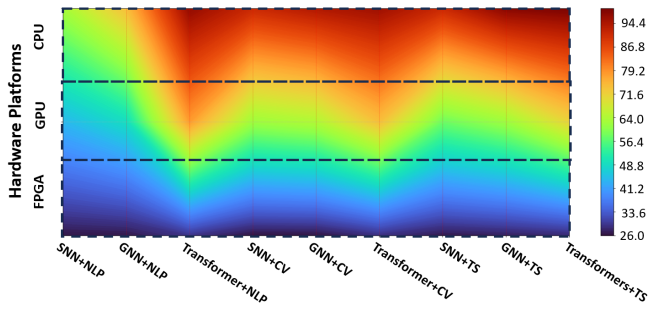
To estimate the energy expended in the algorithms to perform inference operations in different systems, we adopted a top-down approach, as shown in Figure 1. We simulated various ML algorithms across three applications on CPU, GPU, and FPGA platforms. Tools such as Nvidia SMI, Intel Power Gadget, PyJoules, and AMD Vivado were used to evaluate the total energy cost of simulating each algorithm. A crucial metric used for benchmarking efficacy was the energy cost per MAC operation, which determines an algorithm's energy efficiency. This metric is illustrated in Figure 6.

Preliminary findings indicate that of the systems that we analyzed for the specific conditions, computer vision applications use lower energy compared to NLP and time-series applications across all algorithms. SVMs require higher energy given the complexity of the underlying algorithms. Due to their simplicity, ELM algorithms exhibit an edge over other algorithms on CPU and GPU by indicating $5\% - 10\%$ energy per operation compared to other algorithms on CV and time-series applications. SNNs require higher energy on the CPU and GPU since these models operate sequentially. SNNs derive their primary advantages from their event-driven nature and sparsity in the information being processed, eliminating the need for continuous computations. Furthermore, the information in the SNNs is processed as spikes, allowing representation in a binary format that enables hardware architectures to minimize the use of multipliers(38). These optimizations were adapted in the FPGA which resulted in $76\times$ and $23\times$ energy savings compared to simulating the SNN on CPU and GPU, respectively.

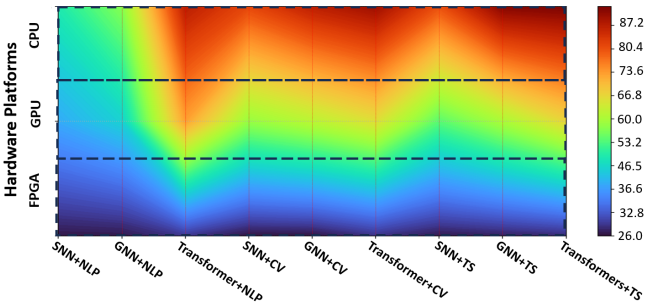
Additionally, in the context of CNNs applied to time series, there is an additional computational demand for non-linear functions per neuron. These non-linear functions, while essential for enhancing the performance of neural network, often come with a significant energy cost. It is imperative to highlight that the energy implications of these non-linear functions aren't encapsulated when solely considering the total number of MAC operations, which explains the higher energy cost per operation for CNN on time-series data. In summary, the findings illustrate that FPGA architectures need lower energies when compared to CPUs and GPUs within the realm of hardware platforms, while also highlighting that transformers require the highest energy among all machine learning algorithms. It is important to note that as the models are scaled to larger problems with complex tasks, these trends may change. Although these findings are preliminary, they offer insights to address these aspects with finer granularity in the future.

7.3. Thermal Analysis

There are trade-offs with high power density and elevated operating temperatures, especially when considering the compact form factor. Heat generated during operation does not dissipate at a rate equal to its generation. This leads to an increase in the junction temperature of the system and, consequently, to a reduction in mean time to failure (MTTF) and also to a higher leakage power (46). Furthermore, the worst-case heating can cause significant problems, ranging from circuit transient timing errors to



(a) Peak Temperature (°C)



(b) Average Temperature (°C)

Figure 8: Peak and average junction temperature of CPU, GPU, and FPGA for three neural networks on three different applications. Red color indicates higher temperature while blue indicates lower temperature.

complete catastrophic burnout, which makes the thermal analysis of the ML algorithms an important criterion.

To assess the thermal implications, we employed Method 1 to gauge temperatures across various algorithm-hardware-application pairings, encompassing NLP, CV, and Time-series applications on both CPU and GPU platforms. Figure 8 visually represents the peak and average junction temperatures of CPUs and GPUs while simulating GNN, SNN, and Transformer algorithms executing three distinct applications. Notably, the results reveal that when performing NLP applications, Transformer algorithms register temperatures approximately 10°C to 20°C higher on both CPUs and GPUs compared to their counterparts, raising concerns for users of large language models (LLMs). While Transformers and other networks involve a relatively similar number of operations, SNNs consistently exhibit considerably lower temperatures. Furthermore, it's worth noting that the temperature increase on CPUs consistently exceeds that on GPUs, regardless of the algorithm or application, by a margin of 19°C to 29°C. This trend of transformers exhibiting higher temperatures is also reflected in their energy consumption, which can be attributed to a higher number of MAC operations required to perform the tasks. The junction temperature and the reported temperature can be understood as approximations, derived from the on-chip sensors. For the purpose of temperature monitoring, data was collected using tools such as *Nvidia-SMI* for NVIDIA GPUs and *Intel Power Gadget* for Intel processors. We used two numerical quadrature methods, Simpson's and trapezoidal methods, when estimating temperatures from the signals. This congruence suggests that the underlying computational techniques of these methods mirror each other closely. The temperature results, encapsulated within the power reports, offer insights into the thermal dynamics of the FPGA during the execution of specific tasks. The temperature readings obtained from the FPGA's on-chip sensors are presented in the vendor software's power report, which meticulously records the junction temperatures during the operation of the device. These readings are instrumental in understanding the thermal characteristics of the FPGA under different computational loads, providing a window into how various machine learning algorithms influence the device's temperature. Incorporating these temperature results into our analysis enhances the depth of our study, allowing us to draw more nuanced conclusions about the energy efficiency and thermal management of FPGAs in the context of machine learning. By benchmarking the thermal performance of FPGAs against that of CPUs and GPUs, we can better understand the trade-offs and advantages inherent in each platform, guiding future design and deployment decisions for energy-conscious machine learning applications.

8. Conclusion

Six widely used machine learning algorithms were systematically simulated on three predominant hardware platforms: CPU, GPU, and FPGA. Complexity of these algorithms on specific hardware based on the specific number of operations, energy based on the total number of operations for a simulation, and the resulting temperature profiles were estimated for three applications: Natural language processing, computer vision, and time series analysis. Our findings indicate that FPGAs are more energy efficient when deploying any of the ML algorithms. This included transformer models for NLP tasks, which are seen to be energy intensive compared to other ML algorithms. As expected, though CPUs are general-purpose and versatile, they consume more energy due to their longer computation times, memory accesses, and data communications. Our thermal analysis illustrates significant temperature variations across different neural network algorithms/architectures, with the Transformer model being notably energy intensive depending on the specific application. From our analysis, it is clear that a comprehensive energy estimation tool for different systems, including custom accelerators and application-specific designs, is needed to quantify the energy and thermal dynamics of popular neural network algorithms. As this is one of the first such studies, we expect to build upon this analysis for more thorough extension to other ML algorithms, different hardware platforms. We think that this will help guide the development of energy-efficient systems combining hardware and software for different applications, as a framework for a sustainable co-design.

9. Acknowledgements

This work was partially supported by the U.S. Department of Energy's Office of Science contract DE-AC02-76SF00515 with SLAC through an Annual Operating Plan agreement WBS 2.1.0.86 from the Office of Energy Efficiency and Renewable Energy's Advanced Manufacturing and Materials Technology Office. We also acknowledge the time-series data provided by SLAC SSRL (Sen Liu, Chriss Tessone, Paul McIntyre). The institutional support from SLAC National Laboratory is also acknowledged.

References

[1] Y. E. Wang, G.-Y. Wei, and D. Brooks, "Benchmarking tpu, gpu, and cpu platforms for deep learning," *arXiv preprint arXiv:1907.10701*, 2019.

[2] S. Shankar and A. Reuther, "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)*. IEEE, 2022, pp. 1–8.

[3] R. Woods, J. McAllister, G. Lightbody, and Y. Yi, *FPGA-based implementation of signal processing systems*. John Wiley & Sons, 2008.

[4] M. Isik, A. Paul, M. L. Varshika, and A. Das, "A design methodology for fault-tolerant computing using astrocyte neural networks," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, 2022, pp. 169–172.

[5] R. Sass and A. G. Schmidt, *Embedded systems design with platform FPGAs: principles and practices*. Morgan Kaufmann, 2010.

[6] A. Boutros, S. Yazdanshenas, and V. Betz, "You cannot improve what you do not measure: Fpga vs. asic efficiency gaps for convolutional neural network inference," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[7] Z. Zong, R. Ge, and Q. Gu, "Marcher: A heterogeneous system supporting energy-aware high performance computing and big data analytics," *Big data research*, vol. 8, pp. 27–38, 2017.

[8] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 83–94, 2000.

[9] D. DeBonis, R. Grant, S. L. Olivier, M. J. Levenhagen, S. M. Kelly, and K. Pedretti, "A power api for the hpc community." Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2014.

[10] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[11] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[12] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, 2019.

[13] D. Li, X. Chen, M. Becchi, and Z. Zong, "Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus," in *2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom)*. IEEE, 2016, pp. 477–484.

[14] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels," in *2019 IEEE international conference on embedded software and systems (ICESS)*. IEEE, 2019, pp. 1–8.

[15] M. Chakib, "Éco-développement: une approche empirique pour réduire la consommation énergétique des logiciels," Ph.D. dissertation, Lille University; Inria Lille-Nord Europe, CRIStAL-Centre de Recherche en ..., 2022.

[16] N. A. Smith, M. Heilman, and R. Hwa, "Question generation as a competitive undergraduate course project," in *Proceedings of the NSF Workshop on the Question Generation Shared Task and Evaluation Challenge*, 2008, pp. 4–6.

[17] A. Krizhevsky, V. Nair, and G. Hinton. (2010) Cifar-10 (canadian institute for advanced research). [Online]. Available: http://www.cs.toronto.edu/kriz/cifar.html

[18] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[19] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[20] S. Gorgannejad, A. A. Martin, J. Wang, J.-B. Forien, M. Strantza, P. Quan, S. Liu, V. Thampy, C. J. Tassone, and N. P. Calta, "Understanding subsurface behavior during metal laser drilling process via in-situ synchrotron x-ray imaging," in *CLEO: Applications and Technology*. Optica Publishing Group, 2023, pp. AM4R–2.

[21] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[22] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[23] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[24] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis,

"Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Frontiers in neuroscience*, vol. 11, p. 324, 2017.
[25] N. Soures, P. Helfer, A. Daram, T. Pandit, and D. Kudithipudi, "Tacos: task agnostic continual learning in spiking neural networks," in *Theory and Foundation of Continual Learning Workshop at ICML'2021*, 2021.
[26] M. Riedmiller and A. Lernen, "Multi layer perceptron," *Machine Learning Lab Special Lecture, University of Freiburg*, pp. 7–24, 2014.
[27] L. Noriega, "Multilayer perceptron tutorial," *School of Computing. Staffordshire University*, vol. 4, no. 5, p. 444, 2005.
[28] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.
[29] J. Wang, S. Lu, S.-H. Wang, and Y.-D. Zhang, "A review on extreme learning machine," *Multimedia Tools and Applications*, vol. 81, no. 29, pp. 41 611–41 660, 2022.
[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[31] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.
[32] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.
[33] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems*, vol. 2, 1989.
[34] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.
[35] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.
[36] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.
[37] H. Elhosary *et al.*, "Low-power hardware implementation of a support vector machine training and classification for neural seizure detection," *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 6, pp. 1324–1337, 2019.
[38] V. Karia, F. T. Zohora, N. Soures, and D. Kudithipudi, "Scolar: A spiking digital accelerator with dual fixed point for continual learning," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022, pp. 1372–1376.
[39] A. Daram, K. Paluru, V. Karia, and D. Kudithipudi, "Scalable ip core for feed forward random networks," in *International Conference on Extreme Learning Machine*. Springer, 2018, pp. 253–262.
[40] "Loihi-2," https://intel-ncl.atlassian.net/wiki/spaces/NAP/pages/1785856001/Get+Started+with+Intel+Loihi+2, accessed: 2023-07-26.
[41] "Loihi-2," https://en.wikichip.org/wiki/intel/loihi_2, accessed: 2023-07-26.
[42] "Lava," https://lava-nc.org/, accessed: 2023-08-2.
[43] H. F. Langroudi, V. Karia, T. Pandit, and D. Kudithipudi, "Tent: Efficient quantization of neural networks on the tiny edge with tapered fixed point," *arXiv preprint arXiv:2104.02233*, 2021.
[44] "PyTorch-OpCounter," https://pypi.org/project/thop/, accessed: 2023-08-26.
[45] "TensorFlow-Profiler," https://tensorflow.org/guide/profiler, accessed: 2023-08-26.
[46] M. Pedram and S. Nazarian, "Thermal modeling, analysis, and management in vlsi circuits: Principles and methods," *Proceedings of the IEEE*, vol. 94, no. 8, pp. 1487–1501, 2006.

Supplementary Material: Hardware Configuration

Supplementary Material: Precision Comparison

Supplementary Material: Machine Learning Algorithms Details

9.1. GNN

The aggregation and update functions are delineated in Equation 4 and 5. In these equations, h_v denotes the feature vector of node v, $N(v)$ signifies the neighbors of node v, W represents the weight matrix, a_v is the aggregated information derived from the neighbors of node v, and h'_v is the updated feature vector of node v post-aggregation. The aggregation function gathers information from a node's neighbors, typically using a weighted sum of their feature vectors. The update function then refines the node's feature representation by combining its current features with the aggregated information, often passed through a non-linear activation function.

$$a_v = \sum_{u \in N(v)} W \cdot h_u \tag{4}$$

$$h'_v = \text{ReLU}(a_v + h_v) \tag{5}$$

9.2. SNN

The SNN architecture consists of two layers with hidden layer size of 200 neurons modeled with leaky integrate and fire units. The input images are rate encoded in a series

of spike trains which are propagated to compute output spikes. During training, the error current is calculated by computing the difference between the rate encoded label spikes and output spikes. This current and its inverse are sent to two error encoding neurons which calculate the false positive and false negative error spikes for the output neurons respectively. The errors for the output layer weights are computed directly using the error spikes, whereas the error spikes are propagated with fixed random weights to the hidden layer neurons.

$$I(t+1) = I(t) + \frac{\Delta t}{\tau_{syn}} \left(\sum_{j=1}^{N} w_j S_j(t) - I(t) \right) \quad (6)$$

$$V(t+1) = V(t) + \frac{\Delta t}{\tau_{mem}} (V_{rest} - V(t) + I(t)R) \quad (7)$$

9.3. MLP

The ELM architecture in this study consists of 784 input neurons, 2000 hidden layer neurons, and 10 output neurons to classify MNIST images. Weight (W) is estimated as shown in Equation 8 and 9 where H is the output of the hidden layer, Y is the output activation, $\hat{Y}$ are the output labels and α is the learning rate.

$$\Delta W = \alpha * (Relu(H) * error) \quad (8)$$

$$error = (Y - \hat{Y}) \quad (9)$$

9.4. Transformer

The attention mechanism within the Transformer architecture discerns the significance of various segments of the input data. This is achieved by initially converting the input, X, into matrices representing query, key, and value through the application of weight matrices, denoted as W_q, W_k, and W_v. Subsequently, "attention scores" are derived by computing the dot product of the query and key matrices, followed by a scaling operation using $\sqrt{d_k}$. These scores highlight the degree of emphasis that each segment of the input should be given. The culmination of this process is an output that represents a weighted amalgamation of the value matrix, contingent on the aforementioned attention scores.

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \quad (10)$$

$$Q = W_q \cdot X, \quad K = W_k \cdot X, \quad V = W_v \cdot X \quad (11)$$

9.5. SVM

The linear kernel function and the decision function for SVM with a linear kernel are shown in Equations 12 and 13, respectively. In these equations, $\mathbf{x}$ and $\mathbf{x}'$ are input vectors, N represents the number of support vectors, α_i denotes the Lagrange multipliers, y_i signifies the labels (either -1 or 1 for binary classification), x_i are the support vectors, and b is the bias term. However, the application of SVM is often limited by the computational requirements of the algorithm, particularly when working with large datasets.

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}' \quad (12)$$

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x_i} \cdot \mathbf{x} + b \quad (13)$$

Table 4: Specifications of the Hardware Devices Used in the Study

Specification	CPU		GPU		FPGA		ASIC
	AMD	**Intel**	**NVIDIA**	**NVIDIA**	**AMD**	**AMD**	**Intel**
Model	Ryzen Threadripper 5995WX	Core i9-12900HA	GeForce RTX 3090	GeForce RTX 3060	Zynq®-7000 series	VC1902 AI Core series	Loihi 2
Technology Node	7nm	10nm	8nm	8nm	28nm	7nm	7nm
Core Count	64	14	10,496	3,584	Single/Dual-core	-	128
Precision	Float32	Float32	Float32	Float32	Fixed32	Fixed32	Fixed32
Operation Frequency	2.7GHz	3.7GHz	1.7GHz	900MHz	100MHz	100MHz	Variable
Memory Technology	DDR4	DDR5	GDDR6X	GDDR6	DDR3	DDR4	On-chip
Platform Years	2022	2021	2021	2021	2018	2022	2022
Maximum Power	280W	157W	365W	170W	?	180W	Low Power

Table 5: Comparison of energy consumption for computing same precision (Float-32) in a GNN algorithm for Computer Vision application across four different hardware types: CPU, GPU, FPGA.

Type	Precision	MACs (OP)	Simulation Time (s)	Power (Watt)	Throughput (OP/s)	Energy Efficiency (Joule/Ops)
CPU	Float32	8.50E+07	0.151	46	5.63E+08	8.17E-08
GPU	Float32	8.50E+07	0.061	51	1.39E+09	3.66E-08
FPGA	Float32	8.50E+07	0.0225	2	3.78E+09	2.56E-10
FPGA	Fixed32	8.50E+07	0.0156	2	5.45E+09	3.67E-10