Evaluating Performance of Spintronics-Based Spiking Neural Network Chips using Parallel Discrete Event Simulation

Elkin Cruz-Camacho
Rensselaer Polytechnic Institute
Troy, New York, USA
cruzce@rpi.edu

Siyuan Qian
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
siyuanq3@illinois.edu

Ankit Shukla
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
ankits4@illinois.edu

Neil McGlohon
Rensselaer Polytechnic Institute
Troy, New York, USA
mcglon2@rpi.edu

Shaloo Rakheja
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
rakheja@illinois.edu

Christopher D. Carothers
Rensselaer Polytechnic Institute
Troy, New York, USA
chrisc@cs.rpi.edu

ABSTRACT

It has long been said that neuromorphic computing will yield enormous energy improvements on machine learning based computations and will be part of the next computing revolution. Yet, how likely is it that these goals are met once hardware-level constraints have been accounted for? In this paper, we benchmark the performance of a spintronics hardware platform designed for handling neuromorphic tasks. Spintronics devices that use the spin of electrons as the information state variable have the potential to emulate neuro-synaptic dynamics in hardware. Unlike their CMOS counterparts, spintronics-based neurons and synapses can be realized within a compact form-factor, while operating at ultra-low energy-delay point.

To explore the benefits of spintronics-based hardware on realistic neuromorphic workloads, we developed a Parallel Discrete-Event Simulation model called Doryta, which is further integrated with a materials-to-systems benchmarking framework. The benchmarking framework allows us to obtain quantitative metrics on the throughput and energy of spintronics-based neuromorphic computing and compare these against standard CMOS-based approaches. Although spintronics hardware offers significant energy and latency advantages, we find that for larger neuromorphic circuits, the performance is evidently limited by the interconnection networks rather than the spintronics-based neurons and synapses. Thus, it becomes imperative to identify interconnect materials that would natively offer low latency and consume less energy than the current copper-based interconnects.

Through Doryta we are also able to show the power of neuromorphic computing by simulating Conway's Game of Life. We show that Doryta obtains over 400× speedup using 1,280 CPU cores when tested on a convolutional, sparse, neural architecture.

SIGSIM-PADS '22, June 8–10, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9261-7/22/06…$15.00
https://doi.org/10.1145/3518997.3531025

CCS CONCEPTS

• **Hardware** → **Integrated circuits**; **Spintronics and magnetic technologies**; **Neural systems**; • **Theory of computation** → *Computability*.

KEYWORDS

spiking neural networks, spintronic devices, chip performance, energy estimation, parallel discrete event simulation, game of life, turing completness

ACM Reference Format:
Elkin Cruz-Camacho, Siyuan Qian, Ankit Shukla, Neil McGlohon, Shaloo Rakheja, and Christopher D. Carothers. 2022. Evaluating Performance of Spintronics-Based Spiking Neural Network Chips using Parallel Discrete Event Simulation. In *SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '22), June 8–10, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3518997.3531025

1 INTRODUCTION

Neuromorphic computing is a non-von Neumann approach to computing that creates a human brain-like computational model to perform machine learning, among several other tasks, in a highly energy-efficient manner. For example, as part of the DARPA SyNASPE program, IBM created an instance of a spiking neuromorphic processor, called TrueNorth. This chip has 4096 neurosynaptic cores with a total of 1 million spiking neurons and 256 million re-configurable synapses and consumes only 63 milliwatts when executing a multi-object detection and classification program using real-time video input [31]. In 2018, Intel created the *Loihi* spiking neuromorphic processor capable of performing on-chip learning [14]. According to Hasler and Marr [18], biological neurons and synapses, if realized truly efficiently in silicon, would be able to compute 10^{18} multiply-accumulate operations (MAC) per second using only 1 watt of power. To that end, it is important to continue improving the state-of-the-art and further reduce the hardware costs associated with neuromorphic computing.

To seek out improvements at the hardware level, one can make use of spintronics devices utilizing magnetic materials, including antiferromagnets and ferromagnets, for the fabrication of electronic neurons and synapses in a brain-inspired architecture. Spintronics devices made using magnetic materials are non-volatile and can mimic the dynamics of biological neurons and synapses in hardware in an energy-efficient and compact form-factor, potentially opening

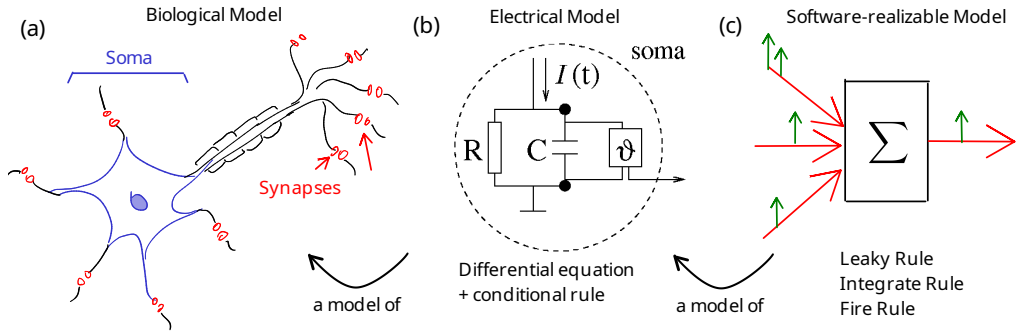

Figure 1: Source of inspiration for Spiking Neural Networks and abstraction levels. (a) The biological spiking neuron. (b) A model (Leaky-Integrate Fire) of the biological spiking neuron [16, Fig. 4.1]. (c) Software-realizable neuron model: neuron (soma) and synapses.

a new artificial intelligence (AI) paradigm endowed with real-time learning, adaptation, and prediction. Such brain-inspired AI hardware can be used in remote, "edge computing" environments with size, weight, and power constraints.

To facilitate our evaluation of spintronics-based neuromorphic hardware, we have developed a multi-scale modeling and simulation approach where physical hardware costs (i.e., energy, area, and latency) are calculated for key neuromorphic operations including neuron integration, neuron fire and signal communication. These performance models are then imported into a new neuromorphic parallel simulation model called Doryta. Doryta enables the energy performance exploration of these devices for neuromorphic applications across a full-scale neural network architecture model.

The key contributions of this work include:

(1) Development of Doryta, a deterministic, parallel spiking neural network simulation platform that is able to execute real neuromorphic applications in simulation, validated against existing spiking neural network tools.
(2) Implementation of a Game of Life as a pure neuromorphic application model for Doryta, demonstrating that spiking neural networks are Turing-complete.
(3) Evaluation of parallel Doryta simulation performance wherein the Game of Life neuromorphic application model is able to obtain over 400× speedup using 1280 CPU cores.
(4) Quantification of the energy, chip area, and runtime performance of spintronics-based neurons and synapses for image classification tasks resulting in three to six orders of magnitude improvement in energy-delay product over CMOS designs.

2 BACKGROUND

In this section, we briefly introduce the component parts of spiking neural networks and summarize parallel discrete event simulation.

2.1 Spiking Neural Networks

Ubiquitous Artificial Neural Networks (ANNs) are not the only relevant bio-inspired development to come out of studying the brain. Spiking Neural Networks (SNNs) [4], as the name implies, are based on the same foundation as ANNs, the biological spiking neurons.

The biological spiking neuron is one of the prevalent neuron types in the brain. It responds to stimuli in the form of *spikes*. The simplest electrical models of the spiking neuron corresponds to the Leaky-Integrate Fire (LIF) Neuron [4], see Figure 1, which can be described by only two procedures: a differential equation that determines how the voltage of the neuron changes as a function of time

$$C\frac{dV(t)}{dt} = I(t) - \frac{V(t) - V_e}{R}; \tag{1}$$

and a conditional rule that dictates how the neuron discharges

$$\texttt{if } V(t) > V_{\text{th}} \texttt{ then set } V(t) = V_{\text{reset}} \texttt{ and fire} \tag{2}$$

where $V(t)$ is the voltage of the neuron at time t, $I(t)$ is the input current to the neuron, C is the capacitance, R is the resistance, and V_e, V_{reset} and V_{th} are the resting, reset and threshold potentials, accordingly.

Notice that there is no general analytical solution for the two rules, leaving us to seek a numerical solution. To numerically simulate Equation 1, we discretize it into Δt time steps

$$V(t + \Delta t) = V(t) + \Delta t\frac{-(V(t) - V_e) + I(t)R}{\tau}, \tag{3}$$

where $\tau \equiv RC$.

The neurons in a neural network are interconnected via axons and synapses. We use the term *synapse* to refer to the connection between two neurons at the software level: from a neuron that fires and sends a spike to a neuron that receives it. At the hardware level, neurons are connected via physical wires or *interconnects*, while the synapse is a non-volatile memory to weight the connection between the neurons. All synapses have two attributes: how much current arrives to the neuron and a delay. For simplicity, we assume that all synapses have the same delay, one clock cycle. This implies that $I(t)$, in Equation 3, represents the summation of all currents from the synapses that receive a spike at time t, i.e.

$$I(t) = \sum_i W_i S_i(t), \tag{4}$$

where W_i is the weight for the i-th synapse, and $S_i(t)$ is either 1 or 0 indicating whether a spike was received or not to the i-th synapse at time t, respectively.

With neurons and synapses, we can construct many complex structures, also called neural-network (NN) architectures or simply architectures. In fact, we can construct the same architectures created for ANNs. Although, it is not trivial to transform an arbitrary ANN and its weights into an SNN with its parameters, an NN architecture can be implemented in either. A full SNN model is then composed of an architecture and three procedures (or operations): the leak operation (Equation 3), the integrate operation (Equation 4) and

the fire operation (Equation 2). At the hardware level, we realize these three operations using spintronics-based spiking neurons and synapses, and metallic interconnects.

2.2 Spintronics Neurons and Synapses

The field of spintronics is expected to support semiconductor-based microelectronics in 'Beyond Moore's' information technologies [17, 29]. In contrast to the conventional electronics that deals with the charge of an electron, spintronics utilizes the spin of an electron to manipulate, transmit, store and detect information. Spintronic devices can be fabricated using back-end-of-the-line CMOS processes and, therefore, realized in modern fabrication facilities without much re-tooling. At the heart of a spintronic device is a magnetic material: ferromagnetic (FM) or antiferromagnetic (AFM), which acts as the active component and displays neuro-synaptic dynamics when perturbed by external input (e.g., current pulse or magnetic fields). The spin configurations of FM and AFM materials are distinct which makes them functionally unique as the building blocks of neuromorphic hardware.

FM-based spintronics nano-devices, such as magnetic tunnel junctions, are commonly used for storage, sensing, logic, interconnections, and as non-linear radio-frequency oscillators [19]. Recently, it was experimentally demonstrated that the FM-based nonlinear oscillators could be used to build circuits that embed neural functionality and can perform speech and digit recognition with high accuracy [43]. FM devices have also been used for realizing synaptic behavior in hardware [6]. However, it is challenging to implement energy-efficient and ultra-fast spiking neurons with FM devices since the underlying physics limits their spiking rate to a few GHz [24]. Unlike FM materials, AFMs have intrinsic frequencies in the THz regime [24]; therefore, in an NN architecture, it is advantageous to use them as hardware emulators of spiking neurons. Meanwhile, ferromagnetic materials are useful to implement the synaptic behavior as they can efficiently store information in a non-volatile manner in their magnetic textures.

2.2.1 Antiferromagnetic Spiking Neurons. Antiferromagnets, such as NiO, Cr_2O_3, alloys of Mn (e.g., Mn_3Ir), etc., are a class of magnetic materials that are internally magnetic on a microscopic scale but possess negligible net magnetization on a macroscopic scale, owing to their atomic arrangement. In principle, they can be used to realize nonlinear signal generators and detectors operating in the GHz to THz frequency spectrum [24]. Such AFM-based signal generators have been theoretically shown to emulate spiking neurons in hardware within a compact form-factor [25, 40], as shown in Figure 2. Their energy efficiency and ultra-fast spiking characteristics will enable significant network-level performance benefits when compared with their CMOS counterparts. In order to excite the spiking response in an AFM, an input current surpassing a threshold is provided to the AFM. Based on the material parameters and the dimensions, the AFM neuron's spike rate and performance can be efficiently tuned [25, 40].

2.2.2 Ferromagnetic Non-volatile Synapses. Memristive dynamics based on domain wall (DW) movement are easily excited in FM structures with a stripe shape as shown in Figure 3a. Here, an input signal, such as current, changes the resistance of the device in an analog manner as shown in Figure 3b, while the synapse also shows plasticity. Thus, FM materials can act as hardware emulators of synapses and store real-valued weights in a non-volatile manner. During the training phase, the input data in the form of current flows between the terminals T2 and T3 of the synapse shown in Figure 3a. The synapse's conductance is set by the magnitude and the duration

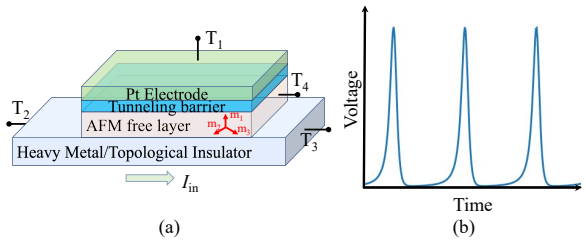

Figure 2: (a) Schematic of a spintronic AFM neuron where the tunneling barrier is Magnesium Oxide (MgO) whereas m_1, m_2, m_3 represent characteristic parameters of the metallic AFM considered here. Input current is applied between terminals T_2 and T_3, while the output voltage is measured across terminals T_1 and T_4. (b) A representative response of the AFM spiking neuron over time due to applied constant input current (I_{in} shown in (a)).

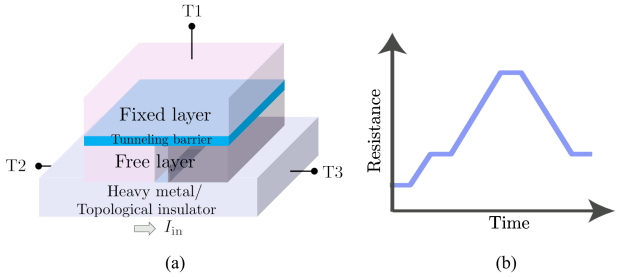

Figure 3: (a) Schematic of a spintronic synapse where the free layer and the fixed layer are made of ferromagnetic materials. Input current is applied across terminals T2 and T3, while the output voltage is measured across T1 and T3. (b) A representative response of the ferromagnetic synapse over time due to applied input current.

of I_{in}. During inferencing, the output current between T1 and T3 terminals is sensed. The output current is given as the product of the reading voltage applied across T1 and T3 and the memristor's conductance, which is set during the training phase. The memristors can be electrically connected in an analog cross-bar architecture such that the net current flowing through the bit line is given as the weighted sum of the memristors' conductance multiplied by the input voltage (corresponding to the data being inferenced).

2.3 Parallel Discrete Event Simulation

Parallel Discrete Event Simulation (PDES) is an efficient method for modeling the behavior of complex systems with discrete interactions between many simulation entities and is a natural match for the independent, discrete behaviors in spiking neural networks. Now, a PDES simulation is made up of agents or entities known as Logical Processes (LPs). These LPs are each mapped to the various cores or Processing Elements (PEs) that may exist. In our simulation environment, a single core or MPI process is home of one PE. Given a constant simulation size with some number of LPs, the more PEs we have, then, the fewer LPs will exist within each PE. Thus, the responsibility for simulating the behavior of all LPs in the simulation becomes more distributed as we increase the number of PEs.

Effective parallelization does not come for free. When distributing computation across multiple PEs, there needs to be some mechanism in place for synchronization, ensuring that correct event ordering is always maintained. In general, there are two approaches for addressing this synchronization problem. *Conservative approaches* [5, 11] ensure that events are processed in timestamp order by waiting until it is "safe" to process each event (i.e., no smaller timestamped events will be received later). *Optimistic approaches* [23] allow events to be processed out of order, but provide a mechanism to detect and erase incorrect event computations. Loosely, each LP have their own clock. The Global Virtual Time (GVT) is the minimum clock time across LP, anything occurred before is considered to "have happened" and any redundant information from that time can be forgotten.

For simulation models developed and used in this work, we leverage the Rensselaer Optimistic Simulation System (ROSS). ROSS is a framework for developing parallel discrete event simulations. ROSS has demonstrated highly scalable, massively parallel event processing capability for both conservative and optimistic synchronization approaches [1, 2, 7, 28, 30, 32]. ROSS' conservative execution is inspired from the YAWNS protocol [33], utilizing a event creation lookahead window restriction that ensures events cannot be created in a way that causes out-of-order processing. ROSS' optimistic execution is accomplished by implementing the Time Warp protocol [20–22] which works with virtual time [23] for event time management. ROSS mitigates Time Warp state-saving overheads via *reverse computation* [8]. In this approach, rollback is realized by performing the inverse of the individual operations that were executed in the event computation. This eliminates the need to explicitly store prior LP state, leading to much more efficient memory utilization.

Another useful feature of ROSS is its ability to deterministically break ties between events that occur simultaneously [30]. This feature orders independent simultaneous events in an unbiased, arbitrary —but reproducible— order. We build on this feature in this work to incorporate *user-defined priorities* to allow for certain events to always be processed before others in the case of event simultaneity. This is discussed in greater detail in Section 3.1.

3 DORYTA: SIMULATING SPIKING NEURAL NETWORKS

Doryta [1] is an architecture-agnostic and deterministic Spiking Neural Network simulator. Doryta is written in C as a ROSS model, and thus can run in virtually any system with a C compiler and a compatible MPI library. Doryta's job is not to train a network for a task, i.e., no neuron parameter is altered in the simulation of the network, instead it is intended to be a reliable, deterministic and time-aware simulator of SNNs.

Doryta differs from its inspirational predecessor, NeMo [37], in several regards. First, NeMo was written in a combination of C, C++ and Lua which increases the complexity of development. Second, NeMo was originally designed to simulate the TrueNorth [9] chip, and generalizing it to any arbitrary architecture proved a difficult task. In contrast, Doryta is written to be architecture agnostic, and therefore more flexible and malleable to a larger array of SNN architectures, such as recursive neuron connections. Lastly, loading trained SNN models into NeMo involved a significant number of steps and additional code per model, whereas Doryta can load models directly from disk given the correct binary format.

Doryta's development is guided by several principles: modularity, reproducibility, determinism, minimal use of third party libraries, and the mantra "the LIF neuron is king". Although modularity has its shortcomings —it tends to produce less efficient code than a monolithic implementation and a high degree of understanding is required to glue all pieces together— we vouch for modularity to be center stage as it has allowed us to extend Doryta's capabilities with minimal refactoring. Modularity also helped us detect bugs by making it easier to unit-test the code in granular detail.

Built into Doryta is a set of tests which allow the developer to check for the unintended injection of bugs or to verify that Doryta compiles and runs as expected in a new machine or architecture. Furthermore, because Doryta, by leveraging the ROSS PDES engine, is deterministic, tests can quickly determine if what was intended to be a minor change resulted in different the simulation output.

We hope that Doryta serves as a platform for other researchers to play and experiment with. This is only possible if Doryta can be compiled and executed without difficulty. Thus, we stand by our decision to restrict Doryta's development to the use of the minimum number of C capabilities and libraries.

We prefer to lean on the LIF neuron model for its combination of simplicity and capability. Although there are dozens of models for the biological neurons and dozens more models implemented in libraries, many of these neuron models, however complex, can be boiled down to behavior similar to that of the LIF neuron, i.e., any neuron can be modelled by the application of three simple rules: leak rule, integrate rule and fire rule. We demonstrate the capabilities and computational power of the LIF neuron model (and its Turing completeness) in Section 4.1.

3.1 Implementation Details

Doryta is implemented as a ROSS model divided up into multiple modules: driver LPs (neuron LP), layouts, model-loaders, and neuron types. Here, we explain in detail each of the modules and related concepts such as "Doryta modes", event types and Δt-step.

3.1.1 LPs and Events. Each LP in Doryta represents a single neuron. Each neuron (LP) is composed of three variables: an ID, a list of synapses (weighted connections to neurons), and, by default, a pointer to the LIF neuron parameters (potential, capacitance, current, resting and reset potential, and threshold). Further neuron models can be implemented as C `structs` and be referenced by the LP instead of LIF neurons.

In a continuous simulation, the set of rules that govern the behaviour of the system are applied every delta-time step (Δt-step) of the simulation. This Δt-step is model dependent and determines the granularity of the simulation. The smaller the Δt-step, the more precise the simulation, at the cost of a larger computational time. Given the nature of PDES, this Δt-step in Doryta has to be explicitly encoded as an event. We call this event a *heartbeat event*.

In the LIF neuron model there are three rules: leak, integrate and fire. Notice that on one hand, in a continuous simulation, all three rules should be executed every Δt-step, but this is not the case in DES where "integration" only occurs when a spike arrives to a neuron and in no other case. On the other hand, leak and fire occur only once even when more than one spike arrives at the neuron within two heartbeats. These two cases are executed by two distinct events:

- **Heartbeat event**: It applies the rules: leak and fire, in that order. It is scheduled every Δt-step. It is received, processed

[1] Doryta is an amalgam composed of the name "Dory" and the Spanish-origin suffix "ita". Dory comes from the movie "Finding Nemo" where a forgetful regal blue tang, called Dory, finds herself accompanying a clownfish in an adventure to find his son, Nemo. Doryta is the spiritual successor to NeMo, another SNN simulator built on ROSS. Doryta, however, is simpler, more modular and flexible — which, we believe, is the essence of Dory the fish.

and sent by each neuron. There is at most one heartbeat event per neuron at any given point.
- **Spike event**: It applies the integration rule (aka, summation) to the neuron, i.e., for the LIF neuron, the spike's current is added up to the input current $I(t)$ at time t. It can be scheduled at any point in time. Spike events can be created in two ways: loaded when the simulation initializes or created by a neuron. There can be as many spikes per neuron as synapses it connects to.

Under optimistic execution, a PE might need to rollback some events in order to keep in sync with all the other PEs. To allow rollbacks, we save the neuron's state before modifying it.

Due to the possibility that two or more events could occur with the same timestamp (an event tie), we lean on the tie-breaking feature of ROSS [30] to maintain simulation determinism and validity. This, however, will also guarantee that the order of any two independent, simultaneous events is explicitly randomized. In most cases for Doryta, this is acceptable; however, we would like for heartbeat events to always be processed before spike events, should they occur at the same time. This use case is an example of *user-defined event priorities*.

To accomplish this, we extended the tie-breaking data structure of ROSS which enforces lexicographic ordering of events based on the values of the items within the structure. Before, ROSS would compare two events based on their regular timestamp; if there was a tie, then it would compare the two based off of their uniquely generated, i.i.d uniform random tie-breaking values.

To implement the user-defined priorities needed for our simulation models, we insert into the structure another value after the regular timestamp but before the random tie-breaker value. This value is defined at event creation depending on the event type; heartbeat events get a higher priority value, spike events get a lower priority value. This allows for the order of simultaneous Doryta events to remain in an unbiased arbitrary order except in the case of comparing a heartbeat and a spike event. In that case, then the heartbeat event will always be processed first.

3.1.2 Modes. Unless a neuron receives a spike it will not fire. This is true only when we can guarantee that the neuron does not have positive leak, which would provoke neurons to fire with no input spikes. Therefore, without positive leak, there is no reason to schedule a heartbeat event for every single Δt-step. We can leverage this fact and run simulations in one of two modes:

- **Needy mode:** A heartbeat event is scheduled every Δt-step regardless of neuron input, creating potentially needless events.
- **Spike-driven mode:** A heartbeat event is scheduled only after a spike is received, preventing the creation of heartbeat events when they aren't needed.

To make simulations on both modes semantically the same, two constraints should always be maintained: heartbeats must always be scheduled at the same timestamps whether they are scheduled by another heartbeat or triggered by a spike, and applying the leak operation consecutively for n Δt-steps must yield (approximately) the same as computing the analytical solution of the leak operation over that same time interval (`big-leak` operation).

As an example, let us assume a Δt-step of 1.0. In needy mode, the first heartbeat would be allocated to time 0.0, and once it is processed, it would allocate the next heartbeat to time 1.0, then to 2.0, and so on. In spike-driven mode, no heartbeat would be allocated to time 0.0 and instead if a spike arrives at time t then a big-leak operation would update the state of the neuron followed by the allocation of a heartbeat for time $\lfloor t \rfloor + 1$. No matter when a spike is received

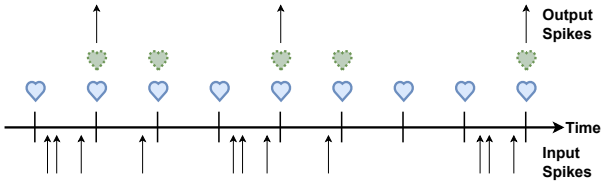

Figure 4: Heartbeat scheduling for needy (solid blue hearts) compared to spike-driven (dotted green hearts) modes.

or for how long the simulation runs, we can ensure that with a Δt-step of 1.0 both simulations will produce the same output spikes. That being said for other Δt-step values like 0.1, although entirely reasonable, one cannot guarantee the same output spikes due to floating-point arithmetic limitations. The important takeaway here is that applying the leak operation iteratively must produce the same result as applying the big-leak operation once and to do that, careful choice of floating-point Δt-steps is crucial, see Figure 4.

3.1.3 Delta-time step and Leak operation. In needy mode, the next heartbeat event is scheduled a Δt-step from whenever the previous heartbeat event is processed. In spike-driven mode, heartbeat events are not scheduled by heartbeat events but by spike events. When a spike arrives, it schedules a heartbeat event (unless already scheduled) for the time:

$$t_{\text{heartbeat}} = \left(\left\lfloor \frac{t_{\text{spike}}}{\Delta t} \right\rfloor + 1 \right) \times \Delta t \tag{5}$$

This equation is sufficient for replicating the behavior of needy mode simulations in spike-driven mode; however, the nature of floating-point number arithmetic could shift the heartbeat timestamps slightly between both modes. We recommend using Δt-steps equal to a power of 2 as that is more resistant to floating-point precision errors.

The leak operation (see Equation 3) is a discretization of the differential equation for the neuron behaviour (see Equation 1). As noted before, when no spikes arrive to the neuron, there is no need to update the state of the neuron. It is possible to determine the state of the neuron if the input current ($I(t)$) is zero (or constant) over any period of time, which means that we can compute the new state at once instead of needlessly applying the leak rule many times. This new computation is called the big-leak operation and it is equal to:

$$V(t) = V_e + e^{\frac{-t}{\tau}} \times (V_i - V_e) \tag{6}$$

where t is the time elapsed since the state of the neuron was last computed.

All of Doryta's capabilities described until now are encapsulated under two modules: the neuron driver module and the neuron model module. The driver module can be configured through a single function that accepts one argument, a pointer to a `struct`. Through this `struct`, each PE loads the parameters for all neurons and their input spikes. The model module contains the leak, integrate and fire operations for all neuron models.

Lastly, Doryta contains two additional modules: layouts and model-loaders. The former determines how neurons are mapped into PEs and how they are connected to each other. The latter defines several hardwired models and a mechanism to load from disk.

3.1.4 LP Mapping and Layouts. Neurons are assigned in groups to each PE. In Doryta, a batch of neurons is reserved to a range of PEs. There can be as many batches as necessary for any simulation.

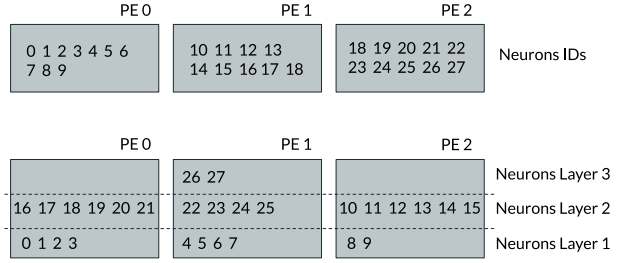

Figure 5: Mapping configurations. Top: A single fully connected network with 28 neurons. Bottom: 3-layer SNN with dimensions [5 10 16 2] (input 5, output 2).

Figure 5 shows two different mappings using the same number of neurons.

On top of the neuron mapping, we have the connections between neurons, the synapses. There are two kinds of connections defined in Doryta currently: 'all-to-all' connections and 2D convolutional connections. An all-to-all connection is defined from a range of output neurons to a range of receiving neurons. The total number of synapses in a all-to-all connection is equal to the number of output neurons times the number of receiving neurons. One possible SNN that can be built out of these components is a simple fully connected layer (see an example in Section 4.2). A 2D convolutional connection is also defined between a range of output neurons and receiving neurons, but it also requires the input image width, the parameters of the kernel (width and height), and the padding and striding of the kernel, both horizontally and vertically. The total number of synapses for the convolutional connection is at most $K_w \times K_h \times N_r$ where N_r is the total number of receiving neurons, and K_w and K_h are the width and height of the kernel. Note that the number of connections/synapses for a convolutional connection are much smaller than for an all-to-all connection.

Contrary to the general consensus on 2D convolutions, which determines that a convolutional layer takes a layer/image with multiple channels/colors and outputs a layer with multiple channels, Doryta's convolutions take images and output images with only one channel. The more general, multi-channel convolutions can be implemented as the accumulation of multiple single-channel, Doryta's convolutions.

All-to-all and convolutional connections are pervasive in ANN applications because of their simplicity and geometric characteristics. In Section 4.1, we show how to construct a grid for Conway's Game of Life made out of four convolutional connections.

3.2 Physical Performance Estimation at the Hardware Level

The performance estimation is accomplished hierarchically by first computing the performance of basic neurons, synapses, and interconnects and projecting them to the core and the chip levels for the neural networks implemented here (see Section 5.2). Note the chip is composed of several cores. Here, we report the methodology and performance metrics for neuronal spiking, synaptic integration, and data communication via metal interconnects.

3.2.1 Antiferromagnetic Neurons. We consider spiking neurons implemented using two different types of AFM materials: Mn_3Ir (metallic) and NiO (insulating). These AFM neurons generate a spike with an auto-reset when triggered with a current pulse that exceeds a threshold value, which depends on the material parameters and dimensions of the neuron. For a typical 5-nm thick NiO-based neuron, the critical electric current density to excite the neuronal dynamics is typically 1.83×10^7 A/cm^2 [36]. The fundamental frequency of spiking in AFM neuron depends on the current overdrive, i.e., the difference between the input electric current and critical current. Assuming the overdrive to be 10% for NiO neuron, we find that it spikes with a fundamental frequency of 20 GHz. For typical material parameters, the energy consumed for a spiking event in a 100 nm $\times$ 15 nm NiO-based neuron is around 1.5×10^{-14} Joules. As for a 5-nm thick Mn_3Ir-based neuron, the critical electric current density is 5.5×10^9 A/cm^2, while its fundamental spiking frequency is around 150 GHz [40]. The higher critical current for the latter case is due to the higher intrinsic damping present in metals as compared to insulators. Assuming a 10% overdrive in input current and 100 nm $\times$ 15 nm cross-section of Mn_3Ir neuron, we estimate its energy dissipation to be 4.5×10^{-15} Joules. Despite the high current density to excite spiking dynamics in Mn_3Ir neuron, its energy dissipation is lower than that of NiO neuron due to the lower input voltage requirement and faster dynamics in Mn_3Ir.

Table 1: Performance metrics of AFM neurons. The minimum feature size $F = 15$ nm. These metrics are obtained for 10% overdrive of input current compared to the threshold input current.

Performance metric	Value	
	NiO (insulator)	Mn_3Ir (metal)
Input electric current density	2×10^7 A/cm^2	2×10^9 A/cm^2
Input voltage	1 V	0.15 V
Fundamental spiking frequency	20 GHz	435 GHz
Output spike voltage	0.232 mV	2.1 mV
Latency	50 ps	2.3 ps
Power dissipation	0.3 mW	0.68 mW
Energy dissipation	1.5×10^{-14} J	1.55×10^{-15} J
Area	$20F^2$	$20F^2$

For comparison, a digital CMOS neuron consisting of two 8-bit registers, an 8-bit adder, 8 NAND gates, 8 inverters, and three 8-state elements, occupies 110 μm^2 area for 15-nm technology node [34]. The energy consumption per spike for the digital CMOS neuron is 136 aJ, while its operating frequency is 1.58 GHz. The analog CMOS neuron is based on an analog-to-digital converter read circuitry composed of 32 CMOS inverter cells and a synapse, which occupies 0.69 μm^2 cross-sectional area [34, 35]. The energy consumption of the analog CMOS neuron is $\approx$ 140 aJ and its operating frequency is $\approx$ 503 MHz.

3.2.2 Ferromagnetic Synapses. In the case of the memristive domain wall-based FM synapses considered here, a 320 nm $\times$ 15 nm device can host up to 64 distinct resistance levels [38]. Ignoring any effects of device-level non-ideality, as shown in Table 2, the conductance ranges from $1.2 \times 10^{-4}\ \Omega^{-1}$ to $2.67 \times 10^{-4}\ \Omega^{-1}$. In order to fire a neuron when the synapse's output is at the maximum level, the read voltage of the synapse is set to 1.125V and the latency of the read process is estimated as 0.27 ps. A reset pulse is applied following the reading pulse, and averaged energy consumption per synaptic operation is estimated to be around 0.081 aJ. By comparison, a digital CMOS synapse, consisting of an 8-bit SRAM register and a state element, consumes $\approx$ 170 aJ energy, while its latency is 0.64 ps [34].

Table 2: Metrics of the DW-based ferromagnetic synapse.

Parameter	Value
Conductance range ($10^{-4}\ \Omega^{-1}$)	1.2–2.67
Sensing current (milli-Amps)	0.18
Pulse width of current (picoseconds)	0.54
No. of conductance states	64
Energy dissipation (atto-Joules)	0.081

The footprint of the digital synapse is estimated as 1.38 μm^2. For an analog CMOS synapse, comprising two operational transconductance amplifiers, the energy consumption is $\approx$ 2 aJ, latency is 19 ps, while its area is 0.17 μm^2 [34, 35].

3.2.3 Interconnects. The aforementioned neurons and synapses are interconnected to each other via Cu/low-κ technology. The energy of these interconnect can be evaluated as

$$E_{\text{ic}} = C_l l V^2, \tag{7}$$

where C_l is the per-unit-length interconnect capacitance, l is the interconnect length, and V is the supply voltage which for Mn_3Ir neuron is 0.25 V while for NiO neuron is 0.87 V. For long, chip-level interconnects ($>$ 0.1 mm), $C_l = C_{l,\text{long}} = 5 \times 10^{-10}$ F/m, while for short interconnects with synapses ($<$ 0.1 mm), $C_l = C_{l,\text{short}} = 9.23 \times 10^{-11}$ F/m. We model the interconnect length for neurons, $l_{\text{ic,neu}}$, and that for synapses, $l_{\text{ic,syn}}$, according to

$$\begin{aligned} l_{\text{ic,syn}} &= \sqrt{a_{\text{syn}} s_{\text{core}}}, \\ l_{\text{ic,neu}} &= \sqrt{a_{\text{core}} c_{\text{layer}}}, \end{aligned} \tag{8}$$

where a_{syn} (a_{core}) is the synapse (core) area, s_{core} is the number of synapses per core, and c_{layer} is the number of cores per layer.

At the core level, the RC-delay dominates the total delay of the interconnects. Thus, we consider it as part of the synapse delay and evaluate it as

$$\tau_{\text{syn,ic}} = (0.38 R_{\text{ic}} C_{l,\text{short}} + R_{\text{eff}} C_{l,\text{short}} + R_{\text{ic}} C_{\text{load}}) l_{\text{ic,syn}}, \tag{9}$$

where $R_{\text{ic}} = 1.1{\times}10^9\,\Omega$ /mΩ is the interconnect resistance per-unit-length; and, $R_{\text{eff}} = 6.075 \times 10^3 \Omega$ is the effective resistance of a synapse, and $C_{\text{load}} = 2.17{\times}\ 10^{-}16$ C is the capacitance of the synapse.

At the chip level, the interconnect delay is associated with the neuron operations and is thus evaluated as

$$\tau_{\text{neu,ic}} = \frac{C_{l,\text{long}} l_{\text{ic,neu}} V_{\text{neu}}}{I_{\text{neu}}}, \tag{10}$$

where I_{neu} is the input current of the neuron, and V_{neu} is the neuron voltage. $I_{\text{neu}} = J_{\text{neu}} a_{\text{neu}}$, where J_{neu} is the neuron's input current density, while a_{neu} is the neuron's cross-sectional area. Table 1 lists the values of J_{neu} and V_{neu} for both Mn_3Ir and NiO neurons.

3.2.4 Chip-level benchmark. Instead of thoroughly designing each part of the chip, we introduce empirical factors to approximate the area associated with peripherals and to accommodate the design rules for component spacing. All cores in our work are connected using a crossbar architecture. The core area is given as

$$a_{\text{core}} = (a_{\text{neu}} n_{\text{core}} F_{\text{neu}} + a_{\text{syn}} n_{\text{in}} n_{\text{out}} F_{\text{syn}}) F_{\text{core}}, \tag{11}$$

where $F_{\text{syn}} = 3$, $F_{\text{neu}} = 3$ and $F_{\text{core}} = 2$ represent the empirical area factor for synapses, neurons and cores, respectively, and n_{in} (n_{out}) is the number of input (output) neurons for the core.

In case the number of input neurons is smaller than the synapses per neuron, the core area is estimated using the convolution core architecture [34]. Replacing the input neuron numbers by the synapse numbers per neuron, we obtain

$$a_{\text{core}} = (a_{\text{neu}} n_{\text{core}} F_{\text{neu}} + a_{\text{syn}} s_{\text{neu}} n_{\text{out}} F_{\text{syn}}) F_{\text{core}} \tag{12}$$

The fan-in of AFM neurons is considered to be infinite because the input current sums up in the interconnects naturally. All the cores in the same layer operate simultaneously and each core operation delay is determined by one synaptic operation and one neuron operation. The delay per core is given as

$$\tau_{\text{core}} = \tau_{\text{neu}} + \tau_{\text{syn}} + \tau_{\text{neu,ic}} + \tau_{\text{syn,ic}}. \tag{13}$$

The energy consumption per core is the summation of energy dissipated in all the synapses and neurons. The energy consumption of a core is workload-dependent because both the active synapse rate and the neuron fire rate depend on the workload-related parameters presented to the neurons and synapses. The active synapses per neuron and the active neurons per inference (per core) are

$$s_{\text{act,neu}} = s_{\text{neu}} s_{\text{act}}, \tag{14}$$

$$n_{\text{act,core}} = n_{\text{core}} n_{\text{act}}, \tag{15}$$

where s_{act} and n_{act} are the ratio of active synapses and neurons per layer, respectively.

Adding all up, the energy consumption per core is

$$E_{\text{core}} = E_{\text{syn}} s_{\text{act,neu}} n_{\text{core}} + E_{\text{neu}} n_{\text{act,core}}. \tag{16}$$

At the chip level, the area and energy consumption are the summation of all cores:

$$a_{\text{chip}} = \sum_i \sum_j a_{\text{core},ij}, \tag{17}$$

$$E_{\text{chip}} = \sum_i \sum_j E_{\text{core},ij}, \tag{18}$$

where i is the layer index, while j is the core index in a specific layer. It is worth noting that all cores of the same layer operate in parallel and thus the chip latency is given as

$$\tau_{\text{chip}} = \sum_i \max_{\text{layer},i}(\tau_{\text{core},j}). \tag{19}$$

4 NEUROMORPHIC APPLICATIONS

We present two outstanding applications to SNNs. First we show how to simulate Conway's Game of Life, and, in doing so, show that SNNs are Turing complete. Secondly, we tackle a classic machine learning problem: image classification.

4.1 Conway's Game of Life and Turing Completeness

Conway's Game of Life [15] (GoL) is an extensively studied cellular automata, a fascinating mathematical construction built out of a grid of cells and a list of rules that determine how the state of the cells change in time. Each cell in the grid is in one of two states: dead or alive, and each cell has eight neighbours. If the grid has a border, then some cells will have less than 8 neighbours. The rules of GoL are simple: a cell stays alive if there are two (2) or three (3) neighbouring cells alive, otherwise it dies; and, a cell comes to life if it has exactly three (3) alive neighbours, otherwise it stays dead.

As noted by [41], it is possible to encode the rules of GoL, and thus simulate GoL, as a multi-layered Neural Network. In Figure 6, we show a 2-layer NN which simulates one step of GoL using as activation function the step function centered on 0.5. Notice that the shape of the network's output (the third layer) is the same as its

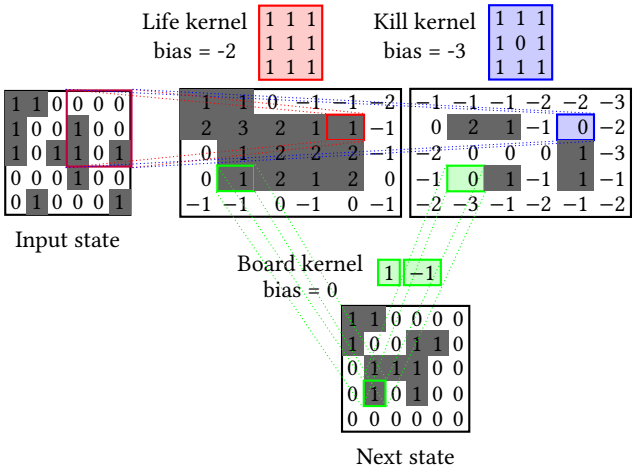

Figure 6: Conway's Game of Life as a 2-layer convolutional Neural Network. The first layer has a padding of 1 and a kernel with two filters (Life Kernel and Kill Kernel). The second layer has no padding and a kernel of 1×1, two input channels and one filter (the Board Kernel). A neuron fires (in grey) when its summation (plus bias) surpasses 0, i.e., the activation function for both layers is the step function centered on 0.5.

Table 3: Crossbar configurations for: (a) Conway's Game of Life (layer 1 receives spikes from layers 2, 3 and the initial input); (b) LeNet as crossbar connection. Boxes indicate crossbar units; lines represent connections between units and input/output.

(a)

Layer	Input Lines	Number of Neurons	Synapses per Neuron
Board	1200	400	3
Life	400	400	8.41
Kill	400	400	8.41

(b)

#	Type	Input Lines	Filters	Number of Neurons	Synapses per Neuron
1	Conv	784	1	784	1
2	Conv	784	6	784	22.90
3	Conv	784	6	196	4
4	Conv	1176	16	100	150
5	Conv	100	16	25	4
6	Full	400	-	120	400
7	Full	120	-	84	120
8	Full	84	-	100	84

input. This allows us to connect the output of the last layer as input to the first layer, thus building a recurrent network.

The first convolution is composed of a kernel of size 3×3 and two filters. We can analyze this kernel by breaking it up into two pieces, one per filter. The first 3×3 kernel corresponds to the Life kernel, which when applied counts the number of cells alive on groups of 9 cells at the time. The bias for the Life kernel (-2) is added up to these counts resulting in the Life number. It can be seen in Figure 6 that the Life number for the uppermost-rightmost quadrant in the input state is equal to 1 (in red). Note that all numbers are natural numbers. If the Life number is bigger than zero, this means that either the neuron was dead and at least three neighbours are alive or it is alive and at least two neighbors are alive; in either, case the neuron output should be 1 (the neuron fires and a spike is sent). The second 3×3 kernel corresponds to the Kill kernel, which when applied counts the number of cells alive around a cell. The bias for the Kill kernel (-3) is added up to this count resulting in the Kill number. If the Kill number is bigger than zero, this means that the number of cells alive around a central cell is bigger than 4, which is not a good outcome for the central cell as it cannot be alive with that number of neighbors regardless of its current state (dead or alive).

In Figure 6, the Kill number for the cell centered in the uppermost-rightmost quadrant is 0 (in blue). The two "images" resulting from the application of the first convolution (Life and Kill images) are binary, containing either 1s and 0s, or spikes and no-spikes, and they tell us which cells have a chance to be alive, and which must be stopped and killed (respectively). The second convolution (between the second and third layers) is in charge of determining the "life" status of each cell given the output from the Life and Kill images. The Board kernel distinguishes the input from the Life image or Kill image. All values in the Life image are multiplied by 1 and all values in the Kill kernel are multiplied by -1. The two resulting values, per cell, are added up, giving us the Board number. If the Board number is zero, then the cell is dead, but if it is one, then it is alive. In Figure 6, the green box on the next state image shows the result of adding up $1 \cdot 1$ and $0 \cdot -1$.

A cell in GoL using the 2-layer NN strategy from above can be represented by three neurons, one for the Board kernel and two for the Life and Kill kernels. The simulation takes two clock cycles to simulate one time step of GoL. In the first clock cycle, the input state is sent as spikes from the first to the second layer following the Life and Kill kernel weights and the neurons fire if the conditions apply. In the second clock cycle, the neurons that fired on the previous cycle send a spike to the first layer where it is determined whether the cell is alive or dead.

The neuron parameters for GoL are: V_e, V_{reset} of 0, C of 0.5, R of 1 and threshold equal to $0.5 - bias_i$, where $bias_i$ is the bias for the neuron i in one of the three possible neuron groups: Board, Life or Kill. The synapses' weights are: 0 if the neuron being connected corresponds to the same cell and the kernel is Kill, -1 if the synapse connects from the group of Kill neurons to the Board neurons, and 1 in any other case. See Table 3a for GoL's crossbar configuration for a grid of size 20×20. The number of input lines for the Board layer is three times that of the size of the grid because it collects the result of the Life and Kill layers as well as the initial state input.

A natural corollary result from simulating GoL as a SNN is that: **SNNs are Turing-complete** given that GoL has already been shown to be Turing-complete [3]. Thus, we have shown that the requirements for SNN Turing-completeness are at most: two neuron parameters (threshold and leak), one synaptic parameter (weight) and recurrent connections. Furthermore, it is possible to modify the 2-layer convolutional NN to make use of only non-negative numbers, which makes it possible to set a fixed threshold for all neurons getting rid off one neuron parameter, leaving us with only three requirements for Turing-Completeness. Similar work [13] made use of the equivalence between μ-recursive functions and Turing Machines to prove that SSNs are Turing-complete. We consider our GoL 2-layer NN to be much a simpler and understandable proof of Turing-completeness for SNNs.

4.2 MNIST Classification

MNIST [27] is a classical Machine Learning dataset consisting of 70,000 grayscale images, each consisting of 28×28-pixels and containing one handwritten digit. This dataset is commonly used to test the ability of new Machine Learning models to classify, and it also works as a bench test to showcase new hardware architectures without the insane requirements of state of the art ANNs.

A simple and popular feed forward network architecture used for image classification on MNIST is LeNet [26]. LeNet can be implemented in virtually any neural network framework, including Doryta and Whetstone [39]. Whetstone is written in Python to train ANNs with some restrictions. An ANN model trained with Whetstone can be encoded into memoryless SNNs little modification. Whetstone's trick is to define an activation function that approximates the step function to act as a threshold in SNNs. We follow the same crossbar structure as proposed by [34]. See Table 3b for the crossbar parameters of LeNet. The number of classes in MNIST is only 10, not 100, but the output of our network is 10 times larger because Whetstone requires redundant neurons to properly classify an image.

Because models trained in Whetstone use simpler, "memoryless" neurons, we have to decide on the parameters for the spiking neurons should take. The spiking neuron parameters we have selected are: V_e and V_{reset} of 0, C of $1/256$, R of 1 and the V_{th} (threshold) is equal to the bias of the neuron (plus 0.5 as per details of Whetstone's implementation). The synapses' weights are in the same manner equal to the weights obtained from Whetstone training.

4.2.1 Validating Doryta against Whetstone. After training LeNet in Whetstone, we saved it into a binary file readable by Doryta. The network was loaded in Doryta and then fed 10,000 test images. LeNet's output in Doryta was compared spike per spike against Whetstone's output. No differences in the output of both Doryta and Whetstone were found. This is remarkable as both Doryta and Whetstone are written in separate languages with completely different underlying libraries. In simulation speed, however, Whetstone takes seconds to infer the class of all 10,000 the images while Doryta takes a couple of minutes. This discrepancy is due to Doryta's nature of simulating individual neurons that forget at an exponential rate, while Whetstone treats each neuron as a summation box of input spikes with no memory of past events. We expect that once transferred to hardware, memoryfull neurons will operate independently and in parallel of each other at much higher rates than what Doryta can simulate. As we show in Section 5.2, inferencing a single image requires only 8 clock cycles (one clock cycle per layer) which in spintronic hardware could take up to only a couple of microseconds.

The trained neuron models, Doryta, and the Python scripts used to verify the equivalence of outputs between Doryta and Whetstone can be found on the supplementary material and code accompanying this paper.

5 EXPERIMENTAL RESULTS

Two sets of experiments were run: performance experiments using GoL for benchmarking, and energy estimation experiments for inference of MNIST images. All experiments in this work were run on up to 32 compute nodes on RPI's AiMOS supercomputer [10]. Each node of AiMOS is composed of two 20-core IBM Power 9 processors clocked at 3.15GHz and 512GiB of RAM.

For simulations performed in this work, Doryta was compiled using IBM's `XLC_r` and the Spectrum MPI library.

5.1 Performance Experiments - Strong Scaling

As seen in Table 4, using a single compute node with up to 40 processors, we ran a Game of Life simulation of size 1000×1000, starting with a random initial state where each cell had a probability of 20% of being alive, for a total of 1000 GoL steps. A total of 3,000,000 LPs are required for the simulation (3 neurons per cell) and 1.75×10^9 events were processed. Because of Doryta's determinism, every single run utilized the same number of net events to work. We found that ROSS's conservative mode produced the best scaling results, showing an almost linear speedup. Efficiency (for Optimistic and Optimistic Realtime modes) refers to $1 - \frac{e_{\text{rollback}}}{e_{\text{net}}}$, where e_{rollback} is the number of events rolled back and e_{net} is the number of net events, events processed. Efficiency is a measure shown in practice to be closely correlated to runtime. Note that even with near 100% efficiency in the simulations with Optimistic and Optimistic Realtime modes, the Conservative execution runs much faster. On a subsequent experiment, we ran the same simulations on up to 32 compute nodes, see Table 5. As one would expect, increasing the number of cores increases the number of remote events, events that have to be sent through the network to a different PE/core.

Notice that other network configurations like a fully connected layer might not scale. The GoL configuration is special and it scales well because the number of connections each neuron has is small (to its neighbours and itself, 9 at most). That is the advantage of 2D convolutional connections as opposed to all-to-all connections; convolutional connections are sparse while all-to-all connections are not. For future work, we have planned to optimize "spike broadcast" as to make these all-to-all connections as efficient and scalable as convolutional connections.

5.2 Hardware Performance Estimation

Following Nikonov and Young's energy estimation strategy [34], we defined a crossbar architecture for the task of digit recognition on MNIST. Then, given a spike workload, we calculated the area, latency and energy of the crossbar architecture for the classification of a single image. In their framework, Nikonov and Young assume an approximate layout for each component of the architecture in the chip, an approximate placement for each neuron, synapse and interconnect in each crossbar. Our crossbar architectures, like theirs, follow the LeNet architecture, as presented in Table 3b (ours only differing on the size of the last layer). For a more accurate performance estimation, we simulate the network in Doryta and gather the usage of each component (leak, fire and integrate) to create a specific workload for LeNet using MNIST and Fashion-MNIST. The performance estimation process takes all equations from Section 3.2 and computes them given the workloads produced by Doryta. This process was automated in a Python script which can be found as supplementary material.

All workload were obtained from Doryta's output. First the trained LeNet model on MNIST (or Fashion-MNIST) was loaded into Doryta, as explained in Section 4.2, and then a total of 10,000 test 28×28 black and white images as spikes were injected into the simulation. From processing these images, Doryta calculated the usage of key operations (fire, integrate and leak) for each layer of the network. The first workload was based on the LeNet model with the MNIST dataset, we nicknamed it the "Small LeNet" (SL) workload (see Table 6). The second workload was constructed by training LeNet on the Fashion-MNIST dataset [44], which is a dataset made to be a "hot swapping" replacement of MNIST where each image belongs to one of ten categories of clothing (the "Small LeNet Fashion" (SLF) workload).

Table 4: Strong scaling for a simulation of GoL. Randomly initialized GoL grid with size of 1000×1000. Sequential execution time of 4746.01 s. Conservative mode execution with lookahead of 0.01. Optimistic mode execution with default values (GVT of 16 and batch size of 16). Optimistic realtime mode with parameters: GVT enforced every 1 ms and a batch of 4.

	Conservative		Optimistic			Optimistic Realtime		
Cores	Runtime (sec)	Speedup	Runtime (sec)	Efficiency	Speedup	Runtime (sec)	Efficiency	Speedup
2	2962.78	**1.60**	4240.68	97.80%	1.12	3425.19	96.37%	1.39
4	1415.07	**3.35**	2743.21	95.79%	1.73	1761.16	90.94%	2.69
8	669.00	**7.09**	1786.44	97.01%	2.66	855.71	92.67%	5.55
16	328.31	**14.46**	1068.36	98.16%	4.44	419.97	95.21%	11.30
32	166.54	**28.50**	609.60	98.83%	7.79	207.87	97.22%	22.83
40	136.37	**34.80**	536.67	98.89%	8.84	171.01	97.39%	27.75

Table 5: Strong scaling on multiple nodes for a simulation of GoL using conservative mode in ROSS. Randomly initialized GoL grid with size of 1000×1000. Sequential execution time of 4746.01 s.

Nodes	Total Cores	Runtime (sec)	Remote Events	Speedup
2	80	68.95	3.32%	68.83
4	160	36.76	6.66%	129.10
8	320	21.31	13.34%	222.75
16	640	14.33	26.65%	331.28
32	1280	10.82	41.66%	438.65

Table 6: Average total number of integration and fire operations utilized on the inference of a black & white, 28×28 image from MNIST and Fashion-MNIST datasets. Accuracy of each network on their respective dataset: Small and Large LeNet on MNIST and Fashion-MNIST.

Workload	Integration	Fire	Accuracy
Small LeNet	73734.32	733.94	96.36%
Small LeNet Fashion	104355.98	1180.18	67.12%
Large LeNet	296092.07	965.04	98.14%
Large LeNet Fashion	752321.71	2612.02	70.06%

To improve accuracy, we increased the number of filters for the four intermediate layers of LeNet from 6 and 16, to 32 and 48, respectively. We found a 2% improvement in accuracy with this larger network on MNIST (see Table 6, "Large LeNet" (LL)), and a 3% improvement on Fashion-MNIST ("Large LeNet Fashion").

Table 7 shows the estimation of the performance per inference with the four different workloads. The energy consumption is composed of four different parts, as illustrated in Figure 7a: neurons, synapses, interconnects connected to the synapses and interconnects connected to the neurons. The energy of neuron firing is negligible, while the energy consumed in interconnects, especially the short interconnects with synapses, is dominant. This helps explain why compared with Mn_3Ir-based network, the NiO-based network consumes more energy despite the energy efficiency of the neuron spiking. Benchmarks corresponding to analog and digital CMOS neural networks are also included as a comparison. Notice in Table 7 the significantly superior performance of spintronics-based neural networks compared with their CMOS counterparts. The AFM neural networks are promising to operate in ~GHz range, while CMOS-based networks typically work in ~MHz frequency range. Energy consumed by analog CMOS devices is 3× higher than that of AFM devices, while the energy consumption of digital CMOS implementation is 10× that of analog CMOS implementation. We also report the compound metric energy-delay product (EDP) for both spintronics-based and CMOS-based networks, and we find that spintronics neural networks offer two orders of magnitude lower EDP compared to CMOS neural networks.

As an additional point, we would like to compare the spintronics performance to that of TrueNorth. According to Cheng et al. [12], TrueNorth could inference images from the MNIST dataset at a rate of 1249 frames per second with an energy performance of 6122.44 frames per second per watt. Given that 1 watt equals 1 J/s, 6122.44 frames/s/W is the same as 6122.44 frames/J or 163.334 μJ/frame (163, 334 nJ/frame). Note that their network architecture is based on CIFAR, a larger dataset than MNIST. TrueNorth is capable of crossbar connections of 256 input lines and 256 neurons only. We approximate that LLF would fit in at least 1400 TrueNorth cores. To balance out the specific architecture differences between Cheng et al. CIFAR-classifying networks and our MNIST networks, we divide their 163, 334 nJ/frame by a factor of 3, which is nearly a 1000× more energy per frame than our AFM design. This factor is the result of dividing the number of cores/crossbar units utilized by Cheng et al., 4064, by the estimated number of 1400 cores for our largest network (LLF). Our estimation for Digital CMOS (4574) is one order of magnitude away from Cheng et al. 54, 444 nJ/frame. This discrepancy can be explained by the cost associated with peripheral circuitry and transducers or amplifiers that will be needed, which we do not account for in our calculations.

Figure 7b shows the energy dissipated in each layer, which is dependent on workload allocation. The SLF workload has approximately 40% more integration and firing operations but the energy consumption is similar to the SL workload. Also, the LLF's integration and firing are 2.5 times that of LL's but less than twice that of energy consumption. The fourth and sixth layer are dominant in energy consumption and the 'Fashion' workload does not consume much more in these two layers so they are more energy efficient.

As mentioned in Equation 13, the latency of each layer is the collaborative contribution of one synaptic operation and one neuron operation and it is dominated by the number of layers and interconnect delay per core. The area of each layer, dominated at the chip-level by the interconnect, is the largest contribution to the delay difference between different workloads implemented using the same neuron device. Figure 7d shows the latency of processing single layers. Note that the only the network size, the number of cores and their size, influence the latency of each layer. There is no difference latency-wise between workloads running on the same network architecture (e.g., no difference in latency between SL and SLF).

Table 7: Performance of various technologies on different workloads. Note that the iso-latency power dissipation of various networks will be directly proportional to their energy consumption and is therefore not reported specifically in the table.

Neuron	Workload	Area (mm^2)	Latency (ps)	Energy (nJ)	E· τ (10^{-18}s· J)
Mn_3Ir	SL	0.045	56	8	0.5
	SLF	0.045	56	7	0.4
	LL	0.259	77	123	10
	LLF	0.259	77	205	16
NiO	SL	0.045	647	8	5.4
	SLF	0.045	647	7	4.7
	LL	0.259	948	124	117
	LLF	0.259	948	206	195
Analog CMOS	SL	1.7	18594	28	523
	SLF	1.7	18594	25	460
	LL	9.7	21864	399	8716
	LLF	9.7	21864	666	14560
Digital CMOS	SL	16	42380	263	11125
	SLF	16	42380	239	10123
	LL	88	80929	2626	212547
	LLF	88	80929	4574	370202

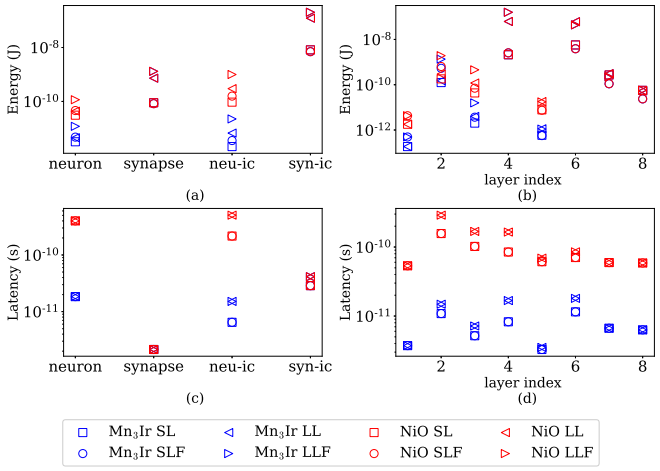

Figure 7: (a) Energy consumed by different components. 'neu-ic' ('syn-ic') is chip-level (core-level) interconnect. (b) Energy consumption reported layer by layer. (c) Latency of devices and interconnects. (d) Latency of each layer.

Figure 7c shows the latency of each components of the network. The neuron and neuron-connected interconnect latency are dominant while the synapse latency is negligible. The synapse-connected interconnect latency is similar for all neuron types and workloads. The interconnect plays a significant role in both the energy consumption and the latency computation, so much in fact that it has become a bottleneck of further reduction in dissipation and latency.

Even though the performance results for spintronic neural networks are exciting, the spintronic devices have some limitations compared to ideal LIF neurons. First, the weights that synapses store can only be non-negative. We found that enforcing non-negative constraints on the synapses weights resulted in trickier to train SNN models, yet, once trained, the workloads were not substantially different from those that we used. Secondly, neuron leak and threshold cannot be tweaked as they are values intrinsic to the materials. Fortunately, when only considering positive weights for synapses, the neuron threshold can be fixed, so that the synapses weights need only to be scaled accordingly to fit the fixed threshold. Lastly, the spintronic synapses have about 64 different levels (6 bits), which—compared to the SNNs trained for this work—is a fraction of what single floating-point numbers can store. We found that restricting the network to 8 bits unsigned values did not affect accuracy by more than half a percent, but when restricting the network to 6 bits there was significant reductions in accuracy. However, 4 bits have been shown to be enough for NNs to learn [42]. Thus, with the right technique to discretize the network, 6 bits could behave as 8 bits, i.e., our workloads are a good enough approximation of what the architecture would encounter when constrained to 6 bits. This is without taking into account different input image encoding. We found positive improvements on accuracy (up to 10%) when spikes are temporally encoded.

6 CONCLUSION

We have presented Doryta, a parallel discrete-event-based, chip-agnostic simulator for neuromorphic applications applied to the energy estimation of novel spintronic-based devices. Due to its modular design and mapping strategy, we've observed that Doryta can be scaled up to well over 1,000 of CPU cores. Additionally, Doryta is able to reproduce the inference results of another spiking-neural network library, Whetstone, with one key advantage: it takes into account time information and delays. We showed that Turing-complete requirements for neuromorphic computing are at most: one neuron parameter (leak), one synaptic parameter (weight) and recurrent connections. Loading Whetstone's models in Doryta, allowed us to determine the workloads that spintronic-based chips would require in terms of basic operations. In an analysis of two spintronic-neuron models, Mn_3Ir and NiO, we found that the bulk of the energy consumption of the chips would be driven by the connection between neurons and not the neurons themselves, the interconnect. Compared to CMOS architectures, our analysis indicates that spintronic-based chips have an energy-delay product that is **three to six orders of magnitude smaller** at inferencing a single image using the LeNet architecture.

For future work, we intend to optimize Doryta's LP model communications and improve their scalability to bring it up to the level of convolutional connections. On the hardware performance estimation, we plan to incorporate the cost associated with peripheral circuitry and transducers or amplifiers that are needed in a working chip. We plan to look into ferromagnetic based spiking neurons due to their ease of fabrication and characterization in the GHz regime as opposed to THz for spintronics. Finally, since we found that interconnects are the bottlenecks, we see an interesting avenue for research in the improvement of interconnects. We believe that building on Doryta and the techniques in this work will be beneficial to deepening our understanding of the neuromorphic computing devices of the future.

ACKNOWLEDGMENTS

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-21-1-1010. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation

thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government. Additionally, computational time on the AiMOS supercomputer was provided by Rensselaer's Center for Computational Innovations.

REFERENCES

[1] Peter D. Barnes, Jr., Christopher D. Carothers, David R. Jefferson, and Justin M. LaPre. 2013. Warp speed: executing time warp on 1,966,080 cores. In *Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation* (Montreal, Quebec, Canada). ACM, 327–336.

[2] D. W. Bauer Jr., C. D. Carothers, and A. Holder. 2009. Scalable Time Warp on Blue Gene Supercomputers. In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, Washington, DC, USA, 35–44.

[3] Elwyn R. Berlekamp, John Horton Conway, and Richard K. Guy. 1982. *Winning Ways for Your Mathematical Plays. 2: Games in Particular.* Academic Press, London.

[4] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M. Bower, Markus Diesmann, Abigail Morrison, Philip H. Goodman, Frederick C. Harris, Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew P. Davison, Sami El Boustani, and Alain Destexhe. 2007. Simulation of Networks of Spiking Neurons: A Review of Tools and Strategies. *J Comput Neurosci* 23, 3 (Dec. 2007), 349–398.

[5] R. E. Bryant. 1977. *Simulation of Packet Communication Architecture Computer Systems.* Ph. D. Dissertation. MIT.

[6] Yi Cao, AndrewW Rushforth, Yu Sheng, Houzhi Zheng, and Kaiyou Wang. 2019. Tuning a Binary Ferromagnet into a Multistate Synapse with Spin–Orbit-Torque-Induced Plasticity. *Adv Funct Mater* 29, 25 (2019), 1808104.

[7] Christopher D. Carothers and Kalyan S. Perumalla. 2010. On deciding between conservative and optimistic approaches on massively parallel platforms. In *Winter Simulation Conference'10*. 678–687.

[8] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto. 1999. Efficient optimistic parallel simulations using reverse computation. *ACM Trans. Model. Comput. Simul.* 9, 3 (1999), 224–253.

[9] Andrew S. Cassidy, Paul Merolla, John V. Arthur, Steve K. Esser, Bryan Jackson, Rodrigo Alvarez-Icaza, Pallab Datta, Jun Sawada, Theodore M. Wong, Vitaly Feldman, Arnon Amir, Daniel Ben-Dayan Rubin, Filipp Akopyan, Emmett McQuinn, William P. Risk, and Dharmendra S. Modha. 2013. Cognitive Computing Building Block: A Versatile and Efficient Digital Neuron Model for Neurosynaptic Cores. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*. 1–10.

[10] Center for Computational Innovations. [n. d.]. Artificial intelligence multiprocessing optimized system (AiMOS). cci.rpi.edu. https://cci.rpi.edu/aimos (accessed Feb 1, 2022).

[11] K. M. Chandy and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. In *IEEE Transactions on Software Engineering*, Vol. 24. 440–452.

[12] Hsin-Pai Cheng, Wei Wen, Chunpeng Wu, Sicheng Li, Hai Helen Li, and Yiran Chen. 2017. Understanding the Design of IBM Neurosynaptic System and Its Tradeoffs: A User Perspective. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, Lausanne, 139–144.

[13] Prasanna Date, Catherine Schuman, Bill Kay, and Thomas Potok. 2021. Neuromorphic Computing Is Turing-Complete. *arXiv:2104.13983 [cs]* (April 2021).

[14] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. H. Weng, A. Wild, Y. Yang, and H. Wang. 2018. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* 38, 1 (January 2018), 82–99.

[15] Martin Gardner. 1970. The Fantastic Combinations of Jhon Conway's New Solitaire Game "Life". *Sci. Am.* 223 (1970), 20–123.

[16] Wulfram Gerstner and Werner M. Kistler. 2002. *Spiking Neuron Models: Single Neurons, Populations, Plasticity.* Cambridge University Press, Cambridge, U.K.

[17] Julie Grollier, Damien Querlioz, KY Camsari, Karin Everschor-Sitte, Shunsuke Fukami, and Mark D Stiles. 2020. Neuromorphic spintronics. *Nature electronics* 3, 7 (2020), 360–370.

[18] Jennifer Hasler and Harry Marr. 2013. Finding a roadmap to achieve large neuromorphic hardware systems. *Front Neurosci* 7 (2013).

[19] Atsufumi Hirohata, Keisuke Yamada, Yoshinobu Nakatani, Ioan-Lucian Prejbeanu, Bernard Diény, Philipp Pirro, and Burkard Hillebrands. 2020. Review on spintronics: Principles and device applications. *J Magn Magn Mater* 509 (2020), 166711.

[20] David Jefferson, Brian Beckman, Fred Wieland, Leo Blume, and Mike Di Loreto. 1987. Time warp operating system. *ACM Symp. Operating Syst. Rev.* 21, 5 (Nov. 1987), 77–93.

[21] David Jefferson, Brian Beckman, Fred Wieland, Leo Blume, Mike Di Loreto, Phil Hontalas, Pierre Laroche, Kathy Sturdevant, Jack Tupman, Van Warren, John Wedel, Herb Younger, and Steve Bellenot. 1987. *Distributed simulation and the time warp operating system.* Technical Report OSTI 5639121. NASA Jet Propulsion Laboratory, Pasadena, CA, USA.

[22] David Jefferson and Henry Sowizral. 1985. *Fast concurrent simulation using the time warp mechanism.* Technical Report ADA129431. Rand. Corp., Santa Monica, CA, USA. Issue 2.

[23] David R. Jefferson. 1985. Virtual time. *ACM Trans. Program. Lang. Syst.* 7, 3 (1985), 404–425.

[24] T Jungwirth, J Sinova, Aurelien Manchon, X Marti, J Wunderlich, and C Felser. 2018. The multiple directions of antiferromagnetic spintronics. *Nat Phys* 14, 3 (2018), 200–203.

[25] Roman Khymyn, Ivan Lisenkov, James Voorheis, Olga Sulymenko, Oleksandr Prokopenko, Vasil Tiberkevich, Johan Akerman, and Andrei Slavin. 2018. Ultra-fast artificial neuron: generation of picosecond-duration spikes in a current-driven antiferromagnetic auto-oscillator. *Scientific reports* 8, 1 (2018), 1–9.

[26] Yann LeCun, Bernhard Boser, John S. Denker, Donnie. Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Comput.* 1, 4 (Dec. 1989), 541–551.

[27] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. 1998. The MNIST Handwritten Digit Database. http://yann.lecun.com/exdb/mnist/.

[28] Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, and Carlos Maltzahn. 2012. On the Role of Burst Buffers in Leadership-class Storage Systems. In *Proceedings of the 2012 IEEE Conference on Massive Data Storage*. Pacific Grove, CA.

[29] Danijela Marković, Alice Mizrahi, Damien Querlioz, and Julie Grollier. 2020. Physics for neuromorphic computing. *Nature Reviews Physics* 2, 9 (2020), 499–510.

[30] Neil McGlohon and Christopher D. Carothers. 2021. Toward Unbiased Deterministic Total Ordering of Parallel Simulations with Simultaneous Events. In *Proceedings of the Winter Simulation Conference* (Phoenix, Arizona) *(WSC '21)*. IEEE Press.

[31] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (2014), 668–673.

[32] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns. 2017. Enabling Parallel Simulation of Large-Scale HPC Network Systems. *IEEE Trans. Parallel Distrib. Syst.* 28, 1 (2017), 87–100.

[33] David M. Nicol. 1993. The cost of conservative synchronization in parallel discrete event simulations. *J. ACM* 40, 2 (April 1993), 304–333.

[34] Dmitri E. Nikonov and Ian A. Young. 2019. Benchmarking Delay and Energy of Neural Inference Circuits. *IEEE J. Explor. Solid-State Comput. Devices Circuits* 5, 2 (Dec. 2019), 75–84.

[35] Chenyun Pan and Azad Naeemi. 2016. Non-Boolean Computing Benchmarking for Beyond-CMOS Devices Based on Cellular Neural Network. *IEEE J. Explor. Solid-State Computat. Devices Circuits* 2 (2016), 36–43.

[36] Arun Parthasarathy, Egecan Cogulu, Andrew D Kent, and Shaloo Rakheja. 2021. Precessional spin-torque dynamics in biaxial antiferromagnets. *Phys Rev B* 103, 2 (2021), 024450.

[37] Mark Plagge, Christopher D. Carothers, and Elsa Gonsiorowski. 2016. NeMo: A Massively Parallel Discrete-Event Simulation Model for Neuromorphic Architectures. In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '16)*. Association for Computing Machinery, New York, NY, USA, 233–244.

[38] Abhronil Sengupta, Aparajita Banerjee, and Kaushik Roy. 2016. Hybrid Spintronic-CMOS Spiking Neural Network with On-Chip Learning: Devices, Circuits, and Systems. *Phys. Rev. Applied* 6 (Dec 2016), 064003. Issue 6.

[39] William Severa, Craig M. Vineyard, Ryan Dellana, Stephen J. Verzi, and James B. Aimone. 2019. Whetstone: A Method for Training Deep Artificial Neural Networks for Binary Communication. *Nat Mach Intell* 1, 2 (Feb. 2019), 86–94.

[40] Ankit Shukla and Shaloo Rakheja. 2022. Spin-Torque-Driven Terahertz Auto-Oscillations in Noncollinear Coplanar Antiferromagnets. *Phys. Rev. Applied* 17, 3 (2022), 034037.

[41] Jacob M. Springer and Garrett T. Kenyon. 2020. It's Hard for Neural Networks To Learn the Game of Life. *arXiv:2009.01398 [cs, stat]* (Sept. 2020).

[42] Xiao Sun, Naigang Wang, Chia-Yu Chen, Jiamin Ni, Ankur Agrawal, Xiaodong Cui, Swagath Venkataramani, Kaoutar El Maghraoui, Vijayalakshmi (Viji) Srinivasan, and Kailash Gopalakrishnan. 2020. Ultra-Low Precision 4-Bit Training of Deep Neural Networks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1796–1807.

[43] Jacob Torrejon, Mathieu Riou, Flavio Abreu Araujo, Sumito Tsunegi, Guru Khalsa, Damien Querlioz, Paolo Bortolotti, Vincent Cros, Kay Yakushiji, Akio Fukushima, et al. 2017. Neuromorphic computing with nanoscale spintronic oscillators. *Nature* 547, 7664 (2017), 428–431.

[44] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747 [cs, stat]* (Sept. 2017).