

Exploring Dendrites in Large-Scale Neuromorphic Architectures

James A. Boyle^{1,2}, Jason Ho¹, Mark Plagge², Suma George Cardwell², Frances S. Chance², Andreas Gerstlauer¹

¹*Electrical and Computer Engineering, The University of Texas at Austin, Texas, USA*

²*Sandia National Laboratories, New Mexico, USA*

{james.boyle, jason_ho, gerstl}@utexas.edu; {mplagge, sgcardw, fschanc}@sandia.gov

Abstract—Dendritic computation, inspired by the complex processing performed by biological dendrites, offers opportunities for designing efficient neuromorphic chips. This paper explores the use of dendrites within large-scale neuromorphic architectures. We show how digital or analog dendritic circuit implementations can be integrated within large-scale spiking architectures and the resulting impact on energy efficiency of sample spiking neural network (SNN) applications. We show a workflow for training and configuring hardware dendrites. We develop extensions and plug-ins for our existing neuromorphic architecture simulator to rapidly simulate digital and analog hardware dendrites. Using four tasks, we demonstrate that a large-scale architecture that leverages the richer functionality of digital hardware dendrites can achieve 56% energy savings compared to a baseline architecture, due to the smaller number of neurons required. Additionally, we show that a design using analog hardware dendrites uses 27% less energy compared to the fully digital implementations, reducing the energy consumed by the dendrites to just 2% of the total on-chip energy.

Index Terms—Neuromorphic computing, performance modeling, design-space exploration

I. INTRODUCTION

Neuromorphic computing applies brain-inspired concepts to devices, circuits, and architectures to design power-efficient hardware. One prominent area of neuromorphic research is concerned with spiking neural networks (SNNs) and the design of hardware optimized to execute SNNs. These networks emulate the spike-driven dynamics of neurons in the brain and can use neural models at varying levels of biological fidelity. While neuromorphic research has typically focused on somatic and synaptic models, an emerging area of interest is dendrites and dendritic computation, i.e., the complex tree-like connectivity that aggregates synaptic inputs into the neuron’s soma. Recent work has highlighted that neuromorphic dendrites have potential for increasing the expressivity of neuron models, and can efficiently perform operations such as non-linear filtering, coincidence and sequence detection, and multiplication for spatial transforms [1]–[3].

Earlier circuits proposed to emulate individual dendrites [3]–[5] have focused on implementing single dendritic branches or trees in isolation, where the ultimate impact on real-world application performance and efficiency is not well understood. Large-scale spiking hardware platforms, such as Intel’s Loihi [6] and BrainScale-S 2 [7], implement a fixed model with limited functionality, supporting only a small set of dendritic dynamics or configurations. An open question

remains how to leverage the richer dynamics and expressivity that dedicated hardware support for dendrites can provide, to achieve optimal power efficiency within the context of large-scale platforms across a wide range of applications.

In this paper, we explore the impact of adding dedicated hardware support for emulation of advanced dendritic behavior within large-scale spiking architectures. We first propose a novel methodology for training analog dendrite circuit parameters, where we train a simplified linear dendrite model and then convert abstract parameters to transistor gate voltages. We then describe an extension to our spiking architecture simulator, SANA-FE [8], [9], to support rapid modeling of different hardware dendrites for design-space exploration. Using SANA-FE, we simulate digital and analog dendritic circuit implementations using a plug-in with custom energy modeling. We use our dendrite plug-in with example applications trained with our methodology to show that digital and analog hardware dendrites can achieve 56% and 68% energy improvement compared to Intel’s Loihi, respectively. Finally, we show that rapid design-space exploration is feasible for large networks by simulating systems with more than 130,000 dendrites.

In summary, our contributions are as follows:

- We propose a novel methodology for training analog dendrites, in which we train an equivalent RC transmission line model and apply conversion steps to calculate various circuit parameters.
- We describe simulator extensions for efficiently modeling digital and analog implementations of hardware dendrites.
- We explore the use of digital and analog dendrites within a hardware platform based on Intel’s Loihi, using four benchmark applications. Results show that designs with digital and analog dendrites can accurately perform all four tasks, while reducing energy compared to Loihi by 56% and 68%, respectively.

The rest of the paper is organized as follows: Section II first describes related work, and Section III gives an overview of different hardware dendrite implementations. Section IV describes our novel methodology for training analog dendrites. Section V details our approach for modeling digital and analog hardware dendrites using our SANA-FE architectural-level neuromorphic hardware simulator. Section VI describes experiments and results, and Section VII concludes the paper with a summary and outlook on potential future work.

II. RELATED WORK

Various hardware implementations of dendrites using either digital logic or analog circuits have been proposed. Analog dendrites have been implemented using custom circuits [10]–[12], but these have mostly been looked at in isolation or only with small numbers of neurons. The modular and flexible circuit from [4] implements dendrites on a Field-Programmable-Analog-Array (FPAA) using a small configurable series of RC compartments. This implementation is easily extended to program larger numbers of dendrites and compartments. As such, we use it as a basis for explorations of analog dendrites in this paper.

Existing large-scale platforms for running real-world SNN applications support dendrites with varying levels of flexibility and capability. Digital platforms such as SpiNNaker 2 [13] and Intel’s Loihi [6] support dendrites either through software or custom logic. SpiNNaker 2 uses a CPU-based design and does not have custom dendrite hardware. Loihi has custom digital logic to support simple multi-compartment dendrites, but can only perform a small set of basic operations. The mixed-signal platform BrainScaleS-2 [7] implements complex, biologically-accurate circuits, but these are limited to a single dendrite model. Furthermore, mixed-signal platforms have limited scalability compared to fully-digital designs, as they require dedicated hardware for every supported neuron. Several architectures have been proposed with support for more advanced digital or mixed-signal dendrites [14], [15]. However, these are still limited to specific operations or small-scale demonstrations.

Dendrites have been modeled at varying levels of biological fidelity and hardware accuracy for exploration of their functional richness and implementation efficiency. Biologically-focused simulators such as NEURON [16], N2A [17], and Dendriify [18] simulate different neuronal dynamics including the behavior of dendrites. These tools are primarily designed for modeling biological features of dendrites and are not well-suited for modeling hardware. Other tools, such as Norse [19] and TorchDendrite [20], implement abstract models suitable for rapid training and simulation of SNNs with dendrites at the application level. However, these training frameworks do not emulate hardware effects and cannot model energy or performance. Dendritic devices and circuits have been modeled using low-level custom circuit simulators such as SPICE [21] or using differential circuit equations solved in MATLAB/Simulink [4]. Such circuit-level simulations are accurate for a small number of neurons but scale poorly and are too slow for design-space exploration.

III. BACKGROUND

In the following section, we describe a general template for large-scale spiking architectures and how they could be adapted to integrate more advanced analog or digital dendrite implementations. We then detail digital and analog dendrite implementations used in our explorations.

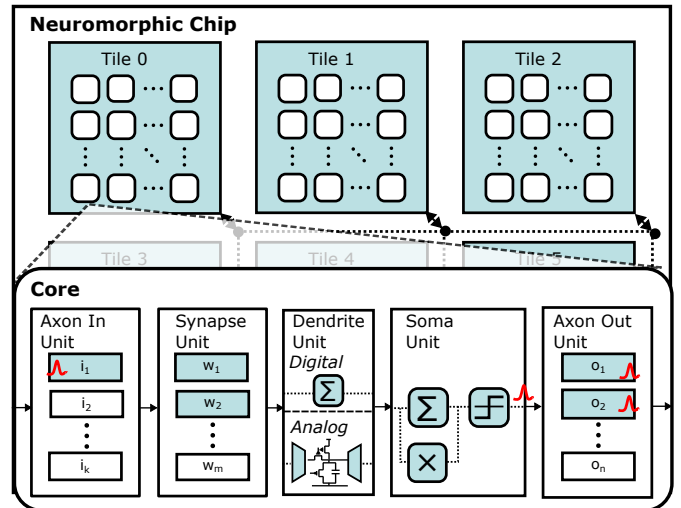


Fig. 1: Example of a large-scale neuromorphic architecture.

A. Large-Scale Spiking Architectures

A number of large-scale spiking architectures have been proposed and deployed, with varying levels of dendrite hardware support [6], [7], [13]. Such designs are typically based on a Network-on-Chip (NoC) and organized hierarchically (Fig. 1): designs are partitioned into one or more network tiles, where each tile contains some shared resource e.g., a network router. Each tile contains a number of neural cores implementing custom hardware for processing spikes. Spike processing pipelines implement a fixed sequence of axonal, synaptic, dendritic, and somatic hardware units, which can be implemented digitally or with analog circuits, i.e., using a mixed-signal approach.

Dendritic processing specifically can be performed digitally or using analog circuits within a mixed-signal SNN base platform, although this may require conversion interfaces. The dendrite hardware unit receives synaptic information to process, e.g., describing the weighted connections between neurons. If the dendrite unit is analog and inputs are represented digitally, there must be a Digital-to-Analog (DAC) converter before the dendritic circuits. Inputs are then processed by the circuit, which may include accumulation and other operations. Processing by an analog dendrite produces a current or voltage to be output to the soma; if the soma unit is digital, then an Analog-to-Digital (ADC) converter is also required.

B. Dendrite Implementations

We explore extensions of the general architecture template described in Section III to incorporate digital and analog dendrite implementations detailed in this section. These implementations are based on modeling a single dendritic branch in a biological neuron (Fig. 2(a)). In these implementations, dendrite dynamics following cable theory are replicated as a variable-length chain or ladder of RC compartments (Fig. 2(b)).

For our digital implementation, we reproduce the compartment based model from TorchDendrite [22]. In this model,

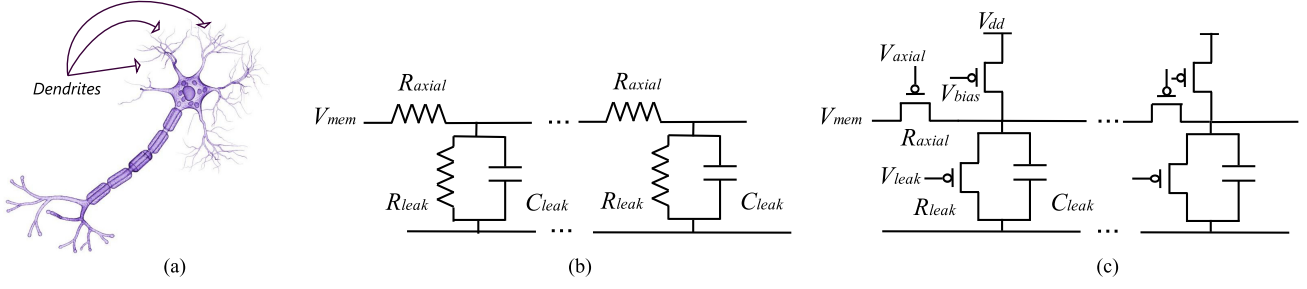


Fig. 2: Abstractions of a biological dendrite using a transistor-based circuit. Figure adapted from [22].

a dendrite has a chain of N compartments, where each compartment $n = 1, 2, \dots, N$ has a voltage value v_n evaluated at discrete time-steps. In each time step, Eq. (1) [22] is used to calculate the compartment voltages v'_n for the next time step, where i_n is an optional input value and compartments can be configured using parameters for decay (α) and axial conductance (β) that reflect the time and space constants of the RC ladder:

$$v'_n = \begin{cases} \alpha_1 v_1 + i_1 + \beta_1 (v_2 - v_1) & \text{if } n = 1 \\ \alpha_n v_n + i_n + \beta_n (v_{n+1} - v_n) + \beta_{n-1} (v_{n-1} - v_n) & \text{if } 1 < n < N \\ \alpha_N v_N + i_N + \beta_{N-1} (v_{N-1} - v_N) & \text{if } n = N \end{cases} \quad (1)$$

The output of the dendrite is given by its first compartment voltage v_1 . Parameters α and β can be trained directly for a given application, as described in more detail in Section IV.

For an analog dendrite implementation, we base our exploration on the circuit from [4] shown in Fig. 2(c). The dynamics of the dendrite can be configured by setting the axial and leakage resistances and the capacitance at each compartment. This circuit has been demonstrated on a Field-Programmable Analog Array (FPAA) [4], which supports rapid reconfiguration of circuit components. Within the FPAA, each compartment has tunable resistances implemented by floating-gate transistors operating in their sub-threshold regime. Specifically, each compartment $n = 1, 2, \dots, N$ has a leakage, axial, and bias resistance, which can be configured by setting the transistor gate voltages $V_{leak,n}$, $V_{axial,n}$ and $V_{bias,n}$ respectively.

The behavior of the analog dendrite, i.e., the voltages at different compartments, can be described by Eqs. (2)–(3) [4]. Circuit parameters and their values in our explorations are summarized in Table I. We first define useful intermediate scaling factors $k_{leak,n}$, $k_{axial,n}$ and $k_{bias,n}$:

$$I'_0 = I_0 \exp\left(V_{dd} \frac{\kappa - 1}{U_T}\right) \quad k_{leak,n} = I'_0 \exp\left(\frac{-\kappa V_{leak,n}}{U_T}\right), \quad (2)$$

where $k_{axial,n}$ and $k_{bias,n}$ can be calculated in the same manner by replacing $V_{leak,n}$ with the corresponding gate voltage $V_{axial,n}$ and $V_{bias,n}$. Applying Kirchhoff's voltage law and creating terms for input, axial (both distal and proximal),

leakage, and bias currents, one compartment's voltage v_n is expressed using the following differential equation:

$$\frac{dv_n}{dt} = \frac{1}{C_{leak}} \left[i_n + k_{axial,n} \left(\exp\left(\frac{v_{n-1}}{U_T}\right) - \exp\left(\frac{v_n}{U_T}\right) \right) + k_{axial,n} \left(\exp\left(\frac{v_{n+1}}{U_T}\right) - \exp\left(\frac{v_n}{U_T}\right) \right) + k_{leak,n} \left(\exp\left(\frac{E_k}{U_T}\right) - \exp\left(\frac{v_n}{U_T}\right) \right) + k_{bias,n} \left(\exp\left(\frac{V_{dd}}{U_T}\right) - \exp\left(\frac{v_n}{U_T}\right) \right) \right]. \quad (3)$$

For brevity, Eq. (3) only gives the general form when $1 < n < N$ without boundary conditions. For the boundary cases: if $n = 1$, the first axial term is omitted, and if $n = N$, the second axial term is omitted.

The entire dendrite's dynamics are then given by solving a system of these differential equations for all N compartments. Similar to the digital dendrite, the output of a dendrite is taken from the first compartment as $v_1 - V_{mem}$.

IV. TRAINING DENDRITES

Training is usually required to tune parameters such as synaptic weights in SNN-based applications. For SNNs using the previously described hardware dendrites, training should also determine each compartment's time (decay) and space (conductance) parameters. However, support for training dendritic hardware is currently limited. SNNAX [23] and SNN-Torch [24] both adapt existing deep-learning frameworks for SNNs and are extensible, but these do not currently support dendrites. TorchDendrite [20] extends SNN-Torch with support for dendritic layers [22]. While TorchDendrite parameters can be directly used for the digital implementation, these are abstract and therefore must be converted before being applied to analog dendrites.

For analog dendrites, each compartment can be configured via $V_{leak,n}$, $V_{axial,n}$ and $V_{bias,n}$, which control the programmable transistor resistances defining the RC dynamics of the compartment. Configuring dendrites for an application requires setting these transistor gate voltages. However, approaches to train these directly with existing deep-learning

TABLE I: Analog dendrite circuit parameters.

Parameter	Value	Description
I_0	1 fA	Transistor current scaling (I_{DS})
V_{dd}	2.4 V	Supply voltage
κ	0.846	Transistor gate modulation factor
U_T	25 mV	Thermal constant
C_{leak}	500 fF	Compartment capacitance
E_k	1.0 V	External potential offset
V_{mem}	1.02 V	Biased steady-state membrane potential
dt	10 μ s	Time-step duration
I_{scale}	100 pA	Input current scaling factor

frameworks are lacking. Instead, it is possible to train a simpler model, such as the one used by digital dendrites, and then convert parameters to gate voltages that achieve similar behavior. To do so, we adapt Eqs. (4) and (5) from [4]. These equations approximately calculate the time (τ_n) and space constants (λ_n) for an RC compartment, given the transistor gate voltage and other constant parameters (Table I):

$$\tau_n = \frac{C_{leak} U_T}{k_{leak,n}} \exp\left(\frac{-V_{mem}}{U_T}\right) \quad (4)$$

$$\lambda_n = \exp\left(\frac{\kappa [V_{leak,n} - V_{axial,n}]}{2U_T}\right) \quad (5)$$

Eqs. (4) and (5) can be rearranged to solve for the leakage (Eq. (6)) and axial (Eq. (7)) transistor gate voltages:

$$V_{leak,n} = \frac{U_T}{\kappa} \left[\ln\left(\frac{I'_0 \tau_n}{U_T C_{leak}}\right) + V_{mem} \right] \quad (6)$$

$$V_{axial,n} = V_{leak,n} - \frac{2U_T}{\kappa} \ln(\lambda_n) \quad (7)$$

Eqs. (6) and (7) can be used to calculate the compartment gate voltages given values for λ and τ . After configuring the leakage and axial transistors, the compartment's resting potential must be biased to V_{mem} using a third transistor. To calculate the bias transistor's gate voltage $V_{bias,n}$, we assume that at steady-state, $I_{bias,n} \equiv I_{leak,n}$. By equating the current flowing across the leakage transistor (Eq. (8)) to the sub-threshold current across the bias transistor (Eq. (9)), we find a solution for $V_{bias,n}$ (Eq. (10)):

$$I_{leak,n} = I_0 \exp\left(\frac{[\kappa - 1] V_{dd} - \kappa V_{leak,n}}{U_T}\right) \cdot \left[\exp\left(\frac{V_{mem}}{U_T}\right) - \exp\left(\frac{E_k}{U_T}\right) \right] \quad (8)$$

$$I_{leak,n} \equiv I_{bias,n} = I'_0 \exp\left(\frac{-\kappa V_{bias,n}}{U_T}\right) \quad (9)$$

$$V_{bias,n} = \frac{-U_T}{\kappa} \ln\left(\frac{I_{leak,n}}{I'_0 \left[\exp\left(\frac{V_{dd}}{U_T}\right) - \exp\left(\frac{V_{mem}}{U_T}\right) \right]}\right) \quad (10)$$

The analysis that these equations are based on is only accurate while compartment voltages remain close to V_{mem} (i.e., within $\pm U_T$). Even when operating within this voltage range, the circuit's behavior may slightly differ from its

equivalent RC transmission line model due to transistor nonlinearities. Nevertheless, the simpler transmission line model is useful because it enables us to train hardware dendrites using existing deep-learning frameworks.

Finally, we apply a conversion step to map trained parameters from TorchDendrite to hardware. TorchDendrite trains abstract leakage (decay) and space (conductance) parameters α and β , which are related to the linear cable equation's time and space constants, τ and λ , by Eq. (11):

$$\tau_n = \frac{dt}{1 - \alpha_n} \quad \lambda_n = \sqrt{\frac{\beta_n \tau_n}{dt}} \quad (11)$$

After calculating τ_n and λ_n , we apply Eqs. (6)–(10) to calculate transistor gate voltages. These are clipped to be within the circuit's operating range i.e., within $[0, V_{dd}]$, before being mapped to hardware.

Once circuit parameters have been calculated, we apply input/output scaling and assign neurons to cores. As mentioned before, SNNs trained with TorchDendrite use abstract parameters that do not represent realistic currents or voltages for analog hardware. In our analog hardware model, we scale inputs and outputs to correct for this. At the dendrite's digital-to-analog interface, weights generate input currents proportional to I_{scale} . At the analog-to-digital output, the recorded voltage is scaled by k_{out} before being sent to the soma. The value of k_{out} can be set by matching the output of the circuit model against the abstract TorchDendrite model for a single compartment. After configuring hardware parameters, we map neurons to hardware cores using a greedy mapping strategy.

To summarize, we train SNNs with digital and analog dendrites using TorchDendrite. While digital dendrites can be trained directly, analog dendrites additionally require the following conversion steps:

- 1) Convert dendrite parameters α and β to equivalent τ and λ values for every compartment using Eq. (11).
- 2) Set other hardware constants (Table I) and calculate transistor gate voltages $V_{leak,n}$, $V_{axial,n}$, and $V_{bias,n}$ using Eqs. (6), (7), and (10), respectively, and clipping to the range $[0, V_{dd}]$.
- 3) Experimentally find the output scaling factor k_{out} by comparing outputs against the TorchDendrite model.

V. MODELING AND SIMULATION

To simulate dendrites for design-space exploration, we have extended our architectural spiking hardware simulator, SANA-FE, to functionally model and predict the energy usage of digital and analog dendrites. SANA-FE, for Simulation of Advanced Neuromorphic Architectures for Fast Exploration [25], is shown in Fig. 3. SANA-FE requires an SNN application and a description of architectures defined using the spiking hardware elements described in Section III-A, including network tiles, cores, and neural-inspired pipeline hardware units. The simulator kernel models the SNN executing on the given architecture and outputs rapid and accurate energy and latency predictions, in addition to detailed hardware traces. To model

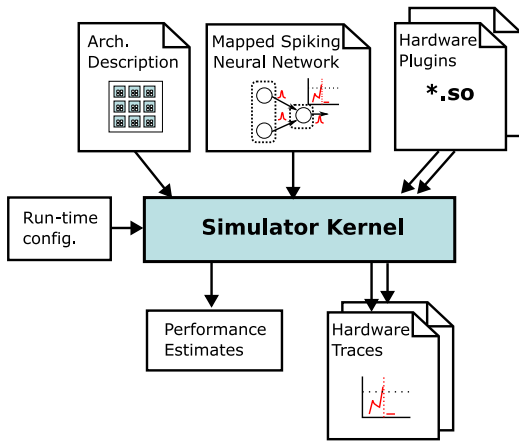


Fig. 3: Overview of SANA-FE with plug-ins.

the dendrite implementations described earlier, we use SANA-FE’s plug-in mechanism to create two new dendrite stage plug-ins with custom energy models.

A. Dendrite Plug-Ins

We have created plug-ins for modeling and simulating both digital and analog dendrites described in Section III-B. To simulate the digital dendrite, we implement the algorithms from [22], including Eq. (1), which calculates updates to compartment voltages. For the analog dendrite, we implemented a fixed-step Runge-Kutta solver to solve Eq. (3), which models the currents flowing through each transistor and can be solved for compartment voltages at discrete time-steps. We step the solver once per time-step for all compartments, and return the first compartment voltage as the dendritic output to the soma. Compartment voltages are also used for predicting energy usage, described next.

B. Energy Prediction

In addition to functional simulation, our plug-ins also dynamically simulate the per-time-step energy with models for both digital and analog implementations. For the digital dendrite, we estimate the dendrite’s energy consumption by multiplying the total number of arithmetic operations by a hardware-calibrated per-operation energy cost. For the analog circuit model, we simulate energy usage by modeling the energy consumed in each compartment and in the digital/analog conversion interfaces. We calculate per-compartment energy usage by numerically integrating each compartment’s power consumption, $V_{dd} \times (\max(0, i_n) + i_{bias,n})$, over the time-step duration, dt . Both input and bias currents, i_n and $i_{bias,n}$, are calculated when simulating the compartment’s dynamics (i.e., by the i_n and $k_{bias,n}$ terms in Eq. (3)). The dendrite’s total energy is then given by the sum of all compartment energies, plus approximate costs for digital-to-analog and analog-to-digital conversions based on existing low-power circuits [26], [27]. Specifically, we add 1 pJ to the total energy for digital-to-analog conversions after receiving a spike, and 100 fJ for analog-to-digital conversions every time-step.

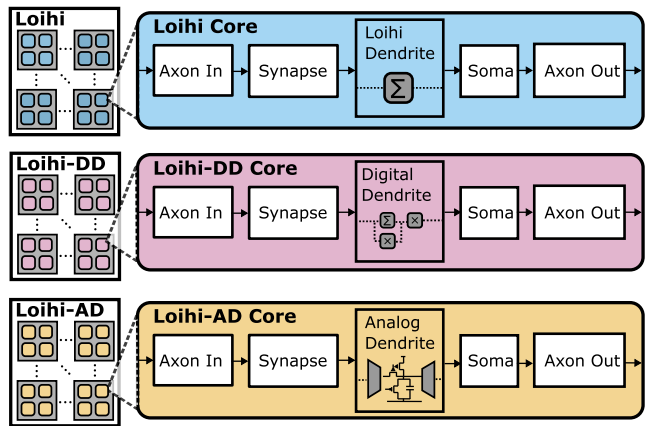


Fig. 4: Intel’s Loihi [6] and two Loihi-based architectures with advanced hardware dendrites.

VI. EXPERIMENTS AND RESULTS

In our experiments we used SANA-FE to model three spiking architectures (Fig. 4): Intel’s Loihi [6], and two hypothetical Loihi-based architectures with advanced dendrites called Loihi-DD and Loihi-AD. The hypothetical architectures are the same as Loihi except for their dendrite unit implementations; **Loihi-DD** replaces Loihi’s dendritic accumulators with the digital dendrites described in Section III-B, whereas **Loihi-AD** replaces accumulators with the previously described analog dendrite circuits (one circuit per neuron). For all architectures, SANA-FE was configured with previously calibrated per-operation energy costs [25]. While SANA-FE is also capable of predicting dynamic latency, in these experiments, the dendrite models require a fixed time-step latency of 10 μ s.

For our benchmark applications, we adopted the regression tasks for approximating \sqrt{x} and $\text{mish}(x)$ [28] from [22], and two categorization tasks for the Yin-Yang [29] and spiking Heidelberg digit (SHD) [30] data-sets. We use two different sets of SNNs with and without dendrite support. SNN applications used in our experiments are summarized Table II.

For the regression tasks (Fig. 5), we broadcast Poisson rate-encoded spike trains to the inputs of a hidden layer. SNNs without dendrites have 256 fully-connected (FC) hidden Leaky-Integrate-and-Fire (LIF) neurons (Fig. 5(a)), whereas SNNs with dendrites are composed out of 16 hidden LIF neurons with each neuron having a 16-compartment dendrite chain (Fig. 5(b)). Due to the smaller number of hidden neurons, SNNs with dendrites have significantly fewer total connections. In the regression SNNs, the hidden layer is connected to a single output neuron configured to accumulate and never spike. The output of the regression SNNs, i.e., an approximation of $f(x)$, is given by the membrane potential of the output neuron after 100 time-steps. For training, we applied the method described in Section IV. We trained using 500 randomly generated samples in the ranges $0 \geq x \geq 1$ for \sqrt{x} and $-3 \geq x \geq 1$ for $\text{mish}(x)$. After training, the output scaling constant k_{out} was measured to be 500.

The categorization SNNs are also composed out of one input layer as external traffic generator, one fully-connected

TABLE II: Three sets of benchmark SNNs with and without 16-compartment dendrite support.

Task	Neuron model	SNN neurons			Total weights	Spk. in /neuron	Spk. out /neuron
		In	Hidden	Out			
Regression [22]	LIF-only	1	256 FC	1	512	60	12
	Dend.+LIF	1	16 FC	1	272	760	20
Yin-Yang [29]	LIF-only	4	256 FC	3	1,792	206	5
	Dend.+LIF	4	16 FC	3	1,072	2,748	26
SHD [30]	LIF-only	70	256 FC-R	20	88,509	3,358	7
	Dend.+LIF	70	16 FC-R	20	11,929	7,818	21

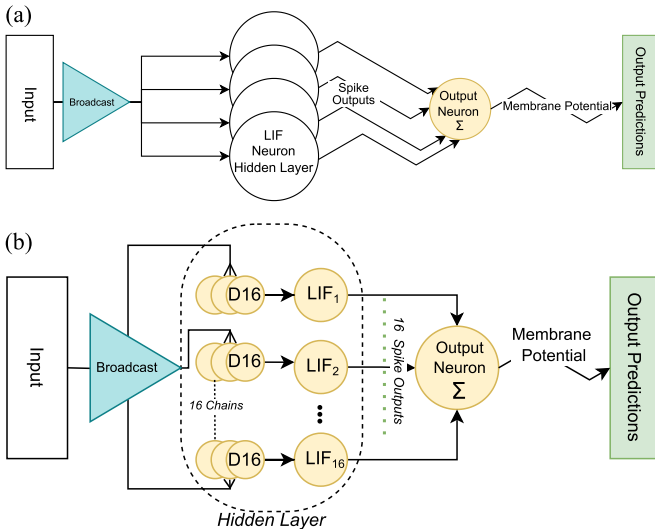


Fig. 5: Two regression SNNs, (a) only using LIF neurons and (b) using dendrites. Reproduced from [22].

hidden layer containing either 256 neurons or 16 dendrite-LIF chains, and one output layer with either integrating or LIF neurons. All layers are fully-connected, but after training and before simulation, edges with absolute weights smaller than 10^{-4} were pruned and did not receive spikes. The Yin-Yang SNNs have four Poisson-encoded input spike-trains and three spiking output LIF neurons, and the predicted class is given by the output neuron with the highest spike count. The spiking digits SNNs have 70 input spike-trains and 20 output neurons that accumulate without spiking, where the predicted class corresponds to the output neuron with the largest maximum potential for each inference test-case. Additionally, for the spiking digit SNNs, the hidden neurons are recurrently connected (R) to match the RSNNs used in [30]. For the Yin-Yang and spiking digits tasks, we trained the SNNs using the suggested configurations in [29] and [30], respectively, and applied the conversion method described in Section IV. However, to improve accuracy for the analog dendrites, we trained a model using a smaller step-size of $dt = 2.5\mu\text{s}$, i.e., four modeled steps per hardware time-step. Training with a finer-grained time resolution more accurately approximates the analog hardware behavior. However, the increased computation results in longer training times.

When mapping SNNs to hardware, we assumed that input spikes were injected by external traffic generators, and there-

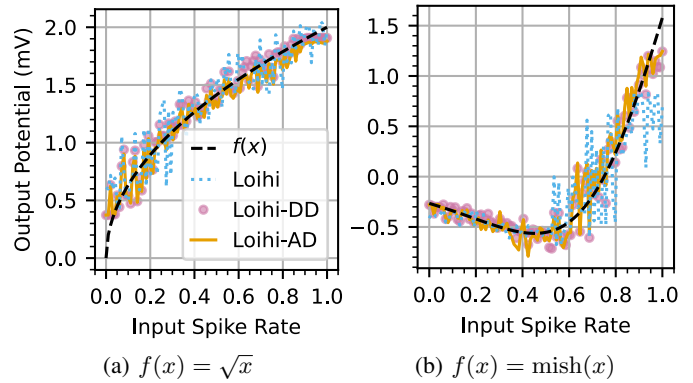


Fig. 6: Output of the regression SNNs from Fig 5.

TABLE III: Classification accuracy.

Task	Loihi	Loihi-DD	Loihi-AD
Yin-Yang [29]	86.0%	93.0%	80.0%
SHD [30]	76.2%	76.1%	70.3%

fore input neurons do not contribute to energy consumption. We mapped the hidden LIF and output LIF or integrator neurons to a single core and measured the mean ingress spikes (including injected spikes) processed per neuron and the average number of egress spikes generated by LIF neurons per test-case (see Table II). LIF neurons with dendrites generated up to $5\times$ more spikes than neurons without dendrites. However, because SNNs with dendrites contain only 1/16 of the hidden neurons compared to LIF-only SNNs, the total number of generated spikes was reduced by including dendrites.

Accuracy results for the regression tasks are shown in Fig. 6, and prediction accuracy results for the categorization tasks are shown in Table III. The LIF-only SNNs (Loihi) approximated the square root and mish functions with a mean absolute error of 0.14 mV and 0.19 mV, respectively. The SNNs using digital dendrites had a mean absolute error of 0.07 mV and 0.11 mV, and SNNs using analog dendrites had a mean absolute error of 0.08 mV and 0.12 mV. For the regression tasks, the SNNs with digital and analog dendrites were slightly more accurate than the larger SNN without dendrites. The SNN with analog dendrites is almost as accurate as the SNN with digital dendrites. The SNNs with dendrites perform comparably to SNNs without dendrites for both categorization tasks, although there is a drop in accuracy between analog and digital dendrites. The conversion step introduces some small error due to nonlinearities in the analog dendrite circuit not accounted for in the parameter conversion. Directly training the analog dendrite without conversion could potentially remove this error.

Fig. 7 shows the energy consumption of the three spiking platforms. The design with digital dendrites (Loihi-DD) on average consumed 44% of the dynamic energy required by the Loihi baseline. This is due to the smaller number of neurons used by the SNN with dendrites, leading to reduced neuron processing overheads and fewer spikes generated and sent to the network. The SNNs with dendrites have 1/16 of the hidden neurons compared to LIF-only SNNs and overall had

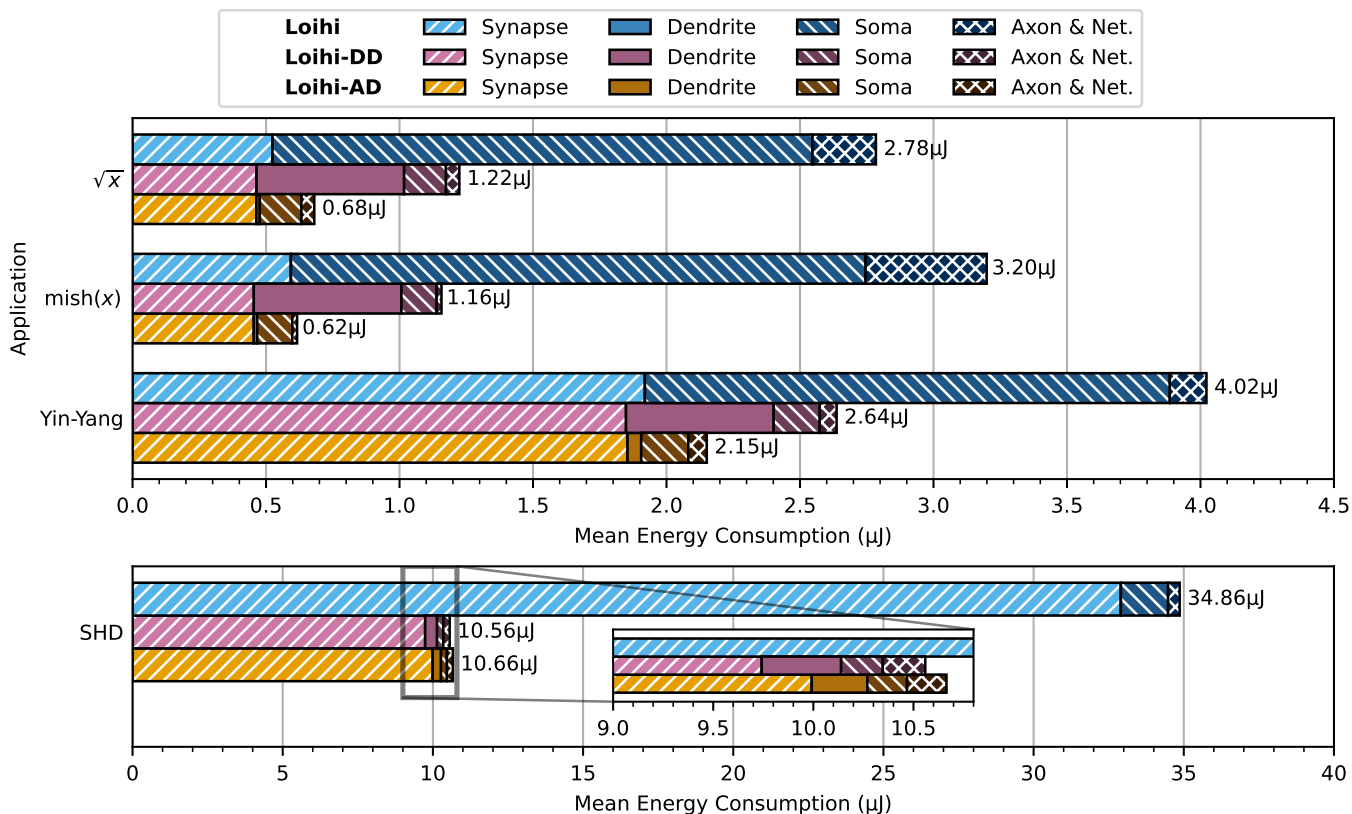


Fig. 7: Energy usage for SNNs executing on Loihi-based architectures with digital (Loihi-DD) and analog (Loihi-AD) dendrites.

78% fewer generated spikes, resulting in mean energy savings across all benchmarks of 91% in the soma units and 68% and 27% in the axon/network and synapse units, respectively. The small energy cost of Loihi’s dendritic accumulator could not be measured separately. As such, the dendrite unit cost is not visible for Loihi and is included in the soma costs.

The design with analog dendrites (Loihi-AD) on average consumed 32% of the energy used by Loihi and 73% of the energy consumed by Loihi-DD. Energy savings over Loihi-DD come mainly from reduced dendrite processing costs. The dendrite circuits’ dynamic energy consumption in Loihi-AD was on average 79% smaller than the digital dendrite energy and only 2% of the total energy consumed by the chip, highlighting the potential for energy efficiency of analog implementations of neuromorphic building blocks. Results also show that the analog dendrite’s efficiency is application dependent, achieving up to 98% energy improvement compared to digital dendrites for the regression tasks but only a 30% improvement for spiking digits. Since DAC overheads dominate energy usage, accounting for over 99% of the dendrite’s energy usage for spiking digits, the analog dendrites in Loihi-AD perform better at tasks with fewer spikes processed per neuron, such as the regression applications. However, in an architecture that does not require digital weights to be converted by a multi-bit DAC, e.g., by implementing analog synaptic memory, the dendrite’s energy efficiency could be improved further.

Other trade-offs should be considered in addition to energy efficiency when comparing dendrite implementations. In general, as shown in our results, more expressive dendrites reduce the number of neurons required, where we demonstrated that 16-neuron SNNs using dendrites performed comparably to 256-neuron SNNs without dendrites. At the same time, custom dendrites require additional dedicated circuitry, taking up area on the chip that may not always be fully utilized. Digital dendrites require custom logic, which is shared between all neurons mapped to the core. To estimate hardware requirements, we synthesized custom Verilog implementations of a 24-bit accumulator (Loihi) and the 16-compartment digital dendrite (Loihi-DD) using Yosys [31]. Compared to a simple accumulator requiring 146 standard cells, a digital dendrite uses 70,261 cells. By contrast, analog dendrites require only a small amount of dedicated hardware compared to digital implementations (48 transistors for a 16-compartment dendrite, plus overhead for ADC/DAC circuits). However, in comparison to digital implementations that allow for time sharing, analog dendrites require a physical circuit for every mapped neuron, limiting scalability. Furthermore, SNN accuracy may be degraded by noise and device variations that are currently not modeled in our setup. Lastly, we found that training SNNs with dendrites requires longer training times compared to SNNs using only LIF neurons. For the tasks used in this paper, training SNNs with dendrites took at least 50% longer compared to training LIF-only SNNs. When training SNNs

with analog dendrites, training took over 200% longer than LIF-only SNNs due to the smaller time-step used.

Our approach for modeling hardware dendrites in SANA-FE supports rapid, early design-space exploration. To evaluate the scalability of such an approach, we duplicated the \sqrt{x} SNN across 130,000 neurons (the maximum supported by Loihi), using 128 cores and over two million dendritic compartments. We measured SANA-FE's run-time executing 1000 time-steps with inputs spiking every time-step. Measurements were taken on an Intel i7-11850H processor. SANA-FE simulated 130,000 digital dendrites (≈ 2 million compartments) at a rate of 14.1 $\mu\text{s/s}$ and with a throughput of 1.41 steps/s. SANA-FE simulated the same number of analog dendrites at a rate of 10.8 $\mu\text{s/s}$ and with a throughput of 1.08 steps/s.

VII. SUMMARY, CONCLUSIONS AND FUTURE WORK

In this paper, we explored the use of digital and analog hardware dendrites within large-scale spiking hardware platforms. We showed how to convert trained parameters for both circuits for use in SNN applications and used our architectural simulator, SANA-FE, to implement plug-ins for both digital and analog dendrites with dynamic energy prediction. Accuracy and energy results for four applications, with and without dendrites, show that a design with advanced hardware dendrites may lead to significant energy savings. Finally, we showed that our approach for modeling and simulation of large-scale dendritic architectures supports rapid design-space exploration and is capable of scaling to large applications.

In future work, we plan to explore the impact of random noise and device variation on analog dendrite accuracy and efficiency. We also plan to explore techniques to improve dendrite training accuracy, e.g., by using trainable delays [32], [33], and to reduce conversion error by directly training analog dendrites, instead of converting from a simpler RC model. Furthermore, we plan to investigate architectures that include other analog neuromorphic elements or complete analog compute cores within an overall large-scale digital back-end, e.g., using memristive crossbar arrays for digital-to-analog synapses and analog LIF soma circuits.

ACKNOWLEDGMENTS

This article has been authored by an employee of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employee owns all right, title and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan <https://www.energy.gov/downloads/doe-public-access-plan>. SAND2025-07998D

REFERENCES

- [1] J. Hasler and B. Marr, "Finding a roadmap to achieve large neuromorphic hardware systems," *Front. Neurosci.*, vol. 7, no. 118, pp. 1–29, 2013.
- [2] K. Boahen, "Dendrocentric learning for synthetic intelligence," *Nature*, vol. 612, no. 7938, pp. 43–50, 2022.
- [3] S. G. Cardwell and F. S. Chance, "Dendritic computation for neuromorphic applications," in *Int. Conf. Neuromorphic Syst. (ICONS)*, 2023.
- [4] S. Nease *et al.*, "Modeling and implementation of voltage-mode CMOS dendrites on a reconfigurable analog platform," *IEEE Trans. Biomedical Circuits Syst. (TBioCAS)*, vol. 6, no. 1, pp. 76–84, 2011.
- [5] S. George *et al.*, "Low power dendritic computation for wordspotting," *J. Low Power Electron. Appl. (JLPEA)*, vol. 3, no. 2, pp. 73–98, 2013.
- [6] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
- [7] J. Kaiser *et al.*, "Emulating dendritic computing paradigms on analog neuromorphic hardware," *Neuroscience*, vol. 489, pp. 290–300, 2022.
- [8] J. Boyle *et al.*, "SANA-FE: Simulating advanced neuromorphic architectures for fast exploration," *IEEE Trans. on Comp.-Aided Des. of Int. Circuits and Syst. (TCAD)*, vol. 44, no. 8, pp. 3165–3178, 2025.
- [9] SANA-FE. [Online]. Available: github.com/SLAM-Lab/SANA-FE
- [10] J. V. Arthur and K. Boahen, "Recurrently connected silicon neurons with active dendrites for one-shot learning," in *Int. Joint Conf. Neural Net. (IJCNN)*, 2004.
- [11] J. Schemmel *et al.*, "An accelerated analog neuromorphic hardware system emulating NMDA- and calcium-based non-linear dendrites," in *Int. Joint Conf. Neural Net. (IJCNN)*, 2017.
- [12] M. Payvand *et al.*, "Dendritic computation through exploiting resistive memory as both delays and weights," in *Int. Conf. Neuromorphic Syst. (ICONS)*, 2023.
- [13] S. Höppner *et al.*, "The SpiNNaker 2 processing element architecture for hybrid digital neuromorphic computing," *arXiv preprint arXiv:2103.08392*, 2021.
- [14] H. Zheng *et al.*, "Temporal dendritic heterogeneity incorporated with spiking neural networks for learning multi-timescale dynamics," *Nat. Commun.*, vol. 15, no. 1, p. 277, 2024.
- [15] S. D'Agostino *et al.*, "DenRAM: neuromorphic dendritic architecture with RRAM for efficient temporal processing with delays," *Nat. Commun.*, vol. 15, no. 1, p. 3446, 2024.
- [16] M. L. Hines and N. T. Carnevale, "NEURON: a tool for neuroscientists," *The Neuroscientist*, vol. 7, no. 2, pp. 123–135, 2001.
- [17] F. Rothganger *et al.*, "N2A: a computational tool for modeling from neurons to algorithms," *Front. Neural Circuits*, vol. 8, p. 1, 2014.
- [18] M. Pagkalos, S. Chavlis, and P. Poirazi, "Introducing the dendrify framework for incorporating dendrites to spiking neural networks," *Nat. Commun.*, vol. 14, no. 1, p. 131, 2023.
- [19] C.-G. Pehle and J. Egholm Pedersen, "Norse—a deep learning library for spiking neural networks," *Zenodo*, 2021.
- [20] DendNet. [Online]. Available: github.com/sandialabs/torchdendrite
- [21] C.-C. Hsu, A. C. Parker, and J. Joshi, "Dendritic computations, dendritic spiking and dendritic plasticity in nanoelectronic neurons," in *Int. Midwest Symp. Circuits Syst. (MWSCAS)*, 2010.
- [22] M. Plagge, S. G. Cardwell, and F. S. Chance, "Expressive dendrites in spiking networks," in *Neur.-Insp. Comput. Elements Conf. (NICE)*, 2024.
- [23] J. Lohoff, J. Finkbeiner, and E. Neftci, "SNNAX—spiking neural networks in JAX," *arXiv preprint arXiv:2409.02842*, 2024.
- [24] J. K. Eshraghian *et al.*, "Training spiking neural networks using lessons from deep learning," *Proc. IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.
- [25] J. Boyle *et al.*, "Performance and energy simulation of spiking neuromorphic architectures for fast exploration," in *Int. Conf. Neuromorphic Syst. (ICONS)*, 2023.
- [26] R. Rubino, P. S. Crovetti, and O. Aiello, "Design of relaxation digital-to-analog converters for internet of things applications in 40nm CMOS," in *Asia Pacific Conf. Circuits Syst. (APCCAS)*, 2019.
- [27] X. Tang *et al.*, "Low-power SAR ADC design: Overview and survey of state-of-the-art techniques," *IEEE Trans. Circuits Syst. I: Regular Papers (TCAS-I)*, vol. 69, no. 6, pp. 2249–2262, 2022.
- [28] D. Misra, "Mish: A self regularized non-monotonic activation function," *arXiv preprint arXiv:1908.08681*, 2019.
- [29] L. Kriener, J. Göltz, and M. A. Petrovici, "The Yin-Yang dataset," in *Neur.-Insp. Comput. Elements Conf. (NICE)*, 2022.
- [30] B. Cramer *et al.*, "The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks," *IEEE Trans. Neural Net. Learning Sys. (TNNLS)*, vol. 33, no. 7, pp. 2744–2757, 2020.
- [31] C. Wolf, J. Glaser, and J. Kepler, "Yosys - a free Verilog synthesis suite," in *Austrian Workshop Microelectronics (Austrochip)*, 2013.
- [32] A. Patiño-Saucedo *et al.*, "Co-optimized training of models with synaptic delays for digital neuromorphic accelerators," in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2024.
- [33] I. Hammouamri, I. Khalfaoui-Hassani, and T. Masquelier, "Learning delays in spiking neural networks using dilated convolutions with learnable spacings," *arXiv preprint arXiv:2306.17670*, 2023.