Towards Efficient Deployment of Hybrid SNNs on Neuromorphic and Edge AI Hardware

James Seekings, Peyton Chandarana, Mahsa Ardakani, MohammadReza Mohammadi, and Ramtin Zand
Department of Computer Science and Engineering, University of South Carolina, Columbia, USA
seekingj@email.sc.edu, psc@email.sc.edu, mahsam@email.sc.edu, mohammm@email.sc.edu, ramtin@cse.sc.edu

***Abstract*—This paper explores the synergistic potential of neuromorphic and edge computing to create a versatile machine learning (ML) system tailored for processing data captured by dynamic vision sensors. We construct and train hybrid models, blending spiking neural networks (SNNs) and artificial neural networks (ANNs) using PyTorch and Lava frameworks. Our hybrid architecture integrates an SNN for temporal feature extraction and an ANN for classification. We delve into the challenges of deploying such hybrid structures on hardware. Specifically, we deploy individual components on Intel's Neuromorphic Processor Loihi (for SNN) and Jetson Nano (for ANN). We also propose an accumulator circuit to transfer data from the spiking to the non-spiking domain. Furthermore, we conduct comprehensive performance analyses of hybrid SNN-ANN models on a heterogeneous system of neuromorphic and edge AI hardware, evaluating accuracy, latency, power, and energy consumption. Our findings demonstrate that the hybrid spiking networks surpass the baseline ANN model across all metrics and outperform the baseline SNN model in accuracy and latency.**

***Index Terms*—Spiking neural network (SNN), edge computing, neuromorphic computing, edge AI accelerators, and heterogenous systems.**

I. Introduction

Spiking Neural Networks (SNNs) [1], [2] are an emerging technology aimed at creating biologically-inspired neural networks for low-power and high-performance computation. They utilize neurons modeled after the brain, enabling them to learn over time and excel at extracting temporal information from event-based data [3]–[5]. In contrast, artificial neural networks (ANNs) like convolutional neural networks (CNNs) are proficient at extracting spatial information but do not handle temporal information well [6]. Recurrent neural networks (RNNs) have gained popularity for their ability to handle temporal information, but they do not offer significant improvements in terms of power or latency [7].

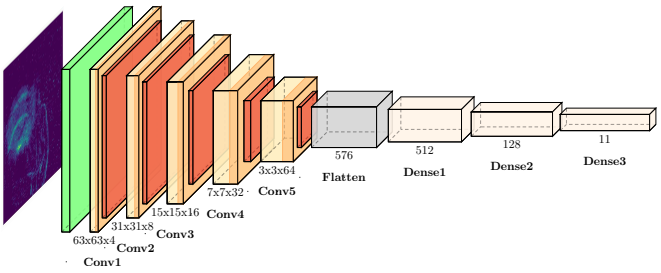
SNNs are being explored as a promising alternative for conventional ANNs due to their low-power computing capabilities, yet when deployed on existing neuromorphic hardware, they often underperform in terms of classification accuracy [8]–[10]. One solution to benefit from the advantage of both SNN and ANN models is to fuse them to create more robust and versatile neural network models capable of addressing complex tasks, including pattern recognition in time-series data and understanding dynamic systems in real-time applications. However, integrating SNNs and ANNs in a single system presents challenges such as developing efficient training algorithms to train across the two domains and optimizing the use of computational resources. Despite these challenges, exploring the viability of combining ANNs and SNNs into one system could yield promising results to advance neural networks capabilities.

In 2021, Kugele et al. [11] proposed a hybrid SNN-ANN architecture with a custom simulator to compile and train a hybrid neural network. Their proposed model consists of an SNN backbone for extracting temporal information and an ANN head for classification. Inspired by [11], Wu et al. [12] investigates appending dense layers to SNN networks to improve accuracy on CIFAR-10 [13]. In [14], a hybrid SNN-ANN model is utilized to process the data captured by dynamic vision sensor (DVS) [15], [16]. Instead of using the SNN for the backbone, other works such as Muramatsu et. al. [17] explore using the ANN as a backbone with an SNN head for classification on the MNIST [18] and CIFAR-10 datasets. Additionally, Muramatsu et. al. performed multiple experiments by modifying the ratios of ANN to SNN layers concluding that models with more ANN layers typically achieve better accuracy. Beyond the domain of image classification, a few papers [19], [20] have also explored the use of hybrid networks in object detection tasks. This is done by combining an SNN backbone with a single-shot detector head and using a surrogate gradient to train the networks.

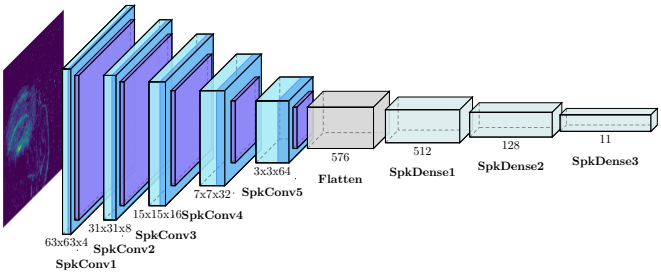
Herein, our work provides a deeper investigation of hybrid SNN-ANN models by offering a unified training mechanism that operates across the SNN and ANN domains. Additionally, we undertake a more extensive performance analysis for different architectures of the hybrid network. The main contributions of our paper compared to the previous works are:

- Developing a unified backpropagation-based training mechanism for hybrid spiking and non-spiking architectures using PyTorch and SLAYER [21] as part of the LAVA neuromorphic computing library [22].
- Investigating the hardware deployment challenges of hybrid architectures. This paper is one of the pioneers in attempting the real hardware implementation of hybrid SNN-ANN models.
- Providing comprehensive performance analyses of hybrid SNN-ANN models deployed on a heterogenous system of neuromorphic and edge AI hardware.

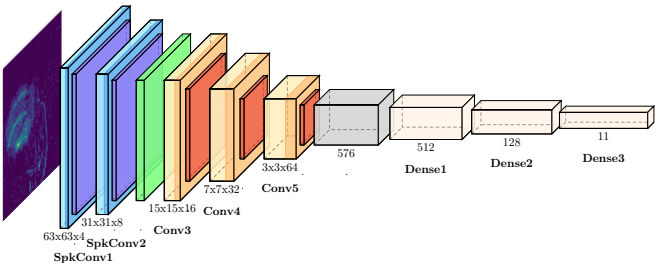
The remainder of the paper is organized as follows. Section II introduces the proposed hybrid SNN-ANN architectures and



(a) ANN Architecture



(b) SNN Architecture



(c) Example S_2A_3 Hybrid Architecture

Fig. 1: The architecture of the CNN models investigated herein to process the data captured by the DVS camera.

their training mechanism. Section III demonstrates the hardware deployment of our hybrid architecture on a heterogeneous system of neuromorphic hardware and edge AI accelerators. The experiments performed and the results obtained are discussed in Section IV . Finally, Section V concludes the paper.

II. Proposed Hybrid SNN-ANN Architecture

A. Hybrid Architecture

Here, we develop a representative CNN architecture with five convolution layers and three dense layers to test the effectiveness of integrating spiking and non-spiking components. Figure 1a shows the baseline ANN architecture using non-spiking convolution and dense blocks that employ ReLU activations and max pooling operations. Additionally, an accumulate operation is included at the front of the network, represented by the green layer, which collapses the temporal dimension from the input data so that the ANN can process it. Figure 1b shows the baseline SNN architecture consisting of Spike Convolutions (SpkConv) and Spike Dense (SpkDense) blocks. These blocks use Current-Based Leaky Integrate and Fire (CUBA-LIF) neurons as activations and spike pooling operations.

A hybrid network is generated by replacing the Conv layers from the ANN with SpkConv layers from the SNN. Layers are replaced in order starting from the beginning of the model and moving deeper. The accumulate operation is then placed between the spiking and non-spiking layers to remove the temporal dimension as the data is passed to ANN, and the dense layers are always implemented via non-spiking blocks. Figure 1c shows an example of a hybrid network with two SpkConv layers and three regular Conv layers, labeled S_2A_3. Table I provides further details of the model architectures investigated in this paper.

B. Accumulator

Before event-based data can be passed to ANN, the temporal dimension must be collapsed. To accomplish this, methods such as those used in [11] involve implementing an accumulator that sums spikes over small time intervals to generate multiple outputs to the ANN. Each output is run through the first layer of the ANN and then concatenated together at the second layer. Our accumulator differs in that, after summation, the outputs are concatenated together before being sent to ANN. The concatenation occurs across the channel dimension, causing it to expand with the size of the temporal dimension.

Herein, the period over which the accumulator sums spikes is referred to as the *accumulate interval*. A large interval sums up many spikes at once, reducing output size at the cost of temporal resolution. On the other hand, smaller intervals better retain temporal resolution but lead to increased model size. In our experiments, we treat the accumulate interval as a hyperparameter to investigate its effect on model performance, which we discuss in Section IV. A model's accumulate interval is denoted by $I = (5/10/25)$, as shown in Table I.

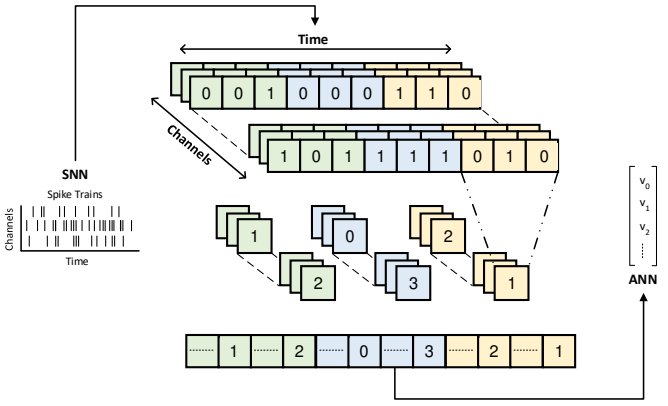


Fig. 2: An example of accumulate operation on a 2-dimensional data with 9 timesteps (T=9) and an accumulate interval of 3 (I=3).

Figure 2 shows the accumulate operation performed on 2-dimensional data although it can be generalized for 4-dimensional input. First, the input data is separated into groups of size I along the temporal dimension, represented by different colors. The groups are then summed together resulting in channel-wide vectors. After summation, these vectors are

TABLE I: Network Architectures. The spiking and non-spiking layers are indicated by "Spk" and "non-Spk", respectively. Accumulate interval is denoted with $I = (5/10/25)$.

	Convolutions		Linear		Parameters			
Model	**Spk**	**Non-Spk**	**Spk**	**Non-Spk**	$I = 5$	$I = 10$	$I = 25$	**Loihi Cores**
ANN	0	5	0	3	223,991	223,631	223,415	0
S_1A_4	1	4	0	3	225,943	224,503	223,639	16
S_2A_3	2	3	0	3	233,727	227,967	224,511	32
S_3A_2	3	2	0	3	264,839	241,799	227,975	36
S_4A_1	4	1	0	3	389,263	297,103	241,807	38
S_5A_0	5	0	0	3	3,041,431	1,566,871	683,135	42
SNN	5	0	3	0	387,229	387,229	387,229	58

concatenated together with the earlier elements appearing first, preserving temporal order. The output of the accumulator is then sent to ANN for further processing.

A mathematical representation of the accumulate operation can be seen in Eq. 1, where S is a $C \times T$ matrix representing spiking input. C is the number of channels and T is the number of time steps. The output is a vector A with length CT/I, where I is the accumulate interval.

$$A_j = \sum_{k=0}^{I-1} S_{(j \ mod \ C),(I\lfloor \frac{j}{C} \rfloor + k)} \tag{1}$$

Each output A_j is defined as the summation of S at channel $j \ mod \ C$ over I timesteps. Once j exceeds C, the summation resets to the first channel through the modulus operation. At this point, $\lfloor \frac{j}{C} \rfloor$ equals 1 which shifts the columns for summation over by I. This repeats for every C indices until the final index.

One challenge of this method is that the channel dimension expands at a rate of T/I to accommodate the shrinking temporal dimension. Reducing the channel count back down to what it was before the accumulate operation can be done with a simple convolutional layer. However, this is not possible in models such as S_5A_0 which does not have a convolution layer following the accumulate operation. As such, those models experience a substantial increase in parameter count at smaller accumulate intervals, as seen in Table I.

C. Training

Our hybrid model was built in PyTorch using the LAVA framework [22] for spiking components. The LAVA library contains a SLAYER-based training algorithm for SNNs which saves the network's previous states to be used during the calculation of gradients via a temporal credit assignment policy [21] and Back Propagation Through Time (BPTT) [23]. The training algorithms for ANN and SNN build computational graphs for calculating gradients and the graphs are combined automatically, allowing the hybrid model to train as a single unified network instead of being trained separately. However, the accumulate operation is non-differentiable, requiring a custom backward pass to be implemented.

In the backward pass of the accumulator, we are faced with an inverse of the forward pass challenge encountered previously. Here, 3-dimensional gradients are received from the ANN while the SNN expects 4-dimensional gradients, thus the temporal dimension needs to be reintroduced or expanded from the ANN domain. This is done by repeating the gradients in-place I times and then reshaping the data to four dimensions. Through this process, spikes that were initially summed together share the same gradient. A mathematical representation following the logic of the forward pass can be seen in Eq. 2.

$$S_{i,j} = A_{C\lfloor \frac{j}{I} \rfloor + i} \tag{2}$$

III. Deployment Methodology

Figure 3 shows the end-to-end system in software and hardware starting from the training phase using our unified training pipeline to deploying the networks on their respective hardware. As shown, the spiking and non-spiking components of our proposed hybrid network have differing hardware constraints that limit deployment options. Due to their asynchronous event-based nature, SNNs cannot be run on GPUs or CPUs without simulation which increases latency and consumes more power. Neuromorphic hardware such as Intel's Loihi chip [24] is specially designed for running spiking models by emulating biologically inspired neuron dynamics in hardware. However, these chips do not support all of the operations in the ANN models making them unfit to run ANNs. In recent years, there has been various research on the deployment of ANN models on edge AI accelerators [25]–[27]. As shown in Fig. 3, we chose to deploy our hybrid spiking model through a distributed system combining a Loihi chip and a Jetson Nano [28]. These devices were profiled separately to isolate their specific impact on the overall system.

1) Deployment of Non-Spiking Components on Edge AI Accelerator: The NVIDIA Jetson Nano is a development board tailored for ML applications. It utilizes the Tegra X1 System on Chip (SoC), which includes a quad-core ARM Cortex A57 CPU clocked at 1.43 GHz. Additionally, the board features four discrete processing clusters, each with 32 CUDA cores, totaling 128 CUDA cores based on the Maxwell architecture. Equipped with 4 GB of RAM, the Jetson Nano provides a robust computational platform for ML acceleration at the edge.

The Jetson Nano operates in two distinct power configurations: a low-power 5 W mode and a higher-performance Max-N mode, both selectable via a software interface. In the 5 W mode, the device restricts itself to utilizing only two ARM A57 cores at a reduced frequency of 0.9 GHz, and

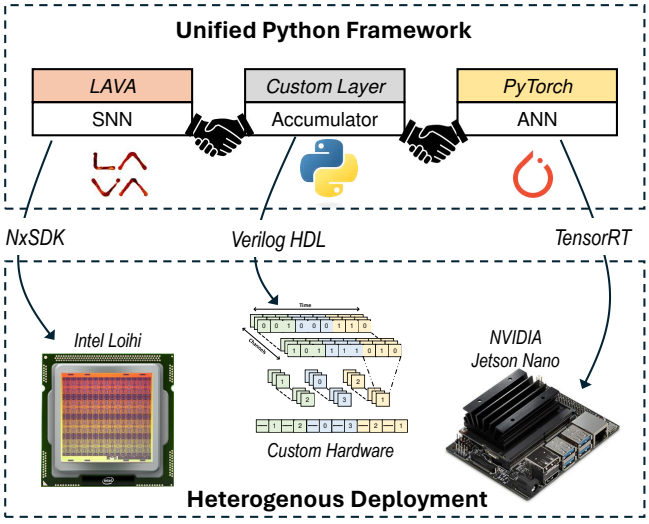


Fig. 3: The proposed unified python framework for training the hybrid SNN-ANN models and the corresponding deployment on a heterogenous system of neuromorphic hardware and edge AI accelerators.

the GPU operates at a reduced clock speed of 0.64 GHz. Conversely, the Max-N mode enables all four ARM A57 cores, running them at an increased frequency of 1.5 GHz, while the GPU operates at a speed of 0.92 GHz. This dual-mode functionality allows developers to balance between power efficiency and computational performance based on the needs of their applications.

Although Jetson Nano is equipped with CUDA capabilities, it primarily utilizes NVIDIA TensorRT [29] to optimize and accelerate deep learning models through quantization and other optimization techniques, thereby enhancing performance. Models can be exported to TensorRT through the Open Neural Network Exchange (ONNX) [30] library.

The proposed hybrid networks were recreated in TensorFlow [31] with the spiking components removed and the input layers adjusted. These partial models were then converted to ONNX and exported to TensorRT for deployment on the Jetson Nano. For inference latency, we calculate the time required for 100 inferences and then determine the average inference time for a single input sample. Input, CPU, and GPU power dissipation were recorded through the Jetson Nano's built-in sensors while running each model for three minutes. The *tegrastats* utility is used to read the sensors automatically [28].

2) Deployment of Spiking Components on Neuromorphic Hardware: SNN simulators such as INIsim [32] or Brian 2 [33] can be used to run SNNs on the CPU through a software abstraction of neural dynamics consuming considerable power and increasing execution latency. Devices that instead emulate neural dynamics, allowing for efficient execution of spiking networks, are referred to as neuromorphic hardware platforms. In our paper, we utilize Intel's Loihi [24], to deploy the spiking components of our hybrid network.

The Loihi chip departs from the typical von Neumann architecture to more closely replicate the neural dynamics of the brain. It contains a network of neurons and synapses that communicate asynchronously through discrete spike events for more efficient computation. Loihi is organized into 128 programmable cores, each containing 1024 neurons connected by a total of 128 million synapses. While not yet commercially available, access to the Loihi chip is provided to us by Intel Labs through membership in the Intel Neuromorphic Research Community (INRC).

While the LAVA framework is utilized in this paper to train the hybrid and spiking networks, we could not measure the power reliably using the power measurement tools available in LAVA at the time of conducting this research. Consequently, we opted for deploying the models on Loihi 1 and employed the NengoLoihi toolchain [34] for deployment and power measurements. NengoLoihi facilitates the conversion of CNNs into SNN architectures through a mapping algorithm, which maps the weights and activations of the CNN onto an equivalent SNN in Intel's NxSDK framework.

These initial CNNs were first developed in PyTorch and then mapped to the spiking domain of each hybrid network. Once converted into SNNs through NengoLoihi, each network is partitioned layer-by-layer across Loihi's neuro-cores. After which, neurons and synapses are mapped together between the chip and network. Once partitioned and mapped, the SNN is run directly on the chip. The power dissipation was recorded through a profiling toolkit in NengoLoihi which also reports the number of Loihi cores allocated to the network. The last column of Table I provides the number of Loihi cores for all the hybrid and SNN architectures studied herein. As listed, one Loihi chip is sufficient to support the implementation of our hybrid SNN-ANN and SNN models. The latency is measured via the spike propagation delay defined as the time between an image being exposed to the network and the output of spikes from the final layer.

IV. Experiments and Results

A. Dataset

Throughout our experiments, we use the DvsGesture dataset which includes 11 hand gestures recorded from 29 subjects under 3 illumination conditions [35]. DVS events are defined by their type (on/off), pixel location (x,y), and a timestamp. To transform the raw DVS data into usable training data, all of the events that occurred in a 10ms time frame were compiled into a 128×128 image. We then took 50 consecutive images to form a single sample of shape (2, 128, 128, 50), representing 500ms of activity. The raw data includes the times when certain gestures are made, which is then used to automatically label the samples. This generates 14,672 training samples and 3,793 testing samples.

B. Accuracy

Figure 4 provides a comparison of the accuracies between baseline ANN and SNN, as well as the hybrid networks. To determine the impact on model performance, the models are evaluated with three distinct accumulation intervals: I=5,

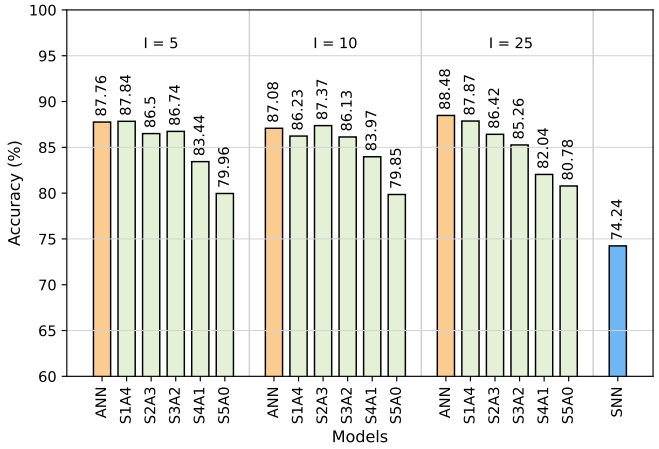


Fig. 4: Comparative analysis of accuracy for various ANN, SNN, and hybrid SNN-ANN models.

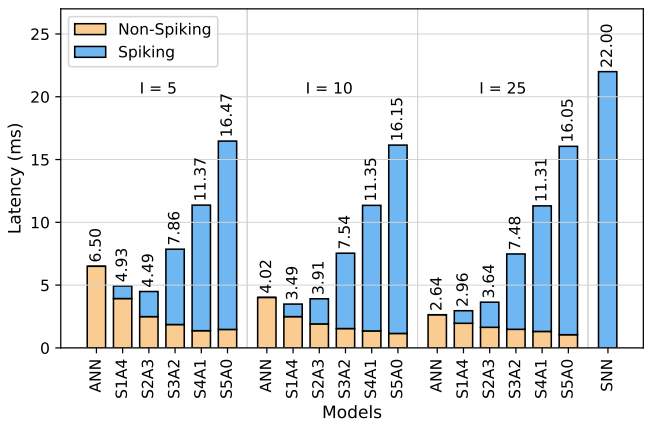


Fig. 5: Inference latency for various ANN, SNN, and hybrid SNN-ANN models.

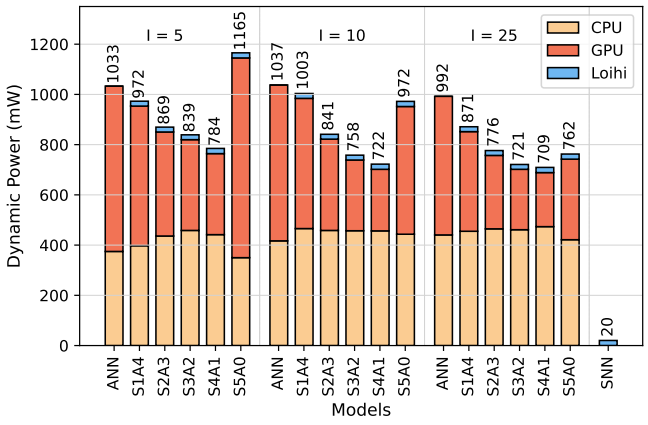


Fig. 6: Dynamic Power Comparison for CPU, GPU, and Loihi for various ANN, SNN, and hybrid SNN-ANN models.

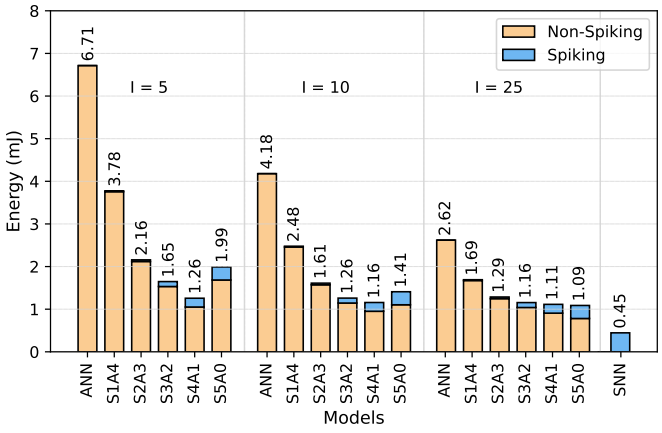


Fig. 7: Energy Consumption Comparison for various ANN, SNN, and hybrid SNN-ANN models.

I=10, and I=25. The data prominently shows that ANNs maintain high accuracy levels across all intervals, achieving their highest at 88.48% with I=25. On the other hand, the SNN baseline significantly underperforms at 74.24%. Analyzing the hybrid network results, we see that the continued addition of SpkConv layers leads to accuracy loss at all intervals, mirroring the SNN performance. However, the inclusion of a few SpkConv layers does not have a significant impact on accuracy and in select cases, such as the S_2A_3 (I=10) configuration, can even lead to slight increases in accuracy.

C. Latency

Figure 5 shows the latency of each model, separated for spiking and non-spiking components. The ANN baseline shows 3.28×, 5.47×, and 8.33× less latency compared to the SNN baseline for I=5, I=10, and I=15, respectively. As shown in the figure, the accumulate interval has a considerable effect on latency, ranging from 6.5 ms to 2.64 ms as it increases. Similar to the accuracy results, it can be observed that the continued addition of SpkConv layers in hybrid networks increases latency to eventually match the SNN's performance. However, a small decrease in latency can be seen when only a few SpkConv layers are present with intervals I=5, and I=10.

D. Power Dissipation

Figure 6 shows a comparison of power consumption among the SNN, ANN, and various hybrid SNN-ANN architectures. The graph illustrates that the ANN baseline, run on CPU and GPU, exhibits significantly higher dynamic power consumption compared to the SNN baseline deployed on Loihi. Adding SpkConv layers reduces power dissipation across the board, although even a very small ANN still consumes vastly more power than a large SNN. A substantial increase in power consumption is observed in configuration S_5A_0 across all intervals, but this is an outlier caused due to the model's increased parameter count as shown in Table I. This occurred because the accumulator expands the channel dimension of the data, which is normally reduced back down by consecutive Conv layers. However, the accumulator in configuration S_5A_0 is followed by a fully connected layer which does not reduce

TABLE II: Accumulator Performance Results. Recorded using Design Compiler.

Model	$I = 5$			$I = 10$			$I = 25$		
	Power (mW)	**Latency (ms)**	**Energy (mJ)**	**Power (mW)**	**Latency (ms)**	**Energy (mJ)**	**Power (mW)**	**Latency (ms)**	**Energy (mJ)**
ANN	0.294	1.28e-2	4.95e-5	0.433	2.56e-2	1.46e-4	0.528	6.40e-2	4.70e-4
S_1A_4	0.294	6.25e-3	2.42e-5	0.433	1.25e-2	7.14e-5	0.528	3.13e-2	2.29e-2
S_2A_3	0.294	3.05e-3	1.18e-5	0.433	6.10e-3	3.48e-5	0.528	1.53e-2	1.12e-4
S_3A_2	0.294	1.45e-3	5.61e-6	0.433	2.90e-3	1.66e-5	0.528	7.25e-3	5.32e-5
S_4A_1	0.294	6.5e-4	2.52e-6	0.433	1.30e-3	7.42e-6	0.528	3.25e-3	2.38e-5
S_5A_0	0.294	2.5e-4	9.67e-7	0.433	5.0e-4	2.86e-6	0.528	1.25e-3	9.17e-6

the overall dimensionality. Overall, our results exhibit that replacing non-spiking layers with spiking ones in a hybrid SNN-ANN architecture can generally lead to decreased power dissipation. Similar to latency, the accumulate operation does not consume a substantial amount of power, being less than 1 mW in all cases, which is further discussed in Section IV-F.

E. Energy Consumption

Figure 7 provides a comparison between the ANN and SNN baselines and the hybrid SNN-ANN models. The results demonstrate that SNNs exhibit significantly lower energy consumption for computational tasks compared to conventional ANNs, highlighting the potential benefits of incorporating spiking layers to reduce energy consumption. It shows that the incremental addition of SpkConv layers significantly decreases energy consumption compared to ANN models. However, there is a similar increase in energy consumption for model S_5A_0 due to the model's aforementioned parameter increase. In our experiments, we found that larger accumulate intervals lead to decreased energy consumption across the board. As shown in Table II, the accumulator only consumes a marginal amount of energy compared to the entire system.

F. Accumulator Overheads

To accurately estimate the overheads of the accumulator when used in a heterogeneous SoC comprising neuromorphic and ANN accelerator cores, we implemented the accumulator in the Verilog Hardware Description Language (HDL).

The accumulator circuit includes k-bit counters with three input ports corresponding to the input clock, reset, and spike signals. The clock signal is dependent on the hardware system's global clock and the reset signal is driven by the accumulator module. The *spike* signal corresponds to whether a spike was emitted by the SNN during the current clock. Thus, if the last layer of the SNN, connected to the accumulator, spikes, the spikes are transmitted to the accumulator's counters which then increments the k-bit counter.

The high-level *accumulator module* combines multiple copies of the k-bit counters into a single hardware accumulator block. The accumulator also has three inputs including the clock signal along with a *sync* signal for synchronizing all of the counters and N input wires to propagate the spike signals to their respective counters. The accumulator also contains an internal k-bit register for controlling when the N k-bit counters should be reset after a predefined number of clock cycles have passed. The output of the accumulator module consists of $N \times k$ bits which can then be connected to an ANN accelerator core for the ANN inference phase of the hybrid model.

Here, we fix the number of inputs into the accumulator to $N = 128$ bits which are then fed into the 128 k-bit counters inside of the accumulator. The number of neurons connected to the accumulator from the SNN part of the model often exceeds the limit of the $N = 128$. To accommodate these larger layer sizes, we divide the total number of output neurons from the last layer into partitions with 128 neurons and send each partition to the accumulator per timestep. Depending on the value of k, the accumulator's output bus would then be $128 \times k$ bits wide. The value of k depends on the accumulate interval (I) value. For example, within the $I = 5$ interval, there will be a maximum of 5 spikes. Thus, a 3-bit counter would adequately store the accumulator's output. Similarly, 4-bit and 5-bit counters can support accumulate intervals of $I = 10$ and $I = 25$, respectively.

Using the Synopsys Design Compiler, we assessed the performance of our accumulator operation on hardware. The findings, detailed in Table II, indicate that the latency, power, and energy overheads attributed to the accumulator circuit are significantly lower, by several orders of magnitude, compared to those of the ANN and SNN models running on the ANN accelerator and neuromorphic cores. This implies that the accumulator overheads are negligible.

V. Conclusion

In this work, we presented a methodology for the efficient deployment of hybrid spiking models on a distributed system of neuromorphic hardware and edge AI accelerators. Our experiments involved testing numerous hybrid models to explore the effect that introducing spiking layers would have on performance. We trained our models on the event-based DvsGesture dataset to perform Gesture Recognition. The models were then profiled on separate devices to record power, latency, and energy. Our findings show that hybrid networks reduce energy consumption compared to homogeneous networks with minimal accuracy loss. We also show that hybrid networks work best with only a few spiking layers serving as feature extractors for following convolution blocks. However, due to the lack of physical hardware, we could not record the cost of communication between devices. This cost would affect latency and energy consumption, potentially

offsetting the benefit of hybrid networks. Despite this, the hybrid networks show promising results that can be explored in further research.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation (NSF) under grant number 2340249. Special thanks to Intel Labs for providing access to the Loihi chips for the experiments performed in this paper.

REFERENCES

[1] J. L. Lobo, J. Del Ser, A. Bifet, and N. Kasabov, "Spiking neural networks and online learning: An overview and perspectives," *Neural Networks*, vol. 121, pp. 88–100, 2020.

[2] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International journal of neural systems*, vol. 19, no. 04, pp. 295–308, 2009.

[3] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

[4] J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, 2023.

[5] M. Davies, "Lessons from loihi: Progress in neuromorphic computing," in *2021 Symposium on VLSI Circuits*, pp. 1–2, 2021.

[6] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 12, pp. 6999–7019, 2021.

[7] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

[8] M. Mohammadi, P. Chandarana, J. Seekings, S. Hendrix, and R. Zand, "Static hand gesture recognition for american sign language using neuromorphic hardware," *Neuromorphic Computing and Engineering*, vol. 2, p. 044005, oct 2022.

[9] P. Chandarana, M. Mohammadi, J. Seekings, and R. Zand, "Energy-efficient deployment of machine learning workloads on neuromorphic hardware," in *2022 IEEE 13th International Green and Sustainable Computing Conference (IGSC)*, pp. 1–7, 2022.

[10] H. Smith, J. Seekings, M. Mohammadi, and R. Zand, "Realtime facial expression recognition: Neuromorphic hardware vs. edge ai accelerators," in *2023 International Conference on Machine Learning and Applications (ICMLA)*, pp. 1547–1552, 2023.

[11] A. Kugele, T. Pfeil, M. Pfeiffer, and E. Chicca, "Hybrid snn-ann: Energy-efficient classification and object detection for event-based vision," in *Pattern Recognition* (C. Bauckhage, J. Gall, and A. Schwing, eds.), (Cham), pp. 297–312, Springer International Publishing, 2021.

[12] N.-C. Wu, T.-H. Chen, and C.-T. Huang, "Hardware-aware model architecture for ternary spiking neural networks," in *2023 International VLSI Symposium on Technology, Systems and Applications (VLSI-TSA/VLSI-DAT)*, pp. 1–4, 2023.

[13] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

[14] A. K. Kosta, M. P. E. Apolinario, and K. Roy, "Live demonstration: Ann vs snn vs hybrid architectures for event-based real-time gesture recognition and optical flow estimation," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 4148–4149, 2023.

[15] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 x 128 120db 30mw asynchronous vision sensor that responds to relative intensity change," in *2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers*, pp. 2060–2069, 2006.

[16] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240 × 180 130 db 3 µs latency global shutter spatiotemporal vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014.

[17] N. Muramatsu and H.-T. Yu, "Combining spiking neural network and artificial neural network for enhanced image classification," 2021.

[18] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[19] L. Cordone, B. Miramond, and P. Thierion, "Object detection with spiking neural networks on automotive event data," in *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2022.

[20] H. Zhang, Y. Wang, and Y. Zhang, "Direct training high-performance spiking neural networks for object recognition and detection," *Frontiers in Neuroscience*, vol. 17, p. 1229951, 2023.

[21] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," *Advances in neural information processing systems*, vol. 31, 2018.

[22] Intel, "Lava: A software framework for neuromorphic computing," 2021.

[23] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[24] 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, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[25] B. C. Reidy, M. Mohammadi, M. E. Elbtity, and R. Zand, "Efficient deployment of transformer models on edge tpu accelerators: A real system evaluation," in *Architecture and System Support for Transformer Models (ASSYST ISCA)*, 2023.

[26] M. Mohammadi, H. Smith, L. Khan, and R. Zand, "Facial expression recognition at the edge: Cpu vs gpu vs vpu vs tpu," in *Proceedings of the Great Lakes Symposium on VLSI 2023*, GLSVLSI '23, (New York, NY, USA), p. 243–248, Association for Computing Machinery, 2023.

[27] M. Mohammadi, S.-E. Huang, T. Barua, I. Rekleitis, M. J. Islam, and R. Zand, "Caveline detection at the edge for autonomous underwater cave exploration and mapping," in *2023 International Conference on Machine Learning and Applications (ICMLA)*, pp. 1392–1398, 2023.

[28] NVIDIA, "Jetson nano developer kit."

[29] NVIDIA, "Tensorrt: Sdk for high-performance deep learning inference," 2024.

[30] ONNX, "The open standard for machine learning interoperability," 2024.

[31] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[32] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience*, vol. 11, p. 294078, 2017.

[33] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *elife*, vol. 8, p. e47314, 2019.

[34] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in neuroinformatics*, vol. 7, p. 48, 2014.

[35] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A low power, fully event-based gesture recognition system," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.