ASNPC: An Automated Generation Framework for SNN and Neuromorphic Processor Co-design

Xiangyu Wang[1*], Yuan Li[1*], Zhijie Yang[2], Chao Xiao[1], Xun Xiao[1],
Renzhi Chen[2], Weixia Xu[1], Lei Wang[2✉]
[1]*National University of Defense Technology, Changsha, China*
[2]*Defense Innovation Institute, Academy of Military Sciences, Beijing, China*

***Abstract*—Spiking neural networks (SNNs) are promisingly considered as energy-efficient alternatives to traditional deep neural networks. At the same time, neuromorphic processors have garnered increasing development to support the efficient execution of large-scale SNNs. However, current works always separate their design to primarily prioritize a single criterion. Hardware-algorithm co-design allows for the simultaneous consideration of hardware and algorithm characteristics during the design process, effectively reducing resource usage while optimizing the algorithm's performance. In light of this, we developed a hardware-algorithm co-design framework named ASNPC for SNNs and neuromorphic processors. Considering the vast mixed-variable co-design space and the time-expensive function evaluations, we employed the surrogate-based multi-objective optimization algorithm MOTPE to identify Pareto solutions that balance algorithm performance and hardware costs. To rapidly obtain hardware results, we designed an end-to-end methodology that can automatically generate the Register-Transfer Level (RTL) code for neuromorphic processors corresponding to each candidate using templates from the hardware library. The evaluated hardware metrics, such as hardware resource and power consumption, are then fed back to MOTPE for the next candidate selection. Compared to existing works, the proposed approach can find better Pareto solutions within a limited search budget, making it effectively adapted to various application scenarios. Additionally, under the same hardware configuration, the neuromorphic processor we generated achieves lower hardware resource usage and higher throughput.**

***Index Terms*—Spiking Neural Network, Neuromorphic Processor, Hardware-Algorithm Co-design**

I. Introduction

Internet of Things (IoT) technology has gained widespread adoption due to its advantages in real-time data collection and automated management, handling vast amounts of data. However, the highly fragmented environment and the diverse application scenarios of IoT have led to an increasing demand for low-power, high-efficiency computing solutions at edge computing. Traditional deep neural networks (DNNs), with their high computational complexity, struggle to fulfill the power constraints required for edge computing. Studies have shown that the human brain can perform extremely complex tasks with only about 20W of power [1]. This has positioned neuromorphic computing, which mimics the mechanisms of the human brain, as an ideal solution for edge computing due to its high energy efficiency and low power consumption.

Spiking Neural Network (SNN) is a type of neuromorphic computing model that processes information through spikes, mimicking the behavior of biological neurons. Liquid State Machine (LSM) is a network type of SNN, first proposed by Maass in 2002 [2]. LSM has gained attention due to its lower training complexity compared to other SNN models. It consists of the input layer, a liquid layer, and a readout layer. The liquid neurons are randomly connected through synapses, converting input into high-dimensional representations, which can effectively extract features and provide richer information for the readout neurons. Since different liquid layer structures exhibit varying capacities for representing input data, it's essential to search for the optimal network structure. The liquid states generated by the liquid layer are used to train the readout layer, enabling rapid adaptation to various applications [3] [4] [5]. Neuromorphic processors [6] [7], designed mainly for supporting SNNs, can accelerate the execution of LSM and also complete complex tasks with low power consumption and high efficiency. Therefore, by optimizing the LSM network architecture and deploying it on the neuromorphic processor, complex tasks can be tackled effectively.

To handle complex application scenarios, traditional methods typically involve running existing algorithms on general-purpose hardware. However, since hardware development often lags behind algorithm advancements, this approach tends to encounter performance bottlenecks due to suboptimal optimization. A viable solution is to design dedicated hardware architectures for new algorithms, but this process of designing and validating specific hardware architectures can be time-consuming and costly. Algorithm-defined hardware is a new design paradigm that aims to translate algorithmic requirements directly into hardware architectures, optimizing performance for specific applications. The main design concept is to design custom hardware based on the characteristics and needs of the algorithm, with hardware serving as a flexible platform that can support dynamic algorithm deployment, satisfying the pressing demands for low power consumption and high efficiency in complex scenarios.

Therefore, we designed ASNPC, a hardware-algorithm co-design framework for LSM, which enables the search for LSM network architectures based on application requirements and

[*] Equal contribution.

This work was supported in part by the National Natural Science Foundation of China under Grants 62372461, 62032001 and 62203457, in part by National Defense Key Laboratory Project of the State Administration of Science and Industry for National Defense under Grant WDZC20235250112 and in part by the Key Laboratory of Advanced Microprocessor Chips and Systems.

[✉] Corresponding author: leiwang@nudt.edu.cn.

978-3-9826741-0-0/DATE25/© 2025 EDAA

the automatic generation of dedicated hardware according to the algorithm output. On the hardware side, we developed an end-to-end framework based on Register-Transfer Level (RTL) code blocks in hardware library for the LSM used in the optimization algorithm. On the algorithmic side, we chose the MOTPE algorithm for the search of LSM and evaluated the algorithm's effectiveness. Our contributions can be summarized as follows:

- To satisfy the diverse task requirements of different application scenarios for edge computing, we propose ASNPC, which adopts the algorithm-defined hardware design philosophy to generate optimized LSM architectures for a range of application scenarios using dedicated neuromorphic processors tailored to algorithmic needs.
- To mitigate the time and effort required for hardware adjustments, we developed an end-to-end design methodology based on configurable RTL code blocks, which can automatically generate LSM hardware implementations tailored to various scenarios, offering advantages in terms of hardware overhead.
- To obtain LSM network architectures suitable for various scenarios, we used the MOTPE algorithm to search for the co-design parameters of LSMs. We demonstrate the co-design efficiency across three datasets.

Experiment results show that our hardware design reduces LUT and FF resource usage by 53.07% and 29.95% compared to current end-to-end works. At the same time, we achieve lower latency and higher throughput. We chose to use MOTPE as our search algorithm because it achieves a better Pareto front. Through hardware-algorithm co-design, our framework attained the accuracies of 93.53%, 89.64%, and 100% on the DVS-128 gesture recognition, binaural sound source localization, and Bern-Barcelona EEG classification datasets, respectively. Additionally, by balancing accuracy and hardware costs, hardware resource usage can be reduced by up to 33.50% with an accuracy loss of no more than 3%.

II. Background

LSM, the spiking version of reservoir computing, is a type of Spiking Neural Network that incorporates both feed-forward and recurrent connections. As shown in Fig. 1, LSM typically consists of three components: the input layer responsible for spike encoding, the liquid layer that generates feature vectors, and the task-related readout layer. LSM tactfully avoids the training dilemma of general recurrent SNN models since only the weights of the readout layer need to be trained. The properties of rich dynamics and extremely low training overhead make it an ideal network model for edge systems.

The liquid layer is the core part of LSM, which is a randomly and circularly interlinked pool of both excitatory and inhibitory neurons. In this work, leaky integrate and fire (LIF) neuron model is adopted to simulate all neurons' behavior, which can be mathematically expressed as:

$$\tau_m \frac{du}{dt} = -[u - u_r] + RI(t) \tag{1}$$

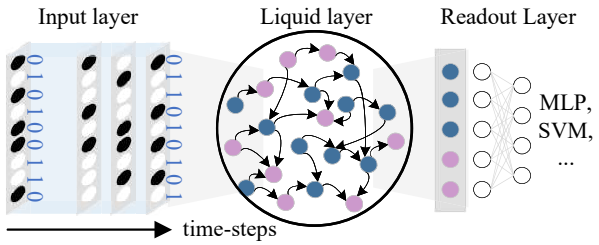

Fig. 1. The structure of LSM.

where τ_m is the membrane constant, and u_r is the resting potential. The LIF neuron model simulates the behavior of biological neurons by simplifying their dynamic characteristics. Its main advantages lie in computational efficiency and a degree of biological plausibility, enabling it to avoid complex biological details.

We adopt a geometrical structure generation method for the liquid layer [8], inspired by the concept that "the closer, the more likely to be interconnected.". Consequently, the probability that **neuron m** synapses to **neuron n** can be formulated as:

$$P_{\mathrm{m,n}} = C \times e^{-(D_{\mathrm{m,n}}/\alpha)} \tag{2}$$

where $D_{\mathrm{m,n}}$ represents the Euclidean distance between two neurons, α is a control factor, and C is the connection coefficient. We use Brian2 [9] for LSM's simulation.

III. Related Work

A. Hardware-Algorithm Co-design

STELLAR, a hardware-algorithm co-design framework developed by Ruixing et al. [10], achieves high energy efficiency and low latency while preserving accuracy by capitalizing on the rich spatiotemporal dynamics inherent in SNNs. Shikhar et al. [11] introduced CODEBench, which facilitated the co-design of neural network architectures and accelerators through Bayesian Optimization with second-order gradients and a heteroscedastic surrogate model. Meanwhile, Hongxiang et al. [12] utilized evolutionary algorithms to explore the Pareto frontier of co-designing CNN and accelerators. DANCE, proposed by Kanghyun et al. [13], is a differentiable NAS framework aimed at co-exploring hardware accelerators and network architectures. Lei et al. [14] introduced a framework that simultaneously identifies multiple DNN architectures and their corresponding heterogeneous ASIC accelerator designs, using a novel RNN-based reinforcement learning controller to predict different neural architectures concurrently. Maryam Parsa et al. proposed PABO [15], a novel pseudo agent-based multi-objective hyperparameter optimization (PABO) for maximizing the DNN performance while obtaining low hardware cost.

In this work, the challenges posed by SNN and neuromorphic co-design problem are two-fold. First, the co-design space consists of both discrete and continuous variables. Second, the real hardware verification process is time-intensive, thus making it unrealistic to conduct too many function evaluations. To address the above-mentioned challenges, we adopt the surrogate-based multi-objective optimization algorithm, MOTPE (Multi-objective Tree-structured Parzen Estimator) [16] to explore

the Pareto frontier balancing SNN algorithm performance and hardware efficiency. The MOTPE algorithm, an extension of TPE, tackles computationally expensive multi-objective optimization problems (MOPs) by modeling the design space with a tree-structured Parzen estimator and selecting solutions based on probability density. It converges faster than many existing methods while maintaining solution diversity.

B. Hardware Generation Frameworks

In the field of hardware acceleration for DNNs, various tools such as FINN [17], Eyeriss [18], and fpgaConvNet [19] provide methods for rapidly converting software models to hardware accelerators to optimize resource utilization and enhance performance. Among these, FINN serves as a significant benchmark for achieving efficient end-to-end network mapping onto FPGAs. In the domain of neuromorphic computing, several renowned neuromorphic processors have their own mapping algorithms to reconfigure the hardware to fit the corresponding network structure. For instance, LCompiler [20] is a compiler specifically designed for spiking neural networks on Loihi [7]. Similarly, BrainScaleS [21] and SpiNNaker [22] have developed their own mapping algorithms [23], [24]. Other efforts, such as S2N2, have extended FINN to support the implementation of LIF neurons. E3NE [25] supports feedforward SNNs based on a custom hardware RTL code library. The work [26] proposes a comprehensive optimization scheme that systematically enhances coding methods, models, and hardware architectures, significantly improving the performance and efficiency of SNNs on FPGAs.

These efforts support rapid conversion from neural networks to hardware architecture, however, most frameworks utilize HLS to implement end-to-end frameworks. While HLS simplifies hardware implementation, the fixed conversion from HLS to RTL makes fine-grained hardware optimization. Furthermore, current frameworks predominantly support feedforward neural networks and cannot accommodate the implementation of LSMs. Therefore, there is a need for a tool that can leverage RTL blocks to rapidly achieve a more comprehensive conversion method.

IV. ASNPC: A Hardware-Algorithm Co-design Framework

A. Overview

We propose a hardware-algorithm co-design method, as shown in Fig. 2. This method consists of three parts. First, we identify the relevant design variables from both the algorithm and hardware design. In terms of algorithms, the design space includes parameters related to network topology and neuronal dynamic properties. On the hardware side, the design space encompasses architecture parameters related to hardware performance.

Second, considering the complexity of the co-design space, we use a multi-objective optimization search strategy, which will be detailed in IV-C. This strategy updates the surrogate model and Pareto front using accuracy results generated by software and hardware performance results, producing better candidate parameters for subsequent searches.

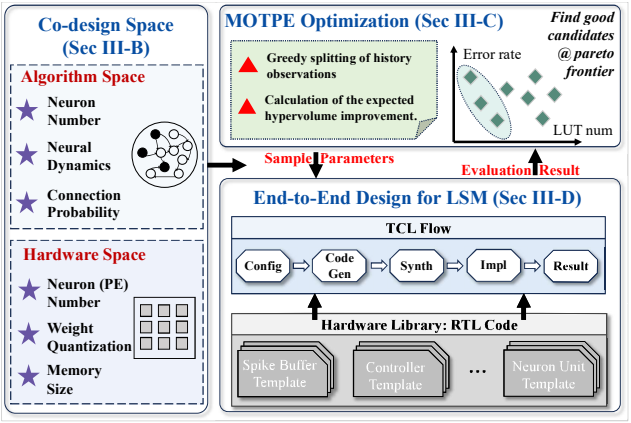

Fig. 2. Overview of our hardware-algorithm co-design method.

Finally, after confirming the relevant parameters, we implement the code on the software framework and use an end-to-end framework to integrate code blocks from the hardware library, generating the corresponding hardware design, followed by outputting evaluation results.

Algorithm 1 Hardware-Algorithm Co-design using MOTPE

Input:
$D = \left\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\right\}$: observations
n_i: number of iterations
n_c: number of candidates per iteration
γ: quantile
Output:
Pareto set of co-design configurations.
for $i \leftarrow 1, \ldots, n_i$ **do**
  $D_l, D_g \leftarrow$ SPLIT_OBSERVATIONS(D, γ) ▷ *Greedy Splitting of the previous observations*
  construct $l(\boldsymbol{x})$ with D_l and $g(\boldsymbol{x})$ with D_g
  $C \leftarrow \{\boldsymbol{x}^{(i,j)} \sim l(\boldsymbol{x}) \mid j = 1, \ldots, n_c\}$ ▷ *sample candidates*
  $\boldsymbol{x}^* \leftarrow \arg\max_{\boldsymbol{x} \in C} \mathrm{EHVI}_\gamma(\boldsymbol{x})$ ▷ *select the best candidate*
  $\boldsymbol{y}^* \leftarrow f(\boldsymbol{x}^*)$ ▷ *Algorithm and hardware assessment*
  $D \leftarrow D \cup \{(\boldsymbol{x}^*, \boldsymbol{y}^*)\}$
  Update the $Pareto\ set$ and $Pareto\ frontier$.
end for
return the $Pareto\ set$ and $Pareto\ frontier$.

B. Co-design Space

Since our work focuses on designing a hardware-algorithm co-design method for the LSM model in SNNs, the design space is divided into two parts: algorithm design space and hardware design space. In the algorithm design space, the recognition accuracy of the LSM is mainly influenced by factors such as the number of neurons n in the liquid layer, the connection probability between excitatory and inhibitory neurons C, neuron threshold parameters θ, and quantization width $quan\_bit$. In the hardware design space, the number of neurons n and quantization width $quan\_bit$ directly affect hardware resource utilization and power consumption. We assume that the number of neuron units (NUs) in the hardware design is the same as the number of neurons. The relevant parameters and their meanings are listed in Table I.

TABLE I
DESIGN SPACE EXPLANATIONS

Name	Description	Search range	Type
n	The number of liquid neurons	32 to 625	int
C_{in}	Connection probability of input to excitatory neurons	0.2 to 1.5	float
C_{ee}	Connection probability of excitatory to excitatory neurons	0.5 to 2.0	float
C_{ei}	Connection probability of excitatory to inhibitory neurons	0.5 to 2.0	float
C_{ie}	Connection probability of inhibitory to excitatory neurons	0.5 to 2.0	float
C_{ii}	Connection probability of inhibitory to inhibitory neurons	0.5 to 2.0	float
θ_e	Threshold of excitatory neurons	-55.0 to -47.0	float
θ_i	Threshold of inhibitory neurons	-43.0 to -37.0	float
quan_bit	Quantization bits for weights	{2,4,6,8,10,12,16,20,24,28,32}	int

C. MOTPE: An Efficient Multi-objective Optimization Method Leveraging Hardware Evaluation information

To address the challenges brought by the mixed-variable co-design space and time-intensive evaluation, we adopt MOTPE to find the Pareto frontier balancing the algorithm performance and hardware efficiency. MOTPE can handle continuous and discrete naturally due to the inherent tree-structured hierarchical processes constructed using adaptive Parzen estimators. The pseudocode of MOTPE is provided in Algorithm 1.

There are two important components in MOTPE, the first one is the greedy splitting method (line 2). At each iteration, the history observations are split into better observations D_l and poor observations D_g for a specific γ. $l(\boldsymbol{x})$ and $g(\boldsymbol{x})$ are the probability density functions using D_l and D_g, respectively. The second important component is the EHVI (expected hyper-volume improvement) calculation (line 5). EHVI is a variant of EI (Expected improvement), one of the commonly used acquisition functions for single-objective optimization algorithms. $\boldsymbol{x}^*$ with the maximum EHVI value is selected as a candidate at the current iteration. Then, the hardware-algorithm pairs defined by $\boldsymbol{x}^*$ are implemented and evaluated (i.e., model training and end-to-end hardware evaluation). $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is added to history observations to guide the optimization process. When the given budget runs out, the $Pareto\ set$ and $Pareto\ frontier$ can be obtained.

D. Configurable RTL-Based End-to-End Design Methodology for Optimized LSM Hardware Implementation

1) Hardware Block Library Designed for LSM: For the network structure used by the algorithm, we use a modular approach to design a hardware architecture tailored for LSM, allowing for fine-grained hardware optimization. We then build a hardware library with these RTL blocks.

To improve the flexibility of the final design, we have made each RTL block configurable. The code library includes various types of configurable code blocks, including neuron modules, controller modules, and memory modules, among others. Moreover, the network configuration information is stored in configuration files, and the end-to-end framework will regenerate the relevant code blocks in the library based on the configuration information. Finally, the generated code blocks and configuration files are used to produce the final neuromorphic processor.

Fig. 4 shows an example of the result from our hardware library. In our design, the input neurons are implemented

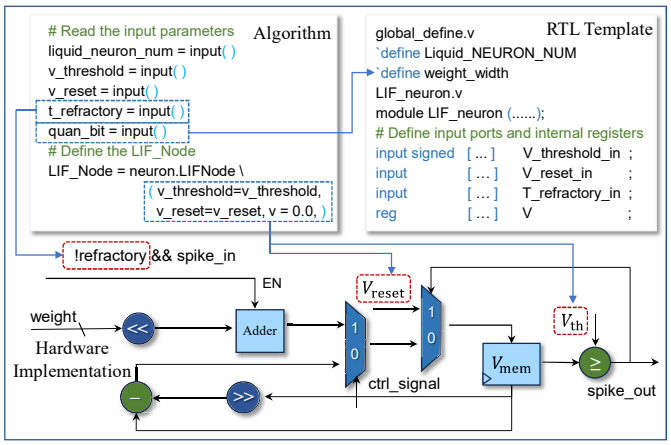

Fig. 3. Diagram of algorithm-defined hardware, showing how the hardware design of LIF neurons is closely tied to the implementation of algorithmic parameters.

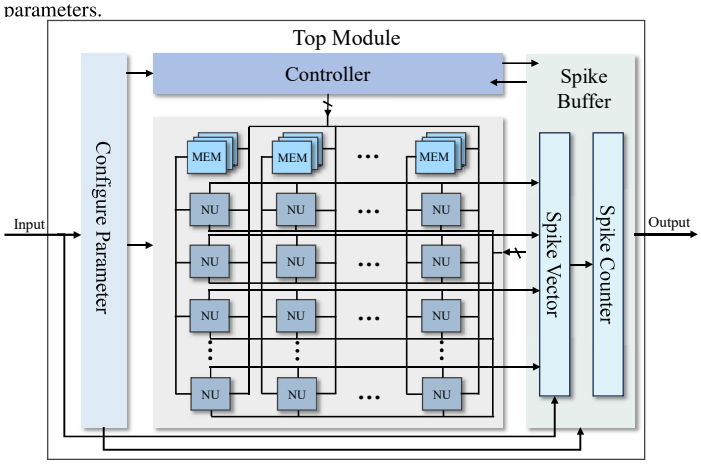

Fig. 4. The overall architecture of neuromorphic processor designed for LSM.

as registers, responsible for generating spike information and transmitting it to the spike buffer. The spike buffer receives both the output spikes generated by the liquid neurons and the input spikes, merging them as input for the next time step. Additionally, the spike buffer is responsible for aggregating the liquid state output, which is then classified by a subsequent MLP. The spike buffer also generates signals based on the current spike processing status, which are handled by the controller. The controller is responsible for generating control signals to ensure the coordinated operation of the neuromorphic processor.

Our neuron units can be configured as various types of

neurons, and each column in the array contains a weight memory to store the synaptic weights for the neurons in that column. During initialization, the controller is responsible for loading the weight information into the weight memory and passing the neuron parameter information to the neuron units. Note that all the liquid weights and readout weights will be stored in the memory at once during initialization. We do not consider partially storing the weights to reduce the memory size, in order to avoid expensive memory access at each time step. We configure the neuron units as LIF neurons, as shown at the bottom of Fig. 3. To efficiently implement the LIF neuron model, we replace the exponential operations in the membrane potential calculation process with shift operations to better suit hardware implementation. Additionally, our design supports weight quantization to reduce hardware resource usage.

As shown in Figure 3, we take the LIF neuron as an example to present the RTL code template corresponding to the algorithm parameters, as well as the implementation of these parameters in the actual hardware. The key neuron parameters in the algorithm include the number of neurons, membrane potential threshold, refractory time, and quantization width. Parameters like the number of neurons and quantization width, which do not affect internal configuration, are defined as global parameters and can be modified by regenerating the global definition file. Neuron-specific configuration parameters, such as membrane potential threshold and refractory time, are embedded in the code templates and can be configured directly via inputs at runtime, ensuring that each neuron can be customized with unique parameter settings. The lower part of Fig. 3 shows a simplified hardware design based on the Verilog code implementation of neurons, where the values for membrane potential threshold, refractory time, and membrane potential correspond directly to the software implementation.

Given the constraints of hardware resources, our neuron units are designed to be reusable during network computation. When reusing neuron units, we set up multiple memory modules to store information and re-partition the weights based on the reuse count. According to the user-specified row and column reuse counts, we can divide the neurons in the original network into multiple blocks for sequential computation. The corresponding row and column modules require additional store blocks to store the weights of neurons in that region, and addressing is managed based on the row and column reuse counts.

This computation flow is illustrated in Fig. 5. During computation, as soon as a column of neurons completes its calculations, it immediately receives its corresponding spike output. When the reuse count reaches its maximum value, the system simultaneously receives the spike outputs and reads the input spikes for the next time step. By adopting this pipelined approach, we can mitigate the increased delay due to neuron reuse, ensuring that the computation flow remains efficient and the performance is enhanced.

2) End-to-End Generation Workflow: We use TCL commands to implement the end-to-end generation workflow. First, based on the parameters provided by the search algorithm, we invoke templates from RTL code blocks and execute Python scripts required for file generation to produce the corresponding code output. Then, a project is created based on the newly generated files, and the project undergoes synthesis and implementation. Finally, regular expressions are used to extract evaluation results from the hardware report. Ultimately, We will obtain a hardware design corresponding to LSM, along with related hardware evaluation results, such as resource utilization and power consumption. Following this, we can leverage this information to carry out the MOTPE algorithm, resulting in the generation of improved candidate solutions.

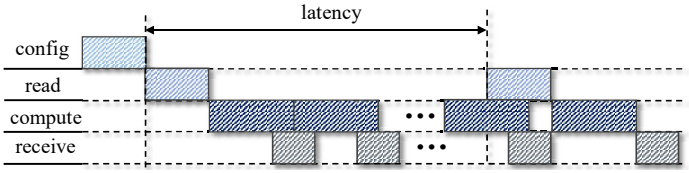

Fig. 5. Execution workflow of LSM, which illustrates how the delay associated with reading and receiving spikes is hided.

TABLE II
THE COMPARISON WITH EXISTING END-TO-END FRAMEWORKS.

Framework	Fang et al. [26]	S2N2 [27]	Ours
Frequency	125MHz	100MHz	100MHz
LUTs	124706	83254	**39074**
FFs	185278	14567	**10204**
DSP	1028	30	**0**
Latency(ms)	22.3	0.80117	**0.79**
Power(W)	4.5	**0.873**	6.065
FPS	717	1248	**1266**

V. EXPERIMENT

A. Experiment Setup

1) Datasets: All experiments are conducted using Python 3.6, the SNN simulator Brian 2.4, and PyTorch 1.0. We introduce three case studies, including DVS-128 gesture recognition, binaural sound source localization, and Bern-Barcelona EEG classification, to demonstrate the efficiency of the proposed algorithm. The DVS-128 gesture recognition dataset contains 10 hand gestures. The second case is binaural sound source localization, comprising 4,200 samples captured from seven distinct directions spanning from -45 to +45 degrees in 15-degree increments. For simplicity, we refer to this as the SSL-7 dataset. The third case is the Bern-Barcelona EEG dataset, Set A (normal EEG signals) and set E (ictal EEG signals) of this dataset were used in this work. For simplicity, we refer to this as the EEG-2 dataset.

2) Hardware Implementation: The SNN accelerator generated by our framework is deployed on Xilinx FPGA board, which is equipped with a large number of LUT and FF resources. To optimize the neuron model, we convert the multiplication operation to shift operations, thereby avoiding the use of DSP for arithmetic functions. Although the maximum clock frequency of the final design depends on the specific FPGA board and resource consumption, we set the clock frequency to 100 MHz.

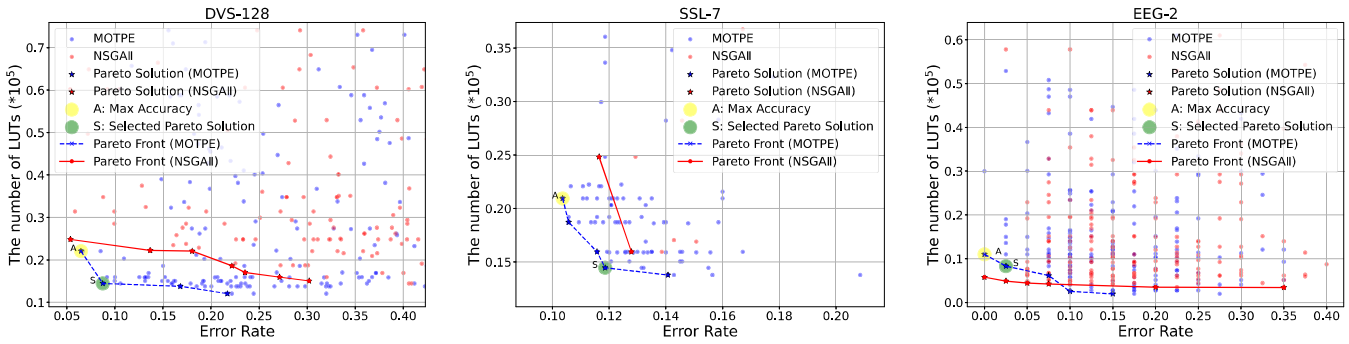

Fig. 6. Comparison between MOTPE (blue points) and NSGAII (red points) on three datasets, within 200 evaluations.

B. Comparison with End-to-End Frameworks

We compared our work with the current end-to-end frameworks, among which S2N2 is an open-source end-to-end framework. To ensure a fair comparison, we selected an MLP network supported by both frameworks, with a network architecture of 784*500*10. In our experiments, we set the relevant parameters of our work to the same values as those in the open-source framework to maintain consistency.

As shown in Table II, compared to the S2N2 framework, our design reduces the use of Look-Up Table (LUT) and Flip-Flop (FF) resources by 53.07% and 29.95%, respectively, and does not use any DSP resources. Additionally, we achieved higher throughput. Although our design shows an increase in power consumption, this is mainly due to the different FPGA boards used, which results in higher static power consumption. In our design, the dynamic power consumption of the FPGA is only 0.243W. For the work in [26], their implemented network structure is 784*500*500*10. However, their design utilizes 3.19 times more LUTs and 18.15 times more FFs compared to ours and additionally uses 1,028 DSP resources. Moreover, their design has lower throughput. Therefore, we think that our design has a performance advantage over that of [26].

C. Multi-objective Optimization (MOTPE)

To test the efficiency of the adopted algorithm MOTPE, it is compared with NSGAII (Non-Dominated Sorting Genetic Algorithm) [28] on three datasets, and the results are shown in Fig. 6. Our motivation is to find the optimum algorithm/hardware pair within as few iterations as possible. Thus, for each dataset, only 200 evaluations are conducted to estimate the Pareto frontier. Each point in Fig. 6 corresponds to one evaluation of the noted algorithm. The solution with the highest accuracy (yellow circle) and the selected Pareto solution (green circle) are highlighted in each figure.

Compared to the NSGAII, it is not hard to find that MOTPE can always find more Pareto solutions, and push the Parto frontier forward. It is analyzed that, as a genetic algorithm-based heuristic, NSGAII generally exhibits slow convergence, particularly in the early stages when population diversity is high. Consequently, a significant number of evaluations are needed to approximate the Pareto frontier.

D. Effectiveness of Co-design

Table III presents the results obtained using the MOTPE algorithm. As shown in table III, the acc column represents the parameter results corresponding to the highest accuracy we found, while the pareto column represents the optimal Pareto front we identified. Compared to the maximum accuracy, the Pareto front we identified, which balances accuracy and hardware overhead, shows a slight accuracy reduction of 2.26%, 1.50%, and 2.50% across the three datasets. However, this comes with significant reductions in LUT overhead by 34.49%, 30.94%, and 24.49%, and in FF overhead by 33.50%, 28.75%, and 22.08%, respectively. Since power consumption evaluation on FPGAs is considered unreliable, we do not use power as an evaluation information.

The above results indicate that the MOTPE algorithm can achieve high-accuracy results in different application scenarios. By selecting appropriate parameters, an effective balance between accuracy and hardware costs can be achieved, greatly reducing hardware resource usage while maintaining a relatively low drop in accuracy. This suggests that, despite a sacrifice in accuracy, such hardware optimization can enhance the overall efficiency of the system in resource-constrained application scenarios.

TABLE III
CO-DESIGN RESULTS WITHIN 200 EVALUATIONS.

Solution	DVS-128		SSL-7		EEG-2	
	acc	*pareto*	*acc*	*pareto*	*acc*	*pareto*
Neuron_num	448	288	416	288	192	144
quan_bit	2	2	2	2	4	4
Accuracy(%)	93.53	91.27	89.64	88.14	100	97.50
LUTs	22075	14462	20940	14462	10990	8298
FFs	15822	10522	14767	10522	8137	6340

VI. CONCLUSION

In this paper, we propose ASNPC, a hardware-algorithm co-design methodology for LSMs. We developed a dedicated end-to-end framework for LSMs to enable rapid deployment on FPGAs and obtain real hardware performance metrics. The MOTPE algorithm was selected and applied across three datasets to demonstrate the effectiveness of the co-design approach. Experimental results show that our end-to-end framework achieves higher hardware performance and lower resource usage compared to other approaches. Additionally, the MOTPE algorithm identifies higher-quality Pareto-optimal solutions and achieves superior accuracy across different datasets. Moreover, by balancing accuracy and hardware costs, our framework significantly reduces hardware resource usage with only a slight loss in accuracy.

REFERENCES

[1] D. Drubach. *The Brain Explained*. Prentice Hall Health, 2000.

[2] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[3] Apostolos Argyris, Julián Bueno, and Ingo Fischer. Photonic machine learning implementation for signal recovery in optical communications. *Scientific reports*, 8(1):8487, 2018.

[4] Kostas Sozos, Adonis Bogris, Peter Bienstman, George Sarantoglou, Stavros Deligiannidis, and Charis Mesaritakis. High-speed photonic neuromorphic computing using recurrent optical spectrum slicing neural networks. *Communications Engineering*, 1(1):24, 2022.

[5] Zhenshan Bing, Claus Meschede, Florian Röhrbein, Kai Huang, and Alois C Knoll. A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in neurorobotics*, 12:35, 2018.

[6] Filipp Akopyan et al. TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 34(10):1537–1557, October 2015.

[7] Mike Davies et al. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, 38(1):82–99, January 2018.

[8] Ch Brücker, D Hess, and J Kitzhofer. Single-view volumetric piv via high-resolution scanning, isotropic voxel restructuring and 3d least-squares matching (3d-lsm). *Measurement Science and Technology*, 24(2):024001, 2012.

[9] Marcel Stimberg, Romain Brette, and Dan FM Goodman. Brian 2, an intuitive and efficient neural simulator. *elife*, 8:e47314, 2019.

[10] Ruixin Mao, Lin Tang, Xingyu Yuan, Ye Liu, and Jun Zhou. Stellar: Energy-efficient and low-latency snn algorithm and hardware co-design with spatiotemporal computation. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 172–185, 2024.

[11] Shikhar Tuli, Chia-Hao Li, Ritvik Sharma, and Niraj K Jha. Codebench: A neural architecture and hardware accelerator co-design framework. *ACM Transactions on Embedded Computing Systems*, 22(3):1–30, 2023.

[12] Hongxiang Fan et al. Algorithm and hardware co-design for reconfigurable cnn accelerator. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 250–255. IEEE, 2022.

[13] Kanghyun Choi, Deokki Hong, Hojae Yoon, Joonsang Yu, Youngsok Kim, and Jinho Lee. Dance: Differentiable accelerator/network co-exploration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 337–342. IEEE, 2021.

[14] Lei Yang et al. Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[15] Maryam Parsa, Aayush Ankit, Amirkoushyar Ziabari, and Kaushik Roy. Pabo: Pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[16] Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, and Masaki Onishi. Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Proceedings of the 2020 genetic and evolutionary computation conference*, pages 533–541, 2020.

[17] Yaman Umuroglu et al. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74, February 2017.

[18] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE J. Solid-State Circuits*, 52(1):127–138, January 2017.

[19] Stylianos I. Venieris and Christos-Savvas Bouganis. fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 40–47. IEEE, May 2016.

[20] Chit-Kwan Lin, Andreas Wild, Gautham N. Chinya, Tsung-Han Lin, Mike Davies, and Hong Wang. Mapping spiking neural networks onto a manycore neuromorphic architecture. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 78–89. ACM, June 2018.

[21] Christian Pehle et al. The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity. *Front. Neurosci.*, 16:795876, February 2022.

[22] Eustace Painkras et al. SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation. *IEEE J. Solid-State Circuits*, 48(8):1943–1953, August 2013.

[23] Matthias Ehrlich et al. A Software Framework for Mapping Neural Networks to a Wafer-scale Neuromorphic Hardware System:. In *Proceedings of the 6th International Workshop on Artificial Neural Networks and Intelligent Information Processing*, pages 43–52, 2010.

[24] Francesco Galluppi, Sergio Davies, Alexander Rast, Thomas Sharp, Luis A. Plana, and Steve Furber. A hierachical configuration system for a massively parallel neural hardware platform. In *Proceedings of the 9th Conference on Computing Frontiers*, pages 183–192, Cagliari Italy, May 2012. ACM.

[25] Daniel Gerlinghoff, Zhehui Wang, Xiaozhe Gu, Rick Siow Mong Goh, and Tao Luo. E3NE: An End-to-End Framework for Accelerating Spiking Neural Networks with Emerging Neural Encoding on FPGAs. *IEEE Trans. Parallel Distrib. Syst.*, pages 1–1, 2021.

[26] Haowen Fang, Zaidao Mei, Amar Shrestha, Ziyi Zhao, Yilan Li, and Qinru Qiu. Encoding, model, and architecture: systematic optimization for spiking neural network in FPGAs. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9. ACM, November 2020.

[27] Alireza Khodamoradi, Kristof Denolf, and Ryan Kastner. S2N2: A FPGA Accelerator for Streaming Spiking Neural Networks. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 194–205. ACM, February 2021.

[28] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.