# NEURAL: An Elastic <u>Neur</u>omorphic <u>Architecture</u> with Hybrid Data-Event Execution and On-the-fly Attention Dataflow

Yuehai Chen, and Farhad Merchant
Bernoulli Institute and CogniGron, University of Groningen, The Netherlands
Email: {yuehai.chen, f.a.merchant}@rug.nl

Abstract-Spiking neural networks (SNNs) have emerged as a promising alternative to artificial neural networks (ANNs), offering improved energy efficiency by leveraging sparse and event-driven computation. However, existing hardware implementations of SNNs still suffer from the inherent spike sparsity and multi-timestep execution, which significantly increase latency and reduce energy efficiency. This study presents NEURAL, a novel neuromorphic architecture based on a hybrid dataevent execution paradigm by decoupling sparsity-aware processing from neuron computation and using elastic first-in-firstout (FIFO). NEURAL supports on-the-fly execution of spiking **OKFormer** by embedding its operations within the baseline computing flow without requiring dedicated hardware units. It also integrates a novel window-to-time-to-first-spike (W2TTFS) mechanism to replace average pooling and enable full-spike execution. Furthermore, we introduce a knowledge distillation (KD)-based training framework to construct single-timestep SNN models with competitive accuracy. NEURAL is implemented on a Xilinx Virtex-7 FPGA and evaluated using ResNet-11, QKFResNet-11, and VGG-11. Experimental results demonstrate that, at the algorithm level, the VGG-11 model trained with KD improves accuracy by 3.20% on CIFAR-10 and 5.13% on CIFAR-100. At the architecture level, compared to existing SNN accelerators, NEURAL achieves a 50% reduction in resource utilization and a 1.97× improvement in energy efficiency.

Index Terms—Spiking neural network, elastic computing, sparsity-aware, knowledge distillation, spiking transformer

# I. INTRODUCTION

Recently, an increasing number of edge devices have gained the capability to perform intelligent processing. Although current mainstream artificial neural networks (ANNs) demonstrate excellent performance, their deployment on resourceconstrained edge devices remains challenging due to their complex computation and high power consumption. In contrast, spiking neural networks (SNNs) deliver information in binary spikes, which inherently exhibit low power consumption and event-driven computation. However, SNNs still have challenges such as high latency due to multi-timestep inference, binary signals that limit backpropagation, and generalpurpose hardware struggling to efficiently support event-driven mechanisms. To address these issues, the algorithm and hardware co-design becomes the key to improving the execution efficiency of SNNs. Fig. 1 shows the differences between the hardware implementation of ANNs and SNNs. Compared to ANNs, SNNs eliminate the need for multipliers and complex

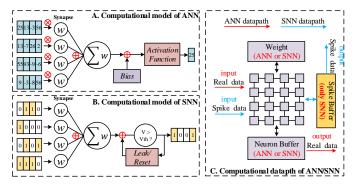


Fig. 1: Differences between ANN and SNN computational models.

activation functions, relying instead on addition and comparator, but require additional memory to store membrane potentials and spike data for temporal processing. Researchers have proposed 3D architectures [1] [2] to process multiple time steps in parallel at the cost of high resource overhead, while sparsity-aware architectures [3] optimize computation taking advantage of the sparsity of spikes. In terms of model optimization, surrogate gradient methods [4] have significantly reduced the number of time steps in recent years while maintaining accuracy. Knowledge distillation (KD) [5] [6] has also been widely used to train low time-step SNNs to guide student networks to learn better representations via ANN teacher networks, achieving a good balance between low latency and high accuracy. Meanwhile, spike-based transformer mechanisms [7] [8] further improve the recognition accuracy of SNN models. However, most existing approaches remain at the algorithmic level, constructing merely spikinglike computation graphs without true spiking execution. This highlights the importance of algorithm-hardware co-design, emphasizing that optimization should be performed not only at the algorithmic level, but also at the architectural level, in order to fully exploit the efficiency potential of neuromorphic computing. To further explore efficient SNN hardware architectures, we focus on building a co-optimized architecture with both full spiking computation and sparsity-aware capabilities that can perform inference in a single time step. Achieving high-accuracy single-timestep SNNs at the algorithmic level eliminates the need for multi-timestep scheduling in hardware, thereby reducing both inference latency and control logic complexity. Furthermore, identifying operations that remain non-spiking and transforming them into spike-based counterparts can further enhance the energy efficiency of event-driven computations. Based on these observations, in this paper, we propose NEURAL, a hybrid data-event execution neuromorphic computing architecture that supports elastic connectivity and on-the-fly attention dataflow. The main contributions are as follows.

- A training framework combining KD and fixed-point quantization, enabling single-timestep SNNs to achieve competitive accuracy compared to multi-timestep models
- Window-to-time-to-first-spike (W2TTFS) mechanism for converting non-spiking average pooling into spike-based computation while maintaining accuracy
- Hybrid data-event execution using elastic FIFO scheduling to enable data-driven control and event-driven neuron computation, while supporting on-the-fly spiking QKFormer [8] without dedicated hardware units
- 4) FPGA implementation of the NEURAL architecture deploying three deep SNNs, VGG-11, ResNet-11, and QKFResNet-11, achieves accuracies of up to 93.46% and 72.1% on CIFAR-10 and CIFAR-100 based on the KD training, respectively. Under a single-timestep execution paradigm, NEURAL significantly outperforms the existing STI-SNN [9] architecture, exhibiting nearly a 3.9× improvement in computing efficiency.

The rest of this paper is organized as follows. Section II summarizes the related neuromorphic algorithms and architectures. Section III introduces the W2TTFS mechanism and the training framework based on knowledge distillation. Section IV describes the detailed design of the NEURAL architecture. Section V evaluates and analyzes the performance of NEURAL. Finally, Section VI concludes this paper and discusses future directions.

## II. RELATED WORKS

Neuromorphic computing evolves in both algorithms and architectures. At the algorithm level, the development of spiking models and training methods has enabled SNNs to approach the performance of ANNs. At the architecture level, the inherent event-driven computation and sparsity of SNNs are leveraged to enhance energy efficiency. Therefore, we summarize related work from both aspects.

Neuromorphic algorithms. With the development of the theory of spiking computation, the network structure has been expanded from the initial simple spiking multi-layer perception (SMLP) to the deep spiking convolutional neural network (DSCNN). Training methods have gone through the transition from bio-inspired spike-timing-dependent plasticity [10] (STDP) and ANN-to-SNN [11] to surrogate gradient-based supervised training methods [4]. In order to further utilize the expressive power of ANN models in deep learning, several studies have explored the use of KD to improve the training of SNNs. Recent work shows that a ResNet-19 SNN model trained with KD can achieve 96. 65% and 81. 47%

accuracy in the CIFAR-10 and CIFAR-100 datasets, respectively, using only 2 time steps [6], significantly improving the practical value of the low-timestep SNNs.

**Neuromorphic architectures.** Compared with generalpurpose processors, dedicated neuromorphic architectures are better suited to the event-driven and sparse computational model of SNN, thus improving energy efficiency and throughput. Existing research on SNN accelerators can be categorized into the following directions: one class of work focuses on the efficient implementation of spiking MLPs [12] [13]; the other class designs sparsity-aware execution mechanisms to leverage the inherent activation sparsity. Sparsity-aware execution mechanisms [3] [14] can dynamically skip zerovalued computation and reduce redundant accesses, thus significantly improving computational efficiency. In addition, to mitigate the latency and computational overhead caused by the multi-timestep nature of SNNs, various studies have explored time-step compression techniques and time-parallel computing architectures [1] [2] [15]. However, these approaches often introduce more complex resource scheduling and higher onchip overhead while saving latency, making them difficult to deploy in resource-constrained edge environments. NEURAL bridges the gap between algorithmic improvements and architectural support, thereby addressing a major limitation of previous SNN hardware. It enables high-accuracy, low-latency, fully spike-based execution through co-optimized training and elastic hardware design with a smaller area, while supporting emerging SNN modules such as QKFormer in on-the-fly execution. This makes NEURAL a practical and scalable solution for next-generation neuromorphic computing systems.

## III. FULL-SPIKE SNN MODEL

SNNs transmit information through spikes, offering inherent advantages in computational and energy efficiency. However, current mainstream models struggle to achieve a fully spike-based computational path, particularly due to the use of average pooling in the downsampling stages. Although average pooling improves training stability under sparse input, it introduces continuous values that break the consistency of spike-based execution and increase computational and energy overhead. To address this, we propose the W2TTFS mechanism, which converts spike windows into time-to-first-spike representations across multiple timesteps during inference. By preserving fully spiking inputs to the classifier, this mechanism balances classification accuracy and hardware efficiency.

#### A. Window-to-Time-to-First-Spike (W2TTFS)

As illustrated in Fig. 2(a), to enhance the model performance, we augment the conventional ResNet-11 [16] backbone with QKFormer Blocks [8] named QKFResNet-11 to incorporate attention mechanisms. The standard average pooling (AP) operation transforms spiking signals into continuous values, leading to non-spiking inputs for the classifier. To tackle this problem, we introduce the W2TTFS method, as depicted in the **Inference** module in Fig. 2(a), which ensures that the classifier receives spike-based input. The specific conversion

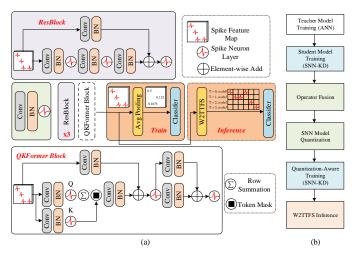


Fig. 2: The overview of QKFResNet-11 model and training framework. (a) QKFResNet-11. (b) SNN model training flow based on knowledge distillation.

flow of W2TTFS is shown in Algorithm 1. Lines 4 and 5 compute the size of the receptive field of the AP operation (denoted as  $window\_size$ ), and initialize a zero matrix with  $window\_size^2$  time steps, according to the dimensions of the feature maps produced by the final convolutional layer. Lines 8–16 iterate over the output feature maps by channel and spatial location, identifying the first valid spike timing by counting the number of spikes within each pooling window, as performed in lines 11-13. Finally, lines 17–20 compute the time-dependent weight scaling factors. For example, with  $window\_size = 4$ , if a valid spike is first emitted at time step t = 3, the corresponding scale factor is 3/16. This factor is subsequently used to scale the weights during the fully connected (FC) computation.

# B. Single-Timestep SNN based on Knowledge Distillation

In this paper, we introduce a KD-based training framework for SNNs, as illustrated in Fig. 2(b). We first train a high-accuracy ANN as the teacher model. Then, an SNN is constructed as the student model and trained via knowledge distillation. As shown in Fig. 2(a), typical SNN models often include layers such as batch normalization (BN), which pose challenges for hardware deployment. To address this, we apply operator fusion and fixed-point quantization to reduce model complexity and hardware resource usage. Since quantization may degrade accuracy, we further adopt KD-based quantization-aware training (QAT) to mitigate accuracy loss. Finally, during inference, we replace the AP layer with the proposed W2TTFS module to allow full spiking execution.

#### IV. NEURAL ARCHITECTURE

The NEURAL architecture, as shown in Fig. 3, comprises three key modules: the elastic processing element array (EPA), the pipelined sparse detection array (PipeSDA), and the W2TTFS-based fully connected computing core (WTFC). Weights are streamed into the EPA via elastic W-FIFO, sourced from the weight management unit (WMU). The WMU dynamically schedules the required weights from off-chip

# Algorithm 1 Window-to-Time-to-First-Spike (W2TTFS)

- 1: **Given:** spike\_map, which represents output from down-sampling convolution layer.
- 2: **Given:** spike\_cnt is the function of getting the number of valid spike.
- 3: **Given:** T, C,  $H_i$ ,  $W_i$ ,  $H_o$ , and  $W_o$  denote the time steps, channels, input height, input width, output height, and output width, respectively.

```
4: window_size \leftarrow H_i//H_o
 5: spike_array_fc \leftarrow Zeros(window_size<sup>2</sup>, C, H_o * W_o)
   for t = 0 to T do
 7:
       spike_array_fc.reset()
       for channel = 0 to C do
 8:
           for h=0 to H_o do
 9:
10:
               for w = 0 to W_o do
                   // window
11:
                   pooling_window \leftarrow spike_map.get(h, w)
                   vld_cnt ← pooling_window.spike_cnt()
12:
13:
                   // time-to-first-spike
                    spike_array_fc[vld_cnt, channel, h*w] = 1
               end for
14:
            end for
15:
       end for
16:
       for tt = 0 to window_size do
17:
            scale = tt / window size^2
18:
            spike_array_fc[tt].flatten.classifier(scale)
19:
       end for
20:
21: end for
```

memory based on the current computation status and feeds them into the FIFO. Input spikes are handled in a similar manner, when valid spike arrays are buffered in the elastic S-FIFO, the EPA reads them and performs parallel computation with the weights. The PipeSDA module identifies the event receptive field of each input spike based on its coordinates, and maps it to the appropriate sparse detection unit (SDU). The SDU generates a local convolution window, which is forwarded to the EPA for further processing. The WTFC module executes the computation defined by the W2TTFS layer, enabling fully spiking inference at the classifier stage. This end-to-end spiking design allows all computational layers of the SNN model to be efficiently executed within the NEU-RAL architecture, thus improving system-level integration and execution efficiency.

#### A. Hybrid Data-Event Execution Dataflow

This study proposes a data-event hybrid execution mechanism based on elastic FIFO, which adopts a data-driven control flow at the architectural level while switching to event-driven execution at the granularity of individual neuron computations. **Data execution.** As illustrated in Fig. 3, the left input of the EPA is the spiking series ①, and the upper side is the corresponding weight matrix ②, enabling the system to trigger computation as soon as data from both ends are available, without relying on centralized control. **Event execution.** As shown in Fig. 3, each PE contains a dedicated event FIFO,

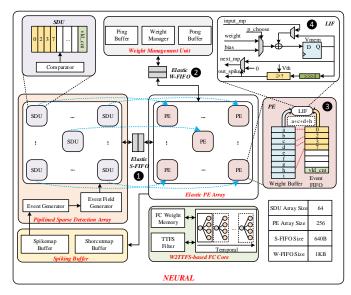


Fig. 3: The overall architecture of NEURAL.

the end register of which stores the number of currently valid events, vld\_cnt ③. During the computation process, the PE reads the event indexes from the FIFO according to the order of vld\_cnt, and obtains the corresponding weight and sends them to the LIF unit, realizing the membrane potential (MP) update computation. The LIF unit updates MP using the corresponding weights and performs threshold comparisons to determine whether to emit a spike ④. It is fully event-driven, thereby avoiding redundant updates during no-spike intervals.

#### B. Pipelined Sparse Detection Array Design

The main stages of PipeSDA include index generation (IG), center position (CP) generation, and mapping the CP to the SDU array (CP Map) for event detection. **Index Generation:** as shown in Fig. 4, the system first generates the index of all valid spikes from the input spiking image and stores them in a buffer. CP Generation: CP generation: the CP of the corresponding receptive field for each spike event is calculated based on the generated index. **CP Map:** these CPs are mapped to specific SDUs in the SDA. Since the coordinates of some CPs may be negative, the virtual SDU is predefined within the SDA architecture to support negative index mapping. Once the mapping is completed, the SDU located at the CP position broadcasts a diffusion signal to its neighboring units (as illustrated on the right of Fig. 4), indicating that the corresponding SDU lies within the active region. Each SDU that receives the signal subsequently updates its internal event FIFO according to the mapping result, preparing for the construction of the convolutional window in later processing.

## C. On-the-fly QKFormer Computation

The proposed NEURAL architecture supports on-the-fly QKFormer computation within the baseline data flow, without requiring a separate spiking transformer unit. As illustrated in Fig. 5, QKFormer operations are directly embedded into the write-back path from the EPA (where spikes are generated by the corresponding PEs) to the Spiking Buffer. As shown in

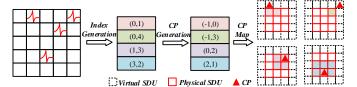


Fig. 4: Pipelined sparse detection data flow.

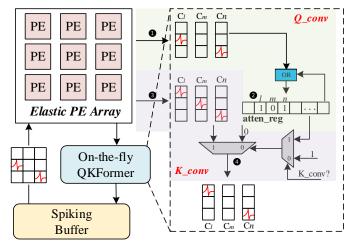


Fig. 5: On-the-fly QKFormer computation data flow.

Fig. 5, after computing the Q matrix ①, we apply bit-wise OR across channels to generate the attention activation state using attention register (atten\_reg) record ②, corresponding to the Row Summation operation along the Q path in Fig. 2. Subsequently, the K matrix is computed ③ and written back to the Spiking Buffer. During this process, the atten\_reg is used to determine per-channel activation status (0/1), which is then applied as a QK token mask ④, aligned with the Token Mask shown in Fig. 2.

## D. W2TTFS-based FC Core

Fig. 6 shows the architecture of the W2TTFS-based FC core, which consists of two core modules: the TTFS Filter and the fully-connected computing unit (FCU). The input spiking feature maps are sequentially fed into the TTFS Filter according to the channel order, and the main function of the TTFS Filter is to count the number of valid spikes in each window (denoted as vld cnt) according to the size of the pooling window, and generate the corresponding weight scaling factor. However, as shown in Algorithm 1, the scale value may have non-shift friendly decimals (e.g., 3/16). NEURAL performs a fine-grained optimization of the scale generation strategy: the scale no longer depends on the specific spiking location but is uniformly set to the inverse unit of the pooling window (e.g., 1/16). Subsequently, the complex scale is approximated by a time reuse strategy. For example, when the algorithm needs to perform the scaling of 3/16, the system repeats the summation of the unit three times to complete the corresponding membrane potential update, avoiding the multiplication and high-precision division operation.

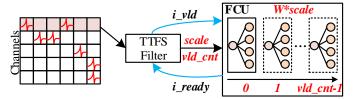


Fig. 6: W2TTFS-Based FC core.

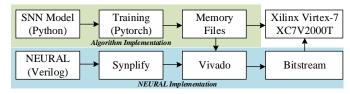


Fig. 7: Design flow of algorithm and hardware implementation.

## V. EXPERIMENTAL EVALUATION

## A. Experimental Settings

Algorithm implementation. We implemented four SNN models: VGG-11, ResNet-11, QKFResNet-11, and ResNet-19 based on PyTorch and SpikingJelly [17]. The LIF neuron decay parameter  $\tau$  is 0.5, and the time step is set to 1. The models were trained and tested on CIFAR-10 and CIFAR-100 datasets using an NVIDIA RTX 2080TI. Training is performed using a logit-based KD framework [6], and the teacher model is ResNet-34 with an SGD optimizer with a drive of 0.9, a batch size of 128, and 300 epochs.

Hardware implementation. After getting a quantized model, the memory files are generated for hardware implementation, as shown in Fig. 7. NEURAL was implemented in Verilog HDL, synthesized using Synplify, and placed and routed using Xilinx Vivado. The design operates at 200MHz on a Xilinx Virtex-7 XC7V2000T FPGA.

#### B. Algorithm Analysis

As illustrated in Fig. 8, it contains four model types: KDT: a full precision model trained using KD; F & Q: a simplified model applying operator fusion and fixed point quantization; KD-QAT: a QAT model based on KD; and W2TTFS: a hardware-friendly model proposed in this paper. As shown in Fig. 8(a), our KD-trained single-timestep VGG-11 achieves 94.06% accuracy on CIFAR-10 in the full precision setting, outperforming [2] by 3.01%, which was evaluated using 4 time steps. After quantization, the KD-QAT VGG-11 exhibits only a 0.17% accuracy loss, which is 0.34% smaller than [2]. In CIFAR-100, the improvement is even more significant, reaching 5.77% and 1.15%, respectively. In addition, KD-QAT has an obvious advantage in maintaining high accuracy. For example, as shown in Fig. 8(b), the accuracy of ResNet-19 decreases by nearly 7% after F & Q, while the loss of accuracy after KD-QAT fine-tuning is only 0.69%. Specifically, QKFResNet-11 improves accuracy by 0.79% over ResNet-11 on CIFAR-100 by incorporating the QKFormer Block. Consequently, the KDbased model training and quantization method can effectively improve the accuracy of single-timestep SNN models, reduce

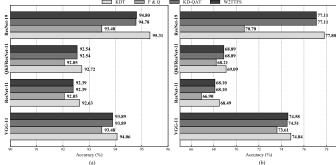


Fig. 8: Accuracy of CIFAR-10/CIFAR-100 based on different models. (a) Accuracy of CIFAR-10. (b) Accuracy of CIFAR-100.

TABLE I: Hardware Resource Cost of NEURAL

Resource	PipeSDA	EPA	WTFC	Total
LUTs	9K (12%)	33K (45%)	1K (1%)	74K
Registers	10K (16%)	15K (24%)	0.7K (1%)	63K
BRAM	3 (2%)	64 (47%)	25 (18%)	137.5

the performance loss due to quantization, and thus enhance the robustness of the models.

## C. Computing Resource and Energy Analysis

Resource Analysis: We deployed the same VGG-11 and ResNet-11 networks used in SiBrain [2] and SCPU [16] on NEURAL and evaluated them on the CIFAR-10 and CIFAR-100 datasets. As shown in Fig. 9, benefiting from the single-timestep execution paradigm, NEURAL consumes only around 70K LUTs, approximately 50% less than other architectures, and also achieves nearly a 50% reduction in RAM usage. In terms of recognition accuracy, the VGG-11 model deployed in NEURAL achieves 93.45% and 72.1% in CIFAR-10 and CIFAR-100, respectively, which is the highest among all the compared architectures. Consistent results are also observed with the ResNet-11 deployment, as shown in Fig. 9. Table I also reports the resources of each key component. The EPA uses nearly half of the total resources since it is the main computing engine. Sparse processing incurs little hardware overhead because its internal logic is simple. For the proposed WTFC, it uses 1K LUTs and 0.7K registers, making it well-suited for edge devices.

Energy Analysis: In edge computing, energy consumption directly affects the deployability and life cycle of the system. As shown in Fig. 10, the energy consumption of NEURAL is reduced by nearly 50% compared to the baseline architecture [2] [16] in the inference process of a single image, and the frame per second (FPS) of image processing is also improved. Specifically, NEURAL achieves a frame rate of 68 FPS while the energy consumption of a single image is only about 10mJ in the VGG-11 (CIFAR-10) task, and the frame rate is increased to 136 FPS in the ResNet-11 (CIFAR-10) task, while the energy consumption remains below 10mJ. The results show that NEURAL is able to achieve a higher energy efficiency ratio while guaranteeing recognition accuracy.

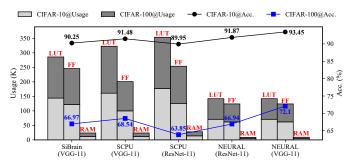


Fig. 9: The execution result of VGG-11/ResNet-11 with CIFAR-10/100 on the different platform: resource and accuracy.

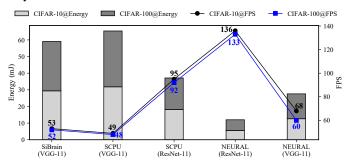


Fig. 10: The execution result of VGG-11/ResNet-11 with CIFAR-10/100 on the different platform: computing energy and throught (FPS).

# D. ResNet-11 vs. QKFResNet-11

Table II depicts the comparison of deploying ResNet-11 and QKFResNet-11 on NEURAL. As shown in Table II, the integration of attention layers in QKFResNet-11 improves classification accuracy across both datasets, with a notable 1.59% gain on CIFAR-100. Due to the increased network depth, QKFResNet-11 incurs an additional latency of approximately 2 ms. Total Spikes (TS) denotes the total spikes generated during inference. As shown in Table II, QKFormer integration can reduce TS on CIFAR-10 for efficient spike suppression, while increasing TS on CIFAR-100 to adapt to higher task complexity. Importantly, compared to [16], our QKFResNet-11 achieves a 4.68% accuracy improvement while reducing energy by 10 mJ.

#### E. Comparison with Prior Neuromorphic Architectures

Giga synaptic operations per second per watt (GSOPS/W) is a common metric for evaluating the energy efficiency of SNN hardware architectures. As illustrated in Table III, when deploying ResNet-11 model on the CIFAR-10 dataset using FP8 precision, NEURAL achieves an accuracy of 91.87%, a frame rate of 136 FPS, a power consumption of just 0.758 W, and an energy efficiency of 46.65 GSOPS/W. Furthermore, with the VGG-11 model on the same dataset, accuracy improves to 93.45%, with a frame rate of 68 FPS, a power consumption of 0.792 W, and an even higher energy efficiency of 52.37 GSOPS/W. Compared with other state-of-the-art platforms, NEURAL consumes substantially less power than SiBrain [2] (1.56 W) and STI-SNN [9] (1.34

TABLE II: ResNet-11 vs. QKFResNet-11 using CIFAR-10/100

Data	Model	Total Spikes	Acc. (%)	Latency (ms)	Energy (mJ)
CIFAR-10	ResNet-11	76K	91.87	7.3	5.56
	QKFResNet-11	72K	92.01 (+0.14)	9.7	8.14
CIFAR-100	ResNet-11	83K	66.94	7.5	6.44
	QKFResNet-11	84K	68.53 (+1.59)	9.9	8.26

TABLE III: Comparison with Existing SNN Accelerators on CIFAR-10

Platform	[2]	[3]	[9]	[18]	NEURAL	
Device	V. 7	Z. 7	Z. U	V. U	V. 7	
Fmax (MHz)	200	200	200	100	200	
Model	VGG- 11	MobileNet	SCNN5	VGG-9	ResNet- 11	VGG- 11
Precision	FP8	N/A	INT8	IN4	FP8	
Acc. (%)	90.25	91.90	90.31	86.6	91.87	93.45
FPS	53	90	397	120	136	68
Power (W)	1.56	1.4	1.53	0.73	0.76	0.79
Efficiency (GSOPS/W)	84.16	31.6	13.46	64.11	46.65	52.37
Norm. Eff (GSOPS/W/ kLUTs)	0.60	0.37	0.52	0.58	0.65	0.73

W), while also achieving superior accuracy and efficiency. Meanwhile, NEURAL achieves up to 3.9× higher computational efficiency than STI-SNN [9], even under the same single-timestep setting. For a fair comparison, we adopt the efficiency per kLUTs (GSOPS/W/kLUTs) as the evaluation metric. As shown in Table III, NEURAL achieves the highest normalized efficiency of 0.73. In particular, compared to [3], NEURAL achieves similar accuracy on CIFAR-10 with only a 0.03% loss, while significantly increasing the FPS by 46 and improving normalized efficiency by 1.97×.

# VI. CONCLUSION

In this paper, we proposed NEURAL, a hybrid data-event execution neuromorphic architecture with elastic interconnection, which enabled efficient SNNs execution and supported on-the-fly computation of spiking QKFormer. We introduced a KD-based training framework to achieve competitive accuracy in single time-step execution. For hardware-friendly deployment, the trained SNN model performed operator fusion, quantization, and KD-based OAT. Furthermore, we proposed the W2TTFS mechanism as a replacement for the AP layer, enabling full-spike execution. Experimental results demonstrated that NEURAL achieves high accuracy and performance with significantly reduced hardware resource consumption. In the future, we plan to explore more applications on NEURAL, including image segmentation [19] and spiking large language models [20], to further promote the practical adoption of energy-efficient neuromorphic computing.

#### REFERENCES

- [1] C. Fang, Z. Shen, Z. Wang, C. Wang, S. Zhao, F. Tian, J. Yang, and M. Sawan, "An energy-efficient unstructured sparsity-aware deep SNN accelerator with 3-D computation array," *IEEE Journal of Solid-State Circuits*, vol. 60, pp. 977–989, Mar. 2025.
- [2] Y. Chen, W. Ye, Y. Liu, and H. Zhou, "SiBrain: A sparse spatio-temporal parallel neuromorphic architecture for accelerating spiking convolution neural networks with low latency," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, pp. 6482–6494, Dec. 2024.
- [3] Q. Chen, C. Gao, and Y. Fu, "Cerebron: A Reconfigurable Architecture for Spatiotemporal Sparse Spiking Neural Networks," *IEEE Transactions* on Very Large Scale Integration (VLSI) Systems, vol. 30, pp. 1425–1437, Oct. 2022.
- [4] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," Frontiers in Neuroscience, vol. 12, May 2018.
- [5] X. Liang, G. Chao, M. Li, and Y. Zhao, "Knowledge distill for spiking neural networks," in 2024 International Joint Conference on Neural Networks (IJCNN), pp. 1–8, June 2024.
- [6] C. Yu, X. Zhao, L. Liu, S. Yang, G. Wang, E. Li, and A. Wang, "Efficient logit-based knowledge distillation of deep spiking neural networks for full-range timestep deployment," May 2025.
- [7] M. Yao, J. Hu, Z. Zhou, L. Yuan, Y. Tian, B. Xu, and G. Li, "Spike-driven transformer," *Advances in Neural Information Processing Systems*, vol. 36, pp. 64043–64058, Dec. 2023.
- [8] C. Zhou, H. Zhang, Z. Zhou, L. Yu, L. Huang, X. Fan, L. Yuan, Z. Ma, H. Zhou, and Y. Tian, "QKFormer: Hierarchical spiking transformer using Q-K attention," *Advances in Neural Information Processing Systems*, vol. 37, pp. 13074–13098, Dec. 2024.
- [9] K. Wang, C. Yang, C. Yu, Y. S. Ang, B. Wang, and A. Wang, "STI-SNN: A 0.14 GOPS/W/PE single-timestep inference FPGA-based SNN accelerator with algorithm and hardware co-design," June 2025.
- [10] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," Frontiers in Computational Neuroscience, vol. 9, Aug. 2015.

- [11] 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, Dec. 2017.
- [12] H. Chu, Y. Yan, L. Gan, H. Jia, L. Qian, Y. Huan, L. Zheng, and Z. Zou, "A neuromorphic processing system with spike-driven SNN processor for wearable ECG classification," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 16, pp. 511–523, Aug. 2022.
- [13] D.-A. Nguyen, X.-T. Tran, K. N. Dang, and F. Iacopi, "A low-power, high-accuracy with fully on-chip ternary weight hardware architecture for deep spiking neural networks," *Microprocessors and Microsystems*, vol. 90, p. 104458, Apr. 2022.
- [14] I. Aliyev and T. Adegbija, "PULSE: Parametric hardware units for low-power sparsity-aware convolution engine," in 2024 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5, May 2024.
- [15] J.-J. Lee, W. Zhang, and P. Li, "Parallel time batching: Systolic-array acceleration of sparse spiking neural computation," in 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 317–330, 2022.
- [16] Y. Chen, Y. Liu, W. Ye, and C.-C. Chang, "The high-performance design of a general spiking convolution computation unit for supporting neuromorphic hardware acceleration," *IEEE Transactions on Circuits* and Systems II: Express Briefs, vol. 70, pp. 3634–3638, Sept. 2023.
- [17] W. Fang, Y. Chen, J. Ding, Z. Yu, T. Masquelier, D. Chen, L. Huang, H. Zhou, G. Li, and Y. Tian, "SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence," *Science Advances*, vol. 9, p. eadi1480, Oct. 2023.
- [18] I. Aliyev, J. Lopez, and T. Adegbija, "Exploring the sparsity-quantization interplay on a novel hybrid SNN event-driven architecture," in 2025 Design, Automation & Test in Europe Conference (DATE), pp. 1–7, Mar. 2025.
- [19] W. Ye, S. Chen, H. Liu, Y. Liu, Y. Chen, Y. Cui, and W. Lin, "The architecture design and training optimization of spiking neural network with low-latency and high-performance for classification and segmentation," *Neural Networks*, vol. 191, p. 107790, 2025.
- [20] H. Zhao, H. Wu, D. Yang, A. Zou, and J. Hong, "Brillm: Brain-inspired large language model," arXiv preprint arXiv:2503.11299, 2025.