Co-designing a Sub-millisecond Latency Event-based Eye Tracking System with Submanifold Sparse CNN

Baoheng Zhang*   Yizhao Gao*   Jingyuan Li   Hayden Kwok-Hay So

The University of Hong Kong

{bhzhang, yzgao, jyli, hso}@eee.hku.hk

Abstract

Eye-tracking technology is integral to numerous consumer electronics applications, particularly in the realm of virtual and augmented reality (VR/AR). These applications demand solutions that excel in three crucial aspects: low-latency, low-power consumption, and precision. Yet, achieving optimal performance across all these fronts presents a formidable challenge, necessitating a balance between sophisticated algorithms and efficient backend hardware implementations. In this study, we tackle this challenge through a synergistic software/hardware co-design of the system with an event camera. Leveraging the inherent sparsity of event-based input data, we integrate a novel sparse FPGA dataflow accelerator customized for submanifold sparse convolution neural networks (SCNN). The SCNN implemented on the accelerator can efficiently extract the embedding feature vector from each representation of event slices by only processing the non-zero activations. Subsequently, these vectors undergo further processing by a gated recurrent unit (GRU) and a fully connected layer on the host CPU to generate the eye centers. Deployment and evaluation of our system reveal outstanding performance metrics. On the Event-based Eye-Tracking-AIS2024 dataset, our system achieves 81% p5 accuracy, 99.5% p10 accuracy, and 3.71 Mean Euclidean Distance with 0.7 ms latency while only consuming 2.29 mJ per inference. Notably, our solution opens up opportunities for future eye-tracking systems. Code is available at https://github.com/CASR-HKU/ESDA/tree/eye_tracking.

1. Introduction

Eye tracking, the monitoring and analysis of eye movement and focus, provides valuable insights into visual attention, cognitive processes, and human-machine interaction. With applications ranging from psychology to marketing, eye tracking enables a deeper understanding of human behavior.

---
*Equal contribution

For example, eye tracking plays a pivotal role in enhancing immersion and interaction in augmented and virtual reality (AR/VR) [11, 14]. Also, it enables researchers to investigate visual perception, study information processing, optimize user interfaces, and enhance the design of human-computer interactions [20, 22].

A standard eye-tracking system is typically housed within embedded and wearable devices, necessitating comprehensive consideration of overall system performance, encompassing factors such as latency, power consumption, and precision. Ensuring low latency guarantees real-time responsiveness and fluid interaction, thereby elevating the user experience. Low power consumption stands as a cornerstone for portable and wearable devices, facilitating prolonged usage periods without the need for frequent recharging. Precision remains paramount for capturing and analyzing eye movements accurately, enabling accurate gaze tracking for diverse applications.

However, achieving optimal performance in all three aspects poses a significant challenge. For instance, in past methodologies employing a frame-based camera in conjunction with a dense Deep Neural Networks (DNN) model, the system may incur noticeable latency and power consumption despite achieving satisfactory accuracy with the latency of 25 ms [31].

To this end, leveraging an event-based camera emerges as a promising solution. Unlike traditional cameras that capture images at fixed time intervals, event cameras detect light intensity changes on a per-pixel basis. This inherently sparse output significantly diminishes the data rate and potential backend processing demands. Moreover, the asynchronous nature of event cameras enables high temporal resolution, which offers a compelling pathway toward achieving low latency and precise eye movement tracking. Nevertheless, to fully exploit the sparsity and the high speed of event cameras, especially in deep learning models, exhibits great challenges. Off-the-shelf hardware platforms, such as GPU and CPU, usually fail to deliver satisfying performances in event processing.

In this work, we provide an efficient **S**parse **E**vent-based

Eye-tracking system called **SEE** by using hardware-software co-design that centralizes the idea of leveraging spatial sparsity. Our system adopts the submanifold sparse convolution neural network (SCNN) to efficiently extract feature vectors from the voxel grid representation of events. The SCNN model is deployed on an FPGA dataflow accelerator that can efficiently operate on sparse activations. The extracted features from the SCNN backbone are then fed into a combination of a GRU+FC module (implemented on the host CPU) to generate the normalized location of the eye center.

Our system is implemented end-to-end on an embedded FPGA SoC and extensively evaluated on the Event-based Eye-Tracking-AIS2024 dataset [1, 6, 36]. Notably, our approach can achieve more than 98% p10 accuracy with 0.7 ms - 0.94 ms inference latency. In comparison with an embedded GPU, our system achieves up to $15.4\times$ and $77.1\times$ speedup compared to standard and submanifold sparse convolution implementations, respectively.

2. Related Work

2.1. Eye tracking

Eye tracking is a technology that involves monitoring and measuring the movement and focus of a person's eyes. It is used to understand and analyze human visual attention, gaze behavior, and eye movements. The process typically involves capturing and analyzing data related to the position, motion, and duration of eye fixations and saccades (rapid eye movements).

Traditional eye-tracking algorithms focus on using image processing methods to extract the center of the eye pupil. For example, [35] uses Harr-like features, K-means, and RANSAC-based ellipse fitting to recognize the pupil. [13] follows a three-step process, using contour segmentation to extract the pupil center. However, these methods are hard to deploy in real scenarios. As pointed out in [23], most of them are developed in controlled environments, and they always fail in some extreme environments, like changing view angles and illuminations.

With the development of deep learning, convolution neural networks (CNN) gradually become the most dominant method in solving computer vision tasks. CNN-based deep learning methods also become the mainstream to solve eye-tracking tasks, achieving much better performance than the traditional methods. For example, PupilNet [12] follows a coarse-to-fine mechanism, using a coarse CNN to obtain sub-regions and a fine CNN to generate the final response. [8] proposed a cascaded pipeline, using SSD [27] to detect the face, CycleGAN [39] to remove the glasses, and FCN [28] to estimate the eye center location.

Despite the notable advancements in accuracy, the efficiency of the algorithm remains a significant challenge. On one hand, the limited frame rate of traditional cameras hampers the system's capability to capture images at a high frequency. On the other hand, the proposed models demonstrate excessive complexity and high computational demands [7, 8, 17, 37], making them difficult to deploy in a real-time system. While some other studies have focused on improving eye-tracking efficiency [21, 24, 31], the latency still exceeds 10 ms. Event-based eye-tracking has recently gained attention as an emerging direction due to its advantages of low latency and low power consumption. However, despite the low latency offered by these approaches, recent works in the field primarily rely on traditional methods [4, 33], which often exhibit reduced accuracy.

2.2. Acceleration of Event-based DNN Models

As the era of deep learning, event vision has achieved remarkable progress in image classification [29, 34], object detection [16, 19], optical flow estimation [25, 30], etc. Despite the high-speed nature of the event sensor, many deep-learning-based event-based solutions are still suffering from the heavy computation. For instance, [29] proposed using a Transformer model for image classification with over 10 ms latency, while some other Convolutional Neural Network (CNN) based solutions [19] can only achieve 7 ms inference using server-level GPUs like V100.

Typically, the deep-learning-based solutions first convert the event streams into 2D images [26] or 3D voxels [5] and use dense models on GPU, losing the spatial sparsity of event streams. To address this challenge from a bottom-up approach, some previous works design special hardware accelerators to explore the sparsity [3, 15] in the model. NullHop [3] introduced an architecture that employs a binary bitmap to depict sparse activation on a per-layer basis, thereby avoiding unnecessary computations for zero values. ESDA [15] proposes an all-on-chip sparse dataflow architecture on FPGA for low-latency and energy-efficient processing of event-based DNN models.

In this work, we build an efficient solution for eye-tracking problems by extending the ESDA framework with additional support of recurrent modules. Through the integration of software-hardware optimization techniques, our model achieves satisfactory accuracy and high hardware efficiency, with an overall latency less than 1 millisecond.

3. Method

3.1. Overall Architecture

To address the eye-tracking problem efficiently, we propose SEE, a hardware-software co-optimization solution. On the software side, our model comprises a SCNN-based backbone for feature extraction, a GRU layer for temporal feature fusion, and a fully connected (FC) layer for eye center regression. Our hardware is heterogeneous, as the FPGA programmable fabric is used for SCNN acceleration and Arm
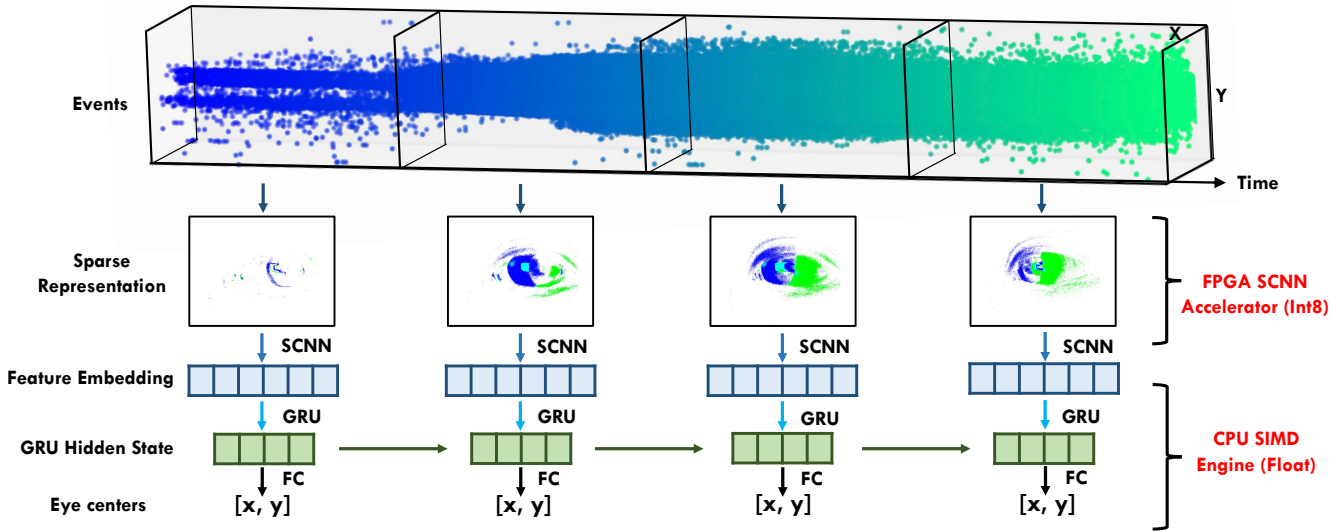
Figure 1. Software architecture: For an event stream, we partition it into multiple consecutive clips. These clips are then transformed into sparse voxel representations. Subsequently, an SCNN is used to generate feature embeddings, which will be then fed into a Gated Recurrent Unit (GRU) module. The GRU module generates the hidden state, and a Fully Connected (FC) layer regresses the eye centers.

Cortex-A series for GRU and FC layers. This heterogeneous architecture allows us to fully exploit the strengths of different hardware devices and deliver an overall low-latency performance. In addition, we also employ hardware-software co-optimization to search for compact models with better tradeoffs between accuracy and hardware latency.

3.2. Software design

3.2.1 Model Architecture

SEE follows the standard voxel grid representations as the input. As depicted in Figure 1, the event clips in a fixed-time interval usually are spatially sparse, which means most of the pixels are zero. These sparse inputs are fed into the SCNN backbone to extract global features. Subsequently, these features undergo further processing through a GRU layer, which captures the temporal information between event frames. The hidden features are then fed into the FC layer, yielding the normalized coordinates of the eye center location, ranging from 0 to 1. The actual eye location pixel coordinates can be obtained directly by multiplying these normalized coordinates with the height and width of the input size.

3.2.2 Submanifold Sparse Convolution

Convolutional layer (standard convolution) has been widely used in all kinds of deep learning architecture. However, the standard convolution algorithm suffers from a dilation effect when taking the spatially sparse input. Here, the spatial sparsity means some pixels in the input image or activations

$$M = A*k_3 + B*k_7 + C*k_5$$

$$N = D*k_3 + E*k_7$$

(a) Standard Convolution with kernel size 3*3

$$M=0$$

$$N = D*k_3 + E*k_7$$

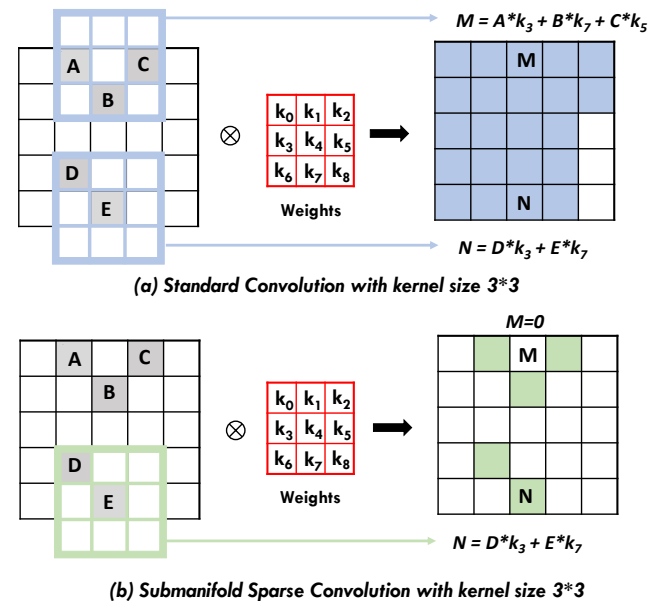(b) Submanifold Sparse Convolution with kernel size 3*3

Figure 2. Standard and submanifold sparse convolution. For standard convolution, all the pixels in an image are processed by the kernels equally, leading to the dilation of spatial intensity. On the other hand, submanifold sparse convolution ensures that the output non-zero pixels locations to be identical as the input.

are completely zero for all the channels. As depicted in Figure 2a, the dilation effect causes the output feature map to be much denser than the input.

To address this issue and preserve spatial sparsity, we

incorporate submanifold sparse convolution [9] originally developed for point cloud networks. As illustrated in Figure 2b, submanifold sparse convolution only outputs non-zeros on identical input non-zero locations. On a valid output location like N, the computation is exactly the same as that of standard convolution. In this way, the sparsity is preserved while meaningful information is propagated.

By leveraging submanifold sparse convolution, we not only mitigate the dilation effect but also enhance the efficiency of inference by reducing unnecessary computations on irrelevant areas.

3.2.3 Quantization

Integer operations are more efficient and require fewer hardware resources for FPGA implementation. To deploy resource-efficient integer arithmetic on FPGA, we adopt HAWQv3 [38] to fine-tune our model, which allows integer-only inference. Specifically, the models tuned by HAWQv3 only require integer multiplication, addition, and shift to be used in the whole computational graph. In our experiments, we first train the model using the float32 data type as standard practice and perform fine-tuning by applying int8 quantization on both input X and weights W of the SCNN backbone. The quantization scheme can be expressed as:

$$Y = S_y \hat{Y} = W \times X = S_w \hat{W} \times S_x \hat{X}$$
$$\hat{Y} = \frac{S_w S_x}{S_y}(\hat{W} \times \hat{X}) = \frac{\hat{S}}{2^n}(\hat{W} \times \hat{X}) \qquad (1)$$

where tensors with ˆ are in integer format. In this dyadic quantization scheme, the division of scaling factor $\frac{S_w S_x}{S_y}$ is replaced by an additional level of quantization with integer multiplication and shift operations (similar to a fixed point format). This allows simple hardware arithmetic to be deployed on the accelerator.

3.3. Hardware design

3.3.1 Overall Architecture

The hardware diagram is shown in Figure 3, which is built upon a Xilinx Zynq UltraScale+ MPSoC device. The proposed hardware system primarily consists of two components: the sparse dataflow SCNN accelerator and the Arm Cortex-A53 processor host. The event-based input is initially fed into the SCNN accelerator to propagate through the submanifold sparse convolutional neural network backbone. Subsequently, the GRU and fully connected layers processes are executed by the host CPU with the Arm NEON SIMD (Single Instruction, Multiple Data) engine.

3.3.2 FPGA SCNN Accelerator

The FPGA SCNN accelerator adopts a dynamic sparse dataflow architecture introduced in [15]. This dataflow accelerator maps all the layers spatially on-chip and pipelines the computation of sparse activations for different modules. The dataflow modules share a unified token-feature streaming interface. A token $[.x, .y, .end]$ marks the current non-zero pixel coordinates. In a nutshell, the design of a dataflow module should comply with three principles: (1) it has the logic to resolve the next non-zero pixel coordinates; (2) it has the logic to compute the corresponding features at the next non-zero pixel; (3) the streaming order of non-zeros should follow the left-to-right, top-to-bottom manner. In this way, different model components, such as conv 1x1, conv 3x3, and pooling layers, can be implemented and cascaded in the dataflow manner, allowing sparse token-features to propagate throughout layers.

Figure 3 shows an example diagram of a submanifold sparse conv 3x3 layer. It's composed of a Sparse Line Buffer (SLB) and a compute engine. Since the submanifold sparse convolution has identical input and output non-zero locations, the tokens can be simply buffered and reused by using a token FIFO. The head and tail tokens are used to control the read and write operations of the buffer. In addition, a kernel offset stream is used to exploit the sparsity within each 3x3 kernel. For example, only the offsets 2, 4, and 6 in the snapshot contain non-zero pixels. The kernel offsets subsequently serve as the index of the weight buffer in the later compute engine.

This sparse dataflow scheme allows the non-zero information/features to be streamed and passed through different modules in the accelerator in an efficient pipeline. As discussed above, weight and activation are quantized into 8-bit integers to allow integer arithmetic units to be deployed while reducing memory consumption. Weights are stored in on-chip BRAM statically to reduce the off-chip communication in our design. However, one potential disadvantage of this approach is that model size can be limited by the on-chip buffer size. Fortunately, we have incorporated a co-optimization framework to trade off between model size and performance, which will be discussed in later sections.

3.3.3 CPU SIMD Implementation of GRU+FC

The main reason for deploying the GRU layer on the CPU is because its complex sigmoid activation functions are difficult to quantize and deploy on FPGA. Fortunately, the Arm SIMD engine has built-in floating point arithmetic units that are capable of handling the remaining GRU and FC layer within a reasonable time.

The GRU and FC layers are implemented using the Eigen [18] C++ library. The computations involving vector operations are realized using several Arm NEON SIMD instructions. The compiled dynamic link library is packaged into Python and integrated with the host PYNQ (Python productivity for Zynq) platform [2].
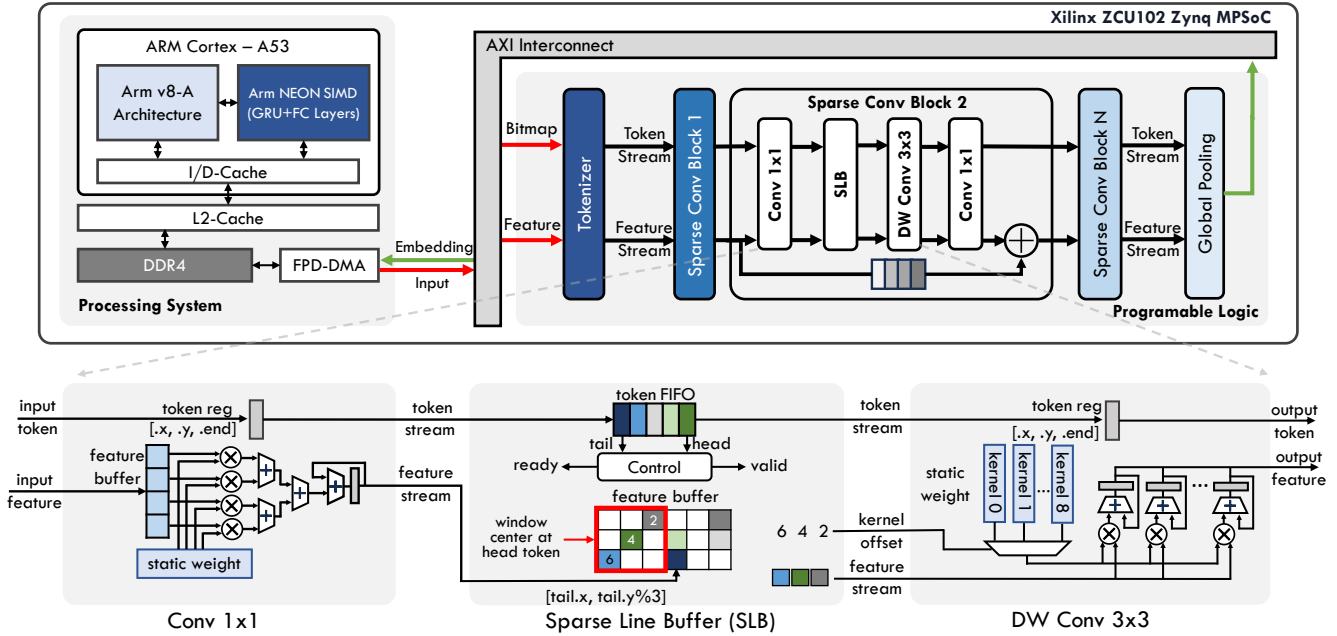
Figure 3. Heterogeneous Hardware architecture. The proposed hardware system primarily consists of the Arm Cortex-A53 acting as a processing system and the SCNN accelerator implemented on programable logic. The input to the SCNN accelerator is the sparse features and a binary bitmap to record the non-zero pixel locations. The GRU and fully connected layers are executed in the processing system with Arm NEON SIMD (Single Instruction, Multiple Data) engine.

By combining the FPGA's parallel computing capability with the Arm Cortex-A series processor's efficient processing of SIMD operations, the proposed system optimizes the utilization of computational resources on Xilinx MPSoC platforms and maximizes both performance and efficiency for real-time eye-tracking applications.

3.4. Software-hardware Co-optimization

Our system requires the entire backbone to fit the on-chip buffer. To achieve this objective, we have developed a software-hardware co-searching framework that aims to generate a compact network by considering both network complexity and hardware resource allocation, which is illustrated in Figure 4. In this framework, we utilize MobileNetV2[32] as a supernet and sample a large number of subnets. The searching space can be divided into four aspects: 1) the number of inverted bottleneck blocks, 2) the channel size of each block, 3) the ratio of expansion for each block, 4) the hidden feature size of the GRU layer.

In the next stage, we select the candidate network architectures using a hardware simulator AGNA[10]. Given a model definition, the simulator uses a Geometric Programming method to estimate the latency based on the hardware constraints. Finally, the models with both lower estimated latency and feasible parameter sizes will be trained. In this pool of trained models, we select the ones lie within the pareto-frontier of accuracy and latency trade-offs.
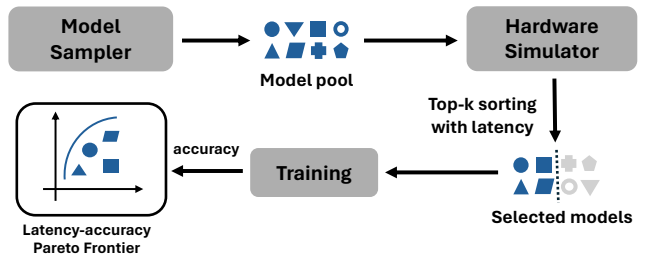

Figure 4. Network searching pipeline: We sample networks from a search pool and use a hardware simulator to select low-latency ones. After training these networks, we create a latency-accuracy Pareto frontier to show the trade-off between accuracy and latency.

4. Experiments

4.1. Implementation Details

We conducted an efficiency verification of our design using a recently released dataset, the Event-based Eye-Tracking-AIS2024 dataset. The dataset comprises a total of 13 subjects, with each subject having 2-6 recording sessions. The subjects were instructed to perform activities belonging to 5 different classes, including random, saccades, read text, smooth pursuit, and blinks. We use the default split of the training and validation set.

The evaluation metrics are "Mean Euclidean Distance" (Dist.) and "pk accuracy". Dist. is the average distance be-

Table 1. Accuracy between standard convolution and submanifold sparse convolution.

	p5 accuracy (%)		p10 accuracy (%)	
	Standard	Sparse	Standard	Sparse
MobileNetV2	87.42	87.63	99.39	99.46
SEE-B	84.87	85.21	99.13	98.86

tween ground truth and predicted locations, while "pk accuracy" denotes the accuracy within a tolerance of k pixels. Specifically, if the Euclidean distance between the ground truth and predicted locations is smaller than k pixels, the sample is considered correct and vice versa. We utilize $k = 5$ and $k = 10$ to measure the prediction accuracy.

In terms of the hardware system, We implemented our hardware design with Vitis HLS 2020.2 and Vivado 2020.2. Then the proposed heterogeneous system is implemented and evaluated on a ZCU102 board with a Xilinx Zynq Ultra-Scale+ MPSoC Device.

In the subsequent sections, we refer to the models trained as the "SEE-series" models, denoting from SEE-A to SEE-D with different performance tradeoffs. MobileNetV2 (width multiplier = 0.5) is utilized as the baseline for comparison.

4.2. Standard vs. Submanifold Sparse Convolution

To demonstrate the model capability of submanifold sparse convolution, we carried out experiments to compare its performance with standard convolution. We use 2 different models including MobileNetV2 baseline and the SEE-B for the experiments, which are trained with standard and submanifold sparse convolution respectively. The results in Table 1 demonstrate that the p5 and p10 accuracy between the standard and submanifold implementations exhibit similarity. Specifically, the submanifold sparse convolution consistently achieves a comparable result in both p5 and p10 accuracy. However, this advantage comes with a notable increase in activation sparsity.

4.3. Latency and Accuracy

To demonstrate the effectiveness of our design, we follow our optimization flow in Figure 4 to generate 20+ different models. The accuracy and latency results are plotted in Figure 5, while the subgraph (a) and (b) show the latency with p5 and p10 accuracy respectively.

When evaluating with p10 accuracy, we observe that the MobileNetV2 and the SEE-series networks achieve comparable high accuracies, mostly exceeding 98%. While considering the p5 accuracy and the mean Euclidean distance, the baseline MobileNetV2 slightly outperforms the SEE-series models. This difference could be attributed to the higher number of network parameters since a larger model size generally provides more capacity to capture richer features. In terms of efficiency, our selected SEE-series model

significantly outperforms MobileNetV2 by a large margin. MobileNetV2 achieves a latency of 1.4 ms, which is more efficient than the previous work. However, our SEE-series model can even achieve a latency of less than 1 ms. Specifically, our SEE-D model achieves a comparable accuracy with MobileNetV2, with around $2\times$ speedup (0.7 ms vs. 1.45 ms). Our SEE-C model (0.6 ms) achieves around $2.5\times$ speedup over MobileNetV2 with only 1% p10 accuracy drops. This highlights the capability of our SEE-framework to push more optimal latency accuracy trade-offs than baseline.

4.4. Hardware Implementation Details

We also conduct further evaluations for our hardware implementation. We record the hardware-related parameters during the experiments, including resource utilization, power, and efficiency. The results are presented in Table 2. Notably, the SEE-series models consistently achieve low latency, with all inference times falling within the 1ms range. Additionally, our models consume lower power and demonstrate superior energy efficiency, as indicated by the reduced mJ per inference metric. These findings demonstrate the effectiveness of our approach in achieving low latency and low power consumption specifically for eye-tracking tasks.

Our system provides a wide spectrum of performance tradeoffs. The SEE-A model obtains the highest p10 accuracy with more power, and the SEE-C model achieves the best overall latency and efficiency at the cost of slight degradation in accuracy. On the contrary, the SEE-B and the SEE-D models strike a more balanced tradeoff between accuracy and efficiency.

4.5. Compare with Embedded GPUs

Finally, we conducted evaluations of our design using the NVIDIA Jetson Xavier NX, a widely-used embedded GPU. Similar to ESDA, we assessed both the dense DNN implementation using PyTorch and the submanifold sparse DNN implementation utilizing the MinkowskiEngine library. We calculated the average latency (batch=1) of the entire test set for the three settings, while the latency of standard implementation is defined as the baseline.

The results are shown in Figure 6. Our SEE implementation achieves a notable speedup ranging from $11.47\times$ to $13.89\times$ compared with the standard one. While compared to the submanifold GPU implementation, the speedup can reach $57.4\times$, $66.1\times$, $72.6\times$, $68.9\times$, and $66.2\times$. The remarkable speedups highlight the significant efficiency improvement achieved by the co-designed hardware accelerator compared to both the standard and submanifold GPU implementations. The GPU implementation of submanifold sparse convolution typically exhibits slower performance than the standard dense baseline. This is primarily due to the significant overhead of sparse coordinate bookkeeping, particularly noticeable during batch 1 inference.

Table 2. Hardware Implementation Details.

|	Accuracy (%)		Dist.	#	Latency (ms)			Power	Efficiency	Utilization			
	p5	p10	(Pixel)	Param.	SCNN	GRU&FC	Total	(W)	(mJ/inf.)	DSP	BRAM	FF	LUT
MobileNetV2	**87.36**	99.53	**3.15**	797K	0.73	0.72	1.45	4.36	3.23	2123	1685	213K	214K
SEE-A	80.83	**99.60**	3.77	465K	**0.49**	0.15	0.64	4.05	1.99	2003	1287	114K	166K
SEE-B	83.32	99.53	3.39	372K	0.79	0.15	0.94	4.17	3.28	2067	1547	117K	170K
SEE-C	75.92	98.39	4.05	180K	**0.49**	**0.11**	**0.60**	**3.86**	**1.88**	1880	1001	94K	135K
SEE-D	81.37	99.53	3.71	**178K**	0.59	**0.11**	0.70	**3.86**	2.29	1606	1092	90K	130K
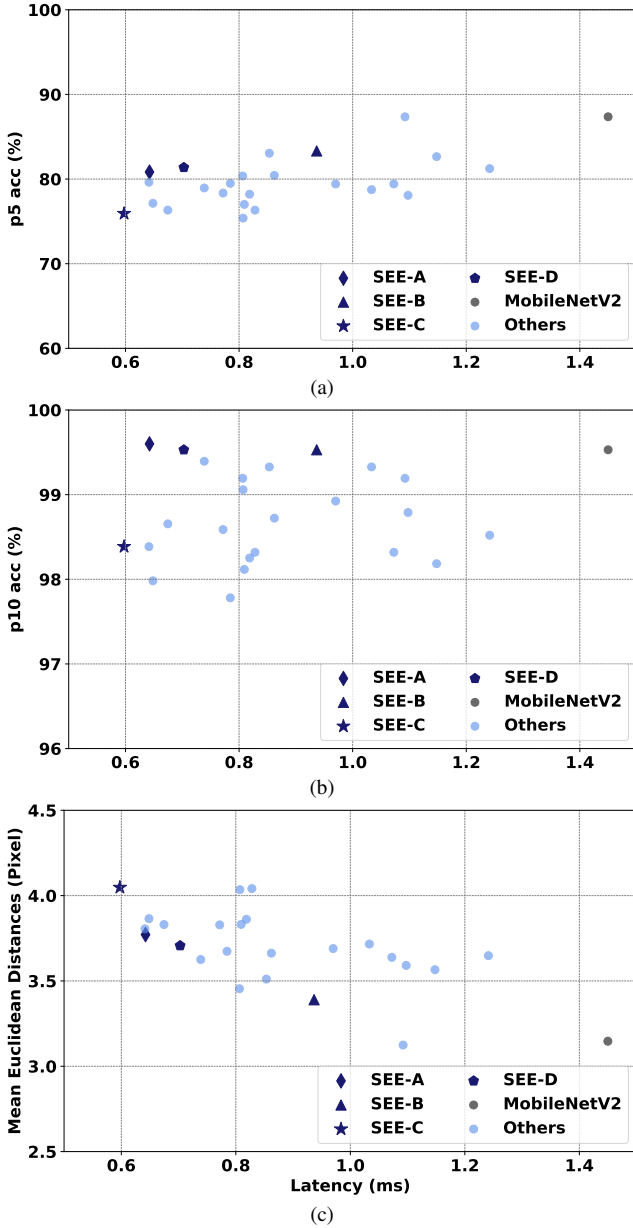
Figure 5. Accuracy vs Latency for sampled models. (a) p5 Accuracy vs. Latency. (b) p10 Accuracy vs. Latency. (c) Mean Euclidean Distances vs. Latency
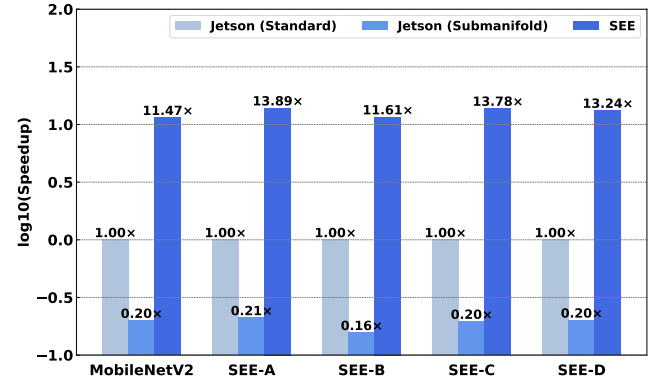

Figure 6. Latency speedup of SEE over an Nvidia Jetson GPU. We measure the latency (batch size = 1) for the standard and sub-manifold implementation using NVIDIA Jetson Xavier NX under MobileNetV2 and SEE-series network and calculate the speedup.

5. Conclusion and Future Work

We present an efficient event-based eye-tracking solution called SEE through software/hardware co-design. SEE models utilize an SCNN backbond for feature extraction, followed by a GRU+FC component for temporal fusion and eye center localization. SEE system leverages the heterogeneous hardware resource of an embedded FPGA SoC platform and accelerates the SCNN using a novel sparse dataflow accelerator. Furthermore, a hardware-software co-optimization framework is developed to obtain compact models optimal accuracy and latency tradeoffs. The results demonstrate impressive system performance, with a latency 0.6 ms to 0.94 ms for each prediction with around 99% p10 accuracy. The overall latency speedups can reach $11.2\times$ to $72.6\times$ when compared to an embedded GPU.

Despite the outstanding performance SEE achieved, we aim to enhance the further latency performances by integrating the recurrent module or attention modules into our FPGA dataflow accelerator. This endeavor necessitates the development of novel quantization techniques or the implementation of some non-linear functions, as well as the support of inter-batch pipeline.

6. Acknowledgment

References

[1] Event-based eye tracking - ais2024 cvpr workshop. https://www.kaggle.com/competitions/event-based-eye-tracking-ais2024/. 2

[2] PYNQ — pynq.io. https://www.pynq.io/. 4

[3] Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Iulia-Alexandra Lungu, Moritz B Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu, et al. Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps. *IEEE transactions on neural networks and learning systems*, 30(3):644–656, 2018. 2

[4] Anastasios N Angelopoulos, Julien NP Martel, Amit PS Kohli, Jorg Conradt, and Gordon Wetzstein. Event based, near eye gaze tracking beyond 10,000 hz. *arXiv preprint arXiv:2004.03577*, 2020. 2

[5] Patrick Bardow, Andrew J Davison, and Stefan Leutenegger. Simultaneous optical flow and intensity estimation from an event camera. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 884–892, 2016. 2

[6] Qinyu Chen, Zuowen Wang, Shih-Chii Liu, and Chang Gao. 3et: Efficient event-based eye tracking using a change-based convlstm network. In *2023 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–5. IEEE, 2023. 2

[7] Warapon Chinsatit, Takeshi Saitoh, et al. Cnn-based pupil center detection for wearable gaze estimation system. *Applied Computational Intelligence and Soft Computing*, 2017, 2017. 2

[8] Jun Ho Choi, Kang Il Lee, and Byung Cheol Song. Eye pupil localization algorithm using convolutional neural networks. *Multimedia Tools and Applications*, 79(43):32563–32574, 2020. 2

[9] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3075–3084, 2019. 4

[10] Yuhao Ding, Jiajun Wu, Yizhao Gao, Maolin Wang, and Hayden Kwok-Hay So. Model-platform optimized deep neural network accelerator generation through mixed-integer geometric programming. In *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 83–93, 2023. 5

[11] Ajoy S Fernandes, T Scott Murdison, and Michael J Proulx. Leveling the playing field: A comparative reevaluation of unmodified eye tracking as an input and interaction modality for vr. *IEEE Transactions on Visualization and Computer Graphics*, 29(5):2269–2279, 2023. 1

[12] Wolfgang Fuhl, Thiago Santini, Gjergji Kasneci, and Enkelejda Kasneci. Pupilnet: Convolutional neural networks for robust pupil detection. *arXiv preprint arXiv:1601.04902*, 2016. 2

[13] Wolfgang Fuhl, Thiago C Santini, Thomas Kübler, and Enkelejda Kasneci. Else: Ellipse selection for robust pupil detection in real-world environments. In *Proceedings of the ninth biennial ACM symposium on eye tracking research & applications*, pages 123–130, 2016. 2

[14] Wolfgang Fuhl, Gjergji Kasneci, and Enkelejda Kasneci. Teyed: Over 20 million real-world eye images with pupil, eyelid, and iris 2d and 3d segmentations, 2d and 3d landmarks, 3d eyeball, gaze vector, and eye movement types. In *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 367–375. IEEE, 2021. 1

[15] Yizhao Gao, Baoheng Zhang, Yuhao Ding, and Hayden Kwok-Hay So. A composable dynamic sparse dataflow architecture for efficient event-based vision processing on fpga. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, page 246–257, New York, NY, USA, 2024. Association for Computing Machinery. 2, 4

[16] Mathias Gehrig and Davide Scaramuzza. Recurrent vision transformers for object detection with event cameras. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13884–13893, 2023. 2

[17] Chao Gou, Hui Zhang, Kunfeng Wang, Fei-Yue Wang, and Qiang Ji. Cascade learning from adversarial synthetic images for accurate pupil detection. *Pattern Recognition*, 88:584–594, 2019. 2

[18] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. 4

[19] Ryuhei Hamaguchi, Yasutaka Furukawa, Masaki Onishi, and Ken Sakurada. Hierarchical neural memory network for low latency event processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22867–22876, 2023. 2

[20] Dan Witzner Hansen and Qiang Ji. In the eye of the beholder: A survey of models for eyes and gaze. *IEEE transactions on pattern analysis and machine intelligence*, 32(3):478–500, 2009. 1

[21] Sangwon Kim, Mira Jeong, and Byoung Chul Ko. Energy efficient pupil tracking based on rule distillation of cascade regression forest. *Sensors*, 20(18):5141, 2020. 2

[22] Ahmad F Klaib, Nawaf O Alsrehin, Wasen Y Melhem, Haneen O Bashtawi, and Aws A Magableh. Eye tracking algorithms, techniques, tools, and applications with an emphasis on machine learning and internet of things technologies. *Expert Systems with Applications*, 166:114037, 2021. 1

[23] Andoni Larumbe-Bergera, Gonzalo Garde, Sonia Porta, Rafael Cabeza, and Arantxa Villanueva. Accurate pupil center detection in off-the-shelf eye tracking systems using convolutional neural networks. *Sensors*, 21(20):6847, 2021. 2

[24] Kang Il Lee, Jung Ho Jeon, and Byung Cheol Song. Deep learning-based pupil center detection for fast and accurate eye tracking system. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX 16*, pages 36–52. Springer, 2020. 2

[25] Haotian Liu, Guang Chen, Sanqing Qu, Yanping Zhang, Zhi-jun Li, Alois Knoll, and Changjun Jiang. Tma: Temporal motion aggregation for event-based optical flow. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9685–9694, 2023. 2

[26] Min Liu and Tobi Delbruck. Adaptive time-slice block-matching optical flow algorithm for dynamic vision sensors. BMVC, 2018. 2

[27] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision– ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016. 2

[28] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 2

[29] Yansong Peng, Yueyi Zhang, Zhiwei Xiong, Xiaoyan Sun, and Feng Wu. Get: group event transformer for event-based vision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6038–6048, 2023. 2

[30] Wachirawit Ponghiran, Chamika Mihiranga Liyanagedera, and Kaushik Roy. Event-based temporally dense optical flow estimation with sequential learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9827–9836, 2023. 2

[31] Nikolaos Poulopoulos and Emmanouil Z Psarakis. A real-time high precision eye center localizer. *Journal of Real-Time Image Processing*, 19(2):475–486, 2022. 1, 2

[32] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zh-moginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 5

[33] Timo Stoffregen, Hossein Daraei, Clare Robinson, and Alexander Fix. Event-based kilohertz eye tracking using coded differential lighting. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2515–2523, 2022. 2

[34] Linhui Sun, Yifan Zhang, Ke Cheng, Jian Cheng, and Hanqing Lu. Menet: A memory-based network with dual-branch for efficient event stream processing. In *European Conference on Computer Vision*, pages 214–234. Springer, 2022. 2

[35] Lech Świrski, Andreas Bulling, and Neil Dodgson. Robust real-time pupil tracking in highly off-axis images. In *Proceedings of the symposium on eye tracking research and applications*, pages 173–176, 2012. 2

[36] Zuowen Wang, Chang Gao, Zongwei Wu, Marcos V. Conde, Radu Timofte, Shih-Chii Liu, Qinyu Chen, et al. Event-Based Eye Tracking. AIS 2024 Challenge Survey. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2024. 2

[37] Zheng Xiang, Xinbo Zhao, and Aiqing Fang. Pupil center detection inspired by multi-task auxiliary learning characteristic. *Multimedia Tools and Applications*, 81(28):40067–40088, 2022. 2

[38] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pages 11875–11886. PMLR, 2021. 4

[39] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017. 2