EyeCoD: Eye Tracking System Acceleration via FlatCam-based Algorithm & Accelerator Co-Design

Haoran You[1]* Cheng Wan[1]* Yang Zhao[1]* Zhongzhi Yu[1]* Yonggan Fu[1] Jiayi Yuan[1]
Shang Wu[1] Shunyao Zhang[1] Yongan Zhang[1] Chaojian Li[1] Vivek Boominathan[1]
Ashok Veeraraghavan[1] Ziyun Li[2] Yingyan Lin[1]
[1]Rice University [2]Meta Reality Labs
{hy34, chwan, zy34, zy42, yf22, jy101, sw99, sz74, yz87, cl114, vivekb, vashok, yingyan.lin}@rice.edu
liziyun@fb.com

ABSTRACT

Eye tracking has become an essential human-machine interaction modality for providing immersive experience in numerous virtual and augmented reality (VR/AR) applications desiring high throughput (e.g., 240 FPS), small-form, and enhanced visual privacy. However, existing eye tracking systems are still limited by their: (1) large form-factor largely due to the adopted bulky lens-based cameras; (2) high communication cost required between the camera and backend processor; and (3) potentially concerned low visual privacy, thus prohibiting their more extensive applications. To this end, we propose, develop, and validate a lensless FlatCam-based eye tracking algorithm and accelerator co-design framework dubbed **EyeCoD** to enable eye tracking systems with a much reduced form-factor and boosted system efficiency without sacrificing the tracking accuracy, paving the way for next-generation eye tracking solutions. **On the system level**, we advocate the use of lensless FlatCams instead of lens-based cameras to facilitate the small form-factor need in mobile eye tracking systems, which also leaves rooms for a dedicated sensing-processor co-design to reduce the required camera-processor communication latency. **On the algorithm level**, EyeCoD integrates a predict-then-focus pipeline that first predicts the region-of-interest (ROI) via segmentation and then only focuses on the ROI parts to estimate gaze directions, greatly reducing redundant computations and data movements. **On the hardware level**, we further develop a dedicated accelerator that (1) integrates a novel workload orchestration between the aforementioned segmentation and gaze estimation models, (2) leverages intra-channel reuse opportunities for depth-wise layers, (3) utilizes input feature-wise partition to save activation memory size, and (4) develops a sequential-write-parallel-read input buffer to alleviate the bandwidth requirement for the activation global buffer. On-silicon measurement and extensive experiments validate that our EyeCoD consistently reduces both the communication and computation costs, leading to an overall system speedup of 10.95×, 3.21×, and 12.85× over general computing platforms including CPUs and GPUs, and a prior-art eye tracking processor called CIS-GEP, respectively, while maintaining the tracking accuracy. Codes are available at https://github.com/RICE-EIC/EyeCoD.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**; **Architectures**; • **Hardware → Emerging technologies**.

KEYWORDS

Eye Tracking Systems, VR/AR, Algorithm-hardware Co-Design

ACM Reference Format:
Haoran You, Cheng Wan, Yang Zhao, Zhongzhi Yu, Yonggan Fu, Jiayi Yuan, Shang Wu, Shunyao Zhang, Yongan Zhang, Chaojian Li, Vivek Boominathan, Ashok Veeraraghavan, Ziyun Li, Yingyan Lin. 2022. EyeCoD: Eye Tracking System Acceleration via FlatCam-based Algorithm & Accelerator Co-Design. In *Proceedings of The 49th Annual International Symposium on Computer Architecture (ISCA '22).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3470496.3527443

1 INTRODUCTION

Eye tracking has emerged as a increasingly crucial component for various applications that require human-machine interactions, e.g., virtual and augmented reality (VR/AR) devices [32, 44, 46]. For example, Foveated Rendering (FR) [25] is one of the core technologies that enables immersive user experiences in VR/AR applications requiring high-performance eye tracking. In particular, FR renders a high-resolution picture only in locations where users are looking at and a low-resolution one for the remaining background. Despite their promise, existing eye tracking systems such as [5] are still limited in their achievable throughput (e.g., still < 30 FPS) and thus cannot fully satisfy the desired real time performance requirements, e.g., > 240 FPS for supporting frequent and substantial human-machine interactions in mobile AR/VR devices of limited computing resources [1, 32]. The bottlenecks are three-fold: First, on the system level, previous eye tracking systems rely on lens-based cameras that have a large form-factor especially thickness and thus can only be placed far away from the backend processor, resulting in a high communication cost between the camera and processor and thus limiting the overall system latency; Second, on the data level, the captured images often contain a significant amount of redundancy as only a small portion of the images contains human eyes; Third,

*These authors are co-first authors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ISCA '22, June 18–22, 2022, New York City, NY
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-8610-4/22/06…$15.00
https://doi.org/10.1145/3470496.3527443

on the model level, current state-of-the-art award-winning solutions for both eye segmentation (e.g., OpenEDS2019 [21]) and gaze estimation (e.g., OpenEDS2020 [35]) require deep neural networks (DNNs) with paramount (e.g., up to 16G) FLOPs.

The above bottleneck analysis of existing eye tracking solutions has uniquely motivated our system design. Specifically, for alleviating the aforementioned system-level inefficiency, lensless cameras [4, 18, 26] have emerged as promising solutions. For example, FlatCam [4] can be 5× ∼ 10× thinner and lighter than lens-based cameras by replacing the focal lenses with a coded binary mask, which encodes the incoming light instead of directly focusing it. The encoded information of FlatCam's sensing measurements can be computationally decoded to reconstruct the captured images with potentially introduced artifacts and noises during the mask fabrication and measurement processes. Furthermore, the reduced form-factor especially thickness leaves room for attaching the backend eye tracking processor to be closer to the front-end cameras, largely reducing the distance between the camera and processor and thus corresponding communication costs for reducing the overall system latency of eye tracking. For tackling the data-level inefficiency, identifying the core eye area in the captured images can potentially reduce both a large amount of computational costs in the required gaze estimation model and corresponding data storage/movement costs of the eye tracking processor. For the model-level inefficiency, a thorough algorithm and corresponding hardware accelerator design space exploration is crucial for largely improving the hardware utilization of eye tracking acceleration.

Motivated by the aforementioned bottleneck analysis and new opportunities, we advocate lensless camera based eye tracking systems for (1) alleviating the bottlenecks in existing eye tracking systems and (2) leveraging the aforementioned opportunities to largely enhance the achievable throughput of eye tracking systems, and make the following contributions:

- We propose a lensless FlatCam-based eye tracking algorithm and accelerator co-design framework dubbed EyeCoD, which aims to leverage FlatCam's much reduced form-factor to design a real-time eye tracking system (i.e., > 240 FPS) by harmonizing both algorithm- and accelerator-level innovations. Specifically, EyeCoD (1) explores the possibility of replacing lens-based cameras with lensless cameras featuring a thinner and lighter form-factor, yet without degrading the tracking accuracy, and (2) further accelerates both eye tracking computations and data movements with a dedicated accelerator attached to the lensless camera to largely reduce the overall system latency.
- On the algorithm level, EyeCoD integrates (1) a sensing-processing interface that directly encodes the first layer of eye tracking models to FlatCam's mask, and (2) a predict-then-focus pipeline that first predicts the region-of-interest (ROI) based on eye semantic segmentation and then only focuses on the ROI parts to estimate the gaze directions, largely reducing the redundant computations and data movements.
- On the hardware level, EyeCoD further develops a dedicated accelerator that can be directly attached to FlatCam for accelerating eye tracking computations and data movements, by (1) enhancing data locality via dedicated workload orchestration between the eye segmentation (predict) and gaze estimation (focus) models; (2) exploring the reuse opportunity for depth-wise layers; and (3) leveraging activation partition and memory access parallelism to save on-chip storage and off-chip bandwidth, respectively.
- On-silicon measurements and extensive experiments validate the effectiveness of our proposed EyeCoD framework. Specifically, EyeCoD leads to 10.95×, 3.21×, and 12.85× overall system speedups over general computing platforms including CPUs and GPUs, and the prior-art eye tracking processor called CIS-GEP [5], respectively, while maintaining the tracking accuracy.

2 RELATED WORKS

Eye Tracking Algorithms. Existing eye tracking algorithms include both model- and appearance-based methods. The former [41, 42] builds a geometric model for eyes to predict the corresponding gaze, including both 2D and 3D models that use near infrared (NIR) illumination to create corneal reflections to estimate the gaze vector. The latter [40] directly maps the raw pixels to the gaze angles. Appearance-based methods in general have surpassed model-based ones for eye tracking, especially when being equipped with advanced deep learning methods. Different DNN structures have been proposed to enhance the performance of gaze estimation. For example, [45] proposed the first DNN model for gaze estimation and [36] further proposed a hybrid network integrating both hourglass [34] and DenseNet [24] to leverage auxiliary supervision based on the gaze-map; [13] introduced ARE-Net, which consists of two smaller modules to first find directions from each eye individually and then estimate the reliability of each eye, respectively; [15] also defined two convolutional neural networks (CNNs) to predict head and gaze angles, respectively. In parallel, different processing pipelines have been developed with diverse focuses on the input features. For example, [45] utilized minimal context by only using grayscale eye images and head poses as inputs; [28] developed a multi-model CNN to extract information from two single eye images, including face image and face grid, for aiding the following gaze estimation; and [19] built an ensemble on top of the features extracted by two eye patches and head pose vectors, and achieved superior performance on several datasets [19, 39].

Lensless FlatCam. As traditional lens-based cameras inevitably require a certain focal length, which prohibits their applications to edge devices with stringent requirements on the form-factor, various lens-less imaging systems have been developed to alleviate the size or thickness bottleneck caused by the lens by capturing an image of a scene without physically focusing the incoming light. Generally, lensless imaging systems capture the scene either directly on the sensor or after being modulated by a mask element. In the latter cases, commonly adopted masks include phase masks [7, 38], diffusers [2], amplitude masks [4, 37], compressive samplers [23], and spatial light modulators [14, 16]. Since directly replacing lens with the aforementioned masks will lead to muddled sensor captures without any resemblance to the scene, either a recovery process is required to transform the captured information to recognizable images or some dedicated functions are adopted to achieve

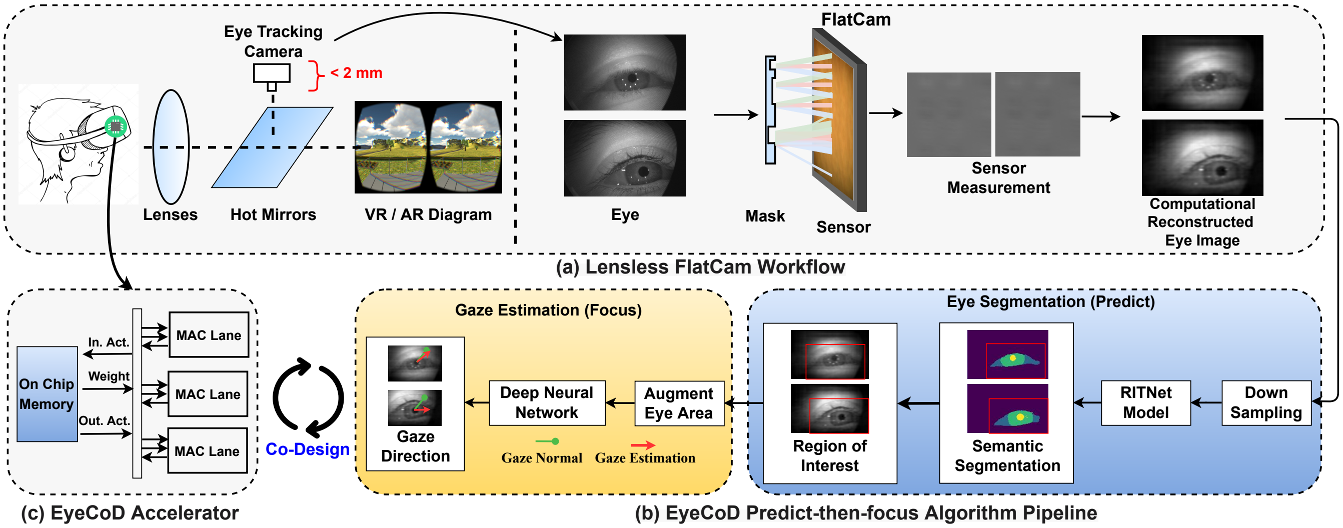


Figure 1: Overview of EyeCoD, an algorithm and accelerator co-design framework for end-to-end eye tracking acceleration.

end-to-end system goals without reconstructing corresponding recognizable images. From the privacy perspective, the captured unrecognizable images can better maintain the visual privacy compared with lens-based cameras. In this work, we adopt a specific lensless camera named FlatCam, which favors general uses and generates phase masks with desired sharp point-spread-functions (PSFs). In particular, phase masks in a FlatCam modulates the phase of incident light according to the principles of wave optics, which allow most of the light to pass through with a high signal-to-noise ratio (SNR). Therefore, FlatCam systems are particularly desirable for low light scenarios and photon-limited imaging, which is very suitable for eye tracking applications on VR/AR devices where human eyes are underexposed.

Eye Tracking Accelerators and DNN Accelerators. Various eye tracking systems with high energy-/latency-efficiency have been proposed for empowering the next-generation VR/AR devices. They are either built on top of commercial devices or supported by customized accelerators. For the former case, [8] presented an accurate infrared eye tracking system on a smartphone equipped with an infrared camera and illumination. For the latter case, [6] developed a CMOS image sensor based gaze estimation processor to reduce power consumption and [22] proposed a low-power single-chip gaze estimation sensor equipped with a novel column-parallel pupil edge detection circuit for supporting their proposed pupil edge detection algorithm, which can achieve a 2.9× power consumption reduction. A recent work [33] designed the first 3D model-based gaze estimator hardware that consumes less than 1mW power and achieves latency of 1ms per frame. In parallel, driven by the success of DNNs in the eye tracking field, there has been an increasing interest in accelerating DNN-based eye tracking systems with customized DNN accelerators [12, 17, 29, 47]. In particular, DNN accelerators have achieved impressive progress and often adopt a carefully designed memory hierarchy and PE arrays to maximize data-reuse opportunities and to enhance parallel processing via dedicated micro-architectures and algorithm-to-hardware mapping methods (i.e., dataflows). For example, representative works, such as ShiDiannao [17] and Eyeriss [12], identified the performance bottleneck caused by the required massive data movements and proposed novel architectures and dataflows that aim to maximize data reuses for reducing the energy/time cost of accessing higher cost memories.

3 EYECOD: MOTIVATION AND OVERVIEW

3.1 Why Existing Eye Tracking Solutions Are Still Inefficient

Eye tracking is known to be a core function for enabling high-quality immersive VR/AR experiences, and requires stringent requirements in terms of both real-time latency and high accuracy for gaze estimation [35]. In general, there still exists a dilemma for designing eye tracking systems: On one hand, the end-to-end system latency needs to meet real-time performance, which desires compact end-to-end processing models/pipelines which can inevitably degrade the achieved tracking accuracy; On the other hand, adopting more complex processing models/pipelines favor the achievable tracking accuracy but can lead to a large system latency of performing eye tracking. For example, a state-of-the-art ASIC eye tracking processor [5] implemented in an 65nm CMOS technology can only achieve a throughput of 30 FPS, limiting their more extensive applications [1, 31, 32].

To better understand the challenges associated with accelerating eye tracking systems, we analyze the bottlenecks from three levels of granularity: (1) On the system level, current lens-based eye tracking camera requires a large form-factor, contradicting the desired small form-factor for mobile VR/AR applications with a head-mounted display (HMD), and thus the camera often locates far away from the central processor, resulting in a high communication cost between the camera and backend processor and thus limiting the achievable end-to-end latency [1, 11]; (2) On the data level,

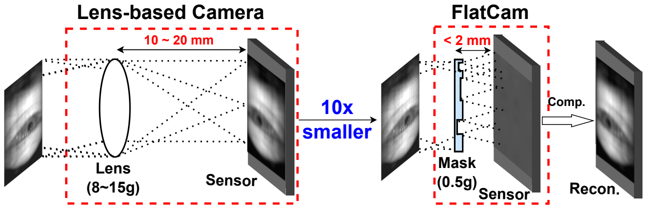


Figure 2: An illustrative comparison between lens-based cameras (left) and lensless FlatCam (right), where Comp. and Recon. denote computation and reconstruction, respectively.

there remains a nontrivial amount of redundancy in the captured images, as only a small portion of which represents human eyes, and thus corresponding redundant acceleration costs. (3) On the model level, current state-of-the-art award-winning solutions for both eye segmentation (e.g., OpenEDS2019 [21]) and gaze estimation (e.g., OpenEDS2020 [35]) require DNNs with paramount (up to 16G) FLOPs. The above analysis regarding the inefficiency and bottleneck of existing eye tracking solutions has uniquely motivated our target dedicated algorithm and accelerator co-design framework for achieving both the real-time processing (e.g., > 240FPS [32]) and the competitive tracking accuracy.

3.2 Why EyeCoD Works and Overview

Fig. 1 shows an overview of the proposed EyeCoD framework, which integrates techniques from various system granularities dedicated to tackle the aforementioned three bottlenecks and thus can largely alleviate the dilemma between the achievable eye tracking efficiency and accuracy. On the system level, we advocate the use of lensless FlatCams instead of lens-based cameras to facilitate the small form-factor needed in mobile eye tracking systems, which also leaves rooms for a dedicated sensing-processor co-design to reduce the required camera-processor communication latency. On the algorithm level, we leverage a predict-then-focus processing pipeline to first identify regions of interest (ROI) via periodic segmentation and then estimate the gaze direction only based on the extracted ROI, eliminating redundant data regions and corresponding algorithmic processing and data movements. Meanwhile, we explore eye tracking model design spaces and compression techniques on top of award-winning SOTA designs [21, 35] to additionally reduce algorithmic redundancy and corresponding acceleration cost. Finally, we develop a dedicated accelerator to leverage the resulting properties from EyeCoD's system and algorithm level optimization, further improving the overall system efficiency.

4 PROPOSED EYECOD'S SENSING AND PROCESSING PIPELINE

In this section, we present our EyeCoD's sensing and processing pipeline. Specifically, we first present (1) the preliminaries of lensless FlatCams in Sec. 4.1, (2) EyeCoD's sensing-processing interface in Sec. 4.2, and (3) EyeCoD's predict-then-focus processing pipeline and its model compression consideration in Sec. 4.3.

4.1 Preliminary of Lensless FlatCams

Replacing Lens With a Lensless Coded Mask. Our EyeCoD advocates replacing the commonly adopted lens-based cameras in eye tracking systems with lensless cameras featuring both smaller thickness and weights (see Fig. 2), in order to reduce (1) the camera-processor distance and (2) communication data volume (i.e., communicating intermediate features with smaller sizes instead of raw images with larger sizes) between the camera and processor, both leading to reduced communication cost between the camera and backend processor for eye tracking. In particular, while EyeCoD can potentially adopts various lensless cameras, in this work we consider FlatCam [3] which replaces the bulky lens in lens-based cameras with a carefully designed thin mask placed on top of the conventional sensor array. During imaging, the incident light is first encoded by the mask and each pixel in the sensor measurement records a linear combination of light from multiple directions. The imaging process can be formulated as follows:

$$y = \Phi_L x \Phi_R^T + e, \tag{1}$$

where x and y are the input light and measurement, respectively, and Φ_L and Φ_R are transfer matrices representing the coded masks and e captures the sensor noise. Replacing lens with thin masks of FlatCams can reduce the form-factor by orders of magnitude [3], making it naturally suitable for mobile eye tracking systems (e.g., VR/AR devices), where there are stringent requirements on the devices' thickness and weight.

Image Reconstruction. As shown in Fig. 2, the sensing outputs of FlatCams do not capture the target scene but its computational combination that often does not convey readable information. To facilitate the following gaze estimation, we first reconstruct the scene image (i.e., captured eyes) by solving an inverse problem of the imaging process with a $\mathcal{L}_2$ norm regularization to reduce the noise during imaging, following [3]. Specifically, the optimization goal of this image reconstruction can be formulated as:

$$\arg\min_X \|\Phi_L X \Phi_R^T - y\|_2^2 + \epsilon \|X\|_2^2, \tag{2}$$

where $\epsilon > 0$ is a regularization parameter and we optimize the reconstructed images X by minimizing the above least-square objective function following [4] towards obtaining the optimally reconstructed images X_{rec}.

Opportunities. From the aforementioned background regarding FlatCam, we can see that as compared to eye tracking systems with lens-based cameras, lensless camera based ones exhibit a great potential in terms of smaller form-factor (e.g., 5× ~ 10× thinner and >10× lighter), reduced communication costs between camera sensors and backend processors, and improved visual privacy, leading to a reduced end-to-end system latency.

4.2 EyeCoD's Sensing-processing Interface

To leverage the aforementioned opportunities offered by lensless cameras, EyeCoD's sensing-processing interface replaces both FlatCam sensing and the first layer of the following eye tracking model with direct optical edge filtering using FlatCam's coded masks, similar to [10, 20], i.e., the coded masks' optical response emulates the first layer of the following DNNs. Such a sensing-processing interface offers two-fold benefits that are highly desirable for mobile

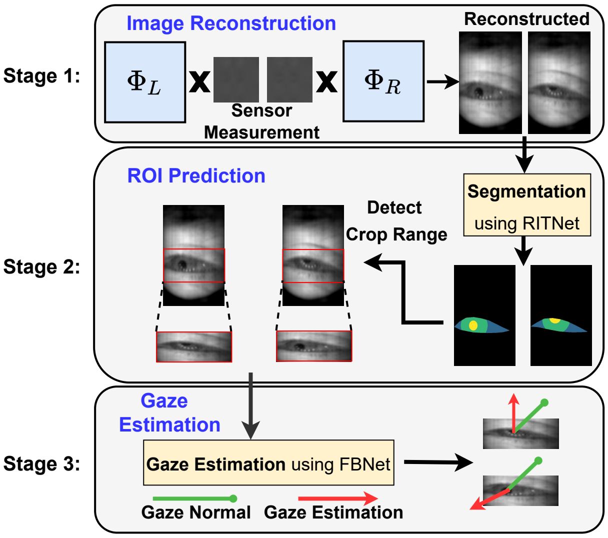


Figure 3: An overview of the proposed predict-then-focus processing pipeline.

eye tracking systems: (1) the enabled sensing-processing co-design within the lensless cameras leads to FLOPs and acceleration cost savings, thanks to the first-layer optical computation, and thus requires a lower electronic power consumption [10], especially for the UNet-like segmentation models [9] of which the first layer has to process images of the highest resolution; and (2) embedding the first-layer of the following eye tracking model into FlatCam's coded masks favors reduced sensing-processor communication volume, since the intermediate sensor measurements now enjoy reduced sizes/channels as compared to the raw images captured by lens-based cameras.

4.3 EyeCoD's Predict-then-focus Processing Pipeline

Pipeline Overview. EyeCoD's processing pipeline consists of three stages as shown in Fig 3: (1) image reconstruction as described in Sec. 4.1, (2) ROI prediction, which aims to predict the ROI centered around the human pupil in each reconstructed image, and (3) gaze estimation, which estimates the gaze based on the ROI derived from the previous stage. Our new contribution lies in the intersection between the second and third stages, where we aim to precisely predict and crop the most informative core eye area (i.e., pupil, iris, and sclera) to estimate the gaze with lower costs. Note that the ROI prediction will only needed once for every 50 frames leveraging the fact that the movement of eyes are much slower than the movement of gaze directions [35], while the gaze estimation will be continuously processed for each frame based on the latest predicted ROI. Note that the costs of ROI prediction are amortized across 50 frames, therefore, the dependent gaze estimation is operating on an ROI extracted 50~100 frames ago.

ROI Prediction. Not all pixels in each image are of the same importance to the corresponding gaze estimation. Ideally, the ROI should contain a small area with pupil, iris and sclera in the center to provide sufficient information with the lowest resolution size for estimating the gaze. However, in the captured image, skin consumes a large portion of area, which have little information for gaze estimation but consumes considerable costs during inference, providing us with a promising source of redundancy.

To reduce computational overhead, we propose to predict the ROI and then only estimate gaze based on the extracted ROI accordingly. We first use a segmentation model to segment the core eye area, of which the advantage is that it favors diverse downstream tasks including gaze estimation and thus general uses. However, the high noise in FlatCam reconstructed images (especially the sclera part) makes it more challenging for the segmentation model to precisely predict the whole core eye area than lens-based camera captured images. To address this issue, directly using segmented core eye areas as the feature for ROI prediction is likely to lead to an inaccurate result, degrading the model accuracy of gaze estimation.

Luckily, we observe that pupils have a significantly different feature than the other parts in the image, as the pupil is usually a circle with a darker color than its surrounding. Therefore, the segmentation model can correctly segment the pupil with a high confidence. Furthermore, as pupils are normally located near the center of human eyes, we propose to use the segmented pupil center as an anchor for generating the ROI. Specifically, we predict the ROI by cropping a rectangle patch centered around the pupils, where the rectangle patch's width and height are of 1.5× more than the average width and height of the segmented sclera area to cover core eye areas according to the statistics of the adopted training dataset. The predicted ROI is then passed to the gaze estimation model for generating the final eye tracking output, i.e., gaze vectors represented in a 3D coordinate system.

5 PROPOSED EYECOD'S ACCELERATOR

This section introduces our EyeCoD's accelerator design. In Sec. 5.1, we first analyze the challenges brought about by EyeCoD's predict-then-focus processing pipeline to derive the design principles of the accelerator design for further minimizing the processing latency and maximizing energy efficiency, and then describe the proposed accelerator with dedicated optimizations in Sec. 5.2.

5.1 Design Challenges and Principles

Design Challenges. As mentioned in Sec. 4.3, EyeCoD's predict-then-focus processing pipeline consists of two DNN models, i.e., a segmentation model for ROI prediction and a gaze estimation model for predicting gazes (see Fig. 3), to collaboratively construct an end-to-end eye tracking pipeline, aiming for both higher eye tracking accuracy and model efficiency. This means that EyeCoD's accelerator is required to efficiently accelerate both of the above segmentation and gaze estimation models that feature diverse model structures as well as layer types and shapes. Hence, it brings about four challenges for effective hardware acceleration to deliver the desired eye tracking latency and efficiency towards practical deployment on mobile devices with both constrained computation and memory resources.

Challenge # I: Workload Orchestration between Segmentation and Gaze Estimation. As introduced in Sec. 4.3, EyeCoD's predict-then-focus processing pipeline processes the segmentation

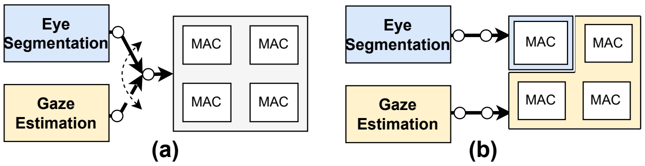


Figure 4: Two classical workload orchestration modes: (a) the time-multiplexing mode and (b) the concurrent mode, for accelerating both the eye segmentation and gaze estimation models.

and gaze estimation models in parallel for reducing the overall latency of eye tracking, where the gaze estimation model continuously operates on every frame while the eye segmentation model executes once out of every N frames ($N = 50$ in our design to balance the achieved eye tracking accuracy and imposed latency and energy costs (see Sec. 6.3)). As such, proper workload orchestration should be considered in our hardware acceleration design.

Potentially, two classical workload orchestration modes, i.e., time-multiplexing and concurrent modes, as shown in Fig. 4, can be adopted to orchestrate the eye segmentation and gaze estimation models on the same accelerator. However, these two workload orchestration modes either require a larger amount of computation resources (i.e., the time-multiplexing mode) or less opportunities for data reuses (i.e., the concurrent mode).

(1) Timing-multiplexing mode. When utilizing the time-multiplexing mode in Fig. 4 (a), only one of the two models' layers occupies the accelerator's computation resources at a given time. As such, to ensure the performance of accelerating the bottleneck layers (i.e., layers with the largest number of operations (FLOPs)) which dominates the overall processing latency, adopting a time-multiplexing mode demands a larger amount of acceleration resources than the models' theoretical requirement. For better understanding, we provide an analysis here. As our eye segmentation's RITNet [9] and gaze estimation's FBNet-C100 [43] contain 140M and 1.06G FLOPs, respectively, the theoretical computation resource requirement is 1024 multiplication-and-accumulations (MACs), if assuming the target eye tracking throughput is 240 FPS at a processing frequency of 350MHz. However, 256 additional MACs (i.e., corresponding to 25% extra MACs if considering a theoretical requirement of 1024 MACs) are required to maintain the target 240 FPS system latency, when running the bottleneck layers (i.e., the third, fifth, forty-second, and forty-forth layer) of the eye segmentation model [9] although the eye segmentation model executes once out of every 50 frames.

(2) Concurrent mode. Considering a concurrent mode for EyeCoD's predict-then-focus processing pipeline, an accelerator spatially executes both the eye segmentation and gaze estimation models simultaneously in two fixed partitions of the accelerator's MACs as shown in Fig. 4 (b) for a given cycle. Different from the time-multiplexing mode, the execution latency of bottleneck layers of the segmentation model are amortized to every 50 frames and do not dominate the overall processing latency. However, the concurrent mode brings about the drawback of less data reuse opportunities, since each of the two partitions has a reduced amount of computation resources as compared to using all resources without partition (i.e., the time-multiplexing mode). Naturally, a good partition scheme should balance the two models' complexity and execution frequency, which in our case will lead to only 4 MACs out of 1024 MACs being assigned to EyeCoD's eye segmentation model and thus result in extremely less reuse opportunities and poor efficiency.

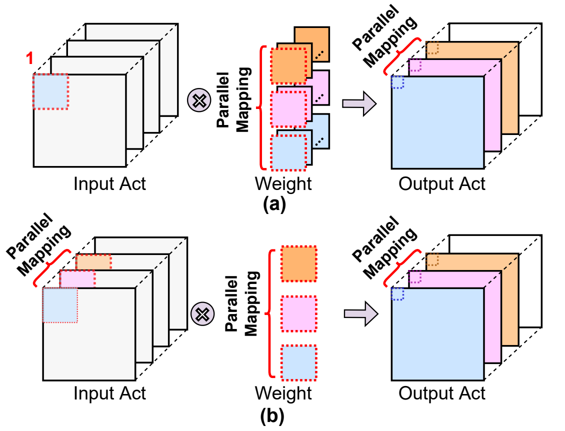


Figure 5: A computation illustration of (a) generic/point-wise convolution and (b) depth-wise convolution layers.

Challenge # II: Support for Various Layer Types. EyeCoD's predict-then-focus processing pipeline consists of RITNet [9] for eye segmentation and FBNet-C100 [43] for gaze estimation, that include generic/point-wise/depth-wise convolution layers, fully-connected (FC) layers, and matrix-matrix-multiplication layers. The efficient processing of various layer types is a design challenge that needs to be addressed. In the following discussion, we compare the number of computation operations of different layer types to find the dominant layer types; analyze the reuse opportunity among various layer types; and show that depth-wise convolution layers require specific optimization.

(1) Dominate layer type analysis. Considering a 50 frame processing of EyeCoD's predict-then-focus processing pipeline when the eye segmentation is needed once, generic convolution, point-wise convolution, depth-wise convolution, FC, and matrix-matrix multiplication layers account for 8.8%, 68.8%, 7.9%, 0.001%, and 14.5% of the overall number of computation operations, respectively. FC is not the dominant layer, since it only accounts for about 0.001% of the overall operations. Matrix-matrix multiplication, can be treated as point-wise convolution layer with a large batch size (i.e., > 1). Therefore, generic convolution, point-wise convolution, and depth-wise convolution are three dominant layer types in EyeCoD's predict-then-focus processing pipeline.

(2) Reuse opportunity analysis. Fig. 5 (a) and (b) illustrate the computation of generic/point-wise convolution and depth-wise convolution layers. There are two kinds of reuse opportunities shared by all convolution layers: *Psum reuse* where the partial sums (Psums) are accumulated to calculate the corresponding output activation (Act) and *intra-channel reuse* where one input channel of

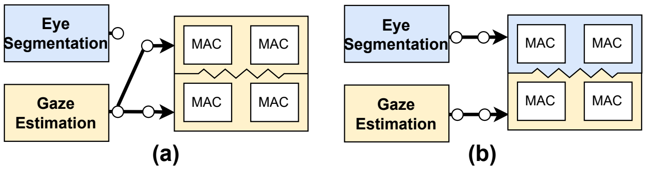


Figure 6: The proposed partial time-multiplexing mode for the workload orchestration of the eye segmentation and gaze estimation models: (a) the gaze estimation model occupies the computation resources; and (b) the eye segmentation and gaze estimation models run simultaneously.

weights are reused by the corresponding channel of input activations to get the Psums (generic/point-wise convolution layers) or output activations (depth-wise convolution layers). Compared with depth-wise layer, generic/point-wise layer has input reuse where one input activation is reused by all 3D weight filters.

(3) Specific optimization requirement for depth-wise convolution layer. Due to the limited reuse opportunities in depth-wise convolution layer, utilizing the same design for both generic/point-wise and depth-wise convolution layers typically leads to a very low MAC utilization or requires a much higher input activation memory bandwidth (see Fig. 5). Our analytical analysis shows that all depth-wise convolution layers account for only 7.9% of the overall number of computation operations, but consume 33.6% of the overall processing time if using the same design as the generic/point-wise layers. Therefore, specific optimizations are necessary for the depth-wise layer to fulfil the goal of real-time performance.

Challenge # III: Workload Partition to Save Activation Memory Size. If using the vanilla layer-by-layer processing, the theoretical on-chip activation memory size should fit the maximum requirement of each layer, i.e., 2.78MB, where the eye segmentation model and the gaze estimation model occupy 2.08MB and 0.70MB, respectively. The 2.78MB on-chip memory size is unacceptable for the eye tracking application; let alone we only count the activation memory. Therefore, proper workload partition is required so that we only need to allocate the activation memory size for the processing of each individual partition.

Challenge # IV: High Activation Memory Bandwidth Requirement. As discussed in Challenge # II, depth-wise convolution layers require a higher activation bandwidth to achieve a satisfying MAC utilization (i.e., 32× ~128× higher bandwidth than the processing of processing generic/point-wise convolution layers for >50% MAC utilization in our design). However, simply enlarging activation bandwidth leads to the bandwidth waste when processing other layers and increases the memory accesses cost. EyeCoD's accelerator design should be optimized to alleviate the stringent requirement of activation memory bandwidth for a better trade-off between depth-wise layers' and other layers' workloads.

Design Principles. Based on the above challenge analysis, we propose the following principles to take full advantage of EyeCoD's predict-then-focus processing pipeline for developing and optimizing the dedicated accelerator. In Sec. 5.2, we incorporate these design principles in our EyeCoD accelerator design.

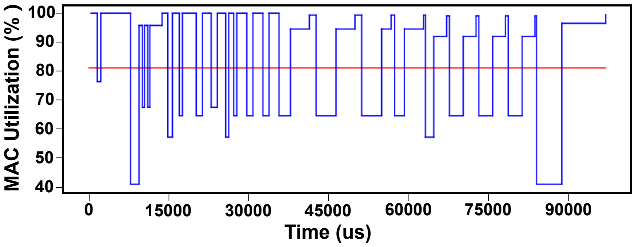


Figure 7: An illustration of the MAC utilization when running the gaze estimation model.

Principle # I: Partial Time-multiplexing Mode for Workload Orchestration. As the time-multiplexing and concurrent modes are not optimized for EyeCoD's predict-then-focus processing pipeline. Adopting them will result in either a larger amount of computation resources or less reuse opportunities, we take advantage of both two modes and propose a partial time-multiplexing mode as shown in Fig. 6. In the proposed partial time-multiplexing mode, the gaze estimation model can fully occupy the computation resources as shown in Fig. 6 (a); or we can run the eye segmentation model and the gaze estimation model simultaneously as shown in Fig. 6 (b). Thanks to the simultaneous processing of both eye segmentation and estimation models (see Fig. 6 (b)), the processing latency of the segmentation model's bottleneck layers are amortized to every 50 frames as in the concurrent mode, which tackles the large computation resource drawback in the time-multiplexing mode. Our evaluation shows that the proposed partial time-multiplexing mode has a 2.31× peak speedup than the time-multiplexing mode with a 10% higher activation global buffer (GB) bandwidth and no computation resource (i.e., MAC) overhead. In addition, the partial time-multiplexing mode provides us the opportunity to better balance the reuse opportunities, the two models' complexity, and the two models' execution frequency to tackle the less reuse opportunity drawback in the concurrent mode. In particular, when the gaze estimation model requires a large amount of computation resources (i.e., generic/point-wise convolution layers), it fully owns the computation resources as shown in Fig. 6 (a). On the other hand, only when the gaze estimation consumes a smaller amount of computation resources (i.e., depth-wise layers), we assign the unused resources to the eye segmentation model to run them simultaneously as shown in Fig. 6 (b). At the same time, the eye segmentation model owns a larger amount of resources than that in the concurrent mode to enable a high reusability.

Principle # II: Intra-channel Reuse for Depth-wise Layer. As the depth-wise layer optimization is critical to speed up the gaze estimation model, the usually-unexplored intra-channel input reuse in generic/point-wise convolution layers need to be explored for depth-wise layer for achieving a high MAC utilization with an acceptable activation memory bandwidth and thus reducing the overall processing time. Our evaluation shows that the proposed intra-channel reuse optimizations can reduce the processing time of depth-wise layers by 71%. The detail description of the proposed intra-channel reuse optimizations is elaborated in Sec. 5.2. It should be note that the intra-channel reuses are limited for the layers with

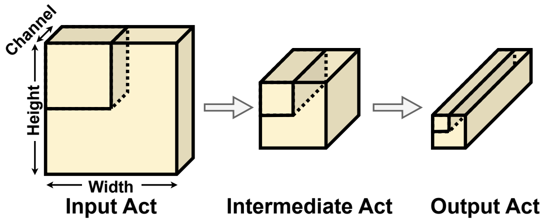


Figure 8: An illustration of input feature-wise partition for cross-layer processing.

a stride of 2 and the last several layers with smaller input activation feature maps (e.g., 7/7 for the height/width of the input feature maps) in the gaze estimation model. As such, further increasing the MAC utilization of these layers is challenging. Thanks to the proposed partial time-multiplexing mode for workload orchestration, we tackle the MAC utilization challenge of these layers by running the segmentation model on the unused MACs. Fig. 7 shows the MAC utilization when running gaze estimation model alone on EyeCoD's accelerator. When the utilization is less than 80% (i.e., the red line in Fig. 7), we can run the eye segmentation model on the unused resources in the proposed partial time-multiplexing mode for a >90% overall MAC utilization.

Principle # III: Input Feature-wise Partition to Save Activation Memory Size. To save the activation memory size, we can partition the input image along the input activation's feature map dimensions (i.e., the height and the width of the input feature map) and process each individual partition through cross-layer processing. As illustrated in Fig. 8, the on-chip activation memory only needs to store the activations of each partition. The overall activation memory size is about 36% (i.e., 1MB) of that before partition.

Principle # IV: Parallelism of Memory Access and Processing to Save Activation Memory Bandwidth. Note that loading the activations from the memory for the next round of processing and the running of the current round of processing can be paralleled, where we denote the one round of processing as the processing using the same activations. Since each round of processing usually takes several cycles (e.g., the number of cycle equals to kernel sizes in our design), we propose to load the next-round activations sequentially from the memory during the current-round processing and then the already loaded next-round activations can be read out in parallel for the next round of processing. This parallelism of memory access and processing can save activation memory bandwidth. A sequential-write-parallel-read input activation buffer is needed to enable the parallelism, which is described in Sec. 5.2. Assuming a commonly-used 3×3 kernel, the propose parallelism saves 50%∼60% memory bandwidth and the sequential-write-parallel-read input activation buffer incurs a negligible area overhead of 0.58%.

5.2 Architecture of EyeCoD's Accelerator

This section describes the proposed accelerator architecture as well as the design optimizations following the principles in Sec. 5.1.

Architecture Overview. Fig. 9 presents the architecture of EyeCoD's accelerator which consists of the following components: (1) on-chip memories for weights, input/output activations, and instructions, (2) computation resources, i.e., 128 MAC lanes, and (3) an on-chip controller. First, two memory hierarchies are adopted. Specifically, the weight GB (i.e., global buffer) stores the parameters of the involved models as well as the reconstruction in EyeCoD's predict-then-focus processing pipeline. Two weight buffers are inserted between weight GB and MAC lanes and work in a "ping-pong" manner to avoid the weight load stalls. Similarly, an input Act buffer and a output Act buffer are inserted between the Act GBs and the MAC lanes to prepare activations for the MAC lanes or Act GB to eliminate input load or output write stalls. Second, each MAC lane is composed of eight MACs and one input Act FIFO to store one row of input activations. The weights of one row are fetched one-by-one from the weight buffer for multiplying one row of input activations in the input Act FIFO. Therefore, each MAC lane is able to reuse the loaded input activation row, i.e., row-wise intra-channel reuse which is ubiquitous among all convolution layers. Third, to implement EyeCoD's predict-then-focus processing pipeline, the on-chip controller reads instructions from the instruction SRAM to control the accelerator.

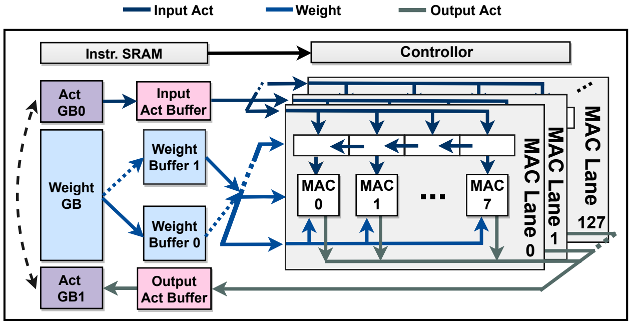


Figure 9: An illustration of EyeCoD's accelerator.

Optimizations for Depth-wise Layer. As discussed in Sec. 5.1, the dataflow for generic/point-wise convolution layers is not sufficient for depth-wise layers, resulting in low MAC utilization or higher input Act bandwidth. To address this, intra-channel reuse is adopted for increasing MAC utilization and better leveraging the limited memory bandwidth. Fig. 10 illustrates two intra-channel reuse opportunities, *column-wise intra-channel reuse* (Fig. 10 (a)) and

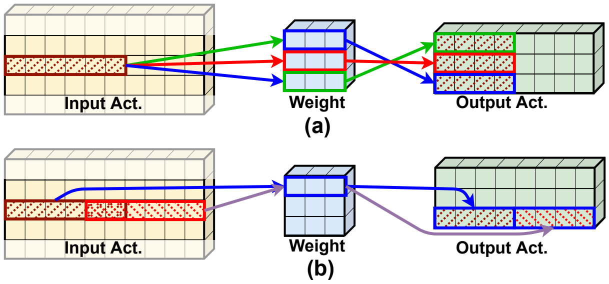


Figure 10: Optimizations for depth-wise convolution layers: (a) column-wise intra-channel reuse and (b) deeper row-wise intra-channel reuse.

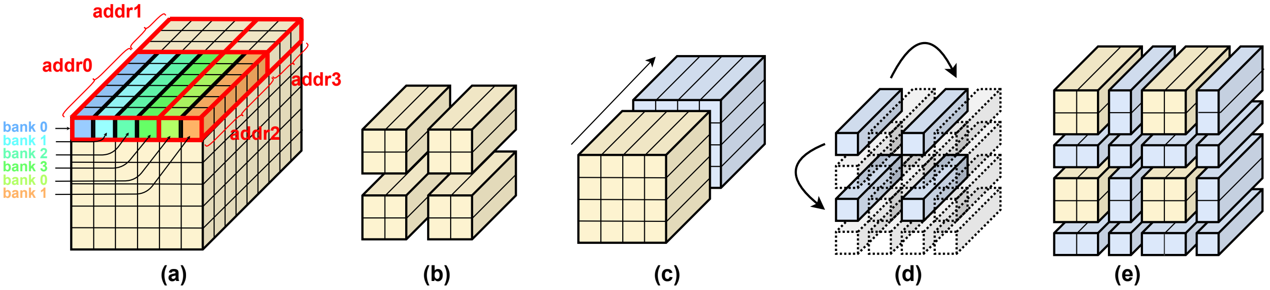


Figure 11: An illustration of the proposed activation GB storage arrangement: an example of (a) the storage arrangement of a 6×6×24 activation tensor, (b) the partition operation, (c) the concatenation operation, (d) the downsampling operation, and (e) the upsampling operation.

deeper row-wise intra-channel reuse (Fig. 10 (b)), for depth-wise convolution layers besides the row-wise intra-channel reuse on each MAC lane. For the former *column-wise intra-channel reuse* method, multiple weight rows in one column of each 3D weight filter reuse one same input activation row and generate multiple output activation rows in the corresponding column. This technique achieves an utilization improvement proportional to the number of available weight rows or the kernel size (e.g., 3 or 5 for the gaze estimation model) as shown in Fig. 10 (a). The latter *deeper row-wise intra-channel reuse* is proposed because the row-wise intra-channel reuse on each MAC lane is limited by the number of MACs of each MAC lane (i.e., 8 MACs/MAC lane in our design). For the latter *deeper row-wise intra-channel reuse*, we tile one input Act row into two sub rows, and further spatially map these two sub input Act rows and their corresponding weight row to two MAC lanes, doubling the MAC utilization.

Activation GB Storage Arrangement. Due the diverse model structures as well as layer types and shapes in EyeCoD's predict-then-focus processing pipeline, various activation reshaping operations are needed. The various reshaping operations impose a challenge for the activation GB storage arrangement, i.e., how to support different activation reshaping operations without complicating controls. We first classify the activation reshaping operations into four classes and then propose an optimized activation GB storage arrangement considering the characterizations of the reshaping operations.

Four classes of reshaping operations are involved in EyeCoD's predict-then-focus processing pipeline: the partition operation (see Fig. 11 (b)) which tiles one input activation tensor into several partitions along activation feature map dimensions to enable the sequential processing of the partitions, the concatenation operation (see Fig. 11 (c)) which concatenates several tiled output activation tensors generated sequentially by the MAC lanes to the output tensor by along the channel dimension, the downsampling operation (see Fig. 11 (d)) for downsampling layers which drops the activations in each activation feature map, and the upsampling operation (see Fig. 11 (e)) for upsampling layers which inserts zeros or duplicates activations in each activation feature map. We propose an activation GB storage arrangement where each activation memory bank address stores one tile of activations along the channel dimension, e.g., 16 activation pixels along the channel dimension per address, This activation GB storage arrangement considers the reshaping operation granularities along activation feature map and channel dimensions to simplify controlling designs. We provide a more detailed explanation below. In particular, this storage arrangement considers that the reshaping operations along feature map dimensions (i.e., the partition, downsampling, and upsampling operations) are usually at the granularity of 1. In contrast, the granularity of the reshaping pattern along channel dimension (i.e., the concatenation pattern) is related to the number of MAC lanes assigned to a certain layer, which is a multiplication of 16 in our design. Fig. 11 (a) gives an example of the storage arrangement of one activation tensor with the shape of 6×6×24. We place four memory banks in parallel for one activation GB and the 6×6×24 activation tensor takes 24 addresses in total. By properly accessing the activation tiles' addresses, the aforementioned four activation reshaping operations are easily supported.

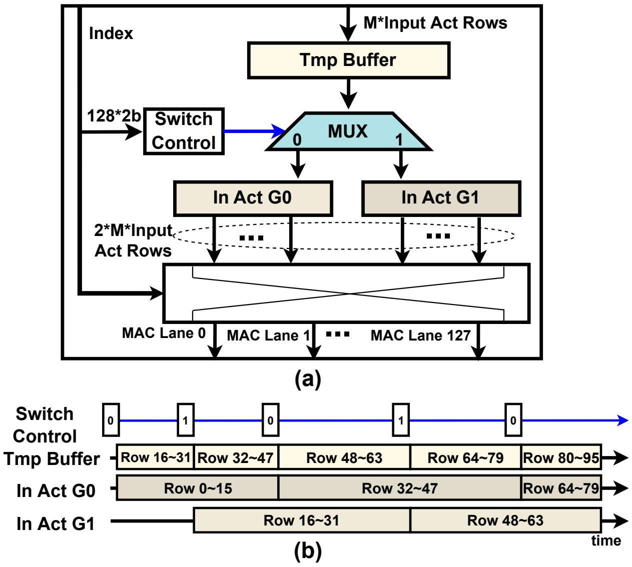
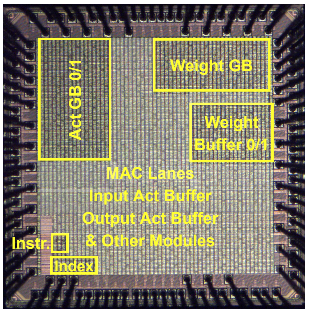


Figure 12: The sequential-write-parallel-read input activation buffer: (a) design scheme and (b) a timing diagram example.

Sequential-write-parallel-read Input Activation Buffer Design. The sequential-write-parallel-read input activation buffer design is demonstrated in Fig. 12 (a) with a timing diagram example in Fig. 12 (b). In the sequential-write-parallel-read input activation buffer, a temp buffer sequentially fetches M input activation rows from the Act GBs ($M = 16$ in this design) for next round of processing and then stores the fetched rows in two interleaved buffers (i.e., In Act G0/G1) following the design principle in Sec. 5.1. After the MAC lanes finish the current round of processing, they can read the input activation rows in parallel from In Act G0/G1. Thanks to this sequential-write-parallel-read buffer design, 2× higher bandwidth (i.e., $2 \times M$) is achieved without memory access stalls which can satisfy the bandwidth requirement for EyeCoD's predict-then-focus processing pipeline..



Technology	28nm
Chip Area	3.00 mm^2
Supply Voltage	0.51-0.8 V (Core) 0.59-0.88 V (Mem)
Core Frequency	370 MHz @ (0.8V, 0.88V)
Total SRAM	316KB
# of MACs	512
Power	154.32 mW @ (0.8V. 0.88V), 370 MHz

Figure 13: EyeCoD silicon prototype: die photo and chip specifications.

Table 1: Accelerator configurations.

Act GB0/GB1	Weight Buffer0/1	Weight GB	Index SRAM	Instr. SRAM
512KB * 2	64KB * 2	512KB	20KB	4KB
MAC Lanes	**MACs/MAC Lane**	**Area**	**Clock frequency**	**Power**
128	8	8 mm^2	370MHz	335mW

6 EXPERIMENTS

In this section, we present a thorough evaluation of the proposed EyeCoD framework, including the experiment setups in Sec. 6.1, the overall benchmark with CPUs/GPUs and previous SOTA eye tracking processors in Sec. 6.2, and the evaluation and ablation studies of EyeCoD's algorithm and accelerator in Sec. 6.3 and Sec. 6.4, respectively.

6.1 Experiment Setups

Model, Datasets, and Training Settings. Model: We use the RITNet [30] and FBNet-C100 [43] as our backbone model for eye segmentation and gaze estimation stage, respectively. Datasets: For evaluating our proposed predict-then-focus pipeline, we use OpenEDS2019 and OpenEDS2020 dataset [21, 35] for segmentation and gaze estimation, respectively. OpenEDS2019 segmentation dataset [21] consists of around 8916 labeled images for training 2403 images for validation. OpenEDS2020 gaze estimation dataset [35] consists of around 128,000 labeled images for training and 70,400 for validation. To simulate the FlatCam reconstructed image, we follow the proposed simulation and reconstruction method in [4]. Training Settings: For evaluating the performance of the entire pipeline, we first crop 512×512 image patches from the center of all images in both datasets for satisfying the requirement that FlatCam's input images are square [4]. (1) For eye segmentation, we downsample the input image from 512×512 resolution to 128×128 resolution before feeding it into the model. Following the award-winning solution [9], we train the model for 300 epochs with a hybrid loss consisting of standard cross entropy loss, generalized dice loss, boundary aware loss, and surface loss. We optimize the model using Adam optimizer [27] with learning rate 1×10^{-3}, and batch size 8. (2) For gaze estimation, we resize the input image to 256×256 resolution and crop a 96×160 ROI region before passing it into the gaze estimation network. Following [35], we train the model with arccosine loss for 25 epochs using Adam optimizer [27] with learning rate of 5×10^{-4}, and batch size of 32.

Baselines and Evaluation Metrics. Baselines: We choose four general computing platforms EdgeCPU (Raspberry Pi), CPU (AMD EPYC 7742), EdgeGPU (Nvidia Jetson TX2), GPU (Nvidia 2080Ti), and one eye tracking processor CIS-GEP [5] as baselines. The batch size for CPU and GPU is set to 1 for a fair comparison. For analyzing the overall estimation accuracy of our FlatCam-based system and the proposed predict-then-focus pipeline, we compare EyeCoD with the winner method in OpenEDS2020 [35]. Metrics: We evaluate all above platforms in terms of both throughput and energy efficiency. In addition, we compare the achieved mIOU and FLOPs comparisons for EyeCoD's segmentation models, gaze estimation error in degrees and FLOPs for EyeCoD's gaze estimation models.

Hardware Platform Setup. Silicon-validated EyeCoD. Fig. 13 illustrates the specifications of the silicon-validated EyeCoD's accelerator which is denoted as the chip for convenience. Specifically, the chip is fabricated in a commercial 28nm HPC CMOS technology, with a total chip area of 3mm^2, a core/memory supply voltage of 0.8V/0.88V, and a power of 154.32mW at a 370MHz frequency. The chip is equipped with 316KB SRAM and 512 MACs. Evaluation Methodology. To enable a fair comparison with the baseline designs with a larger area than the silicon-validated chip, we develop an in-house cycle-accurate simulator of EyeCoD's accelerator, for which the MAC and memory access costs are derived from the real chip measurement or the post-layout simulation. The simulator is verified against the Register-Transfer-Level (RTL) implementation of EyeCoD's accelerator to ensure its correctness, Technology-dependent Parameters. Tab. 1 presents the characteristics of our cycle-accurate simulator of EyeCoD's accelerator used throughout the experiments. Specifically, we implemented 128 MAC lanes with each containing 8 MACs. The SRAM includes two Act GBs with 512KB each (Act GB0/GB1 in Fig. 9), two weight buffers with 64KB each (Weight Buffer0/1 in Fig. 9), one weight GB with 512KB (Weight GB in Fig. 9), one index buffer with 20KB (Index SRAM in Fig. 9), and one instruction buffer with 4KB (Instr. SRAM Fig. 9). Same as the silicon-validated chip, the cycle-accurate simulator assumes a 370 MHz frequency.

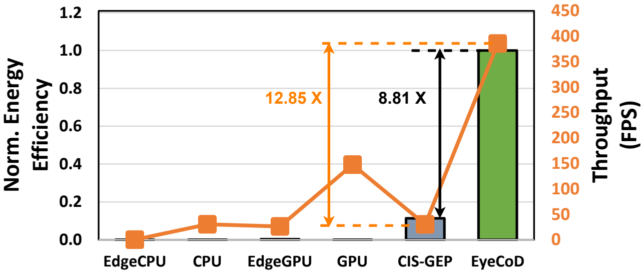


Figure 14: Overall comparison between EyeCoD and other five baselines in terms of both energy efficiency and throughput.

6.2 Overall Performance Comparison

In this part of experiment, we benchmark the proposed EyeCoD's overall performance (normalized energy efficiency vs. throughput) against the CPUs/GPUs and previous SOTA eye tracking processors [5]. As shown in Fig. 14, the proposed EyeCoD consistently achieves both the best normalized energy efficiency and throughput among all the baselines. Specifically, EyeCoD achieves 2966.65×, 12.75×, 14.83×, 2.61×, and 12.86× improvements in terms of throughput as compared to EdgeCPU, CPU, EdgeGPU, GPU, and CIS-GEP baselines, respectively. Meanwhile, EyeCoD maintains a high energy efficiency, achieving 8.81× improvement as compared to the most competitive baseline ASIC accelerator CIS-GEP [5]. We conjure that the improved throughput comes from the sensor-processor co-design and the various optimization techniques as proposed in Sec. 5.2, e.g, customization for different layer types, the proposed sequential-write-parallel-read input activation buffer design, that offer better matched spatial tiling of each operator type to the MAC lanes and reduce the memory stalls, respectively, thus in turn leading to the improved hardware utilization. As for the improved energy efficiency, the tailored handling of the depth-wise convolution layer produces much more intra layer data reuse, resulting in much reduced off-chip memory access and thus also reduced power consumption.

6.3 Evaluation of the EyeCoD Algorithm

Algorithm pipeline evaluation. We first evaluate the necessity of our proposed pipeline with optimized input resolution by benchmarking EyeCoD with vanilla gaze estimation. For the baseline method, we report the winner in [35] that achieves 2.31 degrees error at the cost of 1.82GFLOPs. As shown in Tab. 2, our proposed EyeCoD with the same model (i.e., ResNet18) achieves comparable gaze estimation error (0.10 degree higher) with over 69.2% FLOPs reduction, suggesting that (1) a FlatCam-based eye tracking system does not degrade the accuracy, (2) it is necessary to use an optimized input size for a higher accuracy-efficiency trade-off.

Gaze Estimation On top of the FlatCam system and the pioneering SOTA model ResNet18, we evaluate the effectiveness of various models for gaze estimation. As shown in Tab. 2, EyeCoD with FBNet-C100 (8-int) (highlighted in bold) improves the error of ResNet18 by 0.04 while reducing FLOPs by 78.2%.

Table 2: Benchmark EyeCoD gaze estimation algorithm on OpenEDS'20 dataset with FlatCam reconstructed dataset. The adopted setting in EyeCoD is marked in bold.

Model	Camera	Resolution	Error	Parameter	FLOPs
ResNet18 [35]	Lens	224×224	3.17	11.18M	1.82 G
ResNet18	FlatCam	96×160	3.27	11.18M	0.56G
MobileNet			3.43	2.23M	0.10G
FBNet-C100			3.23	3.59M	0.12G
FBNet-C100 (8-bit)			**3.23**	**3.59M**	**0.01G**

Table 3: Benchmark RITNet performance on OpenEDS'19 dataset under different experiment settings, the adopted setting in EyeCoD is marked in bold.

Model	Resolution	Eye Segmentation mIOU		FLOPs
		Origin Image	FlatCam Image	
U-net	512×512	93.3	92.5	14.1G
RITNet	512×512	95.1	93.6	17.0G
RITNet	256×256	94.7	93.8	4.1G
RITNet (8-bit)	256×256	94.0	92.8	0.3G
RITNet	128×128	94.1	93.5	1.0G
RITNet (8-bit)	128×128	93.3	**92.7**	**0.1G**

ROI Prediction. We further validate the effectiveness of the ROI prediction algorithm we propose by benchmarking our eye segmentation algorithm performance under various settings. Compare to the original result in [21], our proposed algorithm face three more challenges, (1) lower signal-to-noise ratio (SNR) of the FlatCam reconstructed image, (2) smaller resolution ($\frac{1}{16}$ of the original resolution size), and (3) 8-bit quantized model. As shown in the Tab. 3, despite the aforementioned challenges and 16× FLOPs reduction, the segmentation algorithm of EyeCoD (marked in **bold**) still achieves comparable performance (achieving around 93% mIOU on validation dataset) as the award-winning solution in [21]. Moreover, when segmenting on the FlatCam dataset instead of the original image, all networks suffers from performance degradation in terms of mIOU ranging between 1.5% to 0.6%. However, smaller resolution in general suffers less from the adaption of dataset, we suspect this is due to the relatively lower SNR in FlatCam reconstructed images, making it hard for the models to learn the detailed features in the high-resolution images.

Ablation Studies. As shown in Tab. 4, the extracted ROIs augment the gaze estimation by centralizing the eye areas, leading to 9.41 and 8.24 error reductions as compared to random crop or central crop, respectively. This set of experiments validate that extracting ROIs not only reduces the computational cost but also helps to mitigate the undesired effect of FlaCam's blurred images. We also test different ROI sampling frequencies, as shown in Tab. 5, and find that (1) the gaze estimation error and the segmentation FLOPs per frame in general gradually increase alone with the increased ROI sampling frequency, while the error reduction is negligible when sampling frequency is higher than 1 over every 50 frames, and (2) a larger ROI size leads to a better gaze estimation accuracy, where the increase is not significant after the ROI size is larger than 96×160. Thus, our adopted setting (i.e., extracting ROI every

Table 4: Ablation studies of EyeCoD's ROI prediction.

	Random Crop	Central Crop	ROI (Ours)
Gaze Estimation Error	12.64	11.57	3.23

Table 5: Ablation studies of both EyeCoD's ROI prediction frequency and ROI sizes.

ROI Freq.	ROI Size	Gaze Estimation Error	Gaze Estimation FLOPs/Frame	Segmentation FLOPs/Frame
25	96×160	3.23	7.58M	2.5M
50	48×80	3.60	2.28M	1.3M
50	96×160	3.23	7.58M	1.3M
50	144×240	3.19	18.13M	1.3M
100	96×160	3.34	7.58M	0.7M

50 frames with a size of 96×160) achieves an optimal accuracy and inference FLOPs trade-off.

6.4 Evaluation of the EyeCoD Accelerator and System

In this experiment, we perform ablation studies to evaluate EyeCoD's contributions for better understanding its overall superiority. To quantify the impact of different contributions, we build a lens-based system and run the original images of 256×256 resolution on the system. Specifically, the accelerator in the lens-based system removes the hardware-level contributions (including the input activation buffer design, the time-multiplexing workload orchestration, and the intra-channel reuse for depth-wise layers). Please note that the accelerator here keeps EyeCoD's input feature-wise partition to fit the same area and adopts the time-multiplexing mode, where one layer of the eye segmentation model and the gaze estimation model occupy the whole MACs iteratively. We calculate the impact from each of our contributions by applying the FlatCam sensor and the predict-then-focus pipeline, and hardware-level contributions to the lens-based system one-by-one.

Among the 4.00× throughput/energy efficiency improvement over the lens-based eye tracking system, adopting the FlatCam sensor and the predict-then-focus pipeline leads to 1.99× throughput/energy efficiency improvement, while applying the proposed input activation buffer, partial time-multiplexing mode, and intra-channel reuse optimizations further offer 1.22×, 1.28×, and 1.29× throughput/energy-efficiency improvement, respectively. In particular, (1) the proposed predict-then-focus pipeline helps to reduce the image resolution by 76.5%, thus improving the throughput as well as the energy efficiency by 1.99×; (2) the sequential-write-parallel-read activation buffer, which enables the parallelism of memory access and processing, helps to reduce input reading stalls due to the limited activation GB bandwidth and thus improves the performance; (3) the partial time-multiplexing mode, leveraging the two NNs' different execution frequencies for workload-orchestration, achieves 1.28× speedup than the time-multiplexing mode; and (4) the intra-channel reuse further reduces 71% of the depth-wise layers' processing time, resulting in 1.29× speedup.

Table 6: Throughput and normalized energy efficiency of the proposed EyeCoD w/ and w/o predict-then-focus pipeline (P.F.), sequential-write-parallel-read input activation buffer design (Input.), partial time-multiplexing workload orchestration (Partial.), and intra-channel reuse for depth-wise layers (Depth.). The last row is the final adopted EyeCoD system.

System★	Throughput (FPS)	Norm. Energy Eff.
Lens-based System*	96.34	1.00
EyeCoD w/ P.F.*	191.94	1.99
EyeCoD w/ P.F. & Input.	233.64	2.43
EyeCoD w/ P.F. & Input. & Partial.	299.04	3.10
EyeCoD w/ P.F. & Input. & Partial. & Depth.	385.66	4.00

★ All settings use input feature-wise partition.
* Using time-multiplexing mode.

7 CONCLUSION

To this work, we propose, develop, and validate a lensless FlatCam-based eye tracking algorithm and accelerator co-design framework dubbed EyeCoD to enable eye tracking systems with a much reduced form-factor and boosted system efficiency without sacrificing tracking accuracy, targeting next-generation eye tracking solutions. On the system level, we advocate the use of lensless FlatCams instead of lens-based cameras to facilitate the small form-factor need in mobile eye tracking systems. On the algorithm level, EyeCoD integrates a predict-then-focus pipeline that first predicts the region-of-interest ROI and then only focuses on the ROI parts to estimate gaze directions. On the hardware level, we further develop a dedicated accelerator that integrates a novel workload orchestration between the aforementioned segmentation and gaze estimation models, and leverages multiple optimization to further improve the acceleration efficiency. On-silicon measurement and extensive experiments validate advantages of our EyeCoD in enhancing the end-to-end eye tracking throughput while maintaining the tracking accuracy.

ACKNOWLEDGMENT

We would like to acknowledge the funding support from the NSF RTML program (Award number: 1937592) and the NSF EPCN program (Award number: 1934767) for this project.

REFERENCES

[1] Micheal Ambrash. 2021. Creating the Future: Augmented Reality, the Next Human-Machine Interface. In *IEDM*. 1–4.

[2] Nick Antipa, Grace Kuo, Reinhard Heckel, Ben Mildenhall, Emrah Bostan, Ren Ng, and Laura Waller. 2018. DiffuserCam: lensless single-exposure 3D imaging. *Optica* 5, 1 (Jan 2018), 1–9. https://doi.org/10.1364/OPTICA.5.000001

[3] M Salman Asif, Ali Ayremlou, Aswin Sankaranarayanan, Ashok Veeraraghavan, and Richard Baraniuk. 2015. Flatcam: Thin, bare-sensor cameras using coded aperture and computation. *arXiv preprint arXiv:1509.00116* (2015).

[4] Salman Asif, Ali Ayremlou, Aswin Sankaranarayanan, Ashok Veeraraghavan, and Richard Baraniuk. 2015. FlatCam: Thin, Bare-Sensor Cameras using Coded Aperture and Computation. (08 2015).

[5] Kyeongryeol Bong, Injoon Hong, Gyeonghoon Kim, and Hoi-Jun Yoo. 2016. A 0.5° error 10 mW CMOS image sensor-based gaze estimation processor. *IEEE Journal of Solid-State Circuits* 51, 4 (2016), 1032–1040.

[6] Kyeongryeol Bong, Injoon Hong, Gyeonghoon Kim, and Hoi-Jun Yoo. 2016. A 0.5° error 10 mW CMOS image sensor-based gaze estimation processor. *IEEE Journal of Solid-State Circuits* 51, 4 (2016), 1032–1040.

[7] V. Boominathan, J. Adams, J. Robinson, and A. Veeraraghavan. 2020. PhlatCam: Designed phase-mask based thin lensless camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), 1–1.

[8] Braiden Brousseau, Jonathan Rose, and Moshe Eizenman. 2018. Smarteye: An accurate infrared eye tracking system for smartphones. In *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 951–959.

[9] Aayush K Chaudhary, Rakshit Kothari, Manoj Acharya, Shusil Dangi, Nitinraj Nair, Reynold Bailey, Christopher Kanan, Gabriel Diaz, and Jeff B Pelz. 2019. RITnet: Real-time semantic segmentation of the eye for gaze tracking. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, 3698–3702.

[10] Huaijin G Chen, Suren Jayasuriya, Jiyue Yang, Judy Stephen, Sriram Sivaramakrishnan, Ashok Veeraraghavan, and Alyosha Molnar. 2016. ASP vision: Optically computing the first layer of convolutional neural networks using angle sensitive pixels. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 903–912.

[11] Huaijin G. Chen, Suren Jayasuriya, Jiyue Yang, Judy Stephen, Sriram Sivaramakrishnan, Ashok Veeraraghavan, and Alyosha C. Molnar. 2016. ASP Vision: Optically Computing the First Layer of Convolutional Neural Networks using Angle Sensitive Pixels. *CoRR* abs/1605.03621 (2016). arXiv:1605.03621 http://arxiv.org/abs/1605.03621

[12] Y. Chen, T. Krishna, J. Emer, and V. Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *JSSC 2017* 52, 1 (2017), 127–138.

[13] Y. Cheng, F. Lu, and X. Zhang. 2018. Appearance-based gaze estimation via evaluation-guided asymmetric regression. In *Proceedings of The European Conference on Computer Vision*.

[14] Wanli Chi and Nicholas George. 2011. Optical imaging with phase-coded aperture. *Opt. Express* 19, 5 (Feb 2011), 4294–4300. https://doi.org/10.1364/OE.19.004294

[15] H. Deng and W. Zhu. 2017. Monocular free-head 3d gaze tracking with deep learning and geometry constraints. In *Proceedings of the International Conference on Computer Vision*. 3162–3171.

[16] Michael J. DeWeert and Brian P. Farm. 2014. Lensless coded aperture imaging with separable doubly Toeplitz masks. In *Compressive Sensing III*, Fauzia Ahmad (Ed.), Vol. 9109. International Society for Optics and Photonics, SPIE, 180 – 191. https://doi.org/10.1117/12.2050760

[17] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 92–104.

[18] Ryoji Eki, Satoshi Yamada, Hiroyuki Ozawa, Hitoshi Kai, Kazuyuki Okuike, Hareesh Gowtham, Hidetomo Nakanishi, Edan Almog, Yoel Livne, Gadi Yuval, et al. 2021. 9.6 A 1/2.3 inch 12.3 Mpixel with On-Chip 4.97 TOPS/W CNN Processor Back-Illuminated Stacked CMOS Image Sensor. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. IEEE, 154–156.

[19] T. Fischer, Jin Chang, Demiris H., and Y.: Rt-gene. 2018. Real-time eye gaze estimation in natural environments. In *Proceedings of the European Conference on Computer Vision*.

[20] Yonggan Fu, Yang Zhang, Yue Wang, Zhihan Lu, Vivek Boominathan, Ashok Veeraraghavan, and Yingyan Lin. 2021. SACoD: Sensor Algorithm Co-Design Towards Efficient CNN-Powered Intelligent PhlatCam. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 5168–5177.

[21] Stephan J Garbin, Yiru Shen, Immo Schuetz, Robert Cavin, Gregory Hughes, and Sachin S Talathi. 2019. Openeds: Open eye dataset. *arXiv preprint arXiv:1905.03702* (2019).

[22] Injoon Hong, Kyeongryeol Bong, Dongjoo Shin, Seongwook Park, Kyuho Jason Lee, Youchang Kim, and Hoi-Jun Yoo. 2015. A 2.71 nJ/pixel gaze-activated object recognition system for low-power mobile smart glasses. *IEEE Journal of Solid-State Circuits* 51, 1 (2015), 45–55.

[23] G. Huang, H. Jiang, K. Matthews, and P. Wilford. 2013. Lensless imaging by compressive sensing. In *2013 IEEE International Conference on Image Processing*. 2101–2105. https://doi.org/10.1109/ICIP.2013.6738433

[24] G. Huang, Z. Liu, Van Der Maaten, Weinberger L., and K. Q.: Densely connected convolutional networks. 2017. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. *p* (2017), 4700–4708.

[25] Anton S Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem Rufo. 2019. DeepFovea: neural reconstruction for foveated rendering and video compression using learned statistics of natural videos. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–13.

[26] Salman Siddique Khan, Varun Sundar, Vivek Boominathan, Ashok Veeraraghavan, and Kaushik Mitra. 2020. Flatnet: Towards photorealistic scene reconstruction from lensless measurements. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[27] D. P. Kingma and J.: Adam: A Ba. 2015. method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

[28] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik, and A. Torralba. 2016. Eye tracking for everyone. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR*.

[29] Weitao Li, Pengfei Xu, Yang Zhao, Haitong Li, Yuan Xie, and Yingyan Lin. 2020. TIMELY: Pushing Data Movements and Interfaces in PIM Accelerators towards Local and in Time Domain. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 832–845. https://doi.org/10.1109/ISCA45697.2020.00073

[30] Yingyan Lin, Charbel Sakr, Yongjune Kim, and Naresh Shanbhag. 2017. PredictiveNet: An energy-efficient convolutional neural network via zero prediction. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4.

[31] Chiao Liu, Andrew Berkovich, Qing Chao, Song Chen, Ziyun Li, Hans Reyserhove, Syed Shakib Sarwar, and Tsung-Hsun Tsai. 2020. Intelligent Vision Sensors for AR/VR. In *Imaging Systems and Applications*. Optical Society of America, ITu5G–1.

[32] Chiao Liu, Andrew Berkovich, Song Chen, Hans Reyserhove, Syed Shakib Sarwar, and Tsung-Hsun Tsai. 2019. Intelligent Vision Systems–Bringing Human-Machine Interface to AR/VR. In *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 10–5.

[33] Sungmin Moon, Chao Zhang, Sooill Park, Hui Zhang, Woo-Shik Kim, and Jong Hwan Ko. 2021. A Sub-Milliwatt and Sub-Millisecond 3-D Gaze Estimator for Ultra Low-Power AR Applications. In *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*. 481–485.

[34] A. Newell, K. Yang, and J. Deng. 2016. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*. 483–499.

[35] Cristina Palmero, Abhishek Sharma, Karsten Behrendt, Kapil Krishnakumar, Oleg V Komogortsev, and Sachin S Talathi. 2021. OpenEDS2020 Challenge on Gaze Tracking for VR: Dataset and Results. *Sensors* 21, 14 (2021), 4769.

[36] S. Park, A. Spurr, and O. Hilliges. 2018. Deep pictorial gaze estimation. In *European conference on computer vision*.

[37] Takeshi Shimano, Yusuke Nakamura, Kazuyuki Tajima, Mayu Sao, and Taku Hoshizawa. 2018. Lensless light-field imaging with Fresnel zone aperture: quasi-coherent coding. *Appl. Opt.* 57, 11 (Apr 2018), 2841–2850. https://doi.org/10.1364/AO.57.002841

[38] David G. Stork. 2013. Lensless Ultra-Miniature CMOS Computational Imagers and Sensors.

[39] Yusuke Sugano, Yasuyuki Matsushita, and Yoichi Sato. 2014. Learning-by-synthesis for appearance-based 3d gaze estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1821–1828.

[40] K. H. Tan, D. J. Kriegman, and N. Ahuja. 2002. Appearance-based eye gaze estimation. *In: Sixth IEEE Workshop on Applications of Computer Vision* 2002 (2002), 191–195.

[41] E. Wood, T. Baltrušaitis, L. P. Morency, P. Robinson, and A.: A Bulling. 2016. 3d morphable model of the eye region. *In: Proceedings of the* 37 (2016), 35–36.

[42] E. Wood and A.: Eyetab Bulling. 2014. Model-based gaze estimation on unmodified tablet computers. In *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM, 207–210.

[43] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 10734–10742.

[44] Chenhao Xie, Xie Li, Yang Hu, Huwan Peng, Michael Taylor, and Shuaiwen Leon Song. 2021. Q-VR: system-level design for future mobile collaborative virtual reality. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 587–599.

[45] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling. 2015. Appearance-based gaze estimation in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4511–4520.

[46] Shulin Zhao, Haibo Zhang, Cyan Subhra Mishra, Sandeepa Bhuyan, Ziyu Ying, Mahmut Taylan Kandemir, Anand Sivasubramaniam, and Chita Das. 2021. HoloAR: On-the-fly Optimization of 3D Holographic Processing for Augmented Reality. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 494–506.

[47] Yang Zhao, Xiaohan Chen, Yue Wang, Chaojian Li, Haoran You, Yonggan Fu, Yuan Xie, Zhangyang Wang, and Yingyan Lin. 2020. SmartExchange: Trading Higher-Cost Memory Storage/Access for Lower-Cost Computation. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 954–967. https://doi.org/10.1109/ISCA45697.2020.00082