Process Only Where You Look: Hardware and Algorithm Co-optimization for Efficient Gaze-Tracked Foveated Rendering in Virtual Reality

Haiyu Wang
Tandon School of Engineering
New York University
New York, USA
hw3689@nyu.edu

Wenxuan Liu
Tandon School of Engineering
New York University
New York, USA
wl3181@nyu.edu

Kenneth Chen
Tandon School of Engineering
New York University
New York, USA
kennychen@nyu.edu

Qi Sun
Tandon School of Engineering
New York University
New York, USA
qisun@nyu.edu

Sai Qian Zhang
Tandon School of Engineering
New York University
New York, USA
sai.zhang@nyu.edu

Abstract

Virtual reality (VR) plays a crucial role in advancing immersive, interactive experiences that transform learning, work, and entertainment by enhancing user engagement and expanding possibilities across various fields. Image rendering is one of the most crucial application in VR, as it produces high-quality, realistic visuals that are vital for maintaining immersive user experiences and preventing visual discomfort or motion sickness. However, the cost of image rendering in VR environment is considerable, primarily due to the demands of high-quality visual experiences from users. This challenge is even greater in real-time applications, where maintaining low latency further increases the complexity of the rendering process. On the other hand, VR devices, such as head-mounted displays (HMDs), are intrinsically linked to human behavior, using insights from perception and cognition to enhance user experience.

In this work, we aim to reduce the high computational costs of the rendering process in VR by leveraging natural human eye dynamics and focusing on *processing only where you look* (POLO). This involves co-optimizing AI algorithms with underlying hardware for greater efficiency. We introduce POLONet, an efficient multitask deep learning framework designed to track human eye movements with minimal latency. Integrated with the POLO accelerator as a plug-in for VR HMD SoCs, this approach significantly lowers image rendering costs, achieving up to a 3.9× reduction in end-to-end latency compared to the latest gaze tracking methods.

CCS Concepts

• **Computer systems organization** → **Real-time system architecture**; • **Computing methodologies** → **Tracking**; **Virtual reality**; **Mixed / augmented reality**; **Perception**; **Rendering**.

Keywords

Foveated Rendering, Gaze Tracking, Saccade Detection, Hardware Accelerator, Virtual Reality

ACM Reference Format:
Haiyu Wang, Wenxuan Liu, Kenneth Chen, Qi Sun, and Sai Qian Zhang. 2025. Process Only Where You Look: Hardware and Algorithm Co-optimization for Efficient Gaze-Tracked Foveated Rendering in Virtual Reality. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25), June 21–25, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3695053.3731110

1 Introduction

Virtual Reality (VR) is transforming how we engage with digital content by providing immersive experiences that seamlessly blend physical and virtual environments. Its impact spans multiple domains, including entertainment and gaming [25], education [4, 54, 102, 107], healthcare [19, 41, 73, 85], and more [20, 52]. By enabling fully immersive environments, VR allows users to explore scenarios in entirely new ways.

Image rendering is arguably the most critical application in VR systems, as it directly impacts the realism and immersion of the virtual environment. High-resolution, low-latency rendering is essential for a seamless, responsive experience. Conversely, poor rendering quality or visual delays can lead to discomfort and diminished user engagement, including motion sickness. However, rendering high-resolution frames on standalone VR devices can result in significant latency. Traditionally, mobile VR rendering engines have relied on simple rasterization pipelines with limited compute shader support. Nevertheless, ray tracing-based rendering techniques have gained popularity [8, 79, 88, 93, 103] for improving real-time rendering in next-gen standalone VR headsets. Notably, recent advancements have enabled the integration of ray tracing features in Snapdragon GPUs [87, 88] that are increasingly being adopted in VR platforms [89]. This progress allows mobile applications like War Thunder Mobile [88] to deliver enhanced visual experiences despite high computational demands. To investigate this, we perform simulations of image rendering based on ray tracing across various scenes from LumiBench [68] using Vulkan-Sim [91],

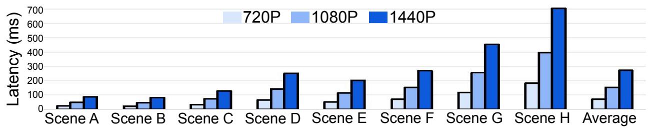

Figure 1: Rendering latencies under different resolutions.

a specialized GPU simulator for image rendering. Vulkan-Sim is configured to emulate the Jetson Orin NX edge GPU [1], which has been frequently used in prior research to model rendering performance in VR devices [42, 45, 82, 91, 117, 124]. As shown in Figure 1, rendering times range from 20 ms to 700 ms depending on the scene and resolution, leading to average latencies of 80 ms, 155 ms, and 282 ms for resolutions of 720P, 1080P, and 1440P, respectively. However, these latencies fall short of meeting the requirements for a smooth visual experience, as earlier studies indicate a per-frame rendering latency of 50–70 ms is necessary [5].

In the realm of VR, the human eyes serve as the primary medium through which users engage with the virtual environment, making gaze tracking the most crucial component for the majority of VR applications. Human visual acuity varies across the visual field. The fovea, the central region of the retina, is responsible for our sharpest vision. As we move away from the fovea, our visual acuity decreases rapidly. Foveated rendering leverages this phenomenon by allocating more computational resources to the fovea while reducing detail in the periphery. This technique significantly enhances VR system performance by lowering the rendering workload without compromising the perceived visual quality, making it a critical innovation in VR [5, 40, 83, 104].

In addition to foveated behavior, the human eye also exhibits saccadic motion. During saccadic motion, the eye rapidly jumps from one point of focus to another, allowing us to scan our surroundings efficiently. These quick movements are crucial for visual tasks like reading or scene exploration, as they enable the brain to gather information from different parts of the visual field in a fraction of a second. During a saccade, the sensitivity of the visual system undergoes a temporary reduction, a phenomenon known as saccadic suppression [59, 76]. This change in sensitivity helps prevent the brain from perceiving the rapid, blurred movement of the visual field as the eyes quickly shift focus.

By leveraging insights from human gaze behavior, this paper aims to lower the high computational costs of visual processing in VR by *Processing Only Where You Look* (POLO). POLO co-optimizes AI algorithms and underlying hardware platform for efficient image rendering in VR. Our contributions can be summarized as follows:

- We propose a deep learning framework termed *POLONet*, that efficiently and accurately detects both gaze direction and saccade occurrences. This approach involves optimizing shared and task-specific components to ensure precise, efficient performance across both tasks, resulting in enhanced overall system performance.
- We design the *POLO Accelerator*, an efficient hardware processor that functions as a plug-in for a VR device SoC. The POLO accelerator significantly enhances the hardware efficiency of POLONet. Moreover, the predicted gaze direction, combined with the generated saccade signal, guides the rendering process in GPU, improving rendering efficiency by adaptively reducing the rendering resolution. We also propose an optimized task execution pattern to efficiently schedule the operations within VR SoC.
- Evaluation results indicate that the POLO system achieves up to a 3.9× reduction in end-to-end latency compared to existing gaze tracking methods. In addition, it provides significant energy savings and enhances the user visual experience, as validated through real user studies.

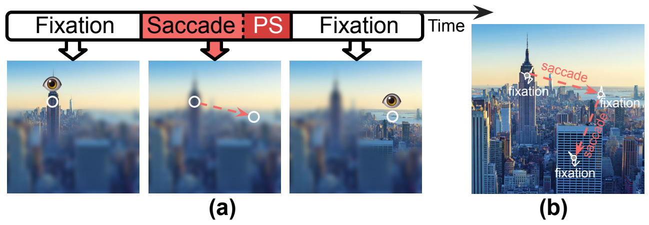

Figure 2: (a) An example on fixation and saccade of human eye. PS indicates the post-saccade duration. (b) An example illustrating gaze movements within a rendered image.

2 Background

2.1 Human Eye Behavior

Human ocular motion can be divided into three principal types, each serving a distinct function: *fixation*, during which the eye remains stationary and converges on a single point; *saccadic movements*, rapid, ballistic shifts that redirect the line of sight from one target to another; and *smooth pursuit*, a continuous, slower tracking of moving objects. Smooth pursuit occurs relatively infrequently, whereas fixation and saccadic movements dominate everyday visual behavior, alternating in quick succession to enable both broad scene exploration and precise focus as illustrated in Figure 2 (a). During a fixation, the gaze remains locked on one location, and spatial resolution varies across the visual field. The fovea, located at the retinal center, provides the highest acuity thanks to its dense packing of photoreceptors. Outside this foveal region, visual sharpness declines steeply, leaving peripheral regions markedly less sensitive to fine spatial detail.

Moreover, humans typically execute one to three saccadic eye movements per second [34, 58, 61], each lasting approximately 20–200 ms [90]. These rapid eye movements play a critical role in efficiently scanning the visual environment. However, during a saccade, the rapid displacement of the retina leads to a temporary degradation of visual input, commonly referred to as saccadic blur [15, 76]. After the eye reaches its new point of fixation, visual clarity is quickly restored, and the brain integrates information from both fixation and saccadic periods to construct a coherent and stable perceptual experience. Previous studies have demonstrated that perceptual suppression during saccades peaks at the moment of maximum eye velocity, leading to a reduction in stimulus detectability by at least 75% [50]. This natural suppression mechanism enables the image rendering process to be temporarily paused during saccades without noticeable degradation of the user's visual experience [55, 70]. Figure 2 (b) shows an example of human gaze movements within a single scene, where the gaze

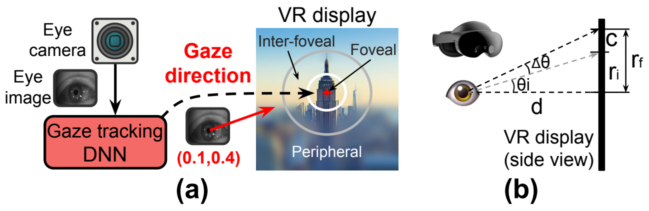

Figure 3: (a) Foveated rendering in VR device. (b) Relation between the tracking error and the size of the foveal region.

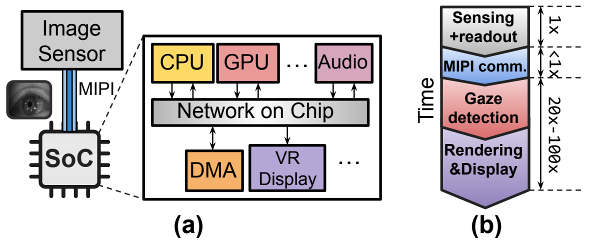

Figure 4: (a) Architecture of TFR system. (b) Normalized breakdown of TFR system latency; illustration not to scale.

typically fixates on multiple points, and saccades occur between these fixation stages before shifting to next scene.

Recent research has shown that even after the gaze lands on a new target at the end of a saccade, the brain still requires time to stabilize the scene and adjust to the new visual input. Visual acuity remains low for an additional 50 milliseconds [61], presenting an opportunity for low-resolution rendering that can save computational resources. This period is referred to as *post-saccadic duration*.

2.2 Gaze-Tracked Foveated Rendering

As virtual reality gains broader adoption, the demand for rendering high-resolution imagery on resource-limited devices, particularly head-mounted displays (HMDs), has intensified [5, 84]. Under these constraints, it is imperative to optimize overall system performance, with a special focus on minimizing latency to suppress visual distortions and artifacts. When rendering falls behind user motion, a mismatch between visual feedback and physical sensation occurs, which can substantially degrade the immersive experience [29, 74].

Gaze-tracked foveated rendering (TFR) is a VR display strategy that exploits real-time eye-gaze estimation, typically via deep neural networks (DNNs), to minimize rendering latency. At each frame, the system analyzes a single eye image to determine the user's gaze point, rendering the *foveal region* (Figure 3 (a)) at full resolution while gradually reducing detail in the surrounding *inter-foveal* and *peripheral regions* without introducing visible artifacts [74, 84, 113]. Figure 3 (b) presents a quantitative model of TFR, where the foveal radius r_f (in pixels) is directly tied to the gaze-tracking error $\Delta\theta$ and is determined as follows:

$$r_f = r_i + c = \rho d \cdot \tan(\theta_i + \Delta\theta) = \rho d \tan(\theta_f)^1 \quad (1)$$

where ρ represents the display's pixel density (pixels per unit length), d denotes the (fixed) distance from the eye to the VR display, θ_i is the eccentricity angle subtended by the foveal region, and $\theta_f = \theta_i + \Delta\theta$ is the resulting eccentricity angle that incorporates the gaze tracking error $\Delta\theta$ due to inaccuracies in the gaze tracking mechanism. The baseline foveal radius without tracking error is $r_i = d\tan(\theta_i)$, and the error-induced increment is $c = d\tan(\theta_f) - d\tan(\theta_i)$, representing the changes on foveal region radius caused by gaze tracking error $\Delta\theta$. Empirical studies typically set θ_i between 5° and 6° depending on user-study results [5, 22, 106]. In [106], the nominal foveal eccentricity θ_i is fixed at 5° around the gaze point, whereas the inter-foveal region extends from 5° to 20°. When a user wears a head-mounted display, d remains constant, as the display is typically mounted directly on the user's head. From equation 1, we observe that a large gaze-tracking error $\Delta\theta$ will expand the foveal and inter-foveal regions, which are rendered at higher resolution to maintain visual quality and ensure a seamless user experience. However, this also increases system overhead, as rendering larger high-resolution regions requires more computational resources.

2.3 TFR System

To implement TFR system in VR hardware, three major components are essential, as illustrated in Figure 4 (a). These components include a near-eye image sensor, a host SoC for visual task processing, and an interconnection link (such as MIPI [62]). For example, in VR HMD like the Meta Quest Pro [86], the SoC is composed of various modules (e.g., CPU, GPU, audio processor).

A standard TFR workflow, illustrated in Figure 4 (a), proceeds as follows: first, the image sensor captures a frame of the user's eye. Next, the image is preprocessed by the image signal processor (ISP) and readout circuitry, then sent over the MIPI link to the host processor. Upon arrival, the host SoC's GPU runs a DNN to infer the gaze direction. Finally, this gaze estimate is supplied to the GPU-based foveated rendering process, which produces the VR scene with spatially varying resolution.

Figure 4 (b) decomposes the end-to-end latency of the TFR pipeline. Camera sensor acquisition delay T_s and MIPI transmission delay T_c together contribute only a small fraction of the total latency, approximately 1ms for image sensing [7, 67, 100, 116] and under 1ms for MIPI transfer [2, 63]. By contrast, gaze inference latency T_d and the subsequent rendering & display latency T_r dominate the total delay, exceeding the sensing and communication overhead by factors of 20×–100× from [5, 96].

Due to the importance of gaze tracking latency T_d and rendering latency T_r, in Section 4.3, we propose an efficient gaze tracking solution that produces accurate gaze tracking results with minimal $\Delta\theta$, thereby reducing the computational cost for foveated rendering and further minimizing T_r. Additionally, in Section 4.2 and Section 4.3 we propose an optimized gaze tracking DNN design with low computational cost, contributing to a reduction in T_d. In Section 5, we propose a hardware accelerator and a computational pattern aimed at further reducing both T_d and T_r and energy consumption for TFR process.

[1]This formula is obtained by assuming the gaze point lies at the center of the forward view, representing the maximum radius of the rendering region.

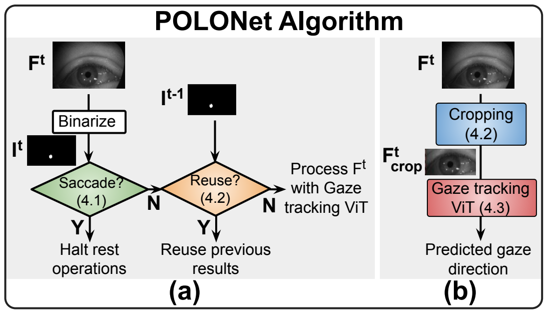

Figure 5: An overview of POLONet.

3 Related Work

3.1 Gaze Tracking Algorithms

Gaze-tracking techniques are generally divided into model-based and appearance-based categories [43, 121]. Model-based methods infer the gaze vector by fitting a 3D eyeball model that captures key anatomical features of the eye [71, 101, 105, 109]. Typically, these methods follow two sequential steps: (1) a neural network extracts eye features, producing a segmentation mask and uses these features to fit a parametric geometric eye model, and (2) the gaze direction is inferred from the parameters of the fitted eye model [36, 38, 64, 115, 120]. Notably, EdGaze [36] reduces computational overhead by evaluating event density between consecutive frames to bypass redundant eye segmentation, reusing previous segmentation results instead. BlissCam [38] leverages the vision transformer [28] to enhance the performance of eye segmentation.

Despite the high eye-segmentation accuracy reported in previous studies [16], geometric model–based gaze estimation often inherently incurs systematic errors exceeding 2° relative to ground truth. This performance degradation stems chiefly from two factors: (1) imprecise estimation in fitting the eye's center and radius during the eye model initialization, and (2) restrictive geometric constraints imposed during optimization stage which limit allowable pupil shapes and positions, thereby amplifying the deviation from true gaze direction [101].

By contrast, appearance-based gaze-tracking approaches operate directly on raw eye images, learning an end-to-end mapping to gaze direction [43, 122, 123]. These methods typically demand larger and more diverse training datasets than model-based techniques. Driven by the scale and complexity of required training data, a broad spectrum of learning paradigms has emerged, including linear regression [72], random forests [99], k–nearest neighbors (KNN) [99, 110], and CNNs [9, 77].

3.2 Saccade Detection Algorithms

Saccade detection methods are broadly categorized into velocity-based and dispersion-based approaches [6, 24]. Velocity-based methods [33, 80, 95] compute angular velocity by differentiating gaze position signals, flagging saccades when the velocity surpasses a predefined threshold. Some studies further refine this process by applying polynomial fitting or neural networks to continuous sequences of velocity and gaze data to model saccadic behavior more accurately [13, 61]. In contrast, dispersion-based methods [92] identify fixations by verifying that the spatial dispersion of gaze points remains below a certain threshold, with segments exhibiting greater dispersion classified as saccades. However, these traditional techniques necessitate continuously running a full high-precision gaze tracking process to obtain accurate gaze landing positions, which incurs significant computational costs. In contrast, POLO offers an efficient alternative by employing a small recurrent neural network to detect saccades, as detailed in Section 4.1. Upon saccade detection, the system bypasses further high-precision tracking, substantially reducing computational load.

3.3 Gaze Tracking Implementations

Recent studies [30, 37, 38, 69] in in-sensor computing focus on improving the efficiency and speed of eye tracking execution directly within sensor hardware, minimizing data transfer to external processors and reducing latency and power consumption. For instance, BlissCam [38] leverages a combination of analog-domain eventification, sampling, and in-sensor computing techniques to detect and reuse the region of interest for gaze tracking, further reducing the sensor-host data volume. In addition, the pixel processor array (PPA) architecture proposed in [30, 69] can sense, store, and reason without relying on external centralized processing units, enabling highly parallel and localized image processing. Despite potential performance benefits, integrating processing units into sensors increases hardware complexity and manufacturing costs, and no current commercial VR products have adopted this approach. Furthermore, the limited computational capacity of sensors can restrict the accuracy and reliability of advanced tracking algorithms.

3.4 DNN Pruning

Pruning is a fundamental strategy for reducing memory footprint and computational costs during DNN inference [14, 60, 65, 66, 78, 119]. Most pruning methods focus on eliminating redundant parameters in the model weights, employing either structured or unstructured approaches to remove redundant weights at various levels of granularity. More recently, pruning has been extended to intermediate token representations in ViTs [28, 53]. Examples include SPViT [57], which employs a token selector to drop non-critical tokens; S^{2}ViTE [18], which leverages sparse training to prune both tokens and attention heads; and Evo-ViT [112], which introduces a slow–fast token evolution mechanism to retain essential information. Prior studies have applied cropping to discard irrelevant pixels from eye-region images [36]. However, performing fine-grained pruning at the token level, such as removing tokens corresponding to eyelashes, can further reduce input redundancy. Unlike tokens that encode iris and pupil information, these extraneous features contribute negligibly to gaze-tracking accuracy [101]. As detailed in Section 4.3, our token-wise pruning approach ranks input tokens based on their importance (attention scores) relative to the final gaze prediction and removes unimportant tokens, significantly reducing computational costs with minimal impact on accuracy. Figure 6 (a) highlights the cropping process.

4 POLO Algorithm

In this section, we discuss the POLONet for efficient gaze tracking and saccade detection, with an overview provided in Figure 5. As

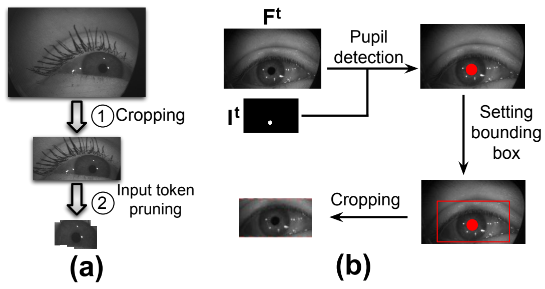

Figure 6: (a) POLONet first applies background cropping for an input eye image (step 1), followed by fine-grained token pruning to remove uninformative regions (step 2). The resulting tokens are then fed into ViT. (b) Illustration of the input cropping procedure based on pupil detection.

indicated in Figure 5 (a), upon receiving an eye image, the system first processes it through a neural network to determine if a saccade has occurred (Section 4.1). If a saccade is detected, the saccade indicator is set to 1, halting further operations. Otherwise the eye image is compared with previous buffer frames to assess whether prior gaze prediction results can be reused, as explained in Section 4.2. If reuse is not feasible, the image is cropped using the approach in Section 4.2 and then passed to the gaze tracking ViT (Figure 5 (b)), as detailed in Sections 4.3.

4.1 Saccade Detection

The occurrence of a saccade can be detected by utilizing the fact that the eye moves rapidly over a short period, resulting in a significant difference between consecutive frames. The workflow for saccade detection is illustrated in the middle part of Figure 5. At time t, the input eye frame F^t first undergoes average pooling with a window size of $M \times M$ to reduce the spatial dimensions and lower computational cost, where M is a hyperparameter that can be calibrated using sampled eye images within the training dataset. The intermediate result is then binarized by comparing it to a predefined threshold γ_1, assigning binary values of 1 to darker regions and 0 to brighter regions. The binary map I^t is passed to the saccade detection DNN, which includes a convolutional layer and a recurrent block for high-level feature extraction. Finally, a linear layer produces a binary output indicating the presence of a saccade. The saccade detection procedure can be formulated as:

$$\begin{aligned} x_t &= Flatten\left(MaxPool(Conv(I^t))\right) \\ h_t &= \beta \cdot h_{t-1} + \alpha \cdot \tanh(Wx_t + Uh_{t-1}) \\ s_t &= \sigma(W_{fc}h_t + b_{fc}) \end{aligned} \tag{2}$$

where h_t is the hidden state of the recurrent network, and s_t is a binary value indicating the occurrence of a saccade. β and α are learnable parameters that control the impact of the current input and historical information. The inclusion of the recurrent block allows the model to capture temporal information of eye movements, enabling the detection of saccadic events by analyzing both current and previous frames, ensuring reliable identification of rapid saccadic motions. If a saccade is detected, the saccade indicator is set to 1, causing all subsequent operations to be skipped. If no saccade is detected, frame F^t is then analyzed for the potential to reuse the gaze data, as outlined in Section 4.2.

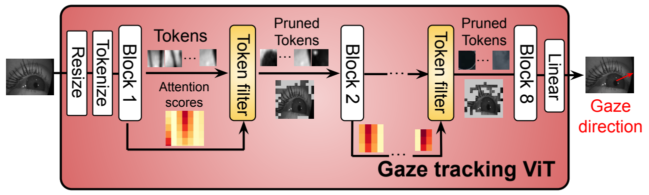

Figure 7: Design of the gaze tracking ViT. The token filter will use the attention scores to eliminate the unimportant tokens inside the activations.

4.2 Gaze Reuse and Event-based Cropping for Efficient TFR

Eye movements are typically minimal across consecutive eye images, resulting in negligible gaze direction changes much of the time [81, 108]. This enables reuse of the previous gaze direction result from the gaze-tracking DNN, significantly reducing computational overhead by running the gaze tracking neural network only when substantial changes are detected in the input eye frame.

To achieve this, we calculate the pixelwise difference between the binary maps I^t and I^{t-1}, and compare it against a predefined threshold γ_2. Based on this comparison, we then decide whether to reuse the existing gaze detection result from the previous frames or perform a new gaze detection. The process is described in the lower part of Figure 5.

If the previous gaze direction is not reused, the input frame F^t will be processed to identify the gaze location. It is important to note that F^t often contains extraneous pixels, such as background or facial muscles, which are irrelevant to gaze tracking. These unnecessary pixels can decrease prediction accuracy and increase computational costs due to the larger input image size. To address this, we propose a method to eliminate the uninformative regions of the eye image. Unlike prior methods that rely on neural networks to identify cropping regions and introduce additional computational and storage overhead [35], our approach is an analytical framework designed to minimize running costs and memory usage.

Our approach begins by roughly locating the pupil, then constructing a bounding box with predefined dimensions centered on it. The process is highlighted in Figure 6 (b). Since the captured eye images by inward eye camera are typically monochromatic with normalized pixel values ranging from 0 to 1, and the pupil is usually darker than the surrounding sclera and iris [97], we can reuse the output of the binarization operation I^t to determine the location of the pupil center. Specifically, we sum the I^t values within a $S \times S$ region centered at each pixel. The pixel with the highest sum means that it is surrounded by many white pixels, identifying it as the center of the pupil. If more than one pixel have the same maximum value, one is randomly chosen as the pupil center. Once the pupil is located, a predefined bounding box, centered on the pupil, crops the original eye image. Because VR HMDs are typically mounted directly on the user's head, the relative position between the eye camera and the eye remains nearly constant, enabling easy determination of hyperparameters, such as the bounding box size and

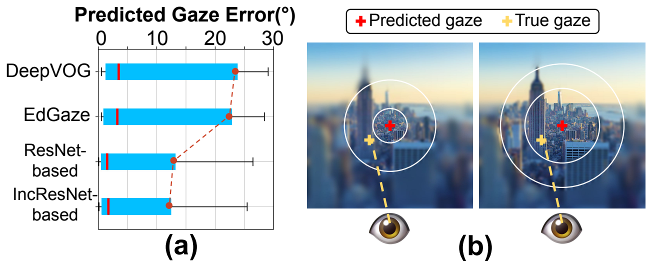

Figure 8: (a) Distribution of gaze errors on OpenEDS 2020 dataset, including mean, 5th, 95th percentile, min, and max angular errors. NVgaze results were excluded due to excessive tracking errors and unstable performance. (b) Visualization of the impact of gaze tracking errors on foveated rendering.

the value of M, using a small calibration dataset. The cropped input images will be processed by the gaze tracking DNN described in Section 4.3.

4.3 Efficient Gaze Tracking Solution

Our efficient gaze-tracking solution PoloNet integrates two components: (1) a lightweight, token-prunable ViT backbone that minimizes compute and memory, and (2) a performance-aware training objective that explicitly suppresses worst-case angular errors. These components achieve low average tracking error while sharply bounding the error tail, ensuring foveated-rendering quality.

Informative regions crops are initially resized to a compact (224×224) resolution before being fed into the gaze-tracking DNN for direction prediction. Following the ViT design of [27], the image is divided into fixed-size patches, tokenized, and enriched with positional encodings. Our backbone consists of eight transformer blocks, each with six self-attention heads and an embedding dimension of 384. The MLP head is modified to regress a 2-D gaze vector (θ_x, θ_y). After each self-attention layer, we compute the attention matrix $A = \mathrm{Softmax}\left(QK^\top/\sqrt{d_k}\right)$ and derive an importance score for every token. Tokens whose maximum attention weight falls below a threshold σ are discarded (Figure 6 (a)). Subsequent blocks therefore process fewer tokens, shrinking intermediate activations and FLOPs. Finally, all activations and weights are 8-bit quantized to further cut bandwidth and storage. The complete network is shown in Figure 7.

Meanwhile, typical gaze-tracking models minimize the average angular error, which leaves a long-tail distribution of outliers (Figure 8 (a)). In foveated rendering, even a small portion of large errors forces the foveal region to be enlarged (Figure 8 (b)), negating performance gains. The experimental results reveal that most previous work has focused solely on minimizing average gaze tracking performance, and none have optimized gaze tracking DNNs by considering the impact of tracking error distribution in foveated rendering applications. To mitigate this problem, we propose a training strategy by minimizing the maximum tracking error during training, which can be formulated as:

$$\min \max_{d \in D_{train}} (||\theta_d - \theta_d^g||^2) \tag{3}$$

Algorithm 1: POLONet Algorithm

```
Input: Current and previous frames $F^t$, $F^{t-1}$, buffered gaze $\theta_d$,
       pooling size $M$ and $S$, threshold $\gamma_1$ and $\gamma_2$.
       $\sigma(.)$ returns 1 is the event inside is true, and 0 otherwise.
       $f_{sac}(.)$, $f_{gaze}(.)$ denote the saccade detection and gaze
       detection ViT described in Section 4.1 and Section 4.3.
       $H_1$ and $H_2$ denote the size of the cropped image.
Output: Gaze direction
1  Initiation
2      Apply $M \times M$ average pooling over $F^t$;
3      $I^t_{ij} \leftarrow \sigma\left(F^t_{ij} < \gamma_1\right)$;
4      if $f_{sac}(I^t) == 1$ then
5          Saccade detect, halt rest operations.
6      else
7          if $\sum_{ij} |I^t_{ij} - I^{t-1}_{ij}| < \gamma_2$ then
8              return $\theta_d$;              // Return previous gaze
9          else
10             Sum $S \times S$ region centered at each pixel within $I^t$, find
                 the pupil center c;
11             Crop an $H_1 \times H_2$ region centered at $c$ within $F^t$.
                 Denote the cropped region $F^t_{crop}$;
12             $\theta_d = f_{gaze}(F^t_{crop})$;
13             return $\theta_d$.
14         $I^{t-1} \leftarrow I^t$;
```

where θ_d and θ_d^g denote the predicted gaze direction and the ground-truth gaze direction (in radians) for the input sample d in the training dataset D_{train}, respectively. To enhance training stability, the DNN is trained using multiple batches of training samples, resulting in Equation 4 being:

$$\min \sum_{b \in B} \max_{d \in D^b_{train}} (||\theta_d - \theta_d^g||^2) \tag{4}$$

where B denotes the set of training dataset batches, and D^b_{train} represents the set of training data in batch b. However, using this formula directly as the loss function can result in underutilization of the training dataset, as it tends to focus on only optimizing the sample with the highest tracking error. Empirically, we find it more effective to optimize an approximate version of Equation 4 by replacing the max operation with an alternative approach, using the approximation $max(x_1, x_2) \approx \frac{1}{N}\ln(e^{Nx_1} + e^{Nx_2})$. To further minimize the average gaze tracking error, we incorporate a weighted mean squared error term, scaled by a small factor λ, into the overall loss function:

$$\sum_{b \in B}\left[\frac{1}{N}\ln\Big(\sum_{d \in D^b_{train}} e^{N\|\theta_d - \theta_d^g\|^2}\Big) + \frac{\lambda}{|D^b_{train}|}\sum_{d \in D^b_{train}} \|\theta_d - \theta_d^g\|^2\right] \tag{5}$$

where N is the scaling factor that controls the degree of the approximation, $|D^b_{train}|$ is the size of the batch and λ represents the relative importance of the average-error term. During the training process, the values of N and λ are tuned carefully to adapt to the value distribution of the input training data to ensure the better convergence of the training process. Algorithm 1 presents the complete version of the POLONet algorithm discussed in Section 4.

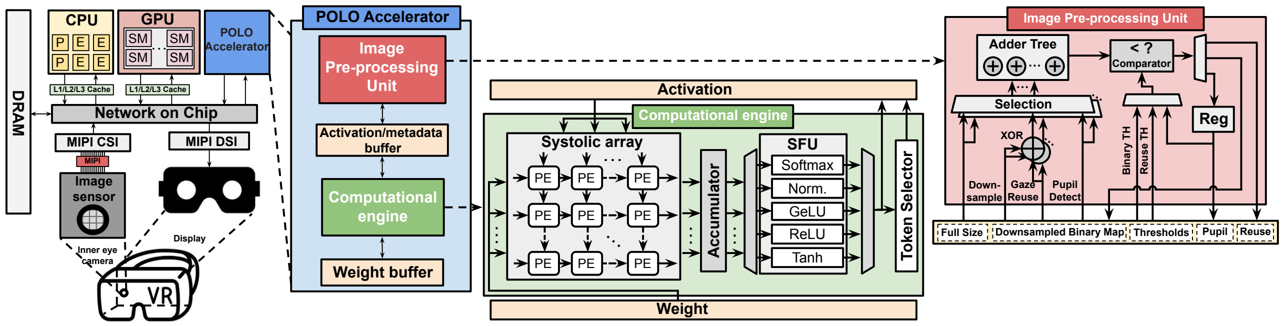

Figure 9: An overview of the POLO accelerator. POLO accelerator is integrated with the SoC of the VR HMD.

5 POLO Accelerator Design

In this section, we introduce the *POLO Accelerator* for efficient gaze processing, highlighted in Figure 9, the POLO accelerator serves as a dedicated module within the SoC of VR HMDs, consisting of three primary components: Image Pre-processing Unit (IPU) (Section 5.1), Computational engine (Section 5.2), and the memory subsystem. The POLO accelerator exclusively executes POLONet, offloading gaze processing from the GPU to reduce its workload and improve rendering efficiency. In Section 5.3, we discuss the computational workflow between POLO accelerator and GPU.

5.1 Image Pre-processing Unit Design

The architectural design of the IPU is illustrated on the far right of Figure 9. The IPU includes three main data paths that sequentially process the full-sized eye image. These paths are responsible for binarization map generation, gaze reuse determination and pupil center detection. The average pooling operation segments the input image into $M \times M$ ($M = 4$) pixel tiles, which are fetched from the buffer and processed through an adder tree within the IPU. The computed sum for each tile is compared to a scaled threshold γ_1 in a comparator for binarization. To reduce latency, division in linear interpolation is bypassed by scaling the threshold by $4 \times 4 = 16$, preserving the same binarization outcome. This process assigns binary values of 1 to darker pupil regions and 0 to other areas, producing a binary map that is stored in the buffer for subsequent processing, the data flow is highlighted in Figure 10 (a).

Gaze reuse is determined by comparing the previous frame's binary map with the current binary map, both stored in the buffer as inputs. Then the pixel differences between these two frames are calculated. Given that the eye image has already been binarized, corresponding pixels from the two binary maps are efficiently processed through an XOR gate array, producing the pixel differences with minimal computational overhead. The event map, representing gaze shifts between frames, is then passed through an adder tree to compute the sum of the entire map, quantifying the overall difference between the two frames. This sum is subsequently compared against a predefined threshold γ_2 to assess whether there is sufficient change in gaze direction. The entire data flow is shown in Figure 10 (b).

To precisely locate the pupil center and crop the image, a 5×5 window slides across the binary map, computing the sum of pixel values within each window. The window with the highest sum is

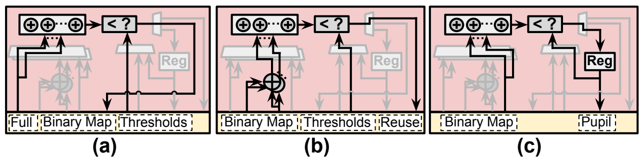

Figure 10: Image pre-processing data flows.

selected, and its center is designated as the pupil center. To identify the highest sum efficiently, each calculated sum is compared with the current highest sum stored in a register. This iterative comparison process, facilitated by a comparator, updates the stored highest sum whenever a new maximum is found. Additionally, to minimize unnecessary computations, the sliding window summation is performed on the binary map only when the center pixel of the window is 1, specifically for the white pixels that may correspond to the pupil. This selective calculation is effective due to the sparse distribution of white pixels, minimizing computation load while ensuring precise detection of the pupil center. The data flow is shown in Figure 10 (c).

To conserve hardware resources and enhance utilization, the three tasks share components such as the adder tree and comparator. This configuration allows for efficient resource reuse while maintaining the distinct functionality of each task.

5.2 Computational Engine

The computational engine is primarily responsible for implementing gaze tracking through a ViT and executing the neural network for saccade detection. Its primary components include a 16×16 systolic array, with each processing element (PE) containing an 8-bit multiply-accumulator (MAC), a token selector, and a special function unit (SFU), as illustrated in Figure 9. During operation, the systolic array processes inputs in a staggered manner, transmitting computed partial sums to the accumulator and SFU. The engine employs a weight-stationary data flow approach, where weights are pre-loaded from the weight SRAM into registers within each PE of the systolic array, while input data is streamed sequentially into the array. This design ensures efficient handling of data for gaze tracking and saccade detection tasks. To facilitate transposed matrix multiplication in ViT, we use a reconfigurable systolic array design proposed in [118], enabling in-place transposed matrix multiplication.

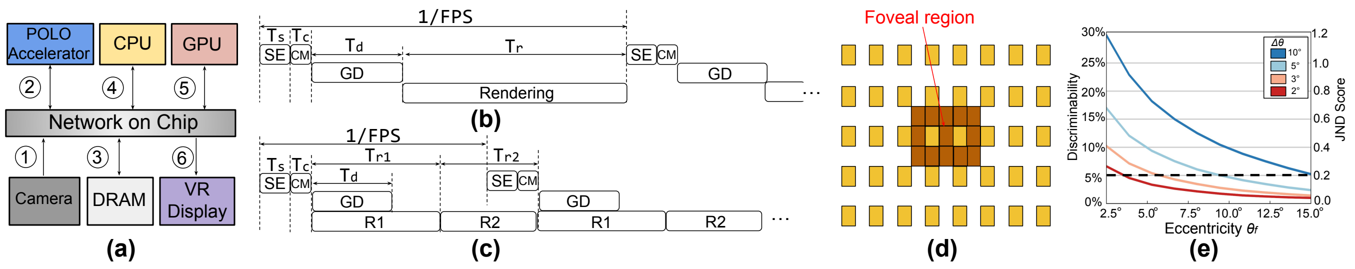

Figure 11: (a) Layout of the SoC of the VR device, with step numbers shown in circles. (b) TFR without parallel processing between gaze tracking (GT) and rendering, where SE, CM denote camera sensing, and MIPI communication. (c) TFR with parallel processing, where $R1$ and $R2$ denote the low-resolution and high-resolution rendering process for the foveal region. (d) The process of hierarchical foveated rendering. (e) Observer's ability to distinguish foveated from full-resolution images under varying θ_f and selected P95 tracking errors $\Delta\theta$. Left y-axis: discriminability probabilities; right y-axis: JND scores.

The special function unit is responsible for executing all nonlinear operations required by the ViT and saccade detection algorithms, including softmax, layer normalization, and activation functions such as GeLU, ReLU, and Tanh, as illustrated in Figure 9. To enhance efficiency in the softmax operation used within ViT, the SFU incorporates a lookup table (LUT) to approximate the exponential function. Another LUT is allocated for square root calculations essential to layer normalization, facilitating precise normalization with minimal computational burden. For activation functions, the SFU accelerates GeLU and Tanh via piecewise linear approximations, balancing computational efficiency and accuracy by segmenting these nonlinear functions into linear intervals. ReLU, specifically used in saccade detection, is implemented through comparator-based logic, eliminating resource-intensive operations. This streamlined SFU design supports diverse nonlinear processing with minimized power and latency overheads, meeting the high-throughput requirements of the accelerator.

The token selection process described in Section 4.3 is implemented by the token selector, which performs pruning after every two Transformer layers. Token importance is calculated by summing each column of the attention map after computing one head, using an adder array. After processing all heads in a layer, the complete importance score is compared in a comparator with a predefined threshold η, tokens with scores below the threshold are pruned by setting their 1-bit mask to 0, excluding them from subsequent computations. With the cropping algorithm implemented by the IPU and the token-wise pruning strategy outlined in Section 4.3, the memory footprint for storing inputs and intermediate results is significantly minimized, allowing for a compact activation buffer of only 128KB. The weight buffer is also configured to 128KB.

5.3 SoC Integration of POLO Accelerator

In this section, we outline how the POLO accelerator coordinates with other components and the computational flow of the TFR process within the SoC. The POLO accelerator is integrated with other SoC components via the Network-on-Chip (NoC), enabling efficient communication with the CPU, GPU, DMA, and additional components. The computational flow of TFR is depicted in Figure 11 (a). In the step 1, the camera captures the eye frame and sends it to the POLO accelerator's SRAM. Next, the processor processes the input image to determine the gaze direction or saccade decision (Step 2), storing the result in memory (Step 3). The CPU then instructs the GPU to use the POLO accelerator results (Step 4), which is applied to perform foveated rendering with high efficiency (Step 5). Finally, the rendered image is transmitted to the VR display and shown to the user (Step 6).

The computational flow of the described process is presented in Figure 11 (b). Given the small size of the gaze direction values, we ignore the DRAM access time, CPU processing time, and NoC transmission time for simplicity. Let T_s, T_c, T_d, and T_r represent the camera sensing time, MIPI communication time, gaze prediction processing time, and foveated rendering latency, respectively. According to the computational pattern shown in Figure 11 (b), The maximum processing throughput in FPS that TFR can support is given by $FPS_{max} = \frac{1}{T_s+T_c+T_d+T_r}$. T_d is the average processing latency of gaze processing and can be expressed as follows:

$$T_d = P_{sac}T_{sac,d} + P_{reuse}T_{reuse,d} + P_{pred}T_{pred,d} \tag{6}$$

where P_{sac}, P_{reuse}, and P_{pred} represent the probabilities of occurrence of saccade, gaze reuse, and gaze prediction, respectively, while $T_{sac,d}$, $T_{reuse,d}$, and $T_{pred,d}$ denote the corresponding processing latencies by the POLO accelerator. T_r, which represents the average time for foveated rendering processing, can be expressed as:

$$T_r = P_{sac}T_{sac,r} + (P_{reuse} + P_{pred}) \times T_{fr,r} \tag{7}$$

where $T_{sac,r}$ represents the processing latency to render the image at low resolution during a saccade. Previous studies [49, 55, 70] have demonstrated that the image can be rendered with uniform low resolution (e.g., 4×4) or directly reuse the last rendering view before the saccade process happen. $T_{fr,r}$ is the latency for foveated rendering during the fixation stage, when gaze direction is available.

To further reduce overall latency and improve throughput, the image rendering process can be performed in parallel with the gaze tracking process, as shown in Figure 11 (c). This parallelism is enabled by the hierarchical structure of foveated rendering, which processes each pixel independently. As illustrated in Figure 11 (d), the initial rendering R1 is done at a lower resolution tailored to the peripheral region's needs (Figure 3 (a)) and represented by the light yellow pixels. This stage can proceed without the need for gaze location data and can, therefore, run in parallel with the POLONet. When gaze tracking information is produced by POLO accelerator,

Table 1: Gaze tracking performance on OpenEDS 2020.

Method	Mean Error(°)	P90 Error(°)	P95 Error(°)
NVGaze	6.81	13.07	18.62
EDGaze	3.25	18.29	22.80
DeepVoG	3.47	17.76	23.77
ResNet-34	1.52	5.96	13.15
IncResNet	1.72	6.23	12.4
INT8-POLOViT(0.4)	2.26	4.93	5.91
INT8-POLOViT(0.2)	1.29	2.31	2.92
INT8-POLOViT(0.0)	**0.98**	**1.48**	**2.3**

the foveal region, indicated by the darker yellow pixels, is rendered next by processing the remaining pixels in this region as R2. The total TFR time T_{tot} and throughput can be expressed as:

$$T_{tot} = T_{r1} + T_{r2}, \quad FPS_{max} = \frac{1}{T_{r1} + T_{r2}} \tag{8}$$

where T_{r1} and T_{r2} are the processing latencies for R1 and R2. We assume T_d is shorter than T_{r1} at a standard image resolution like 720P or higher, based on the results in Section 7.4. With a fixed computational flow between hardware components during the TFR process, it is possible to preconfigure these flows, optimizing operation and communication patterns to enhance system performance.

6 POLONet Evaluation

We evaluate the POLONet using the OpenEDS2020 [81] dataset, which includes 128,000 images from 32 participants in the training set and 70,400 images from 8 participants in the validation set. It also includes annotations specifying the type of eye movement for each frame (*fixation*, *saccade*). We start by assessing the gaze tracking performance in Section 6.1, followed by an evaluation of saccade detection in Section 6.2, and ablation studies in Section 6.3.

6.1 Gaze Tracking Evaluation

In this section, we evaluate the gaze tracking error of the gaze tracking ViT (POLOViT), a core component of POLONet shown in Figure 7 and trained using the performance-aware training strategy described in Section 4.3. We compare POLOViT with five baseline methods, including NVGaze [56], EdGaze [36], DeepVoG [115], ResNet [44], and Inc-ResNet [9]. Each baseline algorithm is trained using its respective loss function from the original work to minimize the average gaze tracking error, with all DNNs trained under the same conditions. For EdGaze, we employ its default configuration, denoted as "eye_net_m" in [39], for evaluation. For training POLOViT, the scaling factor N in Equation 4 is set to 100, and binarization detection threshold γ_1 is set to 40. POLOViT is evaluated across three token pruning ratios by adjusting the threshold η, ranging from 0.0 (no pruning) to 0.4. Both weights and activations quantized to 8 bits. The evaluation metrics include the average gaze tracking error, as well as gaze tracking errors at 90th percentile (P90) and 95th percentile (P95).

Table 1 shows that POLOViT without pruning under 8-bit INT quantization, referred to as **INT8-POLOViT (0.0)**, achieves the optimal P95 error of 2.3° when trained using the performance-aware training strategy. This result is significantly better than those of all baseline algorithms. Furthermore, despite using a loss function

Table 2: Saccade detection performance of POLONet.

Dimension of h_t	16	32 (POLO)	64	128
Accuracy↑	99.0	99.4	99.4	99.6
Macro F1-score↑	0.92	0.95	0.95	0.97

designed to emphasize minimizing the maximum gaze error, INT8-POLOViT (0.0) also achieves the lowest average tracking error. Even with pruning ratios of 0.2 and 0.4, POLOViT continues to outperform the baseline methods in both average and max gaze tracking error. For implementation, we utilize INT8-POLOViT (0.2) to achieve an optimal balance between system performance and tracking accuracy.

6.2 Saccade Detection Evaluation

In this section, we evaluate the saccade detection performance of POLONet, as described in Section 4.1, which is formulated as a binary classification task[2]. As shown in Table 2, POLONet achieves a saccade detection accuracy of 99.5% and a Macro F1 score[3] of 0.95 when configured with an RNN hidden dimension of 32, as specified in Equation 2. Table 2 compares saccade detection accuracy across different RNN hidden dimensions for h_t. Although a hidden dimension of 128 enhances performance, it comes at the expense of increased computational costs. Moreover, the high Macro F1 score demonstrates that our method effectively identifies saccadic occurrences while minimizing the misclassification of non-saccadic events as saccades. This behavior is preferable, as it has a negligible impact on the user's visual experience.

6.3 Ablation Study on Hyperparameter

In this section, we analyze the impact of the hyperparameters γ_1 and γ_2 on gaze tracking error used in Algorithm 1. Specifically, γ_1 plays a critical role in guiding the binarization process and influencing saccade detection performance. As shown in Table 3, POLONet achieves the optimal Macro F1-score on the OpenEDS 2020 dataset with $\gamma_1 = 40$, which is adopted by POLONet.

Next, we investigate the impact of γ_2, which is the frame difference threshold for gaze reuse on the binarized image. We analyze the gaze tracking error of POLONet while accounting for gaze reuse on the OpenEDS 2020 dataset across different γ_2 thresholds. Table 4 specifically presents both the P95 error and the mean gaze tracking error, averaged over eye frames that reuse the results from the previous frame for gaze tracking prediction.

As γ_2 increases, the tracking error also rises. Specifically, when γ_2 is set to 10, the 95th-percentile error and average error are 3.35° and 1.39°, respectively, making this the setting adopted in POLONet. In contrast, setting γ_2 to 15 or 20 results in either higher computational costs or reduced gaze tracking accuracy.

7 POLO System Evaluation

In this section, we evaluate the performance of the POLO system described in Section 5. The proposed POLO accelerator was implemented in Verilog, with RTL synthesized using Synopsys Design Compiler [11] to estimate chip area, timing, and power, based on

[2]The criterion used to evaluate classification performance is the F1 score, which is the harmonic mean of precision and recall.
[3]The Macro F1 score is the average F1 score across all classes.

Table 3: Impact of γ_1

γ_1	Macro F1-score↑
35	0.93
40 (POLO)	0.95
45	0.94
50	0.94

Table 4: Impact of γ_2

γ_2	P95 Error(°)	Mean
≤ 5	3.08	1.32
≤ 10 (POLO)	3.35	1.39
≤ 15	3.8	1.47
≤ 20	4.34	1.68

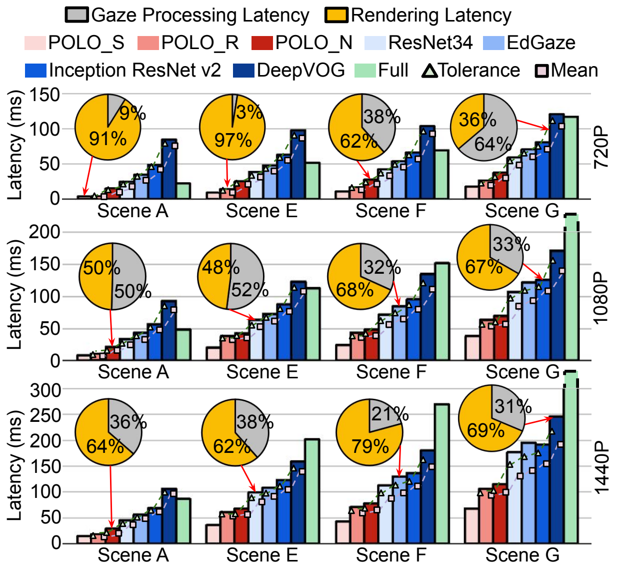

Figure 12: The end-to-end latency in three resolutions, across eight scenes, four scenes are shown due to space limit. Pie charts indicate the end-to-end latency breakdown for selected settings. For POLO_R, its tracking error is adopted using the results shown in Table 4 with $\gamma_2 = 10$.

45nm CMOS technology [51]. The design operates at 1 GHz, with on-chip buffers modeled using CACTI [10]. Synthesis results were scaled to 22nm using DeepScaleTool [94] to better align with current VR SoC technology. The synthesized POLO accelerator occupies an area of $0.75mm^2$ and delivers an average power consumption of $0.15W$. Area breakdown reveals on-chip buffers dominate (72%), followed by the computational engine (24%) and IPU (4%).

We employ Vulkan-Sim [91], a GPU simulator for graphics workloads, to execute ray tracing rendering tasks. Eight scenes from LumiBench [68] were selected to represent various levels of rendering complexity. We configure Vulkan-Sim [91] to simulate Jetson Orin NX 8GB version [3, 21], a widely used edge GPU for VR applications [42, 45, 82, 117, 124]. To precisely match real-world specifications, the simulated GPU is set up with 8 streaming multiprocessors running at a core clock frequency of 765 MHz [3]. Foveated rendering is simulated at input image resolutions of 720P (1280×720), 1080P (1920×1080), and 1440P (2560×1440). The foveal region's eccentricity angle θ_i was set to 5°, with the final angle θ_f calculated using Equation 1, where $\Delta\theta$ represents the 95th percentile gaze tracking error from the OpenEDS 2020 validation set [81] for each method. The eccentricity angle of inter-foveal regions is set 20 degrees larger than that of foveal regions. The resolution drop of the inter-foveal region and the peripheral regions are set

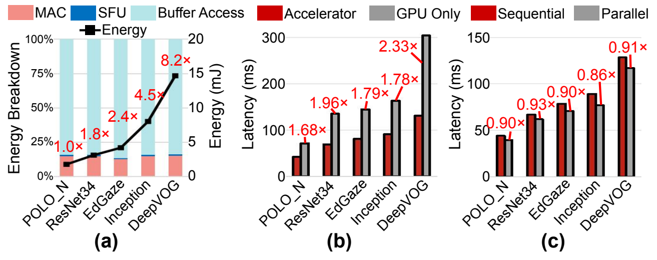

Figure 13: (a) Gaze tracking energy breakdown of all algorithms implemented on their corresponding accelerators. (b) TFR latency with and without accelerator. (c) Impact of computational patterns on TFR latency.

to 4× and 16×, respectively, in line with previous work [5, 22, 84]. We simulate an image sensor design based on the 2-layer stacked CMOS architecture developed in [67] for use in Meta VR HMDs. To align with current standards, the sensor's top layer uses a standard CMOS 65 nm process node, while the bottom layer utilizes a 22 nm process node. For performance evaluation regarding camera and SoC communication, we reference the latency and energy performance over the MIPI CSI interface as reported in [2].

The POLO framework comprises both the POLONet and the POLO accelerator, with the pruning ratio of gaze tracking ViT within POLONet set to 0.2. All other settings for POLONet adhere to those specified in Section 6. During saccade, the image is rendered with a low resolution with a downsampling ratio of 4×4. To evaluate the specific impact of the POLONet, we compare other algorithm baselines including ResNet34, IncResNet, EdGaze, DeepVOG. Each baseline gaze tracking algorithm is implemented on a dedicated accelerator featuring a systolic array, an accumulator, and an SFU for nonlinear operations, mirroring the configuration of the POLO accelerator. The accelerator layout is optimized to enhance performance for each gaze-tracking DNN within the same total chip area. The same synthesis settings as the POLO accelerator are applied to obtain end-to-end performance metrics, ensuring a fair comparison. The tracking error of each algorithm affects the eccentricity angle θ_f, leading to differences in foveated rendering cost. The total end-to-end latency is determined by combining the delays from each component, as illustrated in Figure 11 (b) and (c).

7.1 TFR System Evaluation

We begin by evaluating the TFR performance of POLO accelerator. Specifically, we examine POLO's performance in three scenarios: (1) during a saccadic event (POLO_S), (2) when the gaze location of the previous frame is reused (POLO_R), and (3) for standard gaze tracking without saccade influence or reuse (POLO_N). Since other gaze tracking algorithms do not support saccadic detection or gaze reuse, they are evaluated under standard gaze tracking. We adopt the computational pattern shown in Figure 11 (b).

Figure 12 shows the system performance regarding TFR latency over eight scenes at three resolutions: 720P, 1080P and 1440P. Notably, POLO_S and POLO_R show lower processing latencies compared to POLO_N and other baseline algorithms. This advantage is primarily due to the omission of the gaze tracking ViT, which reduces the execution latency of the POLO accelerator, as detailed

Table 5: Average TFR latency of eight scenes at 1080P with different pruning ratios and comparison with Vive Pro Eye.

Pruning Ratio	0	10%	20% (POLO)	30%	40%	Vive Pro Eye [47]
Latency (ms)	47.6	46.6	45.4	46.0	47.9	86.7

in the pie chart breakdowns of latency components. Furthermore, by leveraging input binarization and cropping (see Section 4.1), our lightweight DNN for saccade detection only requires less than 2% of the latency needed by the gaze tracking ViT on the POLO accelerator. POLO_N achieves gaze tracking latency up to 7.4× lower than the other algorithms for eye tracking, and its minimal gaze error enables a smaller foveal region, which in turn reduces rendering latency by approximately 1.5× compared to other algorithms. Specifically, at 720P, 1080P and 1440P resolutions, POLO_N achieves an average TFR latency reduction of 2.46×, 2.06× times and 1.85× compared to other algorithms on average. By averaging the latencies of POLO_S, POLO_R, and POLO_N based on the proportional occurrence of gaze reuse, saccades, and normal gaze tracking within consecutive frames of OpenEDS 2020, and applying equations 6 and 7, POLO demonstrates performance improvements of 3.42×, 2.50×, and 2.09× compared to other algorithms at 720P, 1080P, and 1440P resolutions, respectively.

Compared to full-resolution rendering, shown in green bars in Figure 12, POLO achieves consistently lower end-to-end latency across a range of scenes and resolutions. POLO_N reduces latency by up to 4.0× in the most complex scene, with an average reduction of 2.5× at 1080P. Specifically, at 720P, 1080P, and 1440P, the average latencies of POLO_N are 26ms, 44ms, and 69ms respectively across weight scenes, all satisfying the 50ms-70ms requirement for foveated rendering [5]. Moreover, DeepVOG, hindered by its computationally heavy gaze tracking neural network, experiences gaze tracking latencies exceeding 70 ms in many cases and results in even higher latencies than full-resolution rendering at 720P.

In addition to setting $\Delta\theta$ to the 95th percentile of gaze tracking error, we configure it to mean gaze tracking error over the OpenEDS 2020 dataset for each algorithm, resulting in varying computational costs for foveated rendering. The latency results, shown by the dotted line with a purple square marker in Figure 12, show that POLO still consistently outperforms baseline solutions in latency.

To preserve the visual experience affected by gaze tracking errors, the resulting foveal angle θ_f is calculated by adding the eccentricity angle θ_i of the foveal region to the P95 value of gaze tracking error $\Delta\theta$. In practice, users may have a degree of tolerance for gaze tracking errors, allowing visual experience to remain comparable to viewing a fully rendered image.

To account for this human effect, we follow prior work [12, 17, 26, 31, 48, 75], and utilize the FovVideoVDP metric [75] to evaluate visual quality in terms of discriminability and just-noticeable difference (JND) [23], as illustrated in Figure 11 (e). Specifically, discriminability represents the probability that a user, viewing a foveated rendering with a central foveal region of θ_f degrees and experiencing a P95 gaze tracking error of $\Delta\theta$, can discern a difference compared to a reference image rendered at full resolution. To ensure this probability of feeling the visual quality drop (i.e., discriminability) remains below a probability (e.g., 5%), θ_f can be

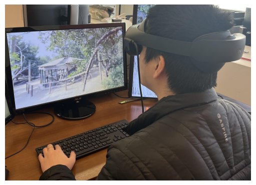
Figure 14: Participants are joining in the user study, which consists of 32 trials.

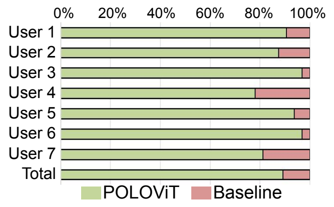

Figure 15: Selection results from the participants.

determined by identifying the intersection of the dotted horizontal line representing 5% with the corresponding curves, each depicting a different gaze tracking error. For example, as shown in Figure 11 (e), under P95 gaze tracking error of $\Delta\theta = 10^\circ$, θ_f needs to be around 15° to ensure the probability of feeling the visual quality drop is less than 5%. Using the θ_f obtained from Figure 11 (e), the performance is shown as a dotted line with a green triangle marker in Figure 12. We observe that a similar trend holds, where POLO achieves an average of 1.4×-1.5× reduction in rendering latency and up to 5.3× reduction in end-to-end TFR latency.

Figure 13 (a) shows the energy breakdown of the gaze-tracking accelerator for each algorithm to process one eye image of OpenEDS 2020, averaged across eight different scenes. The energy consumption of POLO_N is significantly lower than that of other algorithms, achieving an average of 4.1× reduction in energy consumption. Additionally, most of the energy is consumed by memory access, followed by the systolic array operations and SFU execution.

The POLO accelerator extends beyond conventional systolic arrays by integrating several innovative components that significantly enhance both performance and efficiency. The Image-Preprocessing Unit (Section 5.1) generates binarization maps, determines gaze reuse, and detects the pupil center using optimized hardware that maximizes resource reuse through bit-level operations, thereby eliminating the overhead of byte-level processing. The Token Selector (Section 5.2) accelerates the summation of attention scores and token pruning by filtering effective tokens using preset thresholds; a dedicated 1-bit mask flags token validity, obviating the need for computationally intensive sorting and reshaping, and substantially reducing memory accesses and latency associated with element-wise operations. Lastly, the SoC Integration (Section 5.3) employs a parallel processing strategy with hierarchical rendering, reducing end-to-end latency to just 90% of that observed in a sequential scheme (see Section 7.4), and ensuring seamless integration with the overall SoC architecture.

7.2 Impact of Gaze Tracking Accelerator

In this section, we analyze the isolated impact of the gaze tracking algorithms on overall TFR performance. Specifically, we remove the gaze tracking accelerator, making the GPU to handle both gaze tracking and foveated rendering for each baseline algorithm. As shown in Figure 13 (b), we observe that, without using gaze tracking accelerator, TFR latency of POLO_N increases by an average of 1.9× across all 8 scences at 1080P, as the GPU must now process both tasks. Despite this, POLO still achieves the lowest TFR latency among the candidate solutions. Our POLO accelerator outperforms GPU-only solution by efficiently handling nonlinear operations

Figure 16: Four different video clips.

and token-wise pruning in ViT. Specifically, it employs approximate hardware for activation functions on integer data (see Section 5), thereby accelerating computationally intensive tasks such as layer normalization and softmax [111]. Moreover, a dedicated token pruning unit further minimizes memory accesses and avoids high-latency operations such as top-k and reshaping. Finally, in real-time applications with small batch sizes and network architectures, the systolic array within POLO accelerator achieves a near-peak utilization rate, outperforming the GPU.

7.3 Impact on Pruning Ratio

In this section, we evaluate the impact of the token pruning ratio in the gaze-tracking ViT by adjusting the threshold σ to achieve overall pruning ratios of 0%, 10%, 20%, 30%, and 40%, where a higher σ results in a greater token filtering ratio and a smaller gaze-tracking latency. However, this also raises gaze-tracking error, leading to increased foveated rendering latency. As shown in Table 5, a pruning ratio of 20% yields the lowest average TFR latency across four scenes at 1080P, which is the setting adopted by POLO. We incorporate gaze tracking latency and error data from [46, 98] to simulate the TFR latency of a VR HMD equipped with the commercial eye tracker Vive Pro Eye [47]. At 1080P resolution, as shown in Table 5, the commercial system's average end-to-end latency was 1.91× slower than POLO_N.

7.4 Impact on Computational Pattern

We evaluate the TFR latency at 1080P resolution both with and without parallel processing, as shown in Figure 13 (c). For sequential processing, commercial HMDs with integrated eye trackers (e.g., Vive Pro Eye [47]) have a gaze detection delay (T_d) of up to 50 ms [98]. In contrast, under our POLO_N scenario, T_d is only 9.8 ms. Additionally, the rendering time (T_r) varies significantly with the graphics workload; for instance, for ray tracing, T_r may range from tens to nearly a hundred milliseconds as shown in Figure 12. For parallel processing, since R1 operates independently of gaze tracking, it can run concurrently with the gaze tracking process. Using the computational pattern depicted in Figure 11 (c) results in an average 9.4% reduction in TFR latency across various gaze tracking algorithms across eight scenes. Specifically, the latency of R1 averages 22 ms across all scenes, surpassing the 10.7 ms gaze tracking latency of POLO_N while remaining lower than that of other algorithms. For POLO_N, R1 fully overlaps with gaze tracking latency in most scenes, resulting in an average 10% reduction in end-to-end latency, with individual scene reductions ranging from 3 ms to 7 ms.

7.5 User Study

To assess the practical effectiveness of our eye-tracking method, we evaluate the visibility of artifacts in gaze-tracked foveated rendering. Specifically, we focus on measuring the gaze tracking accuracy of the POLOViT architecture, depicted in Figure 7. Seven participants were recruited for the study. As shown in Figure 14, during the experiment, participants remained seated and observed the stimuli via a HMD, the Meta Quest Pro [86]. Users interact with the study using a standard keyboard interface.

The stimuli include monoscopic 360° videos (20 seconds each) with gaze-tracked foveated rendering applied. The participants are instructed to perform a two-interval-forced-choice (2IFC) task [114]. In each trial, users are shown three conditions applied to the same video; the reference (unmodified) video, and two test videos, t1 and t2, with foveated rendering applied. The foveated rendering is applied with traces of gaze tracking errors at varying levels, referred to as e1 and e2, to assess their visual impact. One of the test videos uses gaze data with tracking errors from POLOViT (0.2), while the other uses error traces from ResNet-34, the best performing gaze tracking solution among the baseline algorithms in Table 1. These gaze tracking errors are artificially introduced on top of the eye tracker in the Quest device. During the user study, t1 and t2, which display the same visual content, are randomly paired with e1 and e2. Participants can switch between t1 and t2 using the keyboard to compare their relative visual quality. After observing both videos at least once, the participants were instructed to select the test video with higher visual quality. Four different video clips (sample frames shown in Figure 16) were presented to users. We down-selected these four videos from [32], which were chosen to cover a diverse range of scenes—including static/dynamic, rendered/real-world captured, bright/dark, and indoor/outdoor scenarios. Two videos include significant motion, and the other two are largely static. This results in 4 (videos) × 2 (tracking errors) × 4 (repeats) = 32 trials. For each participant, the 32 trials were presented in a random order to mitigate potential order effects.

Figure 15 depicts the results. Across participants, POLOViT was selected 90%±7% of the time over the baseline algorithm. The trend is consistent across individual videos 93%±9% for video 1, 73%±13% for video 2, 91%±12% for video 3, 100%±0% for video 4. These results evidence that the superior tracking accuracy of our gaze tracking solution can improve the quality of gaze-contingent applications such as foveated rendering.

8 Conclusion and Future Work

The cost of image rendering in VR environments is substantial, driven by the high-quality visual expectations of users. POLO addresses these challenges by leveraging the natural dynamics of human eye behavior to reduce the computational demands of the rendering process. By co-optimizing the AI-based gaze tracking solution with the underlying hardware system, POLO achieves a significant reduction in TFR latency and energy consumption.

Although POLO has been thoroughly evaluated through the methods outlined in Section 6 and Section 7, the impact of TFR latency on user visual experience remains an area for further exploration, even though previous studies suggest that a TFR latency of 50-70 ms is generally acceptable [5]. Additionally, the effect of saccade detection inaccuracies on user experience requires further investigation, and more comprehensive user studies on real HMDs will be necessary in the future.

References

[1] [n. d.]. NVIDIA Jetson Orin. https://www.siliconhighwaydirect.com/product-p/900-13767-0000-000.htm.

[2] [n. d.]. What is Mobile Industry Processor Interface (MIPI) Protocol? https://www.synopsys.com/blogs/chip-design/what-is-mobile-industry-processor-interface-protocol.html

[3] 2023. NVIDIA Jetson Orin NX 8 GB Specs. https://www.techpowerup.com/gpu-specs/jetson-orin-nx-8-gb.c4081 TechPowerUp GPU Database.

[4] Abdullah M Al-Ansi, Mohammed Jaboob, Askar Garad, and Ahmed Al-Ansi. 2023. Analyzing augmented reality (AR) and virtual reality (VR) recent development in education. *Social Sciences & Humanities Open* 8, 1 (2023), 100532.

[5] Rachel Albert, Anjul Patney, David Luebke, and Joohwan Kim. 2017. Latency requirements for foveated rendering in virtual reality. *ACM Transactions on Applied Perception (TAP)* 14, 4 (2017), 1–13.

[6] Richard Andersson, Linnea Larsson, Kenneth Holmqvist, Martin Stridh, and Marcus Nyström. 2017. One algorithm to rule them all? An evaluation and discussion of ten eye movement event-detection algorithms. *Behavior research methods* 49 (2017), 616–637.

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

[8] Arm Community. 2024. Achieving Mobile Gaming Success with Ray Tracing. https://community.arm.com/arm-community-blogs/b/mobile-graphics-and-gaming-blog/posts/mobile-gaming-success-with-ray-tracing Accessed: 2025-02-21.

[9] Rishi Athavale, Lakshmi Sritan Motati, and Rohan Kalahasty. 2022. One Eye is All You Need: Lightweight Ensembles for Gaze Estimation with Single Encoders. arXiv:2211.11936 [cs.CV] https://arxiv.org/abs/2211.11936

[10] Rajeev Balasubramonian, Andrew B Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 2 (2017), 1–25.

[11] B. Jayant Baliga. 2019. Synopsys. *Wide Bandgap Semiconductor Power Devices* (2019). https://api.semanticscholar.org/CorpusID:239327327

[12] David Bauer, Qi Wu, and Kwan-Liu Ma. 2022. Fovolnet: Fast volume rendering using foveated deep neural networks. *IEEE transactions on visualization and computer graphics* 29, 1 (2022), 515–525.

[13] Marie E Bellet, Joachim Bellet, Hendrikje Nienborg, Ziad M Hafed, and Philipp Berens. 2019. Human-level saccade detection performance using deep neural networks. *Journal of neurophysiology* 121, 2 (2019), 646–661.

[14] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the State of Neural Network Pruning?. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 129–146. https://proceedings.mlsys.org/paper_files/paper/2020/file/6c44dc73014d66ba49b28d483a8f8b0d-Paper.pdf

[15] Fergus W Campbell and Robert H Wurtz. 1978. Saccadic omission: why we do not see a grey-out during a saccadic eye movement. *Vision research* 18, 10 (1978), 1297–1303.

[16] 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. https://doi.org/10.1109/iccvw.2019.00568

[17] Kenneth Chen, Thomas Wan, Nathan Matsuda, Ajit Ninan, Alexandre Chapiro, and Qi Sun. 2024. PEA-PODs: Perceptual Evaluation of Algorithms for Power Optimization in XR Displays. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–17.

[18] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. 2021. Chasing Sparsity in Vision Transformers: An End-to-End Exploration. arXiv:2106.04533 [cs.CV] https://arxiv.org/abs/2106.04533

[19] Rajeswari Chengoden, Nancy Victor, Thien Huynh-The, Gokul Yenduri, Rutvij H Jhaveri, Mamoun Alazab, Sweta Bhattacharya, Pawan Hegde, Praveen Kumar Reddy Maddikunta, and Thippa Reddy Gadekallu. 2023. Metaverse for healthcare: a survey on potential applications, challenges and future directions. *IEEE Access* 11 (2023), 12765–12795.

[20] Woranipit Chidsin, Yanlei Gu, and Igor Goncharenko. 2021. AR-based navigation using RGB-D camera and hybrid map. *Sustainability* 13, 10 (2021), 5585.

[21] NVIDIA Corporation. [n. d.]. Jetson AGX Orin. Online. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/ Accessed: 2024-11-07.

[22] NVIDIA Corporation. 2018. Turing Variable Rate Shading in VRWorks. https://developer.nvidia.com/blog/turing-variable-rate-shading-vrworks/.

[23] Dixuan Cui and Christos Mousas. 2022. Estimating the Just Noticeable Difference of Tactile Feedback in Oculus Quest 2 Controllers. In *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 1–7. https://doi.org/10.1109/ISMAR55827.2022.00013

[24] Weiwei Dai, Ivan Selesnick, John-Ross Rizzo, Janet Rucker, and Todd Hudson. 2016. A parametric model for saccadic eye movement. In *2016 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*. IEEE, 1–6.

[25] Mr Ninad Janardan Dani. 2019. Impact of virtual reality on gaming. *Virtual Reality* 6, 12 (2019), 2033–2036.

[26] Nianchen Deng, Zhenyi He, Jiannan Ye, Budmonde Duinkharjav, Praneeth Chakravarthula, Xubo Yang, and Qi Sun. 2022. Fov-nerf: Foveated neural radiance fields for virtual reality. *IEEE Transactions on Visualization and Computer Graphics* 28, 11 (2022), 3854–3864.

[27] Alexey Dosovitskiy. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[28] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV] https://arxiv.org/abs/2010.11929

[29] Andrew T. Duchowski, Donald H. House, Jordan Gestring, Rui I. Wang, Krzysztof Krejtz, Izabela Krejtz, Radosław Mantiuk, and Bartosz Bazyluk. 2014. Reducing visual discomfort of 3D stereoscopic displays with gaze-contingent depth-of-field. In *Proceedings of the ACM Symposium on Applied Perception*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2628257.2628259

[30] Piotr Dudek and Peter J Hicks. 2000. A CMOS general-purpose sampled-data analog processing element. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 47, 5 (2000), 467–473.

[31] Budmonde Duinkharjav, Praneeth Chakravarthula, Rachel Brown, Anjul Patney, and Qi Sun. 2022. Image features influence reaction time: A learned probabilistic perceptual model for saccade latency. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–15.

[32] Budmonde Duinkharjav, Kenneth Chen, Abhishek Tyagi, Jiayi He, Yuhao Zhu, and Qi Sun. 2022. Color-Perception-Guided Display Power Reduction for Virtual Reality. *ACM Trans. Graph.* 41, 6 (2022). https://doi.org/10.1145/3550454.3555473

[33] Ralf Engbert and Reinhold Kliegl. 2003. Microsaccades uncover the orientation of covert attention. *Vision research* 43, 9 (2003), 1035–1045.

[34] Jasper H Fabius, Alessio Fracasso, Tanja CW Nijboer, and Stefan Van der Stigchel. 2019. Time course of spatiotopic updating across saccades. *Proceedings of the National Academy of Sciences* 116, 6 (2019), 2027–2032.

[35] Yao Feng, Haiwen Feng, Michael J Black, and Timo Bolkart. 2021. Learning an animatable detailed 3D face model from in-the-wild images. *ACM Transactions on Graphics (ToG)* 40, 4 (2021), 1–13.

[36] Yu Feng, Nathan Goulding-Hotta, Asif Khan, Hans Reyserhove, and Yuhao Zhu. 2022. Real-Time Gaze Tracking with Event-Driven Eye Segmentation. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 399–408. https://doi.org/10.1109/VR51125.2022.00059

[37] Yu Feng, Tianrui Ma, Adith Boloor, Yuhao Zhu, and Xuan Zhang. 2023. Learned in-sensor visual computing: From compression to eventification. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[38] Yu Feng, Tianrui Ma, Yuhao Zhu, and Xuan Zhang. 2024. BlissCam: Boosting Eye Tracking Efficiency with Learned In-Sensor Sparse Sampling. *arXiv preprint arXiv:2404.15733* (2024).

[39] Yu Feng and Yuhao Zhu. 2022. Real-Time Gaze Tracking with Event-Driven Eye Segmentation. https://github.com/horizon-research/edgaze GitHub repository.

[40] Linus Franke, Laura Fink, Jana Martschinke, Kai Selgrad, and Marc Stamminger. 2021. Time-Warped Foveated Rendering for Virtual Reality Headsets. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 110–123.

[41] Jaris Gerup, Camilla B Soerensen, and Peter Dieckmann. 2020. Augmented reality and mixed reality for healthcare education beyond surgery: an integrative review. *International journal of medical education* 11 (2020), 1.

[42] Antonin Gilles, Pierre Le Gargasson, Grégory Hocquet, and Patrick Gioia. 2023. Holographic near-eye display with real-time embedded rendering. In *SIGGRAPH Asia 2023 Conference Papers*. 1–10.

[43] Dan Witzner Hansen and Qiang Ji. 2010. In the Eye of the Beholder: A Survey of Models for Eyes and Gaze. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32, 3 (2010), 478–500. https://doi.org/10.1109/TPAMI.2009.30

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[45] Tairan He, Zhengyi Luo, Xialin He, Wenli Xiao, Chong Zhang, Weinan Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. 2024. OmniH2O: Universal and Dexterous Human-to-Humanoid Whole-Body Teleoperation and Learning. *arXiv preprint arXiv:2406.08858* (2024).

[46] Baosheng James Hou, Yasmeen Abdrabou, Florian Weidner, and Hans Gellersen. 2024. Unveiling Variations: A Comparative Study of VR Headsets Regarding Eye Tracking Volume, Gaze Accuracy, and Precision. In *2024 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. IEEE, 650–655.

[47] HTC Corporation. 2024. Vive Pro Eye. https://www.vive.com/sea/product/vive-pro-eye/overview/ Accessed: 2024-11-16.

[48] Xincheng Huang, James Riddell, and Robert Xiao. 2023. Virtual reality telepresence: 360-degree video streaming with edge-compute assisted static foveated compression. *IEEE Transactions on Visualization and Computer Graphics* (2023).

[49] Saad Idrees, Matthias Baumann, Felix Franke, Thomas Münch, and Ziad Hafed. 2020. Perceptual saccadic suppression starts in the retina. *Nature Communications* 11 (04 2020). https://doi.org/10.1038/s41467-020-15890-w

[50] Saad Idrees, Matthias P Baumann, Felix Franke, Thomas A Münch, and Ziad M Hafed. 2020. Perceptual saccadic suppression starts in the retina. *Nature communications* 11, 1 (2020), 1977.

[51] Silicon Integration Initiative. [n. d.]. Nangate 45nm Open Cell Library. https://si2.org/open-cell-and-free-pdk-libraries/. Accessed: 2024-11-07.

[52] Ye-Joon Jo, Jun-Seok Choi, Jin Kim, Hyo-Joon Kim, and Seong-Yong Moon. 2021. Virtual reality (VR) simulation and augmented reality (AR) navigation in orthognathic surgery: a case report. *Applied Sciences* 11, 12 (2021), 5673.

[53] Aleksei Karpov and Ilya Makarov. 2022. Exploring Efficiency of Vision Transformers for Self-Supervised Monocular Depth Estimation. In *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 711–719. https://doi.org/10.1109/ISMAR55827.2022.00089

[54] Sam Kavanagh, Andrew Luxton-Reilly, Burkhard Wuensche, and Beryl Plimmer. 2017. A systematic review of virtual reality in education. *Themes in science and technology education* 10, 2 (2017), 85–119.

[55] Jonghyun Kim, Youngmo Jeong, Michael Stengel, Kaan Akşit, Rachel Albert, Ben Boudaoud, Trey Greer, Joohwan Kim, Ward Lopes, Zander Majercik, Peter Shirley, Josef Spjut, Morgan McGuire, and David Luebke. 2019. Foveated AR: dynamically-foveated augmented reality display. *ACM Trans. Graph.* 38, 4, Article 99 (July 2019), 15 pages. https://doi.org/10.1145/3306346.3322987

[56] Joohwan Kim, Michael Stengel, Alexander Majercik, Shalini De Mello, David Dunn, Samuli Laine, Morgan McGuire, and David Luebke. 2019. NVGaze: An Anatomically-Informed Dataset for Low-Latency, Near-Eye Gaze Estimation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300780

[57] Zhenglun Kong, Peiyan Dong, Xiaolong Ma, Xin Meng, Mengshu Sun, Wei Niu, Xuan Shen, Geng Yuan, Bin Ren, Minghai Qin, Hao Tang, and Yanzhi Wang. 2022. SPViT: Enabling Faster Vision Transformers via Soft Token Pruning. arXiv:2112.13890 [cs.CV] https://arxiv.org/abs/2112.13890

[58] Eileen Kowler. 2011. Eye movements: The past 25 years. *Vision research* 51, 13 (2011), 1457–1483.

[59] Bart Krekelberg. 2010. Saccadic suppression. *Current Biology* 20, 5 (2010), R228–R229.

[60] HT Kung, Bradley McDanel, and Sai Qian Zhang. 2019. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 821–834.

[61] Yuna Kwak, Eric Penner, Xuan Wang, Mohammad R Saeedpour-Parizi, Olivier Mercier, Xiuyun Wu, Scott Murdison, and Phillip Guan. 2024. Saccade-Contingent Rendering. In *ACM SIGGRAPH 2024 Conference Papers*. 1–9.

[62] Philippe Lancheres and Mohamed Hafed. 2019. The MIPI C-PHY standard: A generalized multiconductor signaling scheme. *IEEE Solid-State Circuits Magazine* 11, 2 (2019), 69–77.

[63] Pil-Ho Lee and Young-Chan Jang. 2019. A 6.84 Gbps/lane MIPI C-PHY transceiver bridge chip with level-dependent equalization. *IEEE Transactions on Circuits and Systems II: Express Briefs* 67, 11 (2019), 2672–2676.

[64] Bin Li, Hong Fu, Desheng Wen, and WaiLun LO. 2018. Etracker: A Mobile Gaze-Tracking System with Near-Eye Display Based on a Combined Gaze-Tracking Algorithm. *Sensors* 18, 5 (2018). https://doi.org/10.3390/s18051626

[65] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* 461 (2021), 370–403. https://doi.org/10.1016/j.neucom.2021.07.045

[66] Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. 2021. Lost in Pruning: The Effects of Pruning Neural Networks beyond Test Accuracy. arXiv:2103.03014 [cs.LG] https://arxiv.org/abs/2103.03014

[67] Chiao Liu, Lyle Bainbridge, Andrew Berkovich, Song Chen, Wei Gao, Tsung-Hsun Tsai, Kazuya Mori, Rimon Ikeno, Masayuki Uno, Toshiyuki Isozaki, et al. 2020. A 4.6 μm, 512× 512, ultra-low power stacked digital pixel sensor with triple quantization and 127dB dynamic range. In *2020 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 16–1.

[68] Lufei Liu, Mohammadreza Saed, Yuan Hsi Chou, Davit Grigoryan, Tyler Nowicki, and Tor M. Aamodt. 2023. LumiBench: A Benchmark Suite for Hardware Ray Tracing. In *2023 IEEE International Symposium on Workload Characterization (IISWC)*. 1–14. https://doi.org/10.1109/IISWC59245.2023.00011

[69] Yanan Liu, Rui Fan, Jianglong Guo, Hepeng Ni, and Muhammad Usman Maqboo Bhutta. 2023. In-Sensor Visual Perception and Inference. *Intelligent Computing* 2 (2023), 0043.

[70] Lester C. Loschky and Gary S. Wolverton. 2007. How late can you update gaze-contingent multiresolutional displays without detection? *ACM Trans. Multimedia Comput. Commun. Appl.* 3, 4, Article 7 (Dec. 2007), 10 pages. https://doi.org/10.1145/1314303.1314310

[71] Conny Lu, Praneeth Chakravarthula, Kaihao Liu, Xixiang Liu, Siyuan Li, and Henry Fuchs. 2022. Neural 3D Gaze: 3D Pupil Localization and Gaze Tracking based on Anatomical Eye Model and Neural Refraction Correction. In *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 375–383. https://doi.org/10.1109/ISMAR55827.2022.00053

[72] Feng Lu, Takahiro Okabe, Yusuke Sugano, and Yoichi Sato. 2011. A Head Pose-free Approach for Appearance-based Gaze Estimation. In *British Machine Vision Conference*. https://api.semanticscholar.org/CorpusID:7733236

[73] Henna Mäkinen, Elina Haavisto, Sara Havola, and Jaana-Maija Koivisto. 2022. User experiences of virtual reality technologies for healthcare in learning: an integrative review. *Behaviour & Information Technology* 41, 1 (2022), 1–17.

[74] Rados Mantiuk, Bartosz Bazyluk, and Anna Tomaszewska. 2011. Gaze-Dependent depth-of-field effect rendering in virtual environments. In *Proceedings of the Second International Conference on Serious Games Development and Applications*. Springer-Verlag, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-23834-5_1

[75] Rafał K Mantiuk, Gyorgy Denes, Alexandre Chapiro, Anton Kaplanyan, Gizem Rufo, Romain Bachy, Trisha Lian, and Anjul Patney. 2021. Fovvideovdp: A visible difference predictor for wide field-of-view video. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–19.

[76] Ethel Matin. 1974. Saccadic suppression: a review and an analysis. *Psychological bulletin* 81, 12 (1974), 899.

[77] Pier Luigi Mazzeo, Dilan D'Amico, Paolo Spagnolo, and Cosimo Distante. 2021. Deep Learning based Eye gaze estimation and prediction. In *2021 6th International Conference on Smart and Sustainable Technologies (SpliTech)*. 1–6. https://doi.org/10.23919/SpliTech52315.2021.9566413

[78] Bradley McDanel, Sai Qian Zhang, HT Kung, and Xin Dong. 2019. Full-stack optimization for accelerating cnns using powers-of-two weights with fpga validation. In *Proceedings of the ACM International Conference on Supercomputing*. 449–460.

[79] NVIDIA. 2022. Real-Time Ray Tracing for Games and Beyond - SIGGRAPH 2022. https://www.nvidia.com/en-us/on-demand/session/siggraph2022-sigg22-s-11/ Accessed: 2025-02-21.

[80] Marcus Nyström and Kenneth Holmqvist. 2010. An adaptive algorithm for fixation, saccade, and glissade detection in eyetracking data. *Behavior research methods* 42, 1 (2010), 188–204.

[81] 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.

[82] Junrui Pan and Timothy G Rogers. 2024. CRISP: Concurrent Rendering and Compute Simulation Platform for GPUs. In *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–14.

[83] Anjul Patney, Joohwan Kim, Marco Salvi, Anton Kaplanyan, Chris Wyman, Nir Benty, Aaron Lefohn, and David Luebke. 2016. Perceptually-based foveated virtual reality. In *ACM SIGGRAPH 2016 Emerging Technologies* (Anaheim, California) *(SIGGRAPH '16)*. Association for Computing Machinery, New York, NY, USA, Article 17, 2 pages. https://doi.org/10.1145/2929464.2929472

[84] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. 2016. Towards foveated rendering for gaze-tracked virtual reality. *ACM Trans. Graph.* 35, 6 (2016). https://doi.org/10.1145/2980179.2980246

[85] Anitha S Pillai and Prabha Susy Mathew. 2019. Impact of virtual reality in healthcare: a review. *Virtual and augmented reality in mental health treatment* (2019), 17–31.

[86] Meta Quest Pro. 2022. https://www.meta.com/quest/quest-pro/tech-specs/#tech-specs.

[87] Qualcomm. 2024. Snapdragon 8 Gen 3 Mobile Platform. https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-3-mobile-platform Accessed: 2025-02-21.

[88] Qualcomm Technologies, Inc. 2023. Hardware-Accelerated Ray Tracing Improves Lighting Effects in Mobile Gaming. https://www.qualcomm.com/news/onq/2023/05/hardware-accelerated-ray-tracing-improves-lighting-effects-in-mobile-gaming Qualcomm Developer Blog.

[89] Qualcomm Technologies, Inc. 2023. Meta Quest 3. https://www.qualcomm.com/products/mobile/snapdragon/xr-vr-ar/xr-vr-ar-device-finder/meta-quest-3 Qualcomm Product Page.

[90] DA Robinson. 1964. The mechanics of human saccadic eye movement. *The Journal of physiology* 174, 2 (1964), 245.

[91] Mohammadreza Saed, Yuan Hsi Chou, Lufei Liu, Tyler Nowicki, and Tor M. Aamodt. 2022. Vulkan-Sim: A GPU Architecture Simulator for Ray Tracing. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 263–281. https://doi.org/10.1109/MICRO56248.2022.00027

[92] Dario D Salvucci and Joseph H Goldberg. 2000. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 symposium on Eye tracking research & applications*. 71–78.

[93] Samsung Semiconductor. 2024. Mobile Ray Tracing Technology. https://semiconductor.samsung.com/technologies/processor/mobile-ray-tracing/ Accessed: 2025-02-21.

[94] Satyabrata Sarangi and Bevan Baas. 2021. DeepScaleTool: A tool for the accurate estimation of technology scaling in the deep-submicron era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[95] Lars-Michael Schöpper, Markus Lappe, and Christian Frings. 2022. Saccadic landing positions reveal that eye movements are affected by distractor-based retrieval. *Attention, Perception, & Psychophysics* 84, 7 (2022), 2219–2235.

[96] Rahul Singh, Muhammad Huzaifa, Jeffrey Liu, Anjul Patney, Hashim Sharif, Yifan Zhao, and Sarita Adve. 2023. Power, Performance, and Image Quality Tradeoffs in Foveated Rendering. In *2023 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*. 205–214. https://doi.org/10.1109/VR55154.2023.00036

[97] W. Sprague, Zachary Helft, Jared Parnell, J. Schmoll, G. Love, and Martin Banks. 2013. Pupil shape is adaptive for many species. *Journal of Vision* 13 (07 2013), 607–607. https://doi.org/10.1167/13.9.607

[98] Niklas Stein, Diederick C Niehorster, Tamara Watson, Frank Steinicke, Katharina Rifai, Siegfried Wahl, and Markus Lappe. 2021. A comparison of eye tracking latencies among several commercial head-mounted displays. *i-Perception* 12, 1 (2021), 2041669520983338.

[99] Yusuke Sugano, Yasuyuki Matsushita, and Yoichi Sato. 2014. Learning-by-Synthesis for Appearance-Based 3D Gaze Estimation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 1821–1828. https://doi.org/10.1109/CVPR.2014.235

[100] Xiaoyu Sun, Xiaochen Peng, Sai Zhang, Jorge Gomez, Win-San Khwa, Syed Sarwar, Ziyun Li, Weidong Cao, Zhao Wang, Chiao Liu, et al. 2024. Estimating Power, Performance, and Area for On-Sensor Deployment of AR/VR Workloads Using an Analytical Framework. *ACM Transactions on Design Automation of Electronic Systems* (2024).

[101] Lech Świrski and Neil A. Dodgson. 2013. A fully-automatic, temporal approach to single camera, glint-free 3D eye model fitting [Abstract]. In *Proceedings of ECEM 2013* (Lund, Sweden). http://www.cl.cam.ac.uk/research/rainbow/projects/eyemodelfit/

[102] Khaled Takrouri, Edward Causton, and Benjamin Simpson. 2022. AR technologies in engineering education: Applications, potential, and limitations. *Digital* 2, 2 (2022), 171–190.

[103] UploadVR. 2023. NVIDIA Omniverse XR Brings Real-Time Ray Tracing to VR. https://www.uploadvr.com/nvidia-omniverse-xr-ray-tracing/ Accessed: 2025-02-21.

[104] Haiyu Wang, Wenxuan Liu, and Sai Qian Zhang. [n. d.]. Hardware and Algorithm Codesign for Efficient Gaze Tracking in Virtual Reality System. ([n. d.]).

[105] Kang Wang and Qiang Ji. 2017. Real Time Eye Gaze Tracking with 3D Deformable Eye-Face Model. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 1003–1011. https://doi.org/10.1109/ICCV.2017.114

[106] M. Weier, M. Stengel, T. Roth, P. Didyk, E. Eisemann, M. Eisemann, S. Grogorick, A. Hinkenjann, E. Kruijff, M. Magnor, K. Myszkowski, and P. Slusallek. 2017. Perception-driven Accelerated Rendering. *Computer Graphics Forum* 36, 2 (2017), 611–643. https://doi.org/10.1111/cgf.13150 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13150

[107] Thomas Westin, José Neves, Peter Mozelius, Carla Sousa, and Lara Mantovan. 2022. Inclusive AR-games for education of deaf children: Challenges and opportunities. In *European Conference on Games Based Learning*, Vol. 16. 597–604.

[108] Calden Wloka, Iuliia Kotseruba, and John K. Tsotsos. 2017. Saccade Sequence Prediction: Beyond Static Saliency Maps. arXiv:1711.10959 [cs.CV] https://arxiv.org/abs/1711.10959

[109] Erroll Wood, Tadas Baltrušaitis, Louis-Philippe Morency, Peter Robinson, and Andreas Bulling. 2016. A 3D Morphable Eye Region Model for Gaze Estimation. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 297–313.

[110] Erroll Wood, Tadas Baltrušaitis, Louis-Philippe Morency, Peter Robinson, and Andreas Bulling. 2016. Learning an appearance-based gaze estimator from one million synthesised images. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications (ETRA '16)*. 131–138. https://doi.org/10.1145/2857491.2857492

[111] Tianhua Xia and Sai Qian Zhang. 2024. Hyft: A Reconfigurable Softmax Accelerator with Hybrid Numeric Format for both Training and Inference. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*. 1–6.

[112] Yifan Xu, Zhijie Zhang, Mengdan Zhang, Kekai Sheng, Ke Li, Weiming Dong, Liqing Zhang, Changsheng Xu, and Xing Sun. 2021. Evo-ViT: Slow-Fast Token Evolution for Dynamic Vision Transformer. arXiv:2108.01390 [cs.CV] https://arxiv.org/abs/2108.01390

[113] Jiannan Ye, Anqi Xie, Susmija Jabbireddy, Yunchuan Li, Xubo Yang, and Xiaoxu Meng. 2022. Rectangular Mapping-based Foveated Rendering. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 756–764. https://doi.org/10.1109/VR51125.2022.00097

[114] Yaffa Yeshurun, Marisa Carrasco, and Laurence T Maloney. 2008. Bias and sensitivity in two-interval forced choice procedures: Tests of the difference model. *Vision research* 48, 17 (2008), 1837–1851.

[115] Yuk-Hoi Yiu, Moustafa Aboulatta, Theresa Raiser, Leoni Ophey, Virginia L. Flanagin, Peter zu Eulenburg, and Seyed-Ahmad Ahmadi. 2019. DeepVOG: Open-source pupil segmentation and gaze estimation in neuroscience using deep learning. *Journal of Neuroscience Methods* 324 (2019), 108307. https://doi.org/10.1016/j.jneumeth.2019.05.016

[116] Haoran You, Yang Zhao, Cheng Wan, Zhongzhi Yu, Yonggan Fu, Jiayi Yuan, Shang Wu, Shunyao Zhang, Yongan Zhang, Chaojian Li, et al. 2023. EyeCoD: Eye Tracking System Acceleration via FlatCam-Based Algorithm and Hardware Co-Design. *IEEE Micro* 43, 4 (2023), 88–97.

[117] Baoheng Zhang, Yizhao Gao, Jingyuan Li, and Hayden Kwok-Hay So. 2024. Co-designing a Sub-millisecond Latency Event-based Eye Tracking System with Submanifold Sparse CNN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5771–5779.

[118] Sai Qian Zhang, Bradley McDanel, and HT Kung. 2022. Fast: Dnn training under variable precision block floating point with stochastic rounding. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 846–860.

[119] Sai Qian Zhang, Thierry Tambe, Gu-Yeon Wei, and David Brooks. 2024. JointNF: Enhancing DNN Performance through Adaptive N: M Pruning across both Weight and Activation. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*. 1–6.

[120] Tongyu Zhang, Yiran Shen, Guangrong Zhao, Lin Wang, Xiaoming Chen, Lu Bai, and Yuanfeng Zhou. 2024. Swift-Eye: Towards Anti-blink Pupil Tracking for Precise and Robust High-Frequency Near-Eye Movement Analysis with Event Cameras. *IEEE Transactions on Visualization and Computer Graphics* 30, 5 (2024), 2077–2086. https://doi.org/10.1109/TVCG.2024.3372039

[121] Xucong Zhang, Yusuke Sugano, and Andreas Bulling. 2019. Evaluation of Appearance-Based Methods and Implications for Gaze-Based Applications. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3290605.3300646

[122] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. 2015. Appearance-based gaze estimation in the wild. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4511–4520. https://doi.org/10.1109/CVPR.2015.7299081

[123] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. 2017. MPIIGaze: Real-World Dataset and Deep Appearance-Based Gaze Estimation. arXiv:1711.09017 [cs.CV] https://arxiv.org/abs/1711.09017

[124] Ziliang Zhang, Zexin Li, Hyoseung Kim, and Cong Liu. 2024. BOXR: Body and head motion Optimization framework for eXtended Reality. *arXiv preprint arXiv:2410.13084* (2024).