Real-Time Gaze Tracking with Event-Driven Eye Segmentation

Yu Feng†
University of Rochester

Nathan Goulding-Hotta†
Reality Labs Research

Asif Khan‡
Reality Labs Research

Hans Reyserhove§
Reality Labs Research

Yuhao Zhu¶
University of Rochester

1 INTRODUCTION

Gaze tracking is critical to many fields [65], such as medicine [13, 15], human-machine interaction [19, 30, 66, 76], psychology [67], augmented reality (AR) [48, 72], and virtual reality (VR) [23, 46, 71, 83, 89]. Due to its importance, there has been much research improving the tracking accuracy, both using conventional infrared/RGB cameras [20, 50, 53, 56, 86] and emerging event cameras [12].

The speed of gaze tracking is critical. Gaze tracking, in many cases, directly interfaces with users, and is often just the first stage of an end-to-end application. Studies have shown that a tracking speed of at least 30 Hz is required for real-time user-facing applications such as VR [10]. Today’s eye tracking systems, however, are generally an order of magnitude slower than the requirement. For instance, on Nvidia’s Jetson Xavier, a powerful mobile computing platform, two state-of-the-art algorithms, RITNet [20] and DeepVOG [86], operate at a mere 5 Hz (while achieving a sub-0.5° gaze error).

This paper demonstrates a gaze tracking system that achieves 30 Hz tracking speed on a mobile GPU with a sub-0.5° gaze accuracy. Most prior eye tracking algorithms work on a frame-by-frame basis. However, eye tracking in virtually all use-cases processes continuous frames, which exposes temporal information that is often ignored. Our system leverages this temporal correlation for real-time tracking.

**Event-Driven Auto ROI** In particular, we use the temporal correlation across frames to predict the Region of Interest (ROI) in the current frame. We perform processing only in the ROIs whenever possible, falling back to full-resolution processing very rarely (0.6% of the time). The ROIs usually contain only 18%-32% of the pixels compared to full resolution, greatly improving the execution speed.

Our contribution here is a novel ROI prediction algorithm specialized for eye tracking. In particular, we emulate an event camera to extract the temporal correlation across frames, which then guides a lightweight deep neural network (DNN) to predict the ROI. The ROI prediction network essentially gives rise to an *Auto ROI mode* for eye tracking, akin to the conventional 3A algorithms for photography (auto focus, auto white balance, and auto exposure). Similar to the 3A algorithms, this “fourth A” mode operates in a feedback-driven fashion, where we use information extracted from the current frame (e.g., events, edges, ROI) to predict the ROI of future frames.
**Why Events?**  Event cameras report only the changes in pixel brightness level, a.k.a., events. We use events for ROI prediction for two reasons. First, events inherently encode the temporal changes, and thus provide natural cues to ROI prediction. Second, events are naturally sparse and can be efficiently encoded, enabling a small neural network with a low overhead in ROI prediction. Computation for the ROI prediction must be lightweight, in order to not offset the gains from processing only ROI images.

**Why Not an Event Camera?**  We do not directly use an event camera for two main reasons. First, event cameras are not readily available in many real-world systems (e.g., VR) and usually require dedicated downstream applications, which limit their applicability. Second, event cameras can operate at a frequency as high as 10K Hz, useful for capturing precise eye movement, such a high speed is usually an overkill for many consumer eye-tracking applications, e.g., user interface navigation in VR, where the target speed is “only” 30 Hz, easily achievable with conventional cameras.

**Efficiency-Oriented Segmentation**  Coupled with the Auto ROI mode, we propose a lightweight neural network-based algorithm to predict the gaze from an ROI. Like much of the recent work [20, 53, 56, 86], we use a model-based approach, where we first parameterize an eye image through eye segmentation, which is then used to estimate the gaze through fitting a geometric eye model.

We propose a new segmentation network designed with efficiency in mind. We show that depth-wise separable convolution with wider channels provides a segmentation backbone that balances accuracy and compute complexity. Our network requires 13–93 × fewer parameters (30K in total) and 20–31 × fewer arithmetic operations than existing DNN algorithms (RITNet [20] and DeepVOG [86]).

**Reducing Data Transmission**  Apart from delivering real-time tracking on off-the-shelf mobile platforms today, our system has the potential to reduce data transmission overhead, both between the image sensor and the back-end processor and between the processor and memory. Data transmission overhead is a major target for optimization, as the per-byte energy consumption of data transmission is almost three orders of magnitude higher than that of computation [61].

In particular, we show that our ROI prediction network, due to its small memory footprint and lightweight compute complexity, is amenable to execution on compute-capable image sensors, allowing only the ROI pixels or a small amount of metadata, as opposed to full-resolution images, to be transmitted out of the sensor. Using an industry-grade analytical model [61], we show that in-sensor Auto ROI provides 2–4 × reduction in the data transmission energy. Our dataset and code will be released. Our contributions are:

- We introduce event-driven Auto ROI for eye tracking, a novel event-driven ROI prediction method for eye tracking using software-emulated events and temporal feedback.
- We propose an accurate eye segmentation neural network containing with ROI prediction; the segmentation network is an order of magnitude simpler compared to prior algorithms.
- We show how our system can be integrated into emerging compute-capable image sensors to reduce the sensor data transmission energy for eye tracking.

## 2 RELATED WORK

**Eye Tracking Algorithms**  Eye tracking methods fall into two main categories: model-based methods and appearance-based methods [40, 89]. Appearance-based methods directly learn a mapping from an eye image to the gaze directions [64, 84, 90, 91]. We focus on the model-based method, which predicts gazes using a physiology-inspired eye model. Model-based methods are generally regarded as providing better accuracy than appearance-based methods [89].

**ROI Computing**  ROI is widely used to reduce the overall com-
Fig. 3: Overview of our event-driven eye segmentation. Time progresses from top to bottom. The segmentation results are used by a common gaze estimation algorithm \[78\], which is omitted in the figure. We generate an event map from every two consecutive eye frames; the current event map and previous segmentation map are combined to predict the ROI. If the number of events is small (e.g., shown at time \((T+2)\)), we can simply extrapolate the segmentation result rather than performing a full-blown segmentation.

3.1 Overview

Figure 3 shows a high-level flow diagram of our gaze tracking algorithm. We use a model-based, two-stage algorithm for gaze tracking. This paper’s contributions lie in the first stage, i.e., eye segmentation, while relying on a commonly-used gaze estimation model for the second stage \[78\]. Gaze estimation is relatively more mature and is much less compute intensive that the first stage (regression vs. DNN). Figure 3 thus omits the common gaze estimation part.

Initially at time \(T\), the full-resolution eye frame is processed directly by the segmentation network to generate the segmentation map. The next frame at \((T+1)\), instead of being processed directly, generates an event map, which is of the same dimension as the original full-resolution image. Each event map pixel is a 0/1 bitmask, indicating whether the corresponding pixel in the original image has a large change in intensity. This process emulates an actual event camera with simplifications tailored to eye segmentation.

The event map provides a useful guidance to predict the ROI in the current frame. We define the ROI as the minimal bounding box that encloses the three foreground eye parts, i.e., the pupil, iris, and sclera. Our approach can also apply to other tracking algorithms that use fewer or more segments (e.g., \[56, 86\]).

Events alone, however, are not robust enough. When the background eye muscles move significantly and/or when foreground eye parts move little between frames, activated events do not accurately capture the segment boundaries. To improve the robustness, we propose a feedback mechanism, where two important cues from time \(T\) (previous frame) are used to augment the ROI prediction.

For cases where the entire eye moves little between frames, e.g., at time \((T+2)\) in Figure 3, our ROI prediction algorithm would detect the inactivities and opt to extrapolate from the previous segmentation map. Having this move ensures “activity-proportional” tracking, where little tracking work is done when little activity is observed.

In most cases, only the first frame has to be processed in its full resolution. In rare \((<0.02\%)\) cases where the predicted ROI is physically infeasible (e.g., the top-right corner is to the left of the bottom-left corner), we fall back to the full-resolution mode.

3.2 Auto ROI with Event-Driven ROI Prediction

The ROI prediction algorithm has two roles: 1) predicting the ROI of the current eye frame, and 2) deciding whether the current eye frame requires going through a full-fledged segmentation algorithm or can be extrapolated from the previous segmentation map. Figure 3 shows the structure of our ROI prediction network.

3.2.1 The Prediction Algorithm

Intuition The goal of ROI prediction is to filter out background pixels (eye muscles, skin, and eyelids) while leaving only the foreground eye parts, i.e., the pupil, iris, and sclera. We use events to guide the ROI prediction. The intuition is that the background parts do not move as significantly as the foreground eye parts. Thus, activated pixels in the event map mostly correspond to the foreground eye parts and provide a useful guidance to the ROI prediction.

We generate the event map by emulating the high-level process of an event camera \[85\]. We first calculate the pixel-wise absolute difference, \(\Delta F_{t+1}\), between two consecutive frames, \(F_t\) and \(F_{t+1}\). Next, each difference value, \(\Delta F_t(x, y)\), is normalized by the corresponding pixel value in \(F_t\). The normalized pixel values go through an activation function, \(\Phi\), which generates an event when the normalized difference is greater than a predefined threshold value. Mathematically, generating an event map can be expressed as:

\[
E_{t+1}(x, y) = \Phi(|F_t(x, y) - F_{t+1}(x, y)| / F_t(x, y), \sigma) \tag{1}
\]

where \(E_{t+1}(x, y)\) is the value at coordinate \((x, y)\) of the event map at time \((T+1)\), and \(\Phi\) is the activation function which outputs 1 if the difference is greater than the threshold \(\sigma\) (and 0 otherwise). The threshold is a parameter that can be tuned for a specific application or scenario. Section 3.2 will show that the tracking accuracy is robust against the choice of \(\sigma\), but there does exist a sweet spot. Normalizing pixel differences by the previous values mimics the log-scale absolute difference operation done by an actual event
While our algorithm relies on events, the two additional cues of ROI and edge map from the previous frame are critical to the accuracy. We feed the previous ROI into the ROI prediction algorithm (a DNN), which learns to correlate the previous ROI with the current ROI. The intuition is that the ROI of the previous frame provides useful information to predict the ROI in the current frame due to the temporal correlation. Thus, using the previous predicted ROI, however, has an inherent downside: the ROI prediction errors will accumulate and the predicted ROI will drift over time — the previous segmentation map is predicted directly from an actual eye image from the camera; thus, the segmentation map does not accurately represent the foreground eye parts, we feed the previous ROI back to the prediction algorithm. The edge map is a bitmask that indicates whether the two different classes in the segmentation map. We use Canny edge detection on the segmentation map to form the input to the ROI prediction network. The first three layers are Conv layers with $3 \times 3$ kernels followed by a Maxpool layer to reduce each dimension to 1/2. The output of the last Conv layer is vectorized and concatenated with the ROI from time $T$ (a $1 \times 4$ vector). The concatenated vector then goes through two FC layers to generate the predicted ROI of time $(T + 1)$. While our algorithm relies on events, the two additional cues of ROI and edge map from the previous frame are critical to the accuracy.

![Fig. 4: The process of predicting the ROI at time $(T + 1)$ consists of three steps. First, we compute the absolute difference of frames at $T$ and $(T + 1)$ to generate an event map. Second, we use Canny edge detection on the segmentation map at time $T$ to extract an edge map. Finally, we concatenate the edge map and the event map to form the input to the ROI prediction network. The first three layers are Conv layers with $3 \times 3$ kernels followed by a Maxpool layer to reduce each dimension to 1/2. The output of the last Conv layer is vectorized and concatenated with the ROI from time $T$ (a $1 \times 4$ vector). The concatenated vector then goes through two FC layers to generate the predicted ROI of time $(T + 1)$. While our algorithm relies on events, the two additional cues of ROI and edge map from the previous frame are critical to the accuracy.](image)

![Fig. 5: Example of ROI misprediction using only the event map. In the left image, the solid ROI is ground truth, and the dashed ROI is the predicted ROI from using only the event map.](image)

Two Feedback Cues While events are largely effective, there are cases where using events alone fails. In particular, when eye muscles/eyelids move significantly and/or foreground eye parts move little, events do not accurately capture the eye boundary anymore, challenging the assumption of using events to predict ROIs.

To cope with the issue where events in the current event map do not accurately represent the foreground eye parts, we feed the previous ROI back to the prediction algorithm. The intuition is that the ROI of the previous frame provides useful information to predict the ROI in the current frame due to the temporal correlation. Thus, we feed the previous ROI into the ROI prediction algorithm (a DNN), which learns to correlate the previous ROI with the current ROI.

![Prediction Network](image)

Using the previous predicted ROI, however, has an inherent downside: the ROI prediction errors will accumulate and the predicted ROI will drift over time. To mitigate error accumulation, we use an additional cue that does not drift over time — the previous segmentation map. The segmentation map is predicted directly from an actual eye image from the camera; thus, the segmentation map does not drift over time. To reduce data size, we extract an edge map from the segmentation map. The edge is defined as the boundary between two different classes in the segmentation map. We use Canny edge detection to obtain an edge map from a segmentation map. Each pixel value in the edge map is a bitmask that indicates whether the corresponding pixel in the eye frame is a boundary.

### 3.2.2 Activity-Proportional Segmentation

In cases where the entire eyes do not move across frames, detecting the inactivities and skipping segmentation altogether improves the execution speed. This allows our eye tracker to be activity-proportional: no work when no activity is detected. Two issues remain: how to detect inactivities and how to compute an accurate segmentation map extremely fast for inactive eye frames?

Building on top of our ROI prediction algorithm, we detect inactivity by calculating the event density of the event map inside the predicted ROI. If the event density is lower than a threshold $\gamma$, the current frame is deemed inactive, in which case the ROI prediction algorithm sets the extrapolation bit, indicating that no segmentation is to be executed for the current frame. While other inactivity detection schemes are possible, we favor the simple thresholding scheme because the threshold $\gamma$ provides a useful knob to control the speed-vs-accuracy trade-off, allowing our system to potentially adapt to different application requirements and hardware capability.

We experiment with a range of extrapolation schemes, ranging from simply scaling the segments from the previous frame to using a neural network to predict how to morph the segmentation from the previous frame. We eventually settled for the simplest scheme, where the previous segmentation result is reused. In retrospect, this scheme is the most robust because it relies the least on the inactivity detection scheme, which is a simple thresholding.

### 3.3 Efficient Eye Segmentation Network

The Auto ROI mode avoids full-resolution segmentation almost altogether, but the segmentation stage is still the single biggest bottleneck in the system, contributing to over 85% of the total execution time in our measurements. We propose a new segmentation network architecture that is much simpler without sacrificing accuracy.

Figure 6 shows our segmentation network inspired by U-Net [75]. Our architecture contains five downsampling blocks and four upsampling blocks, similar to the original U-Net design. However, unlike...
As we will show, the channel width is an effective knob that provides an opportunity to reduce the data transmission overhead. Our idea is to execute part of the algorithm inside the image sensor to reduce the data transmission volume and thereby the transmission energy. In-sensor computing is possible as image sensors integrate processing capabilities with the conventional pixel array in one sensor, sometimes in one single chip [18,42,58]. A recent example is Sony’s IMX 500 image sensor that provides dedicated memories and a digital signal processor for executing DNNs [27,28].

Table 1: Total number of Floating Point Operations (FLOPs) and the output data size of the four major components in our algorithm for a 640 × 400 pixel grayscale input. For this estimate we assume the ROI image is one-third the size of the full resolution, and that half of the frames are extrapolated. We show the full-resolution image size for reference. Recall that data transfer consumes 800 times more energy than computation per byte [61].

| Stages | Event Map Generation | Edge Map Generation | Prediction Network | Seg. Network | Full-res Image |
|--------|----------------------|---------------------|--------------------|--------------|---------------|
| FLOPs (Million) | 0.3 | 1.9 | 55.4 | 2641.6 | NA |
| Output Data (KB) | 7.8 | 7.8 | 41.7 | 62.5 | 250 |

While conceptually simple, reaping the reward of in-sensor computing is not trivial. Computation inside image sensors consumes more energy than that on a backend processor. This is because the semiconductor fabrication technology, quantified by the process node, of the sensor usually lags at least one generation behind that of the processor. Today, many commercial processors are fabricated using a 7 nm process node or smaller, but even high-end image sensors still use a 14 nm or 28 nm process node [55,57]. The computation power consumption increases quadratically with respect to the process node [25]. Therefore, one must carefully weigh the reduction of data transmission energy against the overhead of computing inside an image sensor on an older process node.

Our goal is thus to map the different components of our eye tracking algorithm to the sensor and processor in such a way that the total energy consumption is minimized. Intuitively, the ideal mapping is one where a trivial amount of computation in the sensor can drastically reduce the data communication.

To identify the optimal mapping, we first quantify the computation cost and the output data volume of the four main components in our algorithm: event map generation, edge map generation, prediction network, and segmentation network; the first three comprise the ROI prediction algorithm as shown in Figure 4. Table 1 shows the results. Note that each image pixel uses 1 byte; each pixel in the event map and the edge map uses 1 bit; and each segmentation map pixel uses 2 bits (4 classes).

---

**3.4 Reducing Sensor Data Transmission**

So far we have focused on improving the compute efficiency of the eye tracking algorithm. However, the data transmission overhead, both between the image sensor and the processor and between the processor and memory, is non-trivial. A recent study from Facebook shows that the average energy of data transmission per byte is about 800 times higher than the computation energy per byte [61]. Similar observations are made in other recent studies [39,54].

The Auto ROI capability of our eye tracking algorithm provides an opportunity to reduce the data transmission overhead. Our idea is to execute part of the algorithm inside the image sensor to reduce the data transmission volume and thereby the transmission energy. In-sensor computing is possible as image sensors integrate processing capabilities with the conventional pixel array in one sensor, sometimes in one single chip [18,42,58]. A recent example is Sony’s IMX 500 image sensor that provides dedicated memories and a digital signal processor for executing DNNs [27,28].

Table 1: Total number of Floating Point Operations (FLOPs) and the output data size of the four major components in our algorithm for a 640 × 400 pixel grayscale input. For this estimate we assume the ROI image is one-third the size of the full resolution, and that half of the frames are extrapolated. We show the full-resolution image size for reference. Recall that data transfer consumes 800 times more energy than computation per byte [61].

| Stages | Event Map Generation | Edge Map Generation | Prediction Network | Seg. Network | Full-res Image |
|--------|----------------------|---------------------|--------------------|--------------|---------------|
| FLOPs (Million) | 0.3 | 1.9 | 55.4 | 2641.6 | NA |
| Output Data (KB) | 7.8 | 7.8 | 41.7 | 62.5 | 250 |

While conceptually simple, reaping the reward of in-sensor computing is not trivial. Computation inside image sensors consumes more energy than that on a backend processor. This is because the semiconductor fabrication technology, quantified by the process node, of the sensor usually lags at least one generation behind that of the processor. Today, many commercial processors are fabricated using a 7 nm process node or smaller, but even high-end image sensors still use a 14 nm or 28 nm process node [55,57]. The computation power consumption increases quadratically with respect to the process node [25]. Therefore, one must carefully weigh the reduction of data transmission energy against the overhead of computing inside an image sensor on an older process node.

Our goal is thus to map the different components of our eye tracking algorithm to the sensor and processor in such a way that the total energy consumption is minimized. Intuitively, the ideal mapping is one where a trivial amount of computation in the sensor can drastically reduce the data communication.

To identify the optimal mapping, we first quantify the computation cost and the output data volume of the four main components in our algorithm: event map generation, edge map generation, prediction network, and segmentation network; the first three comprise the ROI prediction algorithm as shown in Figure 4. Table 1 shows the results. Note that each image pixel uses 1 byte; each pixel in the event map and the edge map uses 1 bit; and each segmentation map pixel uses 2 bits (4 classes).
The average duration of a sequence is 30 seconds with diverse eye movements such as blinks and saccades (rapid eye movements). The data is captured by an infrared camera at 100 Hz with a 640×640-pixel resolution, from 152 participants with various ethnicities and iris colors. All sequences are continuous frames (29,476 frames in total) gathered from the OpenEDS 2020 dataset, which contains 200 sequences of 100 Hz with a 400×640-pixel resolution, which does have ground truth segmentation labels. We then use the trained RITnet to generate the eye segmentation results for the sequential OpenEDS 2020 data.

The generated eye segmentation results are not perfect; we perform a series of data refinement steps to generate the final ground truth. We first apply the DBSCAN spatial clustering algorithm on each initial segmentation result and identify the largest contiguous region as the correct eye region. We then fill the missing holes inside the eye region as a prior study does. Finally, we manually inspected all sequences of the refined data and removed 15 sequences (out of 200), where the eye segmentation results are visibly incorrect. In the end, our sequential dataset contains 185 sequences and 27,431 total frames. Figure 8 shows an example before and after refinement, where the disconnected regions in the initial result are removed, providing a more faithful ground truth.

With the eye segmentation ground truth, we then apply a gaze estimation algorithm commonly used in prior work (e.g., DeepVOG) to generate the gaze ground truth. Our algorithm does not depend on a particular gaze estimation method and, thus, can be integrated with other gaze estimation models. We also use the eye segmentation ground truth to generate the ROI ground truth, which is the bounding box of the foreground segment (pupil, iris, and sclera). We manually verify that using so-defined ground-truth ROIs gives the same gaze results as those from full-frame inference.

Another dataset we used in our evaluation is the TEyeD dataset, which has manually annotated segmentation and gaze ground truth. We select 2 sequences from the dataset. Both sequences are grayscale images at a 384×288 resolution of captured at 25 Hz with an average length of over 30 minutes. In total, the selected data provide over 100,000 images.

On the TEyeD dataset, we report the 2D pupil position error (in pixels) rather than the 3D gaze direction error. This is because the gaze estimation algorithm commonly used in prior work (e.g., DeepVOG) is not based on a particular gaze estimation method and, thus, can be integrated with other gaze estimation models. We also use the eye segmentation ground truth to generate the ROI ground truth, which is the bounding box of the foreground segment (pupil, iris, and sclera). We manually verify that using so-defined ground-truth ROIs gives the same gaze results as those from full-frame inference.

4.2 Training
We first train a standalone eye segmentation network and a standalone ROI prediction network, and then refine eye segmentation using the ROI prediction results.

We follow prior studies and split the entire dataset into the training/validation/test sets with a 80/20 training/validation split, identical to EllSeg. The eye segmentation network is trained using the Adam optimizer, a learning rate of 0.001, and a batch size of 4 for 250 epochs. We use the a loss function that combines both the standard cross-entropy loss and losses that are specialized to the eye structure. To train the ROI prediction network, we use the Adam optimizer with a learning rate of 0.001, a momentum of 0.9, and the mean squared error loss function. We train the ROI prediction network for 100 epochs with a batch size of 8.

We then fine-tune the eye segmentation network using ROI images. In this step, the ROI prediction network first predicts an ROI given an eye image; we then crop the input image based on the predicted ROI and fine-tune the segmentation network with the cropped image. This training procedure is similar to networks based on region proposals such as Faster R-CNN. We use a learning rate of 0.0001, a momentum of 0.9, and 100 epochs.

4.3 Baseline and Evaluation Metrics
We primarily compare against eye segmentation methods that are based on DNNs, which are shown to have superior accuracy compared to non-DNN-based segmentation methods. The baselines are trained using the same procedure as our algorithm.

- **RITnet**: an encoder-decoder network using DenseNet as the backbone. It won first place in the 2019 OpenEDS...
Horizontal Angular Error (deg.)

Vertical Angular Error (deg.)

Fig. 10: The accuracy and speed comparison of different methods. All the subfigures share the same legend. The speedup values are normalized to the speed of RITnet. Ours(S) and Ours(L) are two eye segmentation networks. +ROI denotes ROI prediction is enabled. +E denotes the extrapolation is enabled. +ROI (SIFT) denotes using the SIFT-based ROI prediction.

segmentation challenge [6]. As discussed in Section 4.1, we use RITnet to generate the ground truth for the sequential OpenEDS data. As a result, it is expected that our algorithm will have lower accuracy than RITnet.

- **DenseElNet** [56]: an ellipse segmentation framework for gaze tracking. DenseElNet is originally for ellipse segmentation (three segments). We re-purpose DenseElNet to predict four segments by modifying the channel size of the last layer.
- **DeepVOG** [56]: a popular encoder-decoder network for eye segmentation. DeepVOG was originally designed to generate only two segments (pupil and background). We modified the channel size of the last layer so that it predicts four segments.

We also compare the speed of our algorithms with the baselines. We measure the speed on a state-of-the-art mobile computing platform, Nvidia’s Jetson Xavier board [3]. We use the mobile Volta GPU on the Xavier board for all the networks. The GPU has 512 CUDA cores with a maximum frequency of 1,377 MHz.

### 4.4 Variants

We use two DNN variants of our eye segmentation network, Ours(S) and Ours(L). Both are designed with the same architecture, but differ in the channel width and thus provide a speed-accuracy trade-off. In particular, Ours(S) has, on average, half the channels of Ours(L). Table 2 compares the amount of Floating Point Operations (FLOPs) and parameters of the two networks against the baseline segmentation networks.

Table 2: FLOPs and number of parameters in different eye segmentation networks for an input size of 640 × 400 8-bit pixels.

| Network       | DenseElNet | DeepVOG | RITnet | Ours(L) | Ours(S) |
|---------------|------------|---------|--------|---------|---------|
| FLOPs (Billion) | 53.1       | 36.5    | 23.1   | 2.6     | 1.2     |
| Norm. FLOPs   | 45.3       | 31.1    | 19.7   | 2.3     | 1.0     |
| # of Parameters (Thousand) | 3047.3 | 2835.7 | 391.0  | 73.0    | 30.6    |
| Norm. # of Parameters | 99.6   | 92.6    | 12.8   | 2.4     | 1.0     |

The total computation requirement of Ours(L) is only about 1/9 of RITnet, 1/14 of DeepVOG, and 1/20 of DenseElNet. Ours(S) is even lighter weight, with about 45 × fewer FLOPs and 100 × fewer parameters compared to DenseElNet. Overall, Ours(S) uses only about 30K parameters (~120 KB).

To tease apart the contributions of the different components of our algorithm, we evaluate two variants on top of eye segmentation:

- +ROI: this variant uses only the ROI prediction (Section 3.2.1)
- +ROI+E: this variant uses both ROI prediction (Section 3.2.1) and extrapolation (Section 3.2.2)

5 EVALUATION

We first use OpenEDS 2020 to drive the analysis of the results. We present the overall performance and accuracy comparison (Section 5.1), followed by analyzing the sensitivity of our system to various parameters (Section 5.2) and algorithmic choices (Section 5.3). We perform an ablation study to evaluate the importance of the two feedback cues (Section 5.4). We present the results from the TEyeD dataset (Section 5.5) and analyze failure cases (Section 5.6). Finally, we present the results of in-sensor auto-ROI (Section 5.7).

### 5.1 Accuracy vs. Speed

**Gaze Estimation** Our algorithm achieves a 5.5 × speedup over the baselines with a sub-0.5° gaze error. Figure 10a and Figure 10b compare the vertical and horizontal gaze errors and the speed of our algorithms with different baselines. The speedups are normalized to the speed of RITnet, which runs at 5.4 Hz on a mobile Volta GPU. Note that a 1° error is generally acceptable for gaze tracking [51].

RITnet, DenseElNet, Ours(S), and Ours(S) keep the absolute error rate below 0.1° in both the vertical and horizontal direction. They are all more accurate than DeepVOG. In particular, Ours(S) only has 0.04 and 0.05 higher gaze error than RITnet in each direction, but is 1.9 × faster. Note that this little gaze error loss is achieved with a network that is 12.8 × smaller (Table 2). This highlights our efficient eye segmentation network design (Section 3.3).

By operating on ROI images when possible, Ours(L)+ROI improves the speedup over RITnet to 3.0 ×. Interestingly, Ours(L)+ROI achieves better accuracy than Ours(L). Further inspection of the data shows that this is because by using only the
ROI image for eye segmentation, we remove potential noise in the non-ROI region in the input eye image.

Using activity-proportional segmentation, Ours(L)+ROI+E and Ours(S)+ROI+E further improve the speedup to 4.2× and 5.5×, respectively, while both retaining an angular error rate within 0.5°. The event density threshold used for extrapolation is set to 0.1%. We will study the algorithm’s sensitivity to this parameter in Section 5.2.

To further confirm the robustness of our system, Figure 11 compares the frame-by-frame gaze results of Ours(L), Ours(S)+ROI+E, and the ground truth. Ours(L) virtually matches the ground truth, and Ours(L)+ROI+E has slight deviations (e.g., around frame 10). We show three representative gazes in the bottom panel, where the eye moves moves right, blinks, and moves up left.

Finally, for a comprehensive analysis we show the gaze error distributions across all the evaluated frameworks in Figure 12. Not surprisingly, RITNet has the most compact distribution, because it is used to obtain the ground truth (see Section 4.1). Even the most inaccurate variant of our system, Ours(S)+ROI+E, has a worst-case accuracy below 0.5°, which is generally regarded as an acceptable error bound for eye tracking [51].

Eye Segmentation While gaze accuracy is ultimately what we care about, we show the results of eye segmentation, an intermediate step to gaze estimation, to understand how the two metrics relate.

Figure 10 compares the eye segmentation accuracy and speed across different methods. DenseElNet achieves the highest mIoU at 99.0%. Ours(L) achieves a similar mIoU at 98.6%, but is 2.8× faster and 20× smaller. With a smaller network configuration, Ours(S) introduces a 1.2% mIoU loss. Ours(L)+ROI and Ours(S)+ROI have slightly lower mIoU at 96.5% and 96.4%, respectively. With extrapolation enabled, Ours(L)+ROI+E and Ours(S)+ROI+E can still keep the mIoU loss within 1.5%. Overall, our algorithm has slightly higher accuracy loss over RITNet on the segmentation metric than on the gaze error metric. This shows that the gaze estimation, a model fitting problem, can tolerate segmentation inaccuracy.

Frame Distribution On Ours(L)+ROI+E, 44.9% of frames perform extrapolations. 54.5% of frames are processed in the ROI mode, and only 0.6% of frames require full-resolution eye segmentation. The vast majority of the frames that require full-resolution processing are the initial frames in each sequence. This frame distribution explains the speedup of our algorithm.

Time Distribution The time distribution of different stages of our algorithm points out opportunities for future optimizations. When executing Ours(L)+ROI+E on the mobile Volta GPU, the most time-consuming stage is the segmentation network, which takes 66.9% of total time. The second most time-consuming stage is the ROI prediction network, which takes 18.7% of the time. The event map and edge map are calculated on the CPU and take 3.6% and 3.9% of the time, respectively. The rest of the algorithm (data marshalling) takes 6.1% of the time.

We emphasize that speed we report in the paper should be seen as a lower bound of our algorithm and can be improved with more engineering efforts. For instance, the ROI prediction and eye segmentation networks can be pipelined across frames, and redundant memory copies can be removed if we directly manage the GPU memory rather than through PyTorch. Our goal here is to show that even unoptimized code (speed-wise) achieves several-fold speedup over state-of-the-art algorithms with virtually the same gaze accuracy.

5.2 Sensitivity Study

Event Activation Threshold Our algorithm emulates how an event camera generates events. The event activation threshold (σ in Equation 1) dictates how sensitive our algorithm is to eye activities. So far our evaluation has used a σ of 30% as the default value.

Figure 13 shows how σ affects the speed-vs-gaze trade-off, as σ increases from 15% to 90%. We show only the horizontal error; results for vertical angular error are similar and have been omitted. We retrain the ROI prediction network for each σ for a fair comparison. The speed is normalized to that of RITNet. The gaze errors under all σ settings are consistently lower than 0.5°, indicating an overall robust algorithm. 30% is an empirical sweep spot. A lower threshold, e.g., 15%, degrades accuracy, because a small threshold generates a noisy event map. Conversely, a higher threshold, e.g., 90%, has lower accuracy too, because a large threshold generates a sparse event map.

Event Density Threshold Figure 14 shows how the horizontal gaze error and speed vary with the event density threshold (γ), which we use to determine when to extrapolate (Section 3.2.2). γ is increased from 0.001% to 0.1% from left to right. As γ increases, more frames are extrapolated, so both the speed and the error increase.
3. Impact of edge map.

The gaze errors are consistently below 0.5°. Under a 0.001% γ, we achieve 3.9× speedup over RITnet.

5.3 Design Decision Analysis

ROI Prediction Method To understand the effectiveness of our ROI prediction network, we construct a non-DNN method similar to those used in classic image stitching algorithms [14]. This method predicts the current frame’s ROI by matching SIFT features [63] from two consecutive image frames and using RANSAC [31] to calculate a transformation matrix between the two frames. Using the transformation matrix, we then compute the ROI coordinates of the current frame from the previous ROI. This variant is denoted Ours(L)+ROI(SIFT).

The gaze accuracy and the speed of Ours(L)+ROI(SIFT) are shown in Figure 10 in comparison to other methods. The gaze error of Ours(L)+ROI(SIFT) increases significantly, especially on the vertical direction, to 0.39, much higher than Ours(L)+ROI (0.08). This suggests the effectiveness of our ROI prediction DNN.

Extrapolation Method The extrapolation method that we use in Section 4.2.3 simply reuses the previous segmentation map. To demonstrate its effectiveness, we compare with two other methods:

- Rescale: We first crop the previous segmentation map based on the previous ROI, and rescale the cropped segmentation map based on the size of the current, predicted ROI.
- RANSAC+Rescale: We use the ROI prediction network to first predict the ROI. The ROI is refined based on the SIFT features using RANSAC from the previous ROI. Lastly, we perform the same Rescale method as above.

Figure 15 compares the speedup (over RITnet) and segmentation accuracy of different extrapolation methods. All the extrapolation methods are applied to the Ours(L) network. We also vary the extrapolation threshold from 0.001% (left-most markers, fewest extrapolations) to 0.1% (right-most markers, most extrapolations).

We make two observations. First, our extrapolation method is consistently the most accurate. Combining RANSAC with rescaling is more accurate than simple rescaling, but also is much slower due to the overhead of RANSAC. Second, the accuracy generally drops with more extrapolations. But our method is the most robust to extrapolation. With 37.5% more extrapolations the mIoU degrades by only 0.7% (with 1.3× speedup), whereas the accuracy under rescaling drops by 4.6%. The accuracy drop in rescaling-based methods is likely due to ROI error accumulation.

5.4 Ablation Study

The ROI prediction algorithm uses not only the event map, but also two feedback cues from the previous frame, i.e., the edge map and the ROI. Ablation studies here show the need for these two cues.

Previous ROI Figure 16 compares two ROI prediction networks: one only uses event maps and one uses event maps and previous ROIs. We train two ROI prediction networks using the same training procedure. We compare the mIoU and the speed of these two prediction schemes both with (+ROI+E) and without (+ROI) extrapolation. In the extrapolation mode, we sweep the event density threshold γ from 0.001% through 0.1%, same as in Figure 14. Without extrapolation, using the previous ROI improves the segmentation mIoU by about 1%, confirming that ROI feedbacks help predict future ROIs.

With extrapolation, the mIoU from using the previous ROI drops as γ increases. This shows how ROI prediction errors can accumulate. Interestingly, the mIoU without using the previous ROI increases as γ increases and is even higher than the mIoU without extrapolation at all. The reason, however, is that errors do not accumulate when previous ROIs are not used; when events are sparse (the prerequisite of extrapolation), the ROI naturally moves little, in which case using the previous ROI is the correct decision.

Edge Map Figure 16 compares two ROI prediction networks: one uses the edge map and the other does not, both trained with the same procedure. We show the segmentation accuracy and the speed
We use a recent analytical model published by Meta [61] to provide higher or competitive accuracy.

Figure 17 uses a sequence of five consecutive frames to illustrate how the edge map helps correct the ROI drift. The solid and dashed rectangles are the ROIs that are predicted with and without using the edge map, respectively. The first frame is right after the user’s eye opens after a blink, in which case the event map provides a decent guidance to predict the ROI. However, in the next few frames the eye movements are insignificant, resulting in sparse event maps; in fact, the event map at frame 3 is completely empty. Thus, the ROIs predicted using the event map alone stay almost at the same location. The edge maps during the inactive period, however, still provide decent guidance as they are predicted from the segmentation maps.

### 5.5 Evaluation on TEyeD

Overall, the trend of the results on TEyeD matches that seen in OpenEDS 2020. Table 3 shows the accuracy and speedup across different methods. Compared to RITnet, Ours(S) achieves a 3.0× speedup with better accuracy; Ours(L) is the second best in overall accuracy, next to only DenseElNet, which is 4.8× slower and over 40× larger in model size (Table 2).

By incorporating Auto-ROI and extrapolation, Ours(S)+ROI and Ours(S)+ROI+E achieves a 5.0× and 5.7× speedup over RITnet, respectively, while still maintaining a sub-0.7 pixel pupil position error. The results show that our algorithm is robust on the large-scale TEyeD dataset, providing significant speedups over prior methods with higher or competitive accuracy.

### 5.6 Failure Cases and Limitations

We show a failure case using a sequence of four consecutive frames in Figure 18. The solid and dashed boxes represent the predicted and ground truth ROIs, respectively.

with and without extrapolation enabled. Variants that use the edge map are consistently more accurate than ones that do not (2% mIoU difference). Using the edge map has little impact on speed, because the ROI prediction network is very lightweight to begin with.

Figure 18: A sequence showing failure cases. The solid and dashed boxes are the predicted and ground truth ROIs, respectively.

### 5.7 In-Sensor Auto ROI Results

**Methodology** While compute-capable image sensors such as the Sony IMX 500 [7] are on the horizon, most require special partnership and there is no publicly available samples that we are aware of. We use a recent analytical model published by Meta [61] to provide a first-order estimation. The energy model estimates the energy consumption of data transmission per byte to be 800 times more than that of computation (i.e., FLOPs/Byte). This energy model matches other recent studies [39, 54]. The energy consumption at different process nodes is taken from TSMC’s public data [8, 85].

**Sensitivity Study** Figure 19 shows how the data transmission energy varies with different sensor-vs-processor process node combinations. We consider three such combinations: 1) an ideal case where both use a 7 nm node, 2) a realistic case where the processor uses a 7 nm node and the sensor uses a 16 nm node, one generation behind, and 3) a less common case where the processor uses a 7 nm node and the sensor uses a 40 nm node, three generations behind.

We show results for three variants, one with the ROI mode only and the other two with ROI and extrapolation. We use two extrapolation variants, ROI+E(0.001%) and ROI+E(0.1%), that correspond to the least and most energy-conserving ones in Figure 14. The numbers in parentheses are the event density threshold. Ours(S) is used as the segmentation network. We normalize the energy with respect to the baseline that always transmits the full-frame images. The figure is a stacked bar plot, where each bar is decoupled into three components: the pixel data transmission energy, the overhead of transmitting the edge map, and the additional computation energy.

When the sensor uses a 40 nm node and the processor uses the advanced 7 nm node (an uncommon combination), the total energy increases by about 4 times. This is because the energy overhead of computing inside the sensor on the older process node offsets the energy reduction from reducing the data transmission volume. When the camera sensor uses a more advanced technology node of 16 nm, we start seeing energy reduction. For instance, the total energy is reduced by 14.9% under ROI+E(0.1%).

As the sensor’s process node improves over time, the energy saving from in-sensor ROI increases as well. When both the sensor and the processor use a 7 nm node, in-sensor Auto ROI reduces the data transmission energy by 65.6% even without extrapolation. Extrapolations further reduce the data volume and improves the energy savings. ROI+E(0.1%) reduces energy by 80.5%. The result shows that for in-sensor computing to be beneficial, the fabrication technology of the sensor can not lag too much behind that of the processor. Nevertheless, under a realistic 7nm/16nm setting with extrapolation, our system can still reduce the sensor energy by 14.9%.

![Fig. 19: The normalized energy across different variants under three sensor/processor technology node combinations. Ours(L) is used as the segmentation network for all the results here.](image-url)
6 CONCLUSION

We present an algorithm that enables real-time eye tracking. The algorithm operates at real-time (over 30 Hz on a mobile GPU) thanks to our Auto ROI mode, which judiciously processes only the ROI images whenever possible. Our main contribution is a novel ROI prediction algorithm, which emulates an event camera in software and uses events as natural guidance for ROI prediction. While events are largely effective, we show that feedback cues from previous frames are critical to ensure a robust ROI prediction.

Auto ROI With Event Cameras It is interesting to explore how an actual event camera can help improve ROI prediction. A real event camera samples temporal events much faster than a conventional camera and thus could potentially improve the ROI prediction.

To leverage the high-frequency samples from event cameras, the downstream processing algorithm must be fast enough to keep up with the event rate. A quick ballpark estimation shows that our ROI prediction algorithm itself can comfortably run at 10 KHz even on an embedded processor, matching the frame rate of event cameras.

General Auto ROI An interesting research direction is to generalize our ROI prediction algorithm beyond near-eye images and beyond eye tracking. Many tasks in AR/VR such as gesture tracking and SLAM could benefit from ROI-based processing [54]. Modern image sensors do provide an ROI mode (“windowing” in the sensor parlance) [45], which, however, relies on users to provide the ROIs. Automatic, robust, and general ROI prediction is still an open question; we are optimistic that the notion of events, along with other feedback cues, will continue to be important in predicting ROIs.

REFERENCES

[1] Arm ethos-u processor series. https://armwhl.blob.core.windows.net/developer/files/pdf/arm-ethos-u-processor-series-brief-v2.pdf
[2] Nvidia Jetson AGX Xavier. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/
[3] Omnivision ov9782. https://www.ovt.com/sensors/0V9782
[4] Onsemi noisp1sn025ka datasheet. https://www.onsemi.com/pdf/datasheet/noisp1sn025ka-d.pdf
[5] Opensns 2019 challenge. https://research.fb.com/programs/openeds-challenge/
[6] Sony to Release World’s First Intelligent Vision Sensors with AI Processing Functionality. https://www.sony.com/en/sonyinfo/news/press/202905/20-0637e/
[7] Tsmc logic technology. https://www.tsmc.com/english/dedicatedFoundry/technology/logic
[8] C. Akinlar, H. K. Kucukkartal, and C. Topal. Accurate cnn-based pupil segmentation with an ellipse fit error regularization term. Expert Systems with Applications, p. 11600, 2021.
[9] A. Al-Rahayef and M. Fazzipour. Enhanced frame rate for real-time eye tracking using circular hough transform. In 2013 IEEE Long Island Systems, Applications and Technology Conference (LISAT), pp. 1–6. IEEE, 2013.
[10] M. F. Amir and S. Mukhopadhyay. 3d stacked high throughput pixel parallel image sensor with integrated reram based neural accelerator. In 2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), pp. 1–3. IEEE, 2018.
object detection with region proposal networks. Advances in neural information processing systems, 28:91–99, 2015.

[75] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention, pp. 234–241. Springer, 2015.

[76] T. Strandvall. Eye tracking in human-computer interaction and usability research. In IFIP Conference on Human-Computer Interaction, pp. 936–937. Springer, 2009.

[77] L. Świszcz, T. Baltrušaitis, L.-P. Morency, P. Robinson, and A. Bulling. Learning an appearance-based gaze estimator from one million synthesised images. In Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications, pp. 131–138, 2016.

[78] D. Yu, C. jae Lee, M. Park, J. Park, S. Hwang, J. Lee, S. Yu, H. Shin, B. Kim, J.-W. Choi, et al. 14nm FinFET process technology platform for over 100m pixel density and ultra low power 3d stack CMOS image sensor. In 2019 IEEE International Electron Devices Meeting (IEDM), pp. 8–1. IEEE, 2019.

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

[80] Y. Zhu, A. Samajdar, M. Mattina, and P. Whatmough. Euphrates: Algorithm-soc co-design for low-power mobile continuous vision. arXiv preprint arXiv:1803.11232, 2018.

[81] Z. Zhu and Q. Ji. Eye gaze tracking under natural head movements.