Machine Learning & Heterogeneous Computing for Real-time Eye Tracking

Members: Alek Comstock, Jeffery Kasper, Sandro Panchame, Rudolph Nahra

Advisor: Dr. Rover
Client: JR Spidell
Email: **sddec23-02@iastate.edu**

Ref: https://www.mouser.com/images/marketingid/2021/img/134666503.png

Introduction

The Project

Problem

Eye tracking can require very high frame rates (200+ FPS) to correctly capture some eye movement patterns, such as the saccade.

It is difficult to achieve these frame rates in an embedded system.

Solution

Accurate and fast eye tracking can be achieved using:

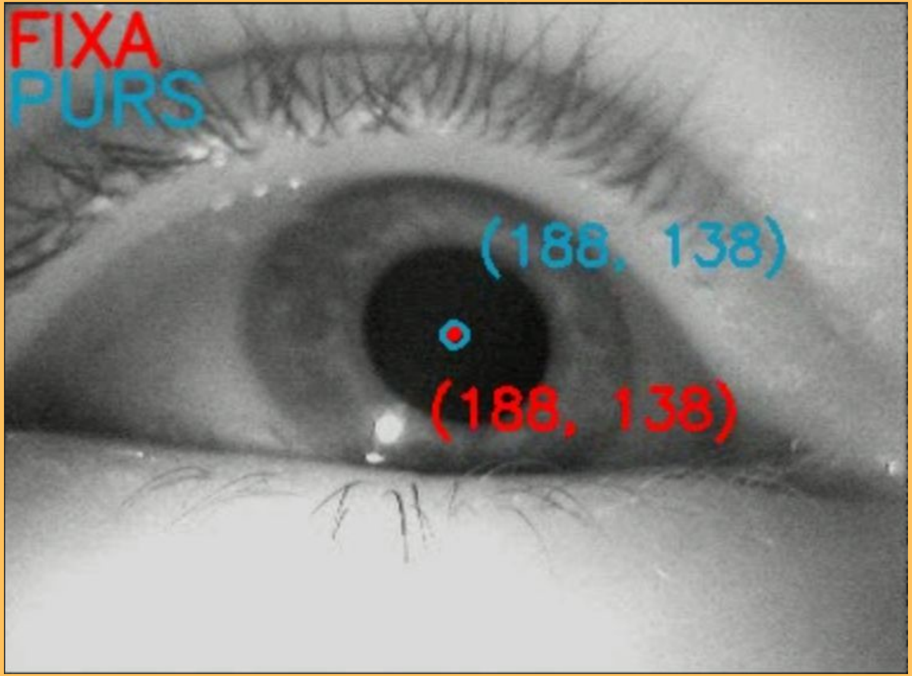
- Heterogeneous hardware specialized for machine learning (ML)
- Implemented on a field programmable gate array system on chip (FPGA SOC)
- Using a custom ML model.

Requirements

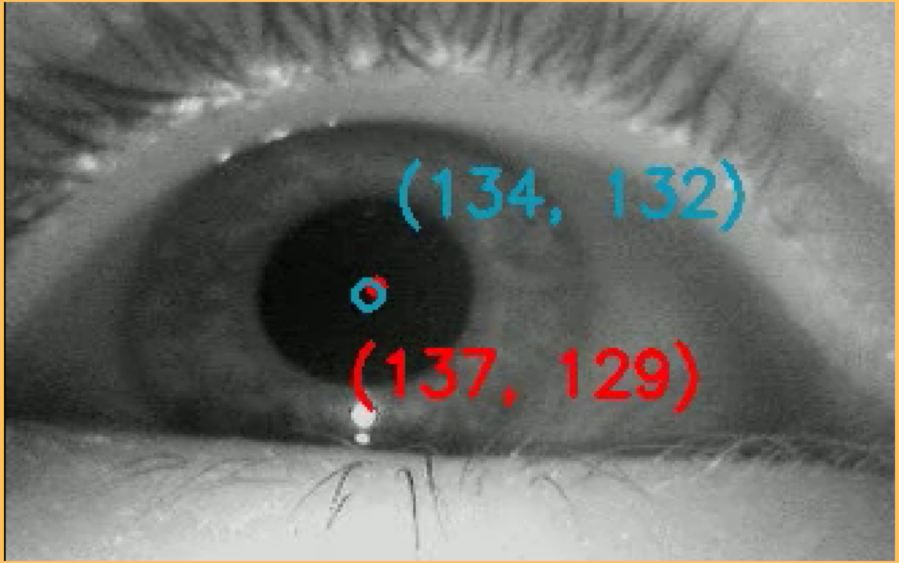
Requirements: Functional

<u>Functional</u>

- Take in many images of eyes
- Output position of pupil and open/close state



Requirements: Nonfunctional



Nonfunctional:

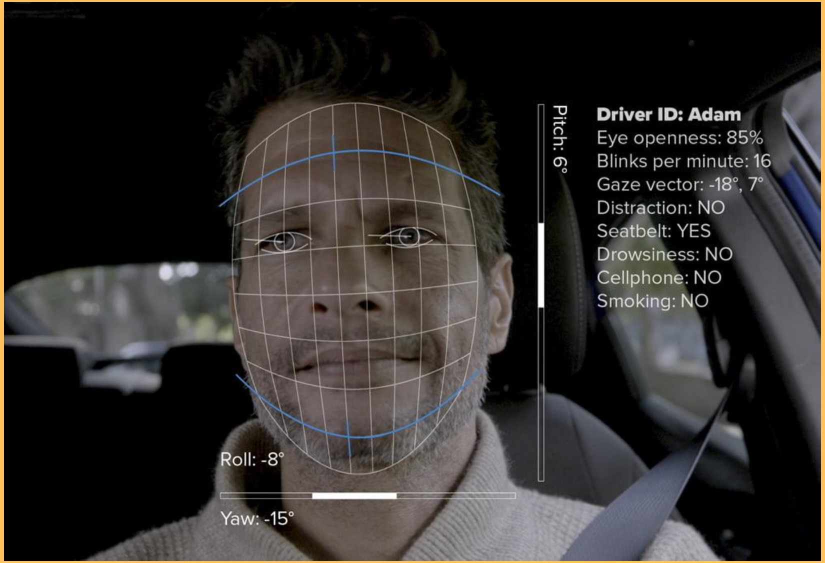
- Process each frame of a video feed with enough throughput to keep up with incoming images
- Root Mean Squared Error (RMSE) of pupil position estimation must be within 3 pixels of the ground truth
- Usage of the Real-time Processing Units (RPU) to enable response to hard time constraints

Constraint:

- Restricted to the Kria KV260 platform

Possible Use Cases

- Knowing the human's eye movement tells you a lot about the state of that human.
- This is helpful in cases where we want to, for example:
    - Monitor vehicle drivers (planes, automobiles, heavy machinery)
    - Diagnose diseases

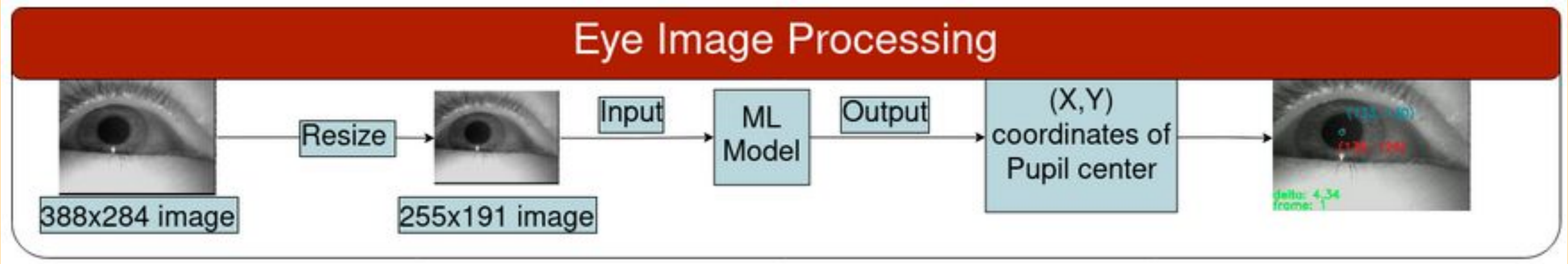


Ref: https://cipia.com/driver-sense/

System Design: ML Model

System Data Processing

- Image preprocessing conducted before neural network
- Neural network outputs an estimate of (X,Y) coordinates of pupil center
- Results are reported



Model Training: Dataset

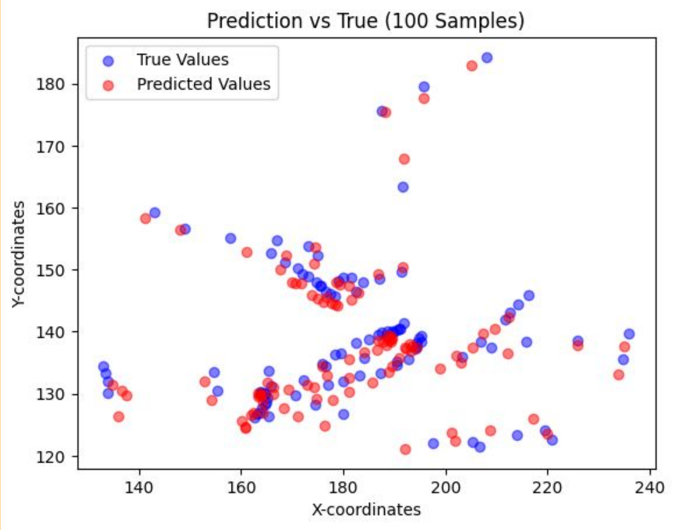
- Dataset sourced from TEyeD
  - Very large collection of real world near eye images
  - Annotations consisted of information such as landmark information, eyeball center, and movement classification
- Roughly 28GB of data was used in training
  - The portion consisted of videos under two and half minutes in length
  - The rest of the dataset was left alone for testing purposes

Model Training: Architecture

- Convolutional Neural Network
- Provided Model Architecture

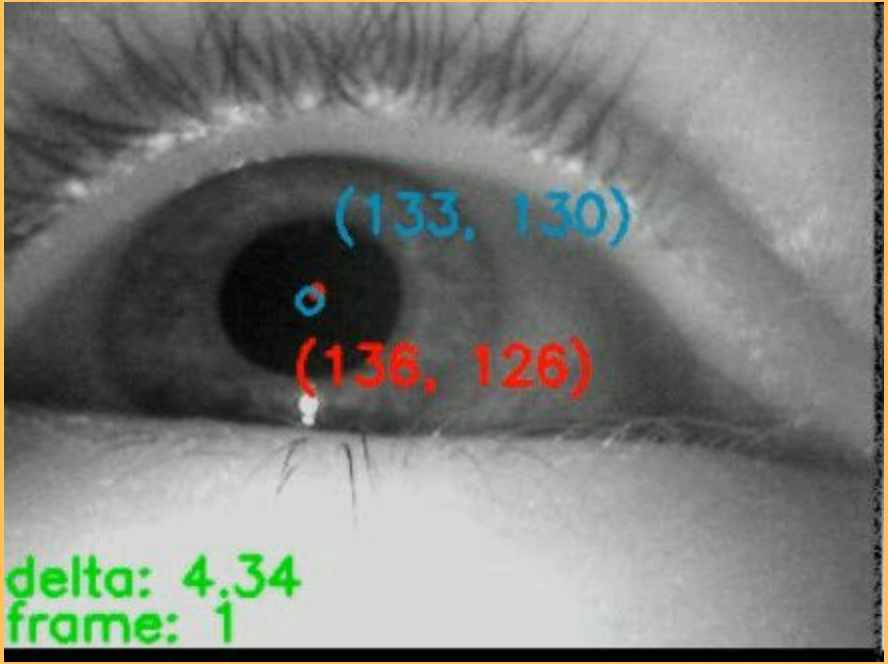
Model Training: Results

- Multiple models were trained until we achieved the following result
  - Roughly 260,000 samples used in training
  - Loss function = RMSE
  - Model resulted in a RMSE of 2.54 pixels



Clip

- Blue = Ground Truth
- Red = Predicted



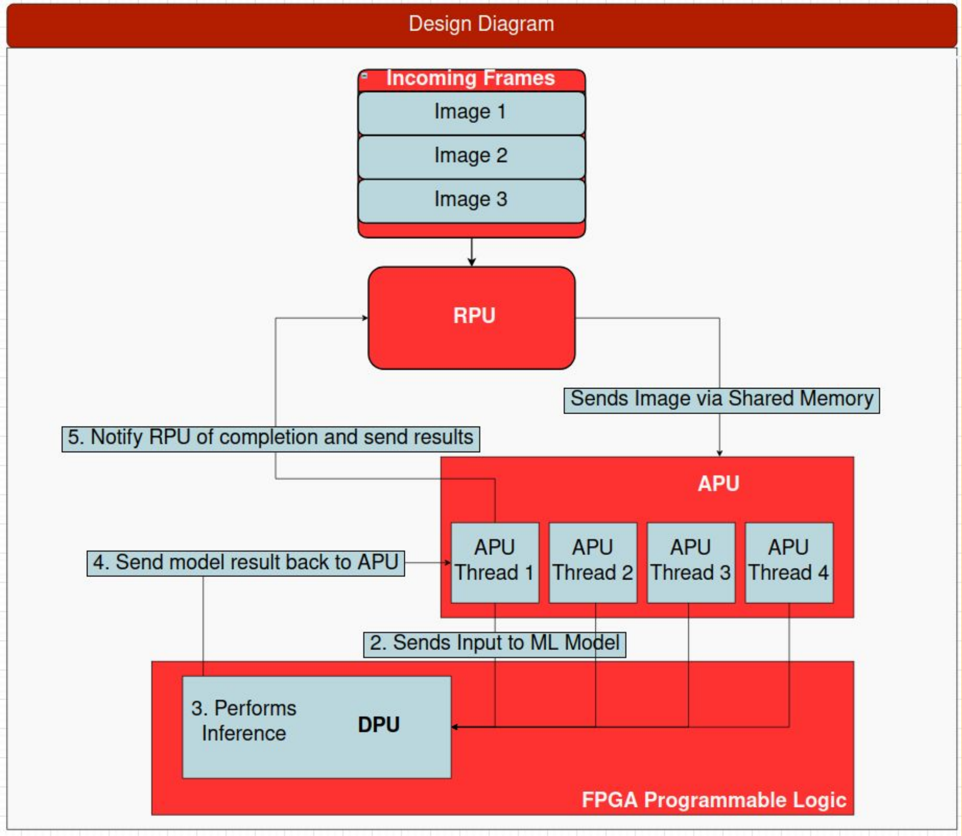
Hardware

- Kria KV260
  - Development board designed by Xilinx.
  - Contains a Kria K26 SOM FPGA.
    - Configured with a Deep-Learning Processing Unit (DPU).
  - 4 ARM Cortex A53 cores running a Petalinux environment.
  - 2 ARM Cortex R5 running in a bare metal environment.
  - 4GB of Memory.

Ref: https://www.mouser.com/images/marketingid/2021/img/134666503.png

Heterogeneous Computing Elements

1. RPU
   - Real time processor
   - Bare metal
2. APU
   - 4 ARM A53 processors
   - Linux-based OS
3. DPU
   - Deep Learning Processor
   - FPGA ML Accelerator

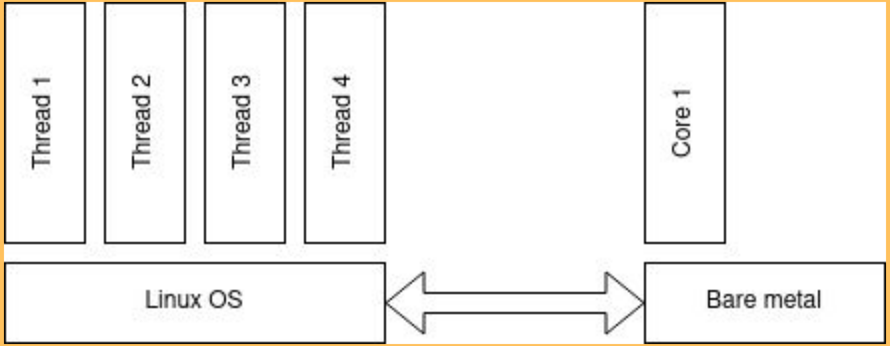


DPU Details

- DPU can perform ML calculations with high parallelization
- Instantiated in FPGA fabric as a systolic array
- Takes instructions like any other processor
- Instructions are compiled from the ML model into a *.xmodel file
- APU feeds instructions to the DPU

Heterogeneous Communication

- OpenAMP
  - Framework facilitating communication between heterogeneous computing elements.
  - Allows sending small messages between APU and RPU via a shared memory region
- Libmetal
  - OpenAMP only allows small messages
  - Need to transfer entire image
  - Libmetal enables management of large shared memory regions



Software Tools by Processing Unit

- RPU code developed with Xilinx Vitis
- APU code developed directly on board
- Hardware design created and synthesized using Xilinx Vivado

	RPU	APU	DPU
Primary Function	Track images in memory	Preprocess image	Perform ML inference
Language	C	C++	XIR (compiled ML model)
Compiler	gcc - GNU ARM Cross Compiler	ARM g++ (directly on board)	Vitis AI Compiler
Middleware	OpenAMP Libmetal	Vitis AI Runtime (VART)	

Challenges Overcome

Accomplishments

- Compile custom operating system (petalinux)
  - Careful creation of memory regions
  - Must compile with correct kernel settings and packages
- Created and synthesized hardware design for FPGA
- Trained the provided machine learning algorithm
- Compiled XIR model of our trained machine learning algorithm
- Implemented an algorithm for passing messages between heterogeneous computing modules
- Accelerated model inference running on board

Testing

- Testing was performed on the Kria board
- Video frames outside of training dataset were put on Kria file system
- APU used DPU installed on FPGA to perform model inference
- Results were compared to ground truth from dataset

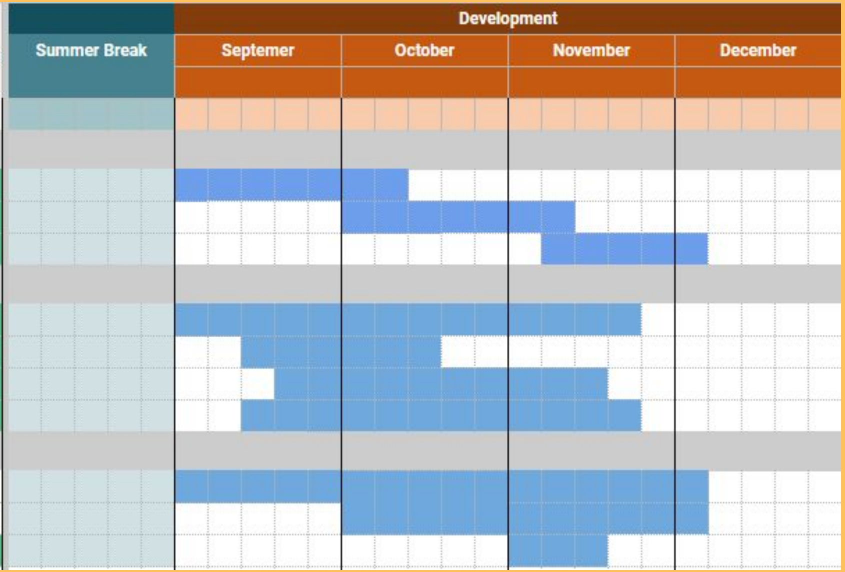
We achieved RMSE of 2.54 pixels and 220 frames per second, meeting our accuracy and throughput objectives

Marabou

- Originally Neural Network Verification was a goal of our project
- NN verification proves that the NN meets specification and does nothing else
- Marabou uses formal methods for verification of NN
- We learned that formal methods is computationally expensive
- Currently, only possible to verify very small networks
- Our network was too big

Gantt Chart/Project Schedule

WBS NUMBER	TASK TITLE	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	**Build Neural Network Model**				
1.1	Preprocess training data	9/11/23	10/13/23	32	100%
1.2	Train Machine Learning Models	10/1/23	11/15/23	44	100%
1.3	Determine Accuracy of the Models.	11/7/23	12/1/23	24	100%
2	**Build Linux OS with drivers**				
2.1	Configure Petalinux Tools project	9/4/23	11/17/23	73	100%
2.2	Identify necessary tools and drivers	9/18/23	10/24/23	36	100%
2.3	Add tools and drivers to Petalinux Tools project	9/20/23	11/15/23	55	100%
2.4	Compile petalinux operating system	9/11/23	11/24/23	73	100%
3	**Develop RPU-APU communication code**				
3.1.1	Develop RPU code	9/4/23	12/4/23	90	80%
3.1.1	Develop APU code	10/2/23	12/4/23	62	80%
3.3	Integrate DPU into design	11/2/23	11/20/23	18	100%



Next Steps

- Receive image feed from camera
- Integrate with larger project
- Reduce error of neural network
- Verify neural network
- Optimize processing algorithms to achieve higher frames per second

Questions?

END

END

END

Appendix

Resources

- TEyeD
  - https://arxiv.org/pdf/2102.02115v3.pdf
- NN Verification
  - https://aisafety.stanford.edu/marabou/fomlas19.html#/sec-architecture-overview
  - https://www.youtube.com/watch?v=KiKS_zaPb64
- Remodnav
  - Dar, A.H., Wagner, A.S. & Hanke, M. REMoDNaV: robust eye-movement classification for dynamic stimulation. Behav Res 53, 399–414 (2021). https://doi.org/10.3758/s13428-020-01428-x