Edge Computing of YOLO for use in Fall Detection

Ben Hempelmann and Cameron Legg

Project Overview

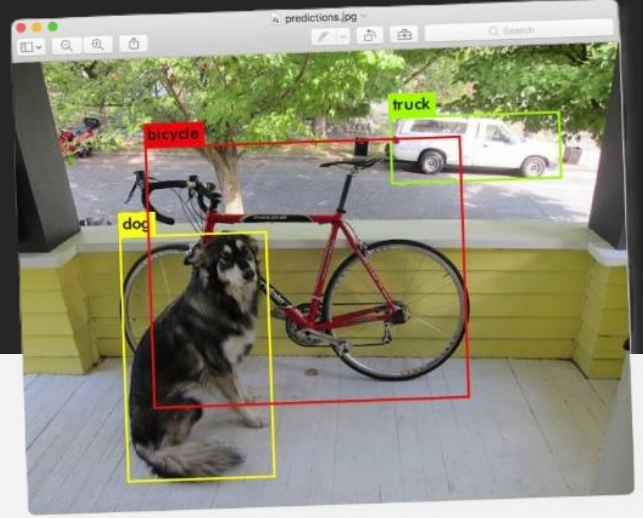
Original Plan

- Detect human falls using a computer vision approach
  - This does not require wearables or other sensors
- Run YOLO Darknet on a Raspberry Pi, using the OAK-D camera and its built-in hardware accelerator and compare with Coral USB TPU

Changed Plans

- Run YOLO Darknet on Nvidia Jetson Orin NX and Oak-D Camera
  - Compute inferences on
    - Oak-D Camera
    - GPU
    - CPU
  - For benchmarking a 5 minute video was passed as a stream to the models
    - The camera feature of the OAK-D was not used, just the processor
    - The benchmarking scripts took the average power consumption and fps every 300 frames.

YOLO



- **Convolutional Neural Network**
- How Fall Detection Works
  - 3 Classes: Sitting, Walking, Fall_Detected
  - Fall_Detected is triggered when someone is laying down
- Parallelization Improves Performance
  - Based on CNNs
  - Applies kernals across different parts of the image
  - Many matrix operations

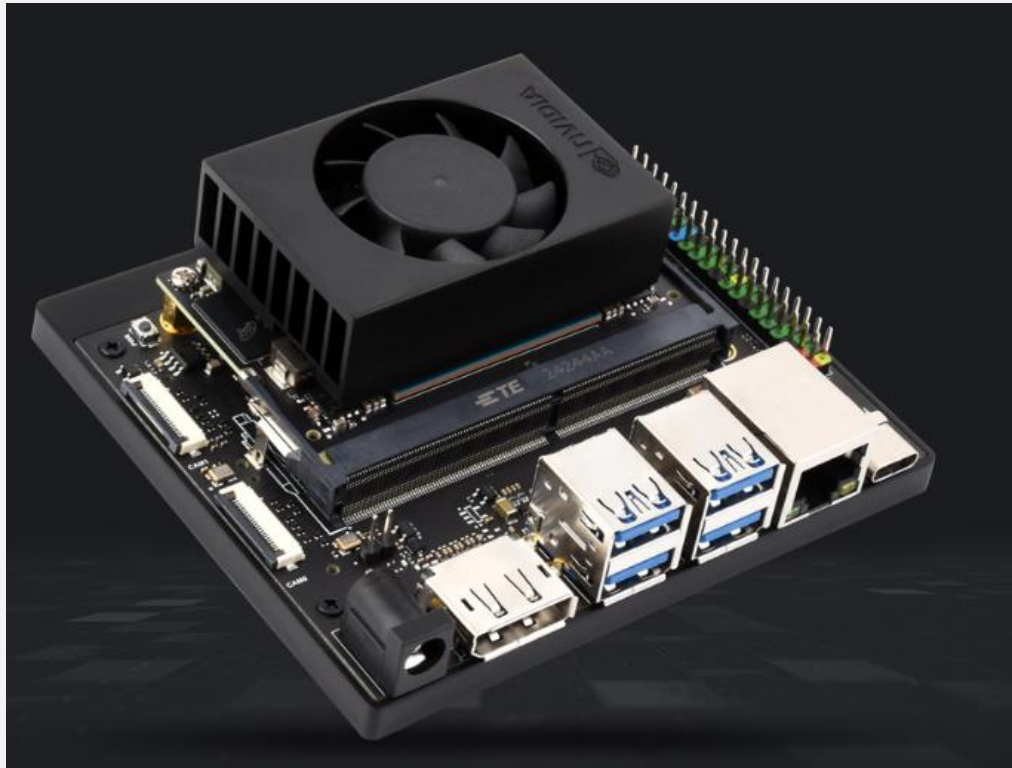
Training YOLO

- Found a dataset of Labeled Walking/Sitting/Fall_Detected images
  - 374 Images for training / 111 for validation
- Trained a YOLOv11 model using Google Colab
- Model trained on 640p images

Acceleration Frameworks

Nvidia Jetson Orin NX

- Ampere GPU – 1024 Nvidia Cuda Cores
- 8 Core Arm Cortex-A78AE CPU

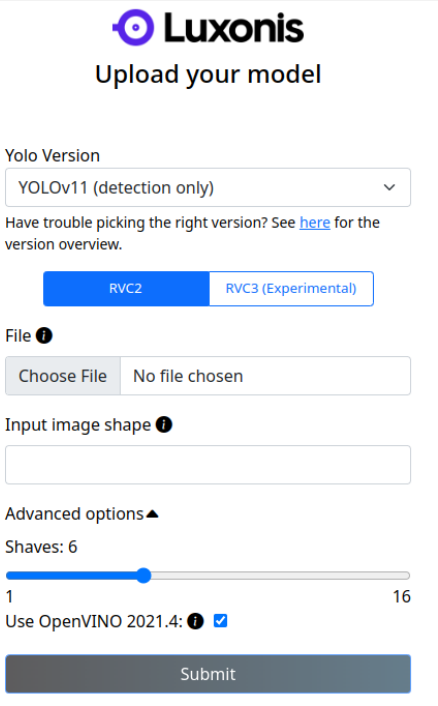


OAK-D Camera

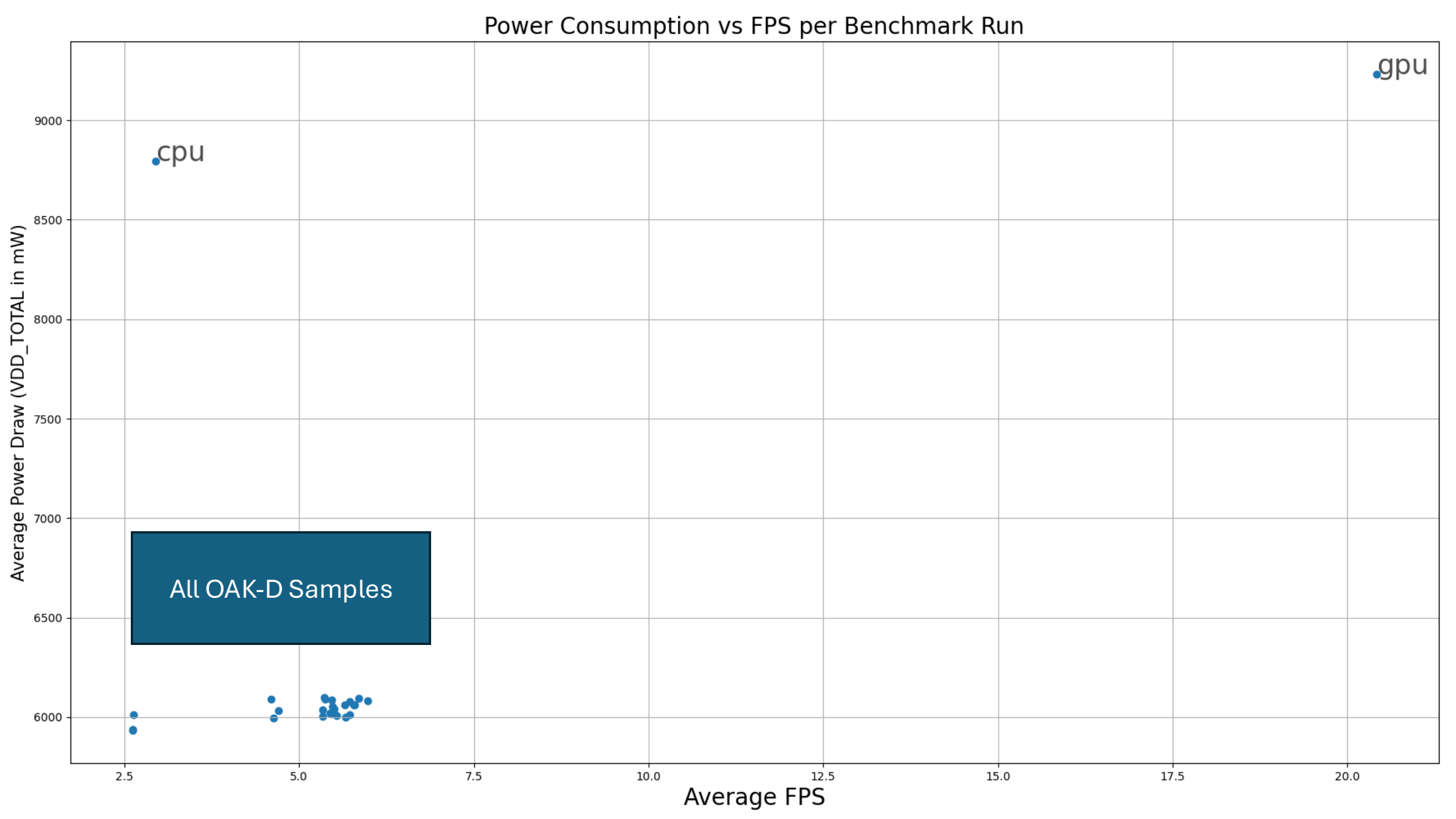
- Based on Robotics Vision Core 2 (RVC2)
  - System on a Chip Architecture
  - Uses Intel Movidius Myriad X Vision Processing Unit
    - Used for on-device deep neural network and computer vision applications
  - 2xLeon CPU cores
  - 16x SHAVE cores
    - Vector Processing Units
  - Max power consumption of 4.5W
  - 2x NCEs (Neural Compute Engines)

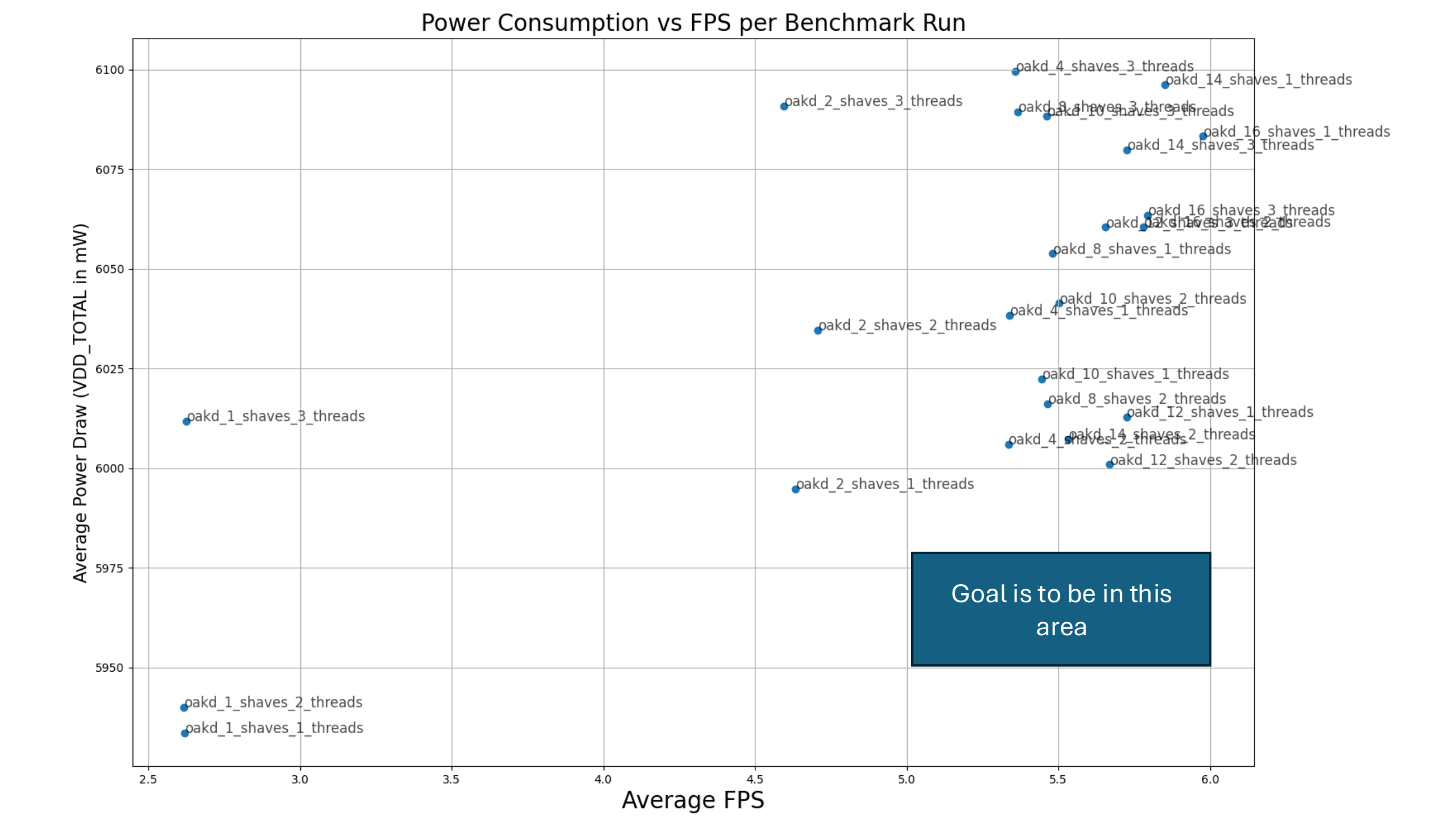
Runtime Framework / Programing Interface

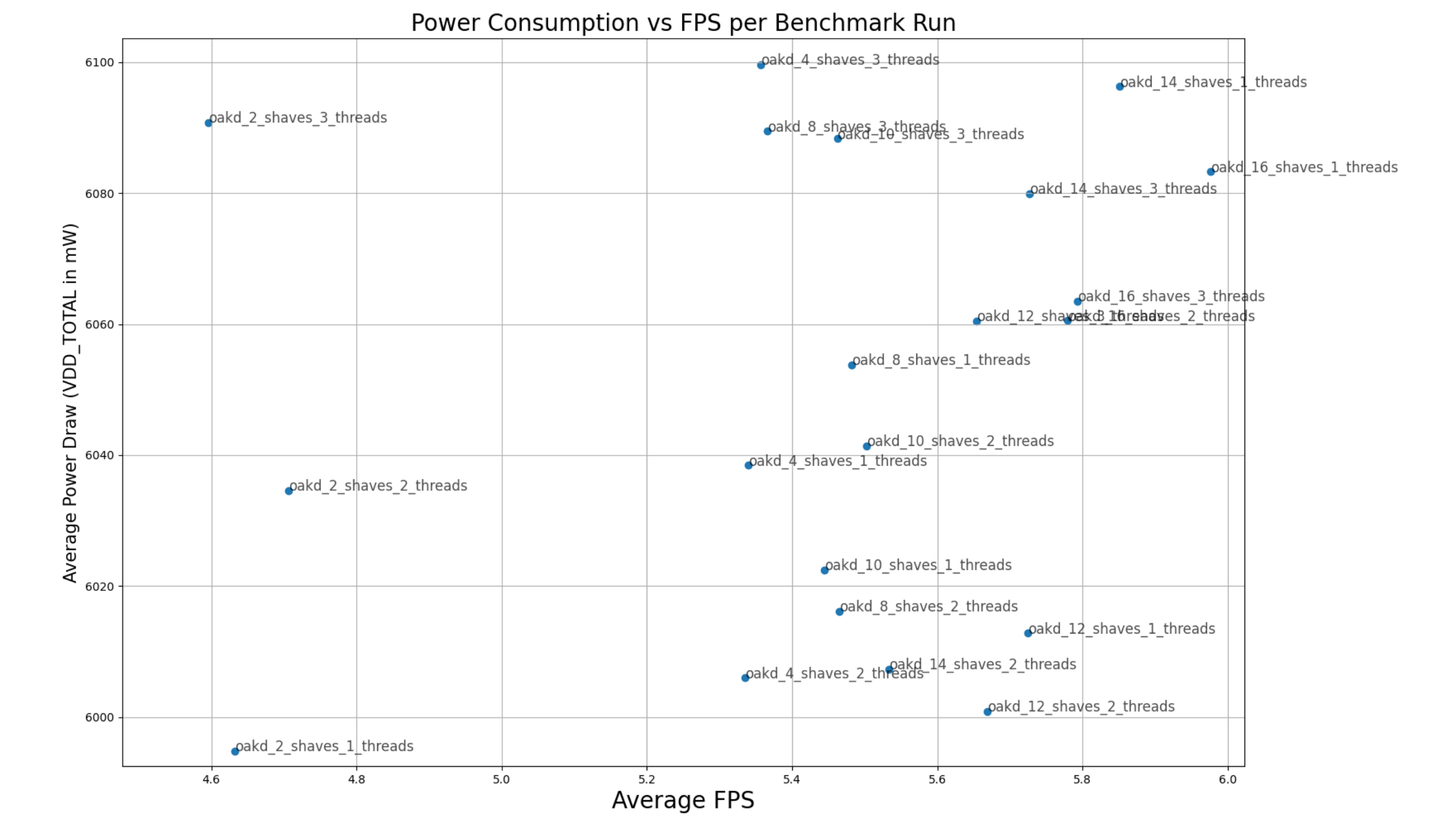
- Used Depth AI Python Library
- Runs the YOLO model after it has been converted to OpenVino format
- OpenVino required for running on Movidius Myriad X
- Oak-Models are compiled with number of shaves in mind
- Difficult part was converting the model
- CPU/GPU
  - PyTorch Framework – CUDA and OpenMP (Via Ultralytics library)

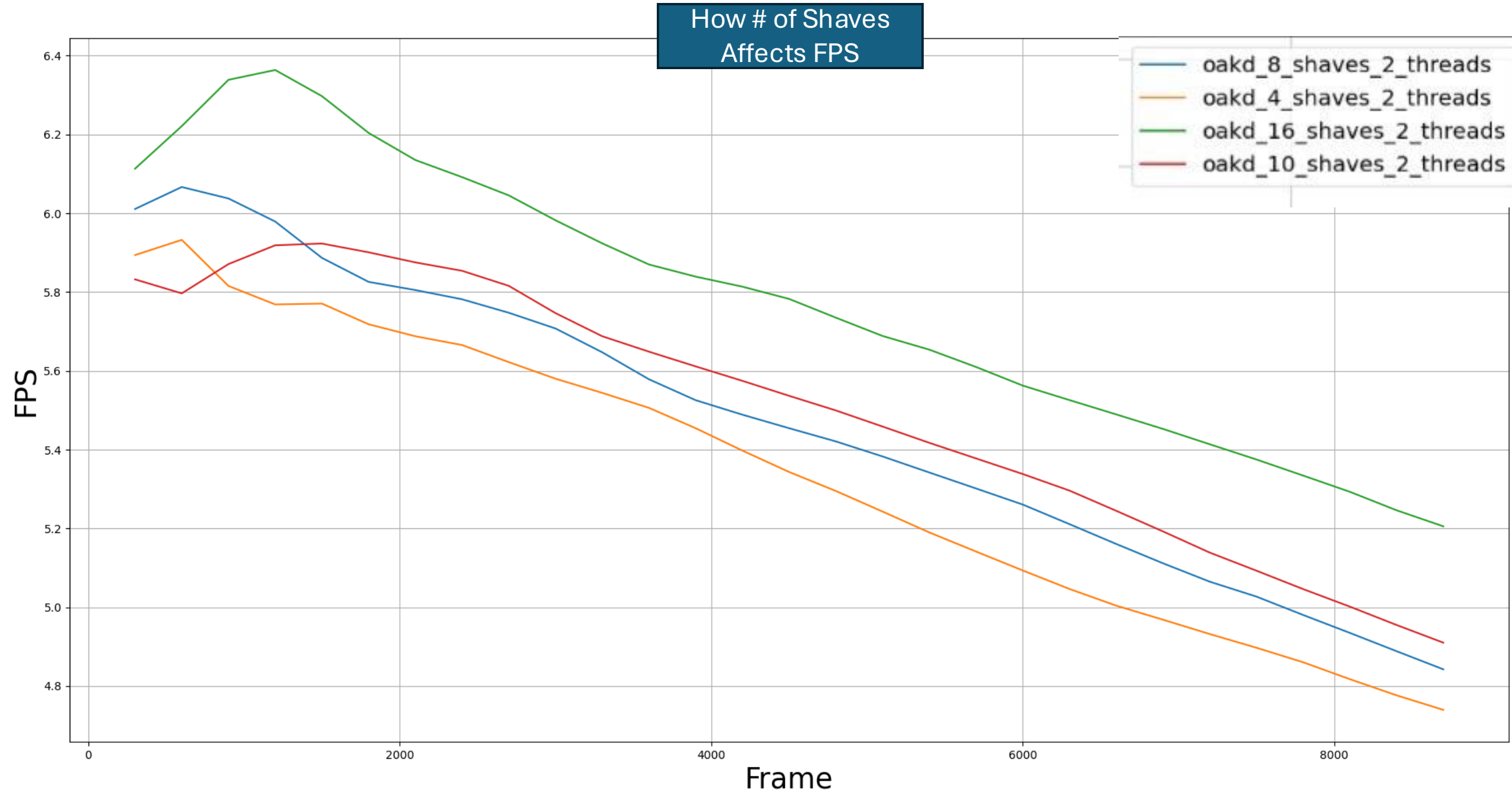


Performance Comparisons

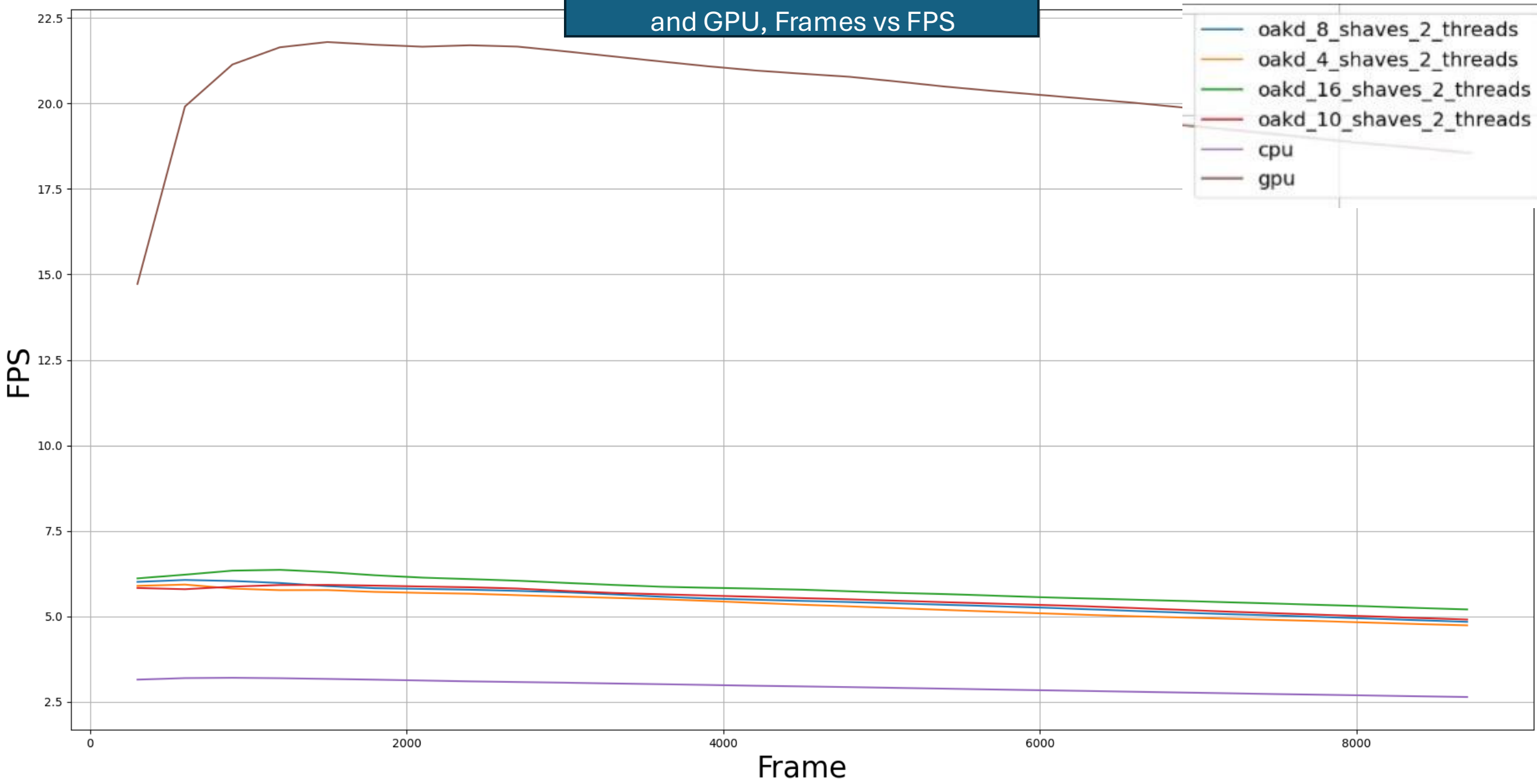

Power Consumption vs FPS per Benchmark Run
gpu
cpu
All OAK-D Samples
Average Power Draw (VDD_TOTAL in mW)
9000
8500
8000
7500
7000
6500
6000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Average FPS

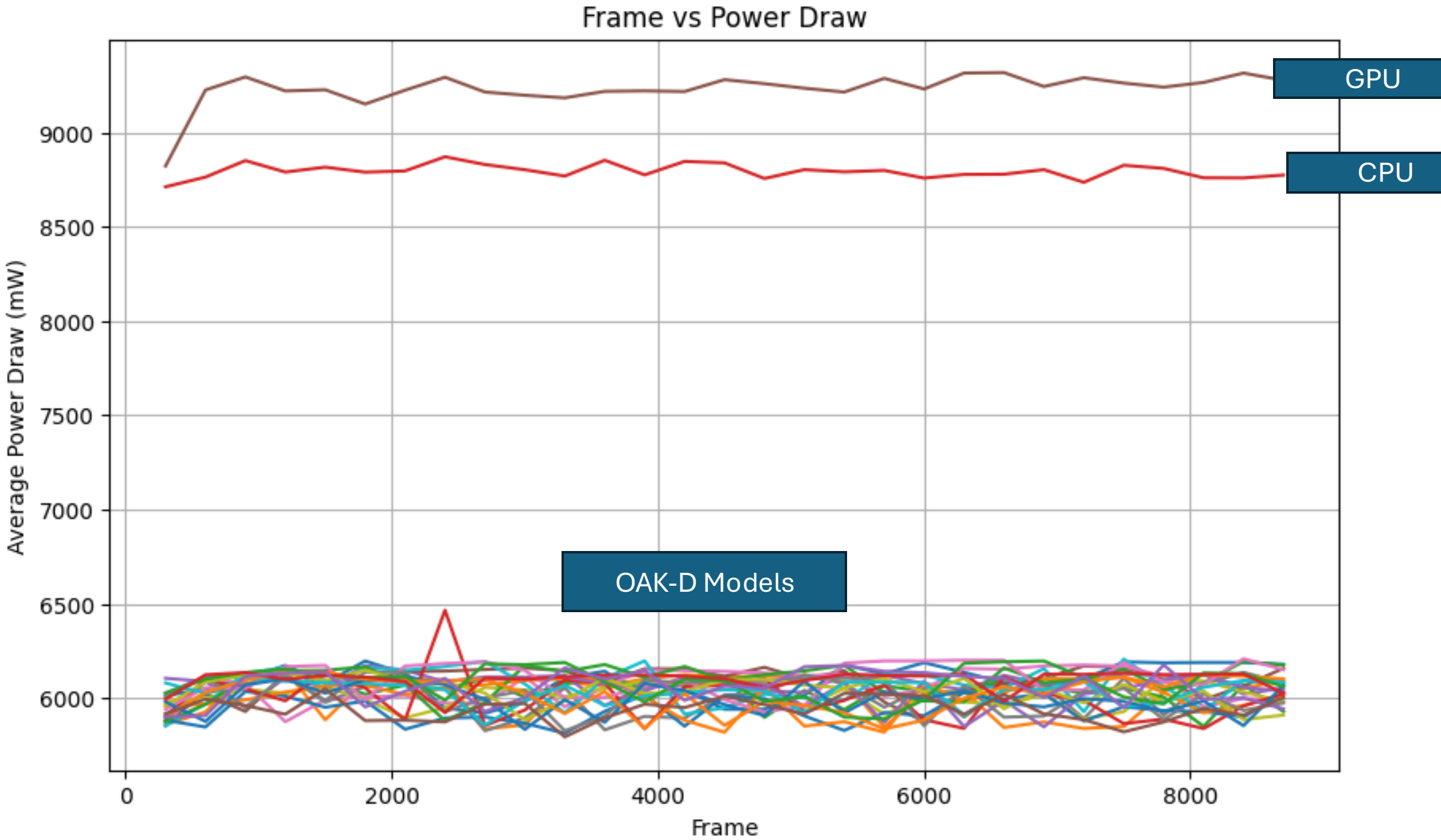

Power Consumption vs FPS per Benchmark Run
Average Power Draw (VDD_TOTAL in mW)
Average FPS
6100
6075
6050
6025
6000
5975
5950
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
oakd_4_shaves_3_threads
oakd_14_shaves_1_threads
oakd_2_shaves_3_threads
oakd_8_shaves_3_threads
oakd_10_shaves_3_threads
oakd_16_shaves_1_threads
oakd_14_shaves_3_threads
oakd_16_shaves_3_threads
oakd_8_shaves_1_threads
oakd_10_shaves_2_threads
oakd_4_shaves_1_threads
oakd_2_shaves_2_threads
oakd_10_shaves_1_threads
oakd_8_shaves_2_threads
oakd_12_shaves_1_threads
oakd_14_shaves_2_threads
oakd_4_shaves_2_threads
oakd_12_shaves_2_threads
oakd_1_shaves_3_threads
oakd_2_shaves_1_threads
Goal is to be in this area
oakd_1_shaves_2_threads
oakd_1_shaves_1_threads

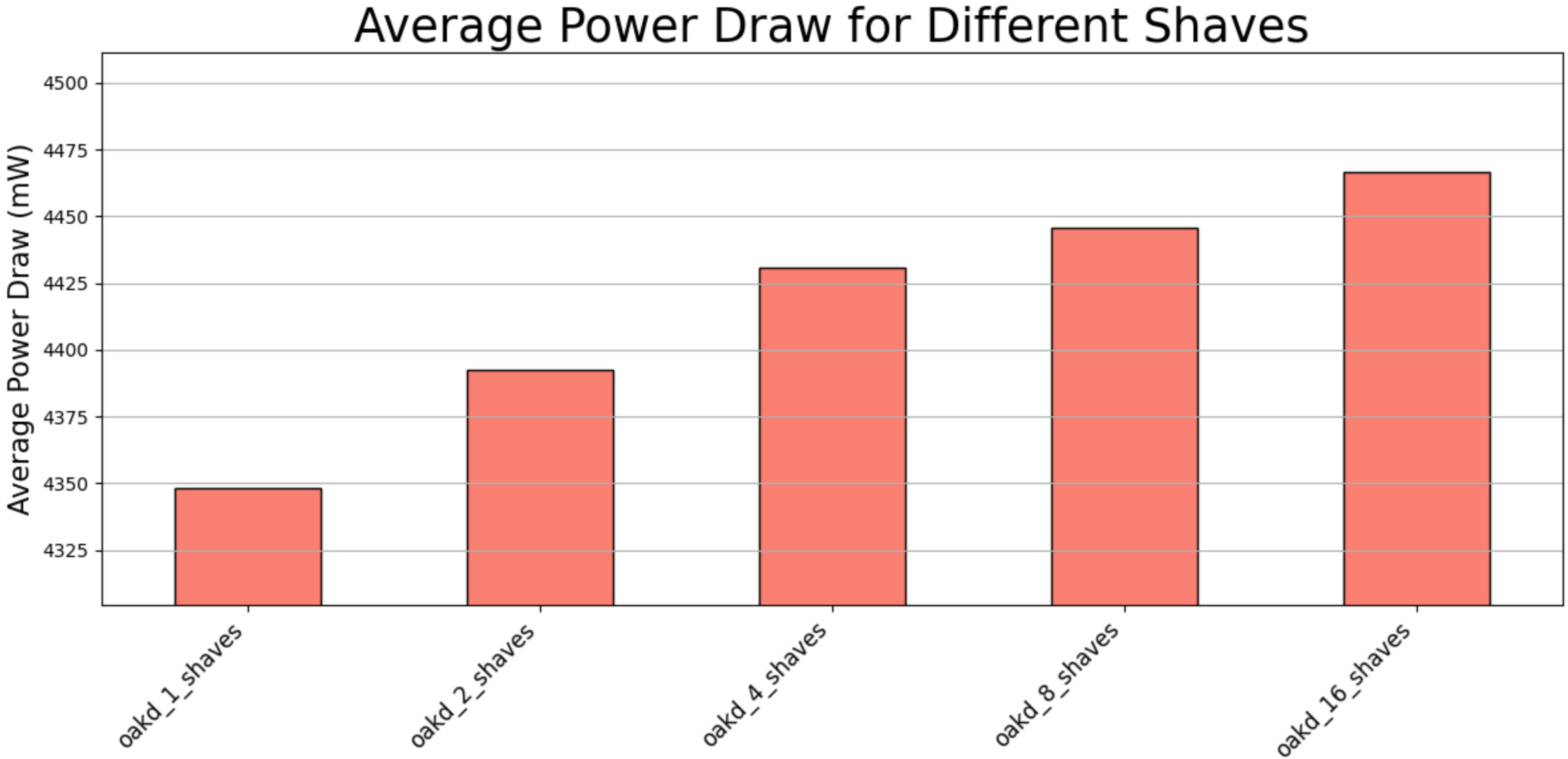

Power Consumption vs FPS per Benchmark Run
Average Power Draw (VDD_TOTAL in mW)
Average FPS
6100
6080
6060
6040
6020
6000
4.6
4.8
5.0
5.2
5.4
5.6
5.8
6.0
oakd_4_shaves_3_threads
oakd_14_shaves_1_threads
oakd_2_shaves_3_threads
oakd_8_shaves_3_threads
oakd_10_shaves_3_threads
oakd_16_shaves_1_threads
oakd_14_shaves_3_threads
oakd_16_shaves_3_threads
oakd_12_shaves_3_threads
oakd_16_shaves_2_threads
oakd_8_shaves_1_threads
oakd_10_shaves_2_threads
oakd_4_shaves_1_threads
oakd_2_shaves_2_threads
oakd_10_shaves_1_threads
oakd_8_shaves_2_threads
oakd_12_shaves_1_threads
oakd_14_shaves_2_threads
oakd_4_shaves_2_threads
oakd_12_shaves_2_threads
oakd_2_shaves_1_threads

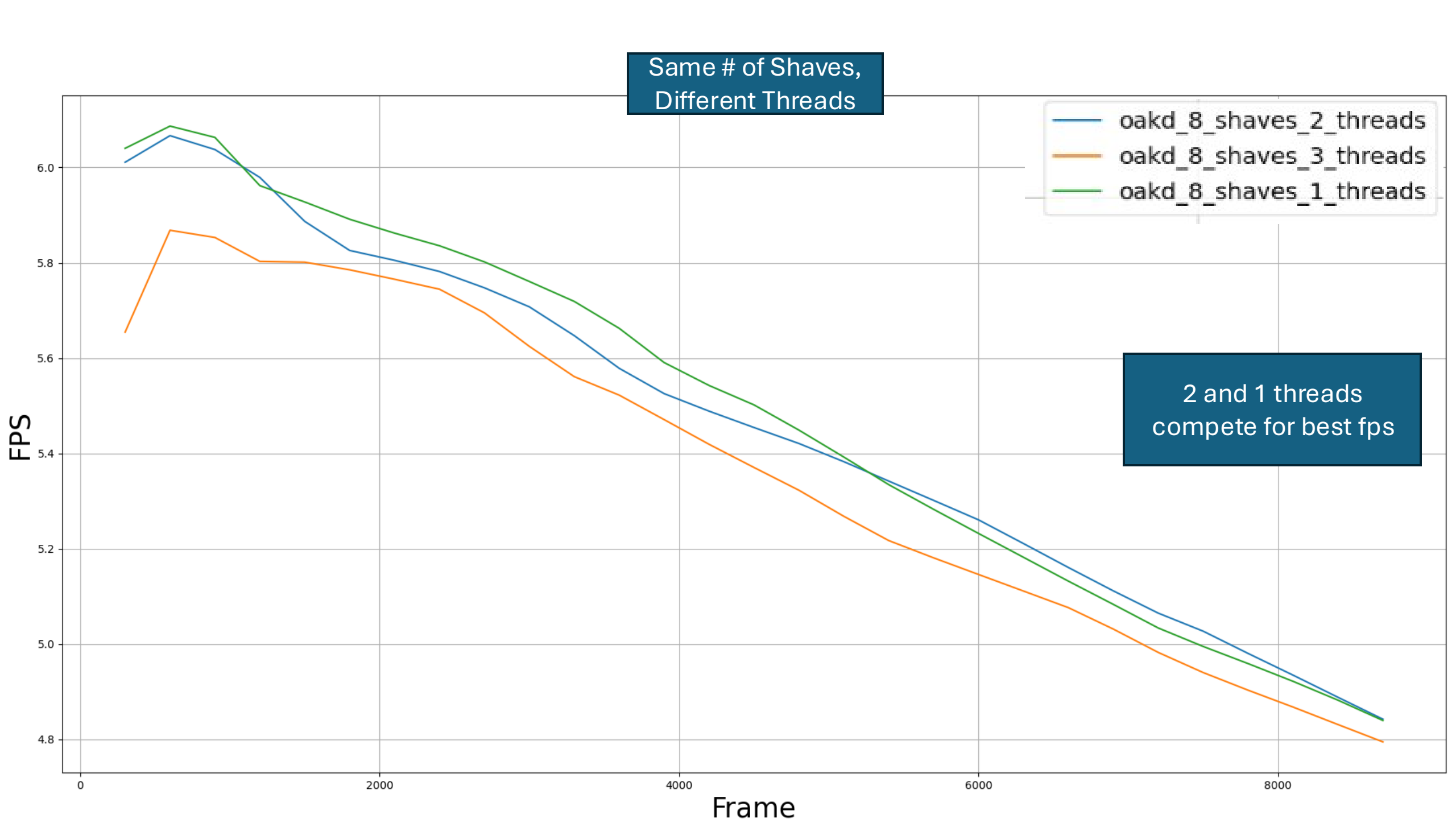

How # of Shaves Affects FPS
oakd_8_shaves_2_threads
oakd_4_shaves_2_threads
oakd_16_shaves_2_threads
oakd_10_shaves_2_threads
FPS
6.4
6.2
6.0
5.8
5.6
5.4
5.2
5.0
4.8
0
2000
4000
6000
8000
Frame

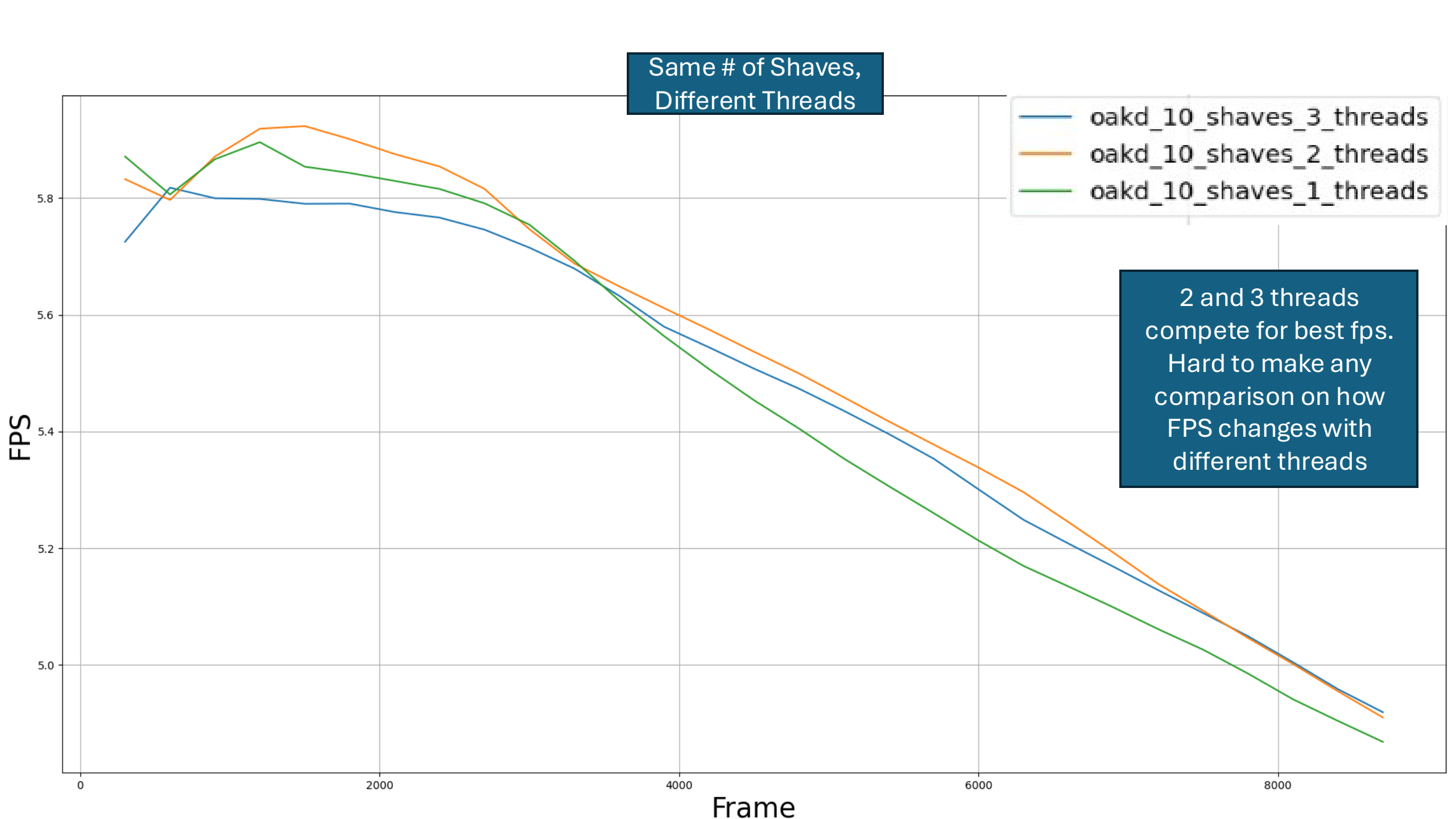
Same Oak-D Benchmarks with CPU and GPU, Frames vs FPS




Frame vs Power Draw
GPU
CPU
OAK-D Models
Average Power Draw (mW)
9000
8500
8000
7500
7000
6500
6000
0
2000
4000
6000
8000
Frame


Average Power Draw for Different Shaves
Average Power Draw (mW)
4500
4475
4450
4425
4400
4375
4350
4325
oakd_1_shaves
oakd_2_shaves
oakd_4_shaves
oakd_8_shaves
oakd_16_shaves


Same # of Shaves, Different Threads
oakd_8_shaves_2_threads
oakd_8_shaves_3_threads
oakd_8_shaves_1_threads
2 and 1 threads compete for best fps
FPS
6.0
5.8
5.6
5.4
5.2
5.0
4.8
0
2000
4000
6000
8000
Frame


Same # of Shaves, Different Threads
oakd_10_shaves_3_threads
oakd_10_shaves_2_threads
oakd_10_shaves_1_threads
2 and 3 threads compete for best fps. Hard to make any comparison on how FPS changes with different threads
FPS
5.8
5.6
5.4
5.2
5.0
0
2000
4000
6000
8000
Frame

Project Learnings

- Increasing number of shaves increased power consumption
- Increasing number of shaves increased inferencing frame rate
- Oak-D outperforms CPU in terms of performance and lower power consumption
- GPU outperforms Oak-D on FPS, but takes much more power
- Oak-D had the best power to FPS Ratio

Hurdles and Future Work

Project Hurdles

- Power Consumption on a Rasberry PI
  - Switched to Orin
- Testing Models with Different Shave Counts
  - You have to recompile the model for each one
- Automating Data Collection
  - Long inference times on videos to get accurate results

Future Work

- Try different file sizes/video qualities
- Inference directly from camera
- Try different models

Questions?