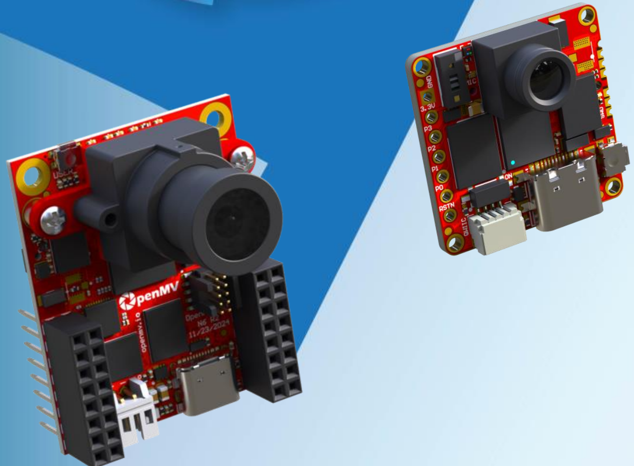
Running Accelerated CNNs on Low-Power Microcontrollers Using Arm Ethos-U55, TensorFlow and Numpy

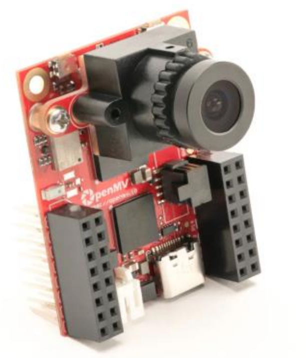


Kwabena W. Agyeman

President

OpenMV, LLC

What is OpenMV?



- Maker of the OpenMV Cam
  - A low-power computer vision platform
    - Directly integrate into products
    - Or licensable for being remixed

Over 100K Sold & Licensed

- What we do:
  - Electrical and PCB design, manufacturing
  - High-performance firmware programming
    - Camera drivers, DMA, cache coherency, etc.
    - SIMD computer vision algorithms, etc.

We make it easy to build a product

Your application

OpenMV provided

MicroPython

Vector (SIMD) accelerated vision algorithms & NPU drivers

Microcontroller support

Camera sensors

Outline

- Market background – what's happening with MCUs?
- Introduce the OpenMV Cam N6 and OpenMV AE3.
- Run ML workloads on microcontrollers using Numpy and TensorFlow.
- Multi-core low-power ML processing using MicroPython.

New AI microcontrollers are here

- **Before:**
  - 600 MHz M7 CPU
    - ~1.2 INT8 GOPS ML performance
    - ~1 MB RAM on chip
      - ~1.2 GB/s bandwidth
    - ~66 MBs FLASH access
  - No MIPI CSI, ISP, NPU

Run 224x224 YOLOv5 Nano at 0.4 FPS @ ~0.8 W

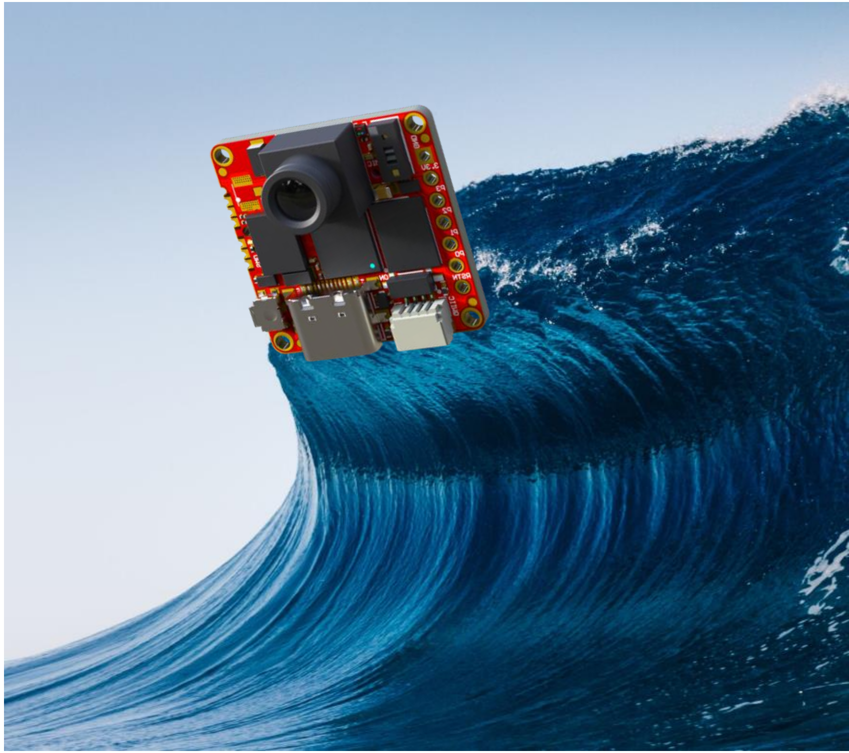
> 200x Better

- **Now:**
  - 400 MHz M55 CPU
    - ~204 INT8 GOPS ML performance
    - ~13 MB RAM on chip
      - ~3.2 GB/s bandwidth
    - ~200 MBs FLASH access
  - MIPI CSI, Helium-ISP, NPU

Runs 224x224 YOLOv5 Nano at 28 FPS @ ~0.25 W

The market wave

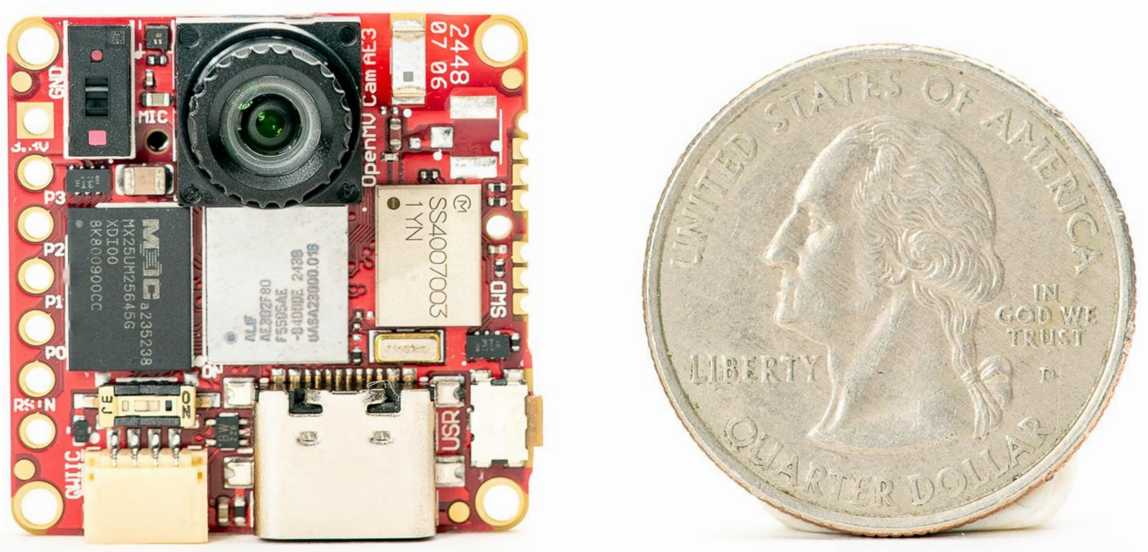
- Running ~2-4 MB YOLO nano models at **30 FPS for < 1 W** is now possible.
  - Or ~8-10 MB YOLO small models at **10 FPS for < 1 W**.
- With **deep sleep power < 1 mW**
  - For years of application battery Life
- <u>*Vision AI for everything, everywhere*</u>



Introducing the OpenMV AE3

- 400 MHz SIMD CPU
  - 204 GOPS NPU
  - 13 MB RAM
  - 32 MB FLASH
- 1 MP color global shutter
  - 30 FPS, 120 FPS @ VGA
  - w/ mic, ToF, accel, gyro
- USB, WiFi, BLE
- GPIO: I2C, SPI, CAN, PWM
- Full power: 60 mA @ 5V (0.25 W)
- Deepsleep: 500 uA @ 5V (2.5 mW)

1” x 1”



OpenMV

And *say hello* to the OpenMV-N6

600 GOPS NPU

MIPI CSI w/ ISP

IMU and user button

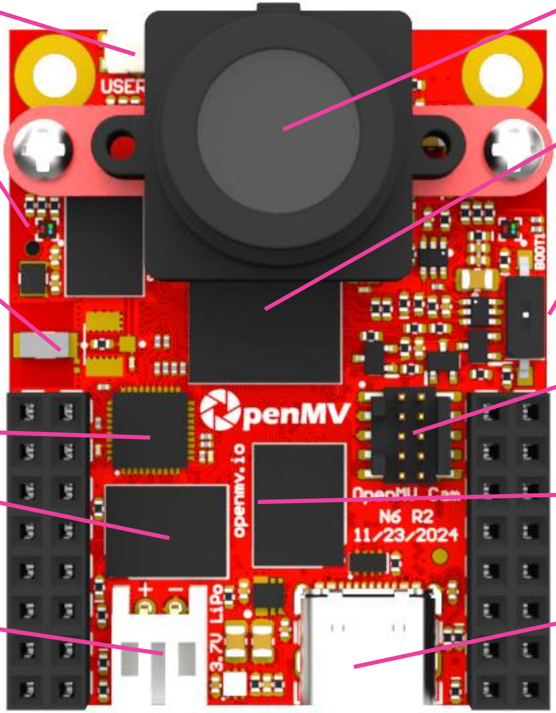
Mic and user RGB LED

2.4 GHz WiFi BLE V5.2

10/100/1000 ethernet

32 MB FLASH @ 400 MB/s

3.7 V LIPO charger



1MP 120 FPS global shutter color camera

STM32N6 MCU

UHS-I µSD card socket (behind camera)

JTAG & SWD

64 MB RAM @ 800 MB/s

USB HS 480 Mb/s

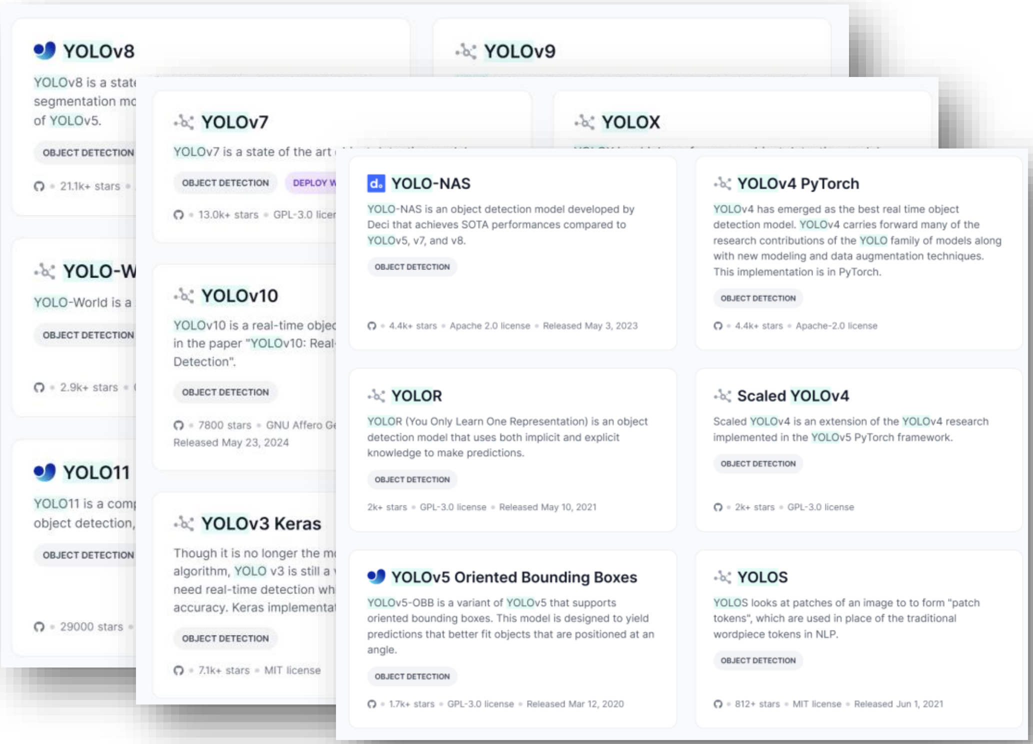
800 MHz SIMD CPU

JPEG & H.264

Full Power: 150 mA @ 5 V (0.75 W)
Deepsleep: 1 mA @ 5 V (5 mW)

NPU Accelerated TensorFlow + NumPy Onboard = Vector Accelerated Python Processing

There are a lot of models



The Problem

- So many vision models!

  How can you quickly support one?

- Quantized models may need tweaking too, custom output modifications and more!

  How to handle this?

NPU accelerated TensorFlow lite for microcontrollers

```
# Reference the YOLO V5 model from ROM to XIP.
model = ml.Model("/rom/yolo_v5_224_nano.tflite")

# YOLO V5 tensor post-processing class.
v5 = yolo_v5_postprocess()

# Take a picture.
img = sensor.snapshot()

# Detect objects in the image using the YOLO V5 model.
boxes = model.predict([img], callback=v5)
```

Accepts a list of Tensors and outputs a list of Tensors for multi-modal inference

OpenMV ML Framework

1. Load a model reference to execute in place from FLASH by the NPU.
2. Create a post-processing object which will receive the tensor output from the model.
3. Run inference using the NPU on image objects and post-process them in Python with Numpy.

Post-process with Numpy on Micropython (1/2)

```
class yolo_v5_postprocess:
    # 0 == x
    # 1 == y
    # 2 == w
    # 3 == h
    # 4 == score
    # 5 == class

    def __init__(self, threshold=0.4):
        self.threshold = threshold

    def __call__(self, model, inputs, outputs):
        oh, ow = model.output_shape[0] # (3087, 6) ~= 72KB of float32s

        # Threshold all rows at the same time
        score_indices = np.nonzero(outputs[:, 4] > self.threshold)[0]

        if not len(score_indices):
            return []

        # Get the bounding boxes that have a valid score
        bb = np.take(colum_outputs, score_indices, axis=0)
```

ARM Helium Accelerated Numpy

1. All YOLO V5 bounding box score outputs are thresholded at the same time using **ARM Helium** accelerated Numpy code!

2. Non-zero indices are then extracted to produce a new array of just the passing bounding boxes.

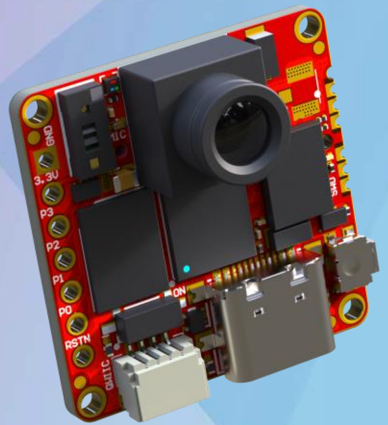
ARM Helium vector acceleration applied to Numpy can be reused by all ML code.

Post-process with Numpy on Micropython (2/2)

```
# Get the score information
scores = bb[:, _YOLO_V5_SCORE]

# Get the class information
classes = np.argmax(bb[:, _YOLO_V5_CLASSES:], axis=1)

# Compute the bounding box information
x_center = bb[:, _YOLO_V5_CX]
y_center = bb[:, _YOLO_V5_CY]
w_rel = bb[:, _YOLO_V5_CW] * 0.5
h_rel = bb[:, _YOLO_V5_CH] * 0.5

# Compute the bounding box coordinates
ib, ih, iw, ic = model.input_shape[0]
xmin = (x_center - w_rel) * iw
ymin = (y_center - h_rel) * ih
xmax = (x_center + w_rel) * iw
ymax = (y_center + h_rel) * ih

# Run NMS to filter out overlapping boxes
boxes = NMS.run(scores, classes, xmin, ymin, xmax, ymax)
```

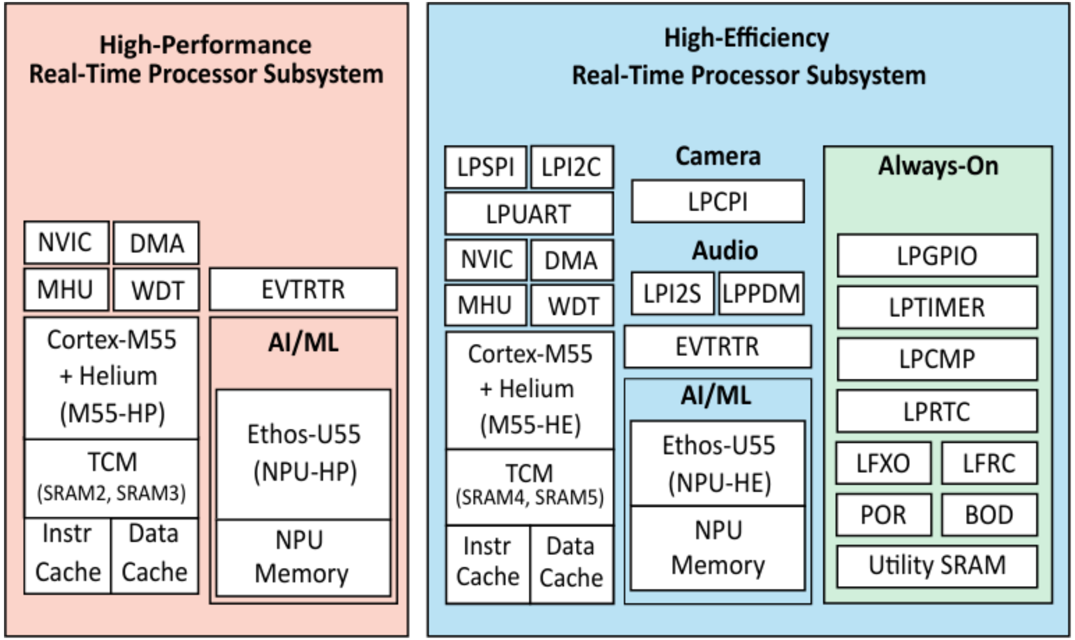
Finishing Up

- Numpy makes it easy to find the maximum class score index of every bounding box row in one line of code!
- Operations to extract the xmin, ymin, xmax, ymax of all bounding boxes are vectorized across all bounding box rows! As fast as C!
- Non-Max-Suppression to filter overlapping bounding boxes, is implemented in Python using Numpy too!



Multi-core processing in MicroPython using OpenAMP on the OpenMV AE3

Easy to use multi-core programming using OpenAMP



The dream

1. High-efficiency core runs AI model on Mic/IMU samples
2. Wake up high-performance core on detection to process images
3. Transmit any detections to the cloud and go back to sleep

One Python script, two processors, two MicroPython VMs

```python
import openamp

# Start the Low Power Core.
openamp.RemoteProc().start()

# This function runs on the Main Core.
def task_callback(data):
    ...

# This function runs on the Low Power Core.
@openamp.async_remote(task_callback)
async def task1(ept):
    ...

# This runs on the Main Core.
while True:
    ...
```

What we've done

1. Python function decorator used to specify **asyncio** co-routines to run on the low-power core.

2. The callback running on the main core will receive messages from the **asyncio** co-routine.

    - Low-power core runs multiple **asyncio** co-routines connected to multiple callbacks.

3. Main core starts the low-power core and enters its own main loop.

A processor and NPU for audio detection

```python
# This function runs on the Low Power Core.
@openamp.async_remote(task_callback)
async def task1(ept):

    import asyncio
    from ml.apps import MicroSpeech

    # Google MicroSpeech ML Keyword Spotting Model
    speech = MicroSpeech()

    while True:
        # Listen for a keyword like ("Ok Google")
        label, scores = speech.listen()

        # Send the detected keyword to the Main Core.
        if label:
            ept.send(label)
```

46 GOPS available for a Wake Word Detector

1. Low power core has its own MicroPython VM, stack, heap, 46 GOPS NPU, and Mic.
2. Low power core runs Google MicroSpeech model to detect a keyword like **“OK Google”**.
3. Low power core sends any detected label strings to the main core via the OpenAMP end-point **“ept”**.

Which triggers NPU image processing

```python
# Reference the YOLO V5 model from ROM to XIP.
model = ml.Model("/rom/yolo_v5_224_nano.tflite")

# YOLO V5 tensor post-processing class.
v5 = yolo_v5_postprocess()

# This function runs on the Main Core.
def task_callback(data):
    if data.decode() == "Ok Google":

        # Take a picture.
        img = sensor.snapshot()

        # Detect objects in the image using the YOLO V5 model.
        boxes = model.predict([img], callback=v5)

        # Do something with the detected objects.
        ...

# Go back to sleep.
while True:
    machine.lightsleep()
```

204 GOPS available for an Object Detector

1. Main core loads YOLO V5 224 nano model reference from ROM to execute-in-place.
2. Main core wakes up when low-power core sends wake word.
3. If **“Ok Google”** the main core takes a picture, runs YOLOv5 on it to detect objects, and transmits the results.
4. The main core then goes back to sleep.

OpenMV

What will you create?

- **The OpenMV AE3**
  - 1x 400 MHz Cortex-M55 w/ 204 GOPS NPU
  - 1x 160 MHz Cortex-M55 w/ 46 GOPS NPU
  - Five sensors:
    - 1MP color global shutter camera
    - 8x8 400 cm ToF distance sensor
    - Accelerometer/gyroscope
    - Microphone
  - Accelerometer/gyroscope/microphone are accessible by the low-power core during **lightsleep()** by the main core.

Resources

Visit us at Booth #909

OpenMV Website

https://openmv.io

OpenMV N6 Product Page

https://openmv.io/collections/cameras/products/openmv-n6

OpenMV AE3 Product Page

https://openmv.io/collections/cameras/products/openmv-ae3