h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Team Project – SS2024

Edge-AI with Google Edge TPU: Sensor Fusion (Camera+Sensor)

Guide

Prof. Schumann

Team

Rakshitha Kukke Prakash (Matriculation No. 1127988)

Sai Swaroop Maram (Matriculation No .1127975)

Mahima Desai (Matriculation No. 1127934)

Sree Samanvitha Manoor Vadhoolas (Matriculation No. 1127974)

Sunil Dabbiru (Matriculation No.1128005)

Table of Contents

1 Introduction

This document provides an in-depth overview of the Team Project titled "Edge-AI with Google Edge TPU: Sensor Fusion (Camera + Sensor),"as part of the Master of Science in Electrical Engineering and Information Technology program of Hochschule Darmstadt. The project integrates Sensor Fusion technology using a Google Coral camera, a carbon monoxide sensor, and the Google Coral Dev Board Mini, enabling rapid prototyping of on-device Machine Learning products. The system is designed to detect human presence using the COCO dataset and carbon monoxide gas (Sensor Fusion), and to trigger an alert on an HMI (currently represented by an LED).

2 Project Scope and Component details

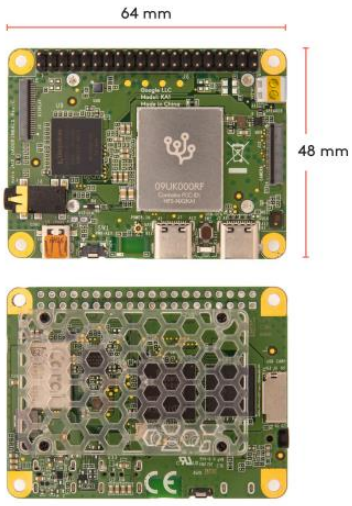
The project's aim is to demonstrate the application of sensor fusion using the Google Coral Dev Board Mini. The focus is on quickly prototyping on-device ML for object detection, triggered by light detection. Initially, the Environmental Sensor Board from Coral was intended for sensor fusion, but extensive testing and manufacturer guidelines, along with online resources, indicated it was incompatible with the Dev Board Mini from Coral.

2.1 Dev Board Mini

The Coral Dev Board Mini is a compact single-board computer engineered for high-speed machine learning inference. It serves as an evaluation platform for the Accelerator Module, featuring the Edge TPU, and also functions as a comprehensive embedded system suitable for diverse on-device ML projects.

2.1.1 Main Characteristics:

- **SoC:** MediaTek 8167s, Quad-core ARM Cortex-A35, Imagination PowerVR GE8300 GPU
- **ML Accelerator:** Google Edge TPU with 4 TOPS peak performance (int8), 2 TOPS per watt
- **Security:** ARM TrustZone
- **Connectivity:** Wi-Fi 5, Bluetooth 5.0
- **Memory and Storage:** 8 GB eMMC, 2 GB LPDDR3
- **Interfaces:**
  - USB 2.0 Type-C OTG
  - HDMI 1.4a (micro)
  - MIPI DSI display
  - MIPI CSI camera
  - 3.5 mm headphone jack
  - Digital PDM microphone
  - 2.54mm 2-pin mono speaker terminal
  - 40-pin GPIO expansion header
  - Mendel Linux OS



Feature	Details
Main system-on-chip (SoC)	MediaTek MT8167s, ARM Cortex-A35, Quad-core, 1.5 GHz
Graphics Processing Unit (GPU)	Imagination PowerVR GE8300, OpenCL 1.2, Vulkan 1.0, OpenGL ES 3.2
Video Processing Unit (VPU)	1080p/60fps HEVC decoder, 720p/30fps H.264 decoder
Memory	2 GB LPDDR3, 1600 MHz DDR clock
Flash memory	8 GB NAND eMMC flash
Network	Wi-Fi 5, Bluetooth 5.0
Interfaces	USB 2.0 Type-C OTG, HDMI 1.4a, MIPI DSI, MIPI CSI, GPIO header

Table 1: Dev Board Mini Components and Features

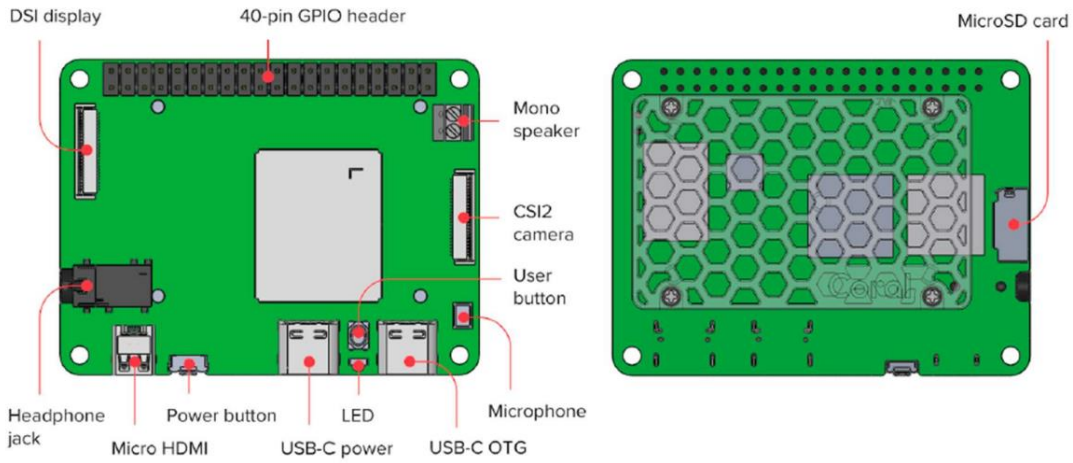


Figure 1: Board Peripherals

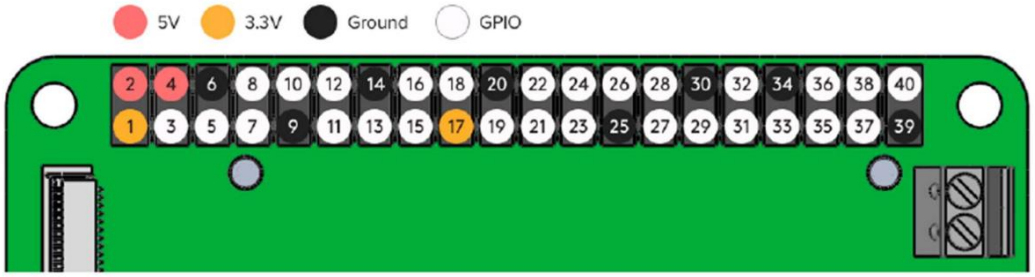


Figure 2: Pin Layout (40 Pins)

2.1.2 Buttons

The board has two tactile switch buttons:

- **Power button:** This controls board power. If only the USB-C power port is connected, you must press this button to boot the board (press firmly). You can power-off the board by pressing it again. If the USB OTG port is connected to power, then the board boots automatically and this button instead reboots the board.
- **User button:** This does nothing by default. You can program your own behavior with GPIO2 ("gpio389"; active low).

2.1.3 USB-C ports

The board has two USB ports:

- **USB-C power:** This is for power input only.
- **USB-C OTG:** This supports USB 2.0 and is a dual-role port, allowing the board to operate as a host (such as when you connect a keyboard) or as a peripheral (such as when you connect to a computer to flash the board). This can also power the board for "always on" operation—the board will not fully shut down (see section 5.1 Power supply).

2.1.4 Micro HDMI port

This port supports HDMI 1.4a with a micro connector, and a resolution up to 1920x1080.

2.1.5 MicroSD slot

The microSD card supports SD/SDHC/MMC (card capacities up to 32 GiB) and meets the SDIO 2.0/3.0 standard to provide expanded memory for the system. Insert the card with the contacts facing up (toward the board).

Parameter	Min	Typical	Max
Board supply voltage (VIN)	4.5 V	5 V	5.5 V
General purpose I/O voltage (VCC3IO)	2.97 V	3.3 V	3.63 V
Operating temperature	0 °C	-	50 °C
Storage temperature	-40 °C	-	85 °C

Table 2: Electrical Specifications

In order to connect to the actual pins of the board, we need the following instantiations in the python script:

Note:” dev” stands for device.

```
gpio22 = GPIO("/dev/gpiochip0", 22, "in")  # pin 7
gpio9 = GPIO("/dev/gpiochip0", 9, "in")    # pin 11
gpio36 = GPIO("/dev/gpiochip0", 36, "in")  # pin 12
gpio10 = GPIO("/dev/gpiochip0", 10, "in")  # pin 13
gpio0 = GPIO("/dev/gpiochip0", 0, "in")    # pin 16
gpio1 = GPIO("/dev/gpiochip0", 1, "in")    # pin 18
gpio7 = GPIO("/dev/gpiochip0", 7, "in")    # pin 22
gpio8 = GPIO("/dev/gpiochip0", 8, "in")    # pin 26
gpio37 = GPIO("/dev/gpiochip0", 37, "in")  # pin 35
gpio13 = GPIO("/dev/gpiochip0", 13, "in")  # pin 36
gpio45 = GPIO("/dev/gpiochip0", 45, "in")  # pin 37
gpio38 = GPIO("/dev/gpiochip0", 38, "in")  # pin 38
gpio39 = GPIO("/dev/gpiochip0", 39, "in")  # pin 40
```

Figure 3: GPIO Instantiations

2.2 Additional Information:

The Coral Dev Board Mini integrates the Edge TPU, which is a small ASIC designed by Google for accelerating TensorFlow Lite models efficiently. It is capable of performing 4 trillion operations per second (4 TOPS), using just 2 watts of power, thus achieving 2 TOPS per watt. This on-device ML processing capability reduces latency, increases data privacy, and eliminates the need for a constant internet connection.

The board also features various peripherals and interfaces, making it versatile for different embedded applications.

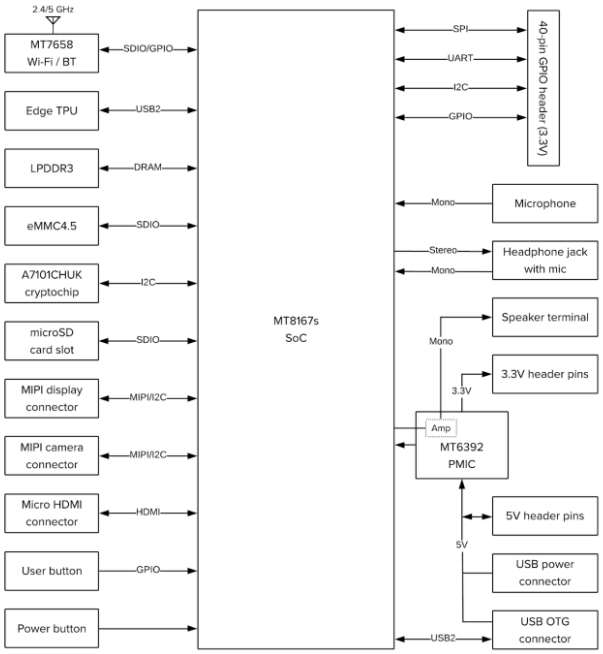


Figure 4: Block Diagram

3 Google Coral Camera

The Google Coral Camera is a high-performance 5-megapixel camera module designed specifically to work with the Coral Dev Board and Dev Board Mini. This camera module supports high-resolution image capture and video streaming, making it ideal for machine learning applications such as object detection and image classification.

3.1 Main Features:

- **Resolution:** 5 megapixels
- **Sensor Type:** Omnivision OV5645
- **Field of View (FOV):** 84.1° diagonal
- **Lens:** Fixed focus, F/2.0 aperture
- **Interface:** MIPI CSI-2
- **Connector:** 24-pin FFC
- **Frame Rate:** Up to 30 fps at full resolution

Specification	Detail
Image Sensor	Omnivision OV5645
Pixel Size	1.4 µm x 1.4 µm
Active Array Size	2592 x 1944 pixels
Shutter Type	Rolling shutter
Maximum Frame Rate	30 fps at 2592 x 1944 pixels
Output Format	RAW, YUV, JPEG
Operating Voltage	1.8V for I/O, 2.8V for analog
Power Consumption	Low-power mode: 90 mW, Active mode: 170 mW
Dimensions	25 mm x 24 mm x 8 mm

Table 3: Coral Camera Specifications

The Coral Camera connects to the Dev Board Mini through a MIPI CSI-2 interface, ensuring high-speed data transfer for real-time image processing. The fixed focus lens and large field of view make it suitable for a variety of computer vision applications.

4 Environmental Sensor Board

The Environmental Sensor Board from Coral is designed to add multiple environmental sensing capabilities to projects using the Coral Dev Board Mini. It includes a suite of sensors for monitoring air quality, temperature, humidity, and atmospheric pressure.

Main Features:

- **CO2 Sensor:** Measures carbon dioxide concentration

- **TVOC Sensor:** Measures total volatile organic compounds
- **Temperature and Humidity Sensor:** Measures ambient temperature and relative humidity
- **Barometric Pressure Sensor:** Measures atmospheric pressure

The Environmental Sensor Board connects via I2C to the Dev Board Mini and provides a comprehensive solution for environmental monitoring. It allows developers to create applications that respond to changes in the environment, making it ideal for smart home, industrial, and environmental monitoring projects.

Sensor	Model	Measurement Range	Accuracy
CO2	CCS811	400 – 8192 ppm	±30 ppm + 3% of reading
Temperature	HDC1080	-20°C to 85°C	±0.2°C
TVOC	CCS811	0 – 1187 ppb	±10 ppb or ±15% of reading
Humidity	HDC1080	0% to 100% RH	±2% RH
Pressure	BMP280	300 – 1100 hPa	±1 hPa

Table 5: Environmental Sensor Board Specifications

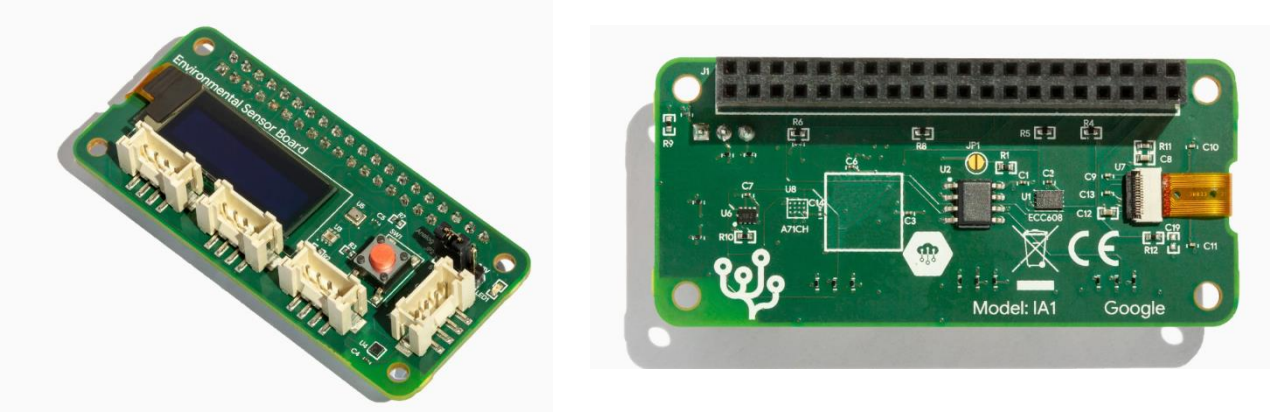


Figure 5: Environmental Sensor Board

5 Sensor Fusion

It is a process in which data from multiple sensors are combined to obtain a more accurate and comprehensive understanding of an environment or a phenomenon than would be possible with a single sensor alone. The goal of sensor fusion is to improve the reliability, accuracy, and robustness of information gathered from various sensors, allowing for better decision-making and perception in a wide range of applications. It plays a pivotal role in numerous fields, enabling the development of advanced applications that require accurate and reliable

perception and decision-making. There are 2 main types of sensor fusion techniques: Decision Level Fusion and Feature Level Fusion.

5.1 Decision Level Fusion

Decision-level fusion is a process in which multiple independent decisions or outputs from different sources or subsystems are combined or aggregated to make a final decision or inference. It is a technique commonly used in various fields, including signal processing, data fusion, machine learning, and artificial intelligence. The goal of decision-level fusion is to improve the overall decision-making performance by leveraging the diverse and complementary information provided by multiple sources. By combining the outputs of multiple subsystems or models, decision-level fusion can enhance decision accuracy, reliability, robustness, and generalization capabilities. Decision-level fusion can be performed using different methods, depending on the specific application and the nature of the decision being made. Some common techniques for decision-level fusion include:

● **Majority Voting**: Each source or subsystem provides a decision, and the final decision is determined based on the majority vote among all the sources. This method is effective when there is a low probability of all sources providing incorrect decisions simultaneously.

● **Weighted Voting**: Similar to majority voting, but each source's decision is assigned a weight based on its reliability or expertise. The final decision is determined by aggregating the weighted decisions. This approach allows more influential sources to have a greater impact on the final decision.

● **Dempster-Shafer Theory**: This theory of evidence combines decisions or beliefs from multiple sources using probability theory. It considers both the reliability of individual sources and the degree of agreement or conflict among them to reach a final decision.

● **Decision Trees or Rule-Based Systems**: The decisions from different sources are combined using decision trees or rule-based systems, where the rules or branches of the tree are based on the decisions provided by each source. This method allows for complex decision-making processes and can incorporate decision dependencies. Decision-level fusion is often used in applications such as sensor networks, multi-sensor systems, pattern recognition, data mining, and decision support systems.

5.2 Feature Level Fusion

It is a technique that involves combining or fusing the extracted features from multiple sensors to create a new set of features that capture the complementary or correlated information from each sensor. This fusion process occurs at the feature level before any decision or inference is made. In feature-level fusion, the goal is to create a consolidated feature representation that improves the performance of subsequent processing steps, such as classification/detection, recognition, or tracking. By combining features from multiple sensors, the fusion process aims to exploit the strengths of each sensor and compensate for their individual limitations. Here are the general steps involved in feature-level fusion for sensors:

● **Sensor data acquisition**: Collect data from multiple sensors capturing the same or related phenomenon. For example, you might have visual data from a camera sensor and depth data from a depth sensor.

● **Feature extraction**: Process the data from each sensor independently to extract relevant features. This step involves applying signal processing techniques, image processing algorithms, or any other methods suitable for the specific sensor modality.

● **Feature representation**: Represent the extracted features from each sensor as feature vectors or matrices. Each feature vector corresponds to the feature representation of a particular sample or instance.

● **Feature fusion**: Combine the feature vectors or matrices obtained from different sensors to create a fused feature representation. This fusion can be performed using various fusion techniques, such as concatenation, averaging, weighted summation, or principal component analysis (PCA).

● **Feature normalization**: Normalize the fused feature representation to ensure consistency and eliminate any scale or range differences. Common normalization techniques include mean normalization, min-max scaling, or z-score normalization.

● **Use fused features for subsequent processing**: Once the fused feature representation is obtained and normalized, it can be used as input for subsequent processing steps, such as image/object classification/detection algorithms, pattern recognition models, or machine learning classifiers.

Feature-level fusion can provide several benefits, including increased discriminative power, improved robustness to noise, enhanced generalization capabilities, and more comprehensive representation of the underlying data. It is widely used in applications such as multimodal biometrics, surveillance systems, autonomous vehicles, and remote sensing, where combining information from multiple sensors can lead to more accurate and reliable results. It's important to note that the specific fusion techniques and algorithms used for feature-level fusion can vary depending on the application, the nature of the sensor data, and the desired outcome. Experimentation and adaptation to the specific context are often necessary to achieve optimal fusion performance.

5.3 Importance of Sensor Fusion

Sensor fusion is important for several reasons, and it can be explained through the following key points:

5.3.1 Enhanced Accuracy and Reliability

Complementary Information: Different sensors provide various types of data that complement each other. For instance, in autonomous vehicles, cameras can provide detailed visual information, while LiDAR can give precise distance measurements. Combining these data sources leads to more accurate and reliable information.

Error Reduction: By integrating data from multiple sensors, the system can mitigate the errors and uncertainties inherent in individual sensors. If one sensor fails or gives incorrect data, others can compensate for it.

5.3.2 Robustness and Redundancy

Fault Tolerance: In critical applications, such as aerospace and autonomous driving, having multiple sensors ensures that the system can continue to function correctly even if one sensor fails. This redundancy enhances the overall robustness and safety.
Increased Confidence: Combining data from different sensors can provide higher confidence levels in the measurements and decisions made by the system.

5.3.3 Improved Situational Awareness

Comprehensive View: Sensor fusion allows for a more comprehensive understanding of the environment. For example, in robotics, combining vision sensors with touch and proximity sensors helps the robot navigate and interact with its environment more effectively.
Contextual Understanding: Different sensors can capture different aspects of the environment, providing a richer context. This is crucial for applications like surveillance, where combining video, audio, and motion sensors can lead to better threat detection and assessment.

5.3.4 Enhanced Performance

Data Integration: Fusing data from multiple sensors can enhance system performance by providing a more detailed and nuanced view of the situation. In healthcare, for example, combining data from heart rate monitors, blood pressure sensors, and glucose meters can lead to more accurate patient monitoring and diagnosis.
Real-Time Decision Making: In dynamic environments, such as autonomous driving or industrial automation, real-time sensor fusion allows for quicker and more informed decision-making.

5.3.5 Versatility and Flexibility

Adaptability: Sensor fusion systems can adapt to different environments and conditions by leveraging the strengths of various sensors. For instance, a drone equipped with both GPS and vision-based navigation can operate effectively in both open skies and cluttered indoor spaces.
Extended Capabilities: By combining different types of sensors, new capabilities and applications can be developed. For example, in augmented reality (AR), combining gyroscopes, accelerometers, and cameras enables more precise tracking and interaction with the real world.

5.3.6 Cost Efficiency

Optimized Resources: Sensor fusion can optimize the use of sensor data, potentially reducing the need for expensive, high-precision sensors. Instead, a combination of lower-cost sensors can achieve similar or even better performance.
Resource Management: Efficient sensor fusion algorithms can manage and process sensor data in a way that conserves computational resources, leading to more efficient and cost-effective systems.

By combining the strengths of multiple sensors, sensor fusion creates systems that are more accurate, reliable, and capable than those relying on single sensors. This leads to better performance in a wide range of applications, from autonomous vehicles and robotics to healthcare and environmental monitoring.

6 Object Detection Coco

6.1 Image Classification

Image classification is the task of categorizing images into predefined classes, such as identifying whether an image contains a cat, dog, or bird. This technique is fundamental in applications like medical imaging, where it can assist in diagnosing diseases from X-rays or MRIs, and in social media, where it helps in organizing and tagging photos.

- Simplifies the process of identifying the primary subject of an image.
- Helps in organizing and managing large datasets of images.
- Useful in specific applications where identifying the presence of a single object type is necessary.

6.2 Object Detection

Object detection extends image classification by not only identifying objects within an image but also determining their precise locations. This technology is crucial for applications such as autonomous driving, where the system must detect and react to pedestrians and other vehicles, security systems that monitor for intruders, and retail inventory management, where it helps in tracking product availability. It not only determines what objects are presented but also draws bounding boxes around them to indicate their positions.

Object detection provides both class labels and precise bounding box locations for multiple objects in an image, whereas image classification assigns a single class label to the entire image or a region of interest.

6.3 Introduction to COCO Algorithm

The COCO (Common Objects in Context) algorithm refers to the dataset and evaluation metrics used in the COCO dataset, a large-scale dataset designed for object detection, segmentation, and image captioning tasks. The COCO dataset contains over 330,000 images with more than 200,000 labeled images and 80 object categories. It is widely used in the computer vision community for benchmarking the performance of algorithms and models.

This dataset is well-suited for our project due to its comprehensive approach to object detection. Our project requires precise detection of humans and specific environmental conditions to trigger alerts. The COCO dataset includes detailed annotations for a wide range of objects, including humans, which is essential for our project's objective of detecting human presence accurately. This dataset is designed to handle complex real-world scenarios with varying lighting conditions, occlusions, and backgrounds. This robustness is beneficial for our project, which needs to operate effectively in diverse environments where the camera and sensors are deployed, especially for a fire detection application where the clarity of the image is not accurate.

The COCO algorithm is compatible with TensorFlow Lite, which is crucial for our project's edge device implementation. TensorFlow Lite is optimized for mobile and embedded devices, ensuring that our object detection model can run efficiently on the Google Coral Dev Board Mini without compromising performance.

7 Framework

- TensorFlow is an open-source AI Framework for machine learning and high-performance numerical computation from Google. It is a Python library that invokes C++ to construct and execute dataflow graphs.
- Tensor Flow Mobile: Directed to port tensor flow models into a mobile environment. Challenges:
    - Building the model for mobile application.
    - Adding libraries to mobile application.
    - Preparing the model file.
    - Optimizing binary size, file size, RAM usage, etc.

Common Cases for mobile ML

- Image recognition
- Speech recognition
- Gesture Recognition

7.1 Overview of TensorFlow Lite

TensorFlow Lite is a deep learning framework designed to enable on-device machine learning for mobile and edge devices. Developed by Google, TensorFlow Lite is a lightweight version of TensorFlow, optimized for performance and efficiency on devices with limited computational resources. It allows developers to deploy machine learning models on Android, iOS, and embedded Linux devices. TensorFlow Lite supports a range of machine learning tasks, including image classification, object detection, and natural language processing, making it versatile for various applications.

A TensorFlow Lite model is represented in a special efficient portable format known as FlatBuffers. This provides several advantages over TensorFlow's protocol buffer model format such as reduced size (small code footprint) and faster inference (data is directly accessed without an extra parsing/unpacking step) that enables TensorFlow Lite to execute efficiently on devices with limited compute and memory resources.

A TensorFlow Lite model can optionally include *metadata* that has human-readable model description and machine-readable data for automatic generation of pre- and post-processing pipelines during on-device inference.

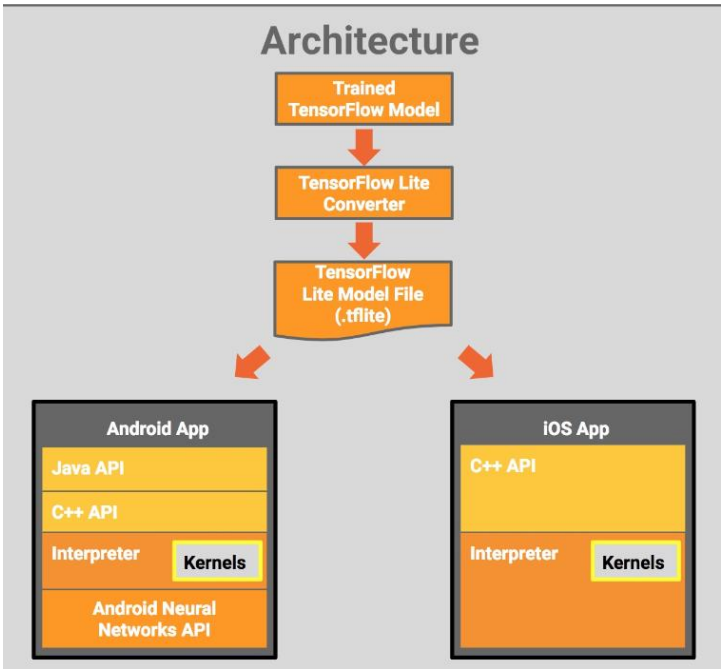


Figure 6: Tensorflow Framework Architecture

(Source: https://blog.tensorflow.org/2018/03/using-tensorflow-lite-on-android.html)

7.2 Key Features of TensorFlow Lite

TensorFlow Lite offers several key features:

- **Small Binary Size**: TensorFlow Lite models are significantly smaller than their TensorFlow counterparts, making them suitable for deployment on devices with limited storage.
- **Fast Execution**: Optimized for mobile and embedded environments, TensorFlow Lite ensures fast inference times.
- **Cross-Platform Compatibility**: Supports multiple platforms, including Android, iOS, and Linux.
- **Support for Hardware Acceleration**: Can leverage hardware acceleration through frameworks like NNAPI (Android), Core ML (iOS), and GPU delegate, enhancing performance on supported devices.

8 Board Setup and Configuration

The step by step set up was taken from the manufacturer manual of configuration (Source: https://coral.ai).

8.1 Requirements

To get started, we need the following:

1. A host computer running Linux
2. Python 3 installed.

3. One USB-C power supply (2 A / 5 V).
4. One USB-C to USB-A cable (to connect to a computer).
5. An available Wi-Fi internet connection.

The board is intended as a headless device. That is, you typically do not connect a keyboard and monitor, and use it as a desktop computer connecting to the board remotely with a secure shell terminal.

However, the Mendel OS does include a simple desktop with a terminal application. So, if you prefer, you can connect a monitor to the micro-HDMI port, and connect a keyboard and mouse to the USB OTG port.

8.2 Install MDT

MDT is a command line tool for your host computer that helps you interact with the Dev Board Mini. For example, MDT can list connected devices, install Debian packages on the board, and open a shell terminal on the board.

You can install MDT on your host computer follows:

```
python3 -m pip install --user mendel-development-tool
```

You might see a warning that mdt was installed somewhere that's not in your PATH environment variable. If so, be sure you add the given location to your PATH, as appropriate for your operating system. If you're on Linux, you can add it like this:

```
echo 'export PATH="$PATH:$HOME/.local/bin"' >> ~/.bash_profile

source ~/.bash_profile
```

8.3 Connect the board

- Connect your power supply to the board's USB power plug (the left plug, as shown in figure and connect it to an outlet.
- Connect your USB data cable to the other USB plug and to your host computer.

When you connect the USB cable to your computer, the board automatically boots up, and the board's LED turns green. It then takes 20-30 seconds for the system to boot up. If the green light doesn’t glow after some time (around 15 mins), there is a chance that board is not booting up. Try to connect to the mdt shell using serial console (refer appendix 15.2). If this step also doesn’t work, force-boot the board to fastboot mode (refer appendix 15.1).

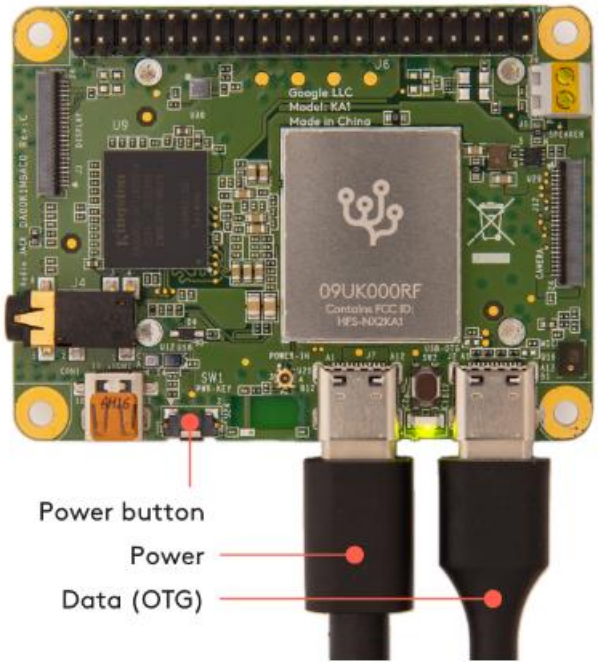


Fig 7: USB power and data cable connection to board

For more information, see section 5.1 in the Dev Board Mini datasheet.

8.4 Connect to the board's shell

MDT provides an easy way to establish an SSH connection with the board over USB, as follows.

Make sure MDT can see your device by running this command using the **Windows command prompt** and you should see output showing your board hostname and IP address:

```
C:\Users\Admin>mdt devices
indigo-valet            (192.168.101.2)
```

If you don't see your device, it might be because the system is still booting up. Wait a moment and try again.

Now to open the device shell, run this command. After a moment, you should see the board's shell prompt, which looks as below and now you're in the board!

```
C:\Users\Admin>mdt shell
Waiting for a device...
Connecting to indigo-valet at 192.168.101.2
Linux indigo-valet 4.19.125-mtk #1 SMP PREEMPT Thu Dec 10 02:36:13 UTC 2020 aarch64

The programs included with the Mendel GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Mendel GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jun 11 18:24:22 2024
mendel@indigo-valet:~$
```

If you are unable to enter the shell, refer appendix 15.2.

If you have already connected the board to a previous computer and you want to connect to a new computer, follow the steps in this link.

8.5 Connect to the internet

You'll need the board online to download system updates, TensorFlow models, and code examples.

Enter your network name and password with this command:

```
mendel@indigo-valet:~$ nmcli dev wifi connect "Mahima's" password "Mahi##4523" ifname wlan0
Device 'wlan0' successfully activated with '3fe0ce9a-4dac-4709-b2b5-197afe4f36ca'.
mendel@indigo-valet:~$
```

Verify your connection with below command and you should see your selected network listed in the output. For example:

```
mendel@indigo-valet:~$ nmcli connection show
NAME                UUID                                  TYPE      DEVICE
gadget0             cb42258b-fe89-4c14-9265-a4b766ef020b  ethernet  usb0
gadget1             966937e5-dab3-442b-b136-99cabc536a42  ethernet  usb1
Mahima's            8b44e298-feb2-45ff-a506-573332f06dbe  wifi      wlan0
```

To update mendal software, refer appendix 15.3 (recommended to be done only if required)

9 Implementation

9.1 Installation of Object Detection Model on board

The model used for this project is a pre-trained model and quantized model MobileNet V1.0. 224.l2norm. This model can be retrained or replaced. For installing this model to the Google Coral Mini, we need to clone pycoral library into the board by typing the following commands:

```
mkdir coral && cd coral

git clone https://github.com/google-coral/pycoral.git

cd pycoral
```

After library is installed, we need to add the model labels too and overall, all the requirements for our project:

```
bash examples/install_requirements.sh
```

9.2 Connect the camera with the Coral Mini Board

In order to connect the Coral Camera to the Coral Mini Board, the connection is made via the CSI connector as shown on Figure 7. (Source: https://coral.ai)

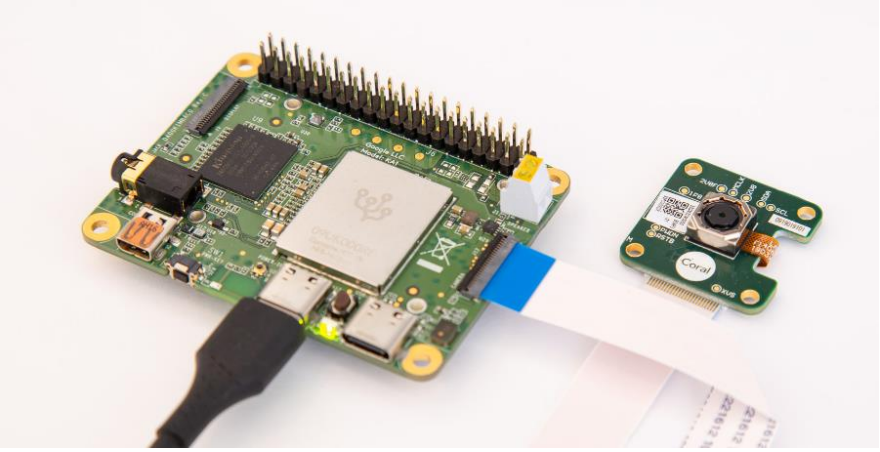


Fig 8: Coral camera and board connection

9.3 Connect the Carbon Monoxide Sensor to the circuit

We are using the MQ-7 Carbon Monoxide sensor. It has 4 pins – VCC (supply), Ground, Analog Output and Digital output. This sensor is used to detect high levels of carbon monoxide and then a LED is used to indicate the fire detection.

The MQ-7 sensor is a widely used gas sensor for detecting carbon monoxide (CO). It operates on the principle of a tin dioxide (SnO2) sensing layer, whose conductivity varies in the presence of CO gas. Here is a summary of the key features and how to interpret the data from an MQ-7 sensor:

Key Features

- **Detects**: Carbon Monoxide (CO)
- **Concentration Range**: 10 to 10,000 ppm (parts per million)
- **Power Consumption**: High heater current (~5V)
- **Sensitivity**: Good sensitivity to CO, lower sensitivity to alcohol, methane, and LPG
- **Response Time**: Fast response and recovery time

Pin Configuration

- **A0, A1**: Analog output pins (can be connected to a microcontroller's ADC)
- **D0**: Digital output pin (threshold adjustable via onboard potentiometer)
- **GND**: Ground
- **VCC**: Power supply (typically 5V)
- **H1, H2**: Heater pins

Basic Usage

- **Power the Sensor**: Connect VCC to 5V and GND to ground.
- **Preheat the Sensor**: Allow the sensor to preheat for at least 24 hours before taking accurate measurements.
- **Analog Reading**: Connect A0 (or A1) to an ADC pin of a microcontroller to get the sensor reading.
- **Digital Reading**: Connect D0 to a digital input pin of a microcontroller to get a high/low signal based on the CO concentration threshold set by the onboard potentiometer.

Data Interpretation

- **Analog Output**: The analog output voltage varies with the concentration of CO. You will need to convert the analog value (ADC reading) to a concentration in ppm using a calibration curve or reference data from the sensor's datasheet.

- **Digital Output**: The digital output goes high or low based on whether the CO concentration crosses the preset threshold.

The circuit implementation is as below:

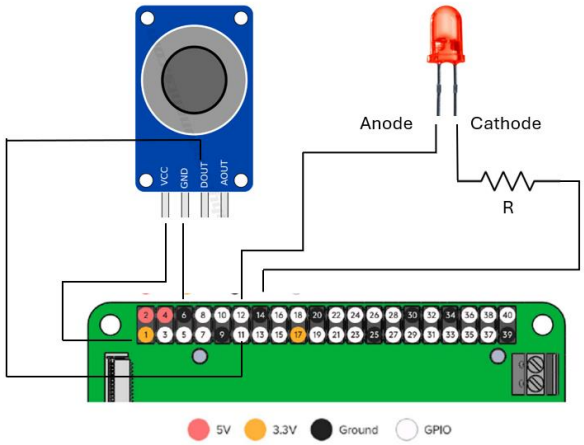


Fig 9: CO sensor and LED connection

9.4 Python Code for sensor fusion

First, we need to calibrate the sensor to have a desired threshold value that is done by running the code to read the sensor value and then changing the potentiometer value.

```
# Import libraries
import time
from periphery import GPIO

# Configure GPIO pins
gpio_in = GPIO("/dev/gpiochip0", 9, "in")  # pin 11

def main():
    """Main function to read sensor value from pin 11"""
    try:
        while True:
            # Read the input pin
            input_value = gpio_in.read()

            # Print the sensor value
            print(f"Sensor value : {input_value}")

            # Delay for a short period of time
            time.sleep(0.1)

    except KeyboardInterrupt:
        # Clean up GPIO pins and exit
        gpio_in.close()
        print("\nExiting gracefully.")

# Main
if __name__ == '__main__':
    main()
```

Then we can start with the complete system implementation. First, we need to take an image with the Coral Camera with the following code.

```python
def capture_and_save_image(image_path):
  """This function captures an image by using the Coral Mini Camera"""
  error = False

  #Initialize Google Camera
  cap = cv2.VideoCapture('/dev/video1')

  # Capture a frame
  ret, frame = cap.read()

  if ret:
    # Save the captured frames
    cv2.imwrite(image_path, frame)
  else:
    print("Failed to capture image")
    error = True

  # Release Camera
  cap.release()

  return error
```

The object detection function is configured to return a Boolean value which is true only when a human is detected. The model used was the coco model and in the following code the labels will be uploaded to the python program and be appended in a list.

```python
def object_detection():
    """This function runs the object detection TFLite model. First it defines the paths to be used, will then
    take a camera capture and process it into the model."""
    # Definition of the different needed paths
    base_path = "/home/mendel/pycoral/test_data/"
    label_path = base_path + "coco_labels.txt"
    interpreter_path = base_path + "ssd_mobilenet_v2_coco_quant_postprocess_edgetpu.tflite"
    capture_path = base_path + "capture.jpg"

    # Upon call of main
    capture_and_save_image(capture_path)
    image_path = capture_path
    output_path = base_path + "capture-processed.jpg"

    # Fetch the label list from coco_labels.txt to a list.
    label_list = []
    with open(label_path, "r") as file:
        # Read each line from the file and append it to the list
        for line in file:
            # Remove newline characters and add the line to the list
            label_list.append(line.strip())
```

After having the image and the labels of the model, now the image detection is implemented to analyse the given image, compare it with the model information and the labels provided on the image detection model alongside the threshold and other parameters.

```
parser = argparse.ArgumentParser(
    formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('-m', '--model', required=False,
                    help='File path of .tflite file')
parser.add_argument('-i', '--input', required=False,
                    help='File path of image to process')
parser.add_argument('-l', '--labels', help='File path of labels file')
parser.add_argument('-o', '--output',
                    help='File path for the result image with annotations')
parser.add_argument('-t', '--threshold', type=float, default=0.4,
                    help='Score threshold for detected objects')
parser.add_argument('-c', '--count', type=int, default=5,
                    help='Number of times to run inference')
args = parser.parse_args()

# Here we fill our arguments manually
labels = read_label_file(label_path) if args.labels else {}
interpreter = make_interpreter(interpreter_path)
interpreter.allocate_tensors()

# Open camera capture and pre-process it for the model
image = Image.open(image_path)
_, scale = common.set_resized_input(
    interpreter, image.size, lambda size: image.resize(size, Image.ANTIALIAS))

print('----INFERENCE TIME----')
print('Note: The first inference is slow because it includes',
      'loading the model into Edge TPU memory.')

# Run n rounds (depending on count argument) of the model and print each round's performance.
for _ in range(args.count):
    start = time.perf_counter()
    interpreter.invoke()
    inference_time = time.perf_counter() - start
    objs = detect.get_objects(interpreter, args.threshold, scale)
    print('%.2f ms' % (inference_time * 1000))
```

After invoking the image detection model, the results are given in the objs list, now the squares with the area of the object alongside the level of certainty and the name will be shown alongside with the image taken by calling the function draw_objects().

```
    # Print results
    print('-------RESULTS--------')

    # If no objects detected in the picture
    if not objs:
        print('No objects detected')

    # Check if a human is detected
    human_present = False

    # Print the results of the detected objects and the score. It also prints the box coordinates.
    for obj in objs:
        print(labels.get(obj.id, obj.id))
        print('  object:    ', label_list[obj.id])
        print('  score: ', obj.score)
        print('  bbox:  ', obj.bbox)

        # Check if the detected object is a human
        if obj.id == 0:
            human_present = True

    # Draw the rectangles over the capture and save it into the output path
    image = image.convert('RGB')
    draw_objects(ImageDraw.Draw(image), objs, labels, label_list)
    image.save(output_path)
    # image.show()

    return human_present
```

The draw_objects() function is as follows.

```
def draw_objects(draw, objs, labels, label_list):
    """Draws the bounding box and label for each object. If object is human, box is green, else red"""
    for obj in objs:
        bbox = obj.bbox
        label_string = label_list[obj.id]
        # Drawing of the rectangles of detected objects.
        if obj.id == 0:
            draw.rectangle([(bbox.xmin, bbox.ymin), (bbox.xmax, bbox.ymax)],
                           outline='green')
            draw.text((bbox.xmin + 10, bbox.ymin + 10),
                      '%s\n%.2f' % (label_string, obj.score),
                      fill='green')
            # Turn on the LED by setting the output pin to True (high)
            gpio_out.write(False)
        else:
            draw.rectangle([(bbox.xmin, bbox.ymin), (bbox.xmax, bbox.ymax)],
                           outline='red')
            draw.text((bbox.xmin + 10, bbox.ymin + 10),
                      '%s\n%.2f' % (label_string, obj.score),
                      fill='red')
```

The important part of the program is to configure the pins which is done at the start of program and the main function which invokes and object detection model and also checks for the sensor data.

```
# Configure GPIO pins
gpio_out = GPIO("/dev/gpiochip0", 36, "out")  # pin 12
gpio_in = GPIO("/dev/gpiochip0", 9, "in")  # pin 11
```

```
# Main
if __name__ == '__main__':
    """This main function is in charge of detecting the light presence and triggering the object detection model"""
    try:
        var = True

        while True:
            print("Checking Sensor Data....")
            # Check if human is detected
            human_detected = object_detection()
            sensor_value = gpio_in.read()

            if human_detected and sensor_value:
                print("Human Detected and Smoke Detected")
                # Turn on the LED by setting the output pin to True
                gpio_out.write(True)
                print("Fire Detected!!")
                var = False
                time.sleep(5)
                sys.exit()

            else:
                # Turn off the LED by setting the output pin to False
                time.sleep(5)
                gpio_out.write(False)

            # Delay for a short period of time
            time.sleep(0.1)

    except KeyboardInterrupt:
        # Clean up GPIO pins and exit
        gpio_in.close()
        gpio_out.close()
```

9.5 Push the code to the Coral Mini board

We can push the implemented code to the coral mini board using the below command by specifying the desired path.

```
C:\Users\Admin>scp -i "C:\Users\Admin\.ssh\id_rsa" "E:\TeamProject\Project_2024\scf_demo2.py" mendel@192.168.101.2:/home
/mendel/test
scf_demo2.py                                                  100% 6842     6.6MB/s   00:00
```

9.6 Get the captured image from the Coral Mini board

We can copy the captured image from the coral mini board using the below command by specifying the desired path.

```
C:\Users\Admin>scp -i "C:\Users\Admin\.ssh\id_rsa" mendel@192.168.101.2:/home/mendel/pycoral/test_data/capture-processed.jpg C:\Users\Public
capture-processed.jpg                                                                                    100%   35KB   4.0MB/s   00:00
```

10 How to run it

To summarize the script uses a COCO (Common Objects in Context) dataset with a quantized Single Shot MultiBox Detectro (SSD) object detection model with the MobileNetV2 backbone. The model has already been optimized for Edge TPUs, so the model is smaller and has a faster interface by using quantization techniques. The model is used for object detection on edge devices. It can identify objects into ~90 different classes and draws the box coordinates for the labelled objects. Since this model is quantized, it makes it suitable for real-time interface on resource-constrained devices.

Modifications were made to support the following features:

- Support for Coral Mini peripherals (GPIO)
- Continuously monitor if a Human is detected. Specific Human Detection on model:
    - When detecting a Human, capture an image and the processed image will draw a GREEN box around humans instead of a RED box.
- Updated boxing and labelling so processed image show the name of the actual labels and not the IDs.
- As an action of this, the COx sensor data is read and checked if it has reached a pre-defined threshold. If the threshold is met, a “Fire Detected” log is sent.

In order to execute the program, the following steps need to be done:

1. Enter Google Coral Mini's Terminal with “mdt shell” in the computer’s command.
2. Upload to the Coral Mini Board:
    a. The desired TFLite model (see steps on section 9.1.1)
    b. Python script that executes the model, scf_demo2.py in our case. See the steps in 9.1.5 on how to get an edited code in the Host OS (Windows in our case) to the board.
3. Run the modified COCO SSD script by typing “python3 scf_demo3.py” or whatever name was used for the python script.
4. The system keeps printing the objects it detects on the terminal. This happens because the system captures a snapshot and runs the image detection model inside the Coral Mini Board.
5. Ensure that a human in front of the camera.
6. Provide sensor input to the COx sensor such that the threshold is met. For simulating the sensor value, we have configured the sensor to detected small amount of CO2. Threshold can be adjusted as per requirement using the potentiometer.
7. Once the threshold is met and the human is detected, a message “Fire detected” is printed on the terminal and the snapshot of the environment is stored.

In order to see retrieve the image resulted from the image detection follow these steps:

1. Exit the Google Coral Mini Board by typing “exit” on the terminal.
2. Pull the image from the Google Coral Mini Board to a path in the Host Computer (Windows in our case). See the steps in 9.1.6

11 Results

The resulting script output of the terminal can be seen in Figure 10.

```
Checking Sensor Data....
----INFERENCE TIME----
Note: The first inference is slow because it includes loading the model into Edge TPU memory.
299.85 ms
59.33 ms
59.35 ms
58.91 ms
59.20 ms
-------RESULTS--------
72
  object:     laptop
  score:  0.671875
  bbox:   BBox(xmin=322, ymin=297, xmax=622, ymax=415)
Checking Sensor Data....
----INFERENCE TIME----
Note: The first inference is slow because it includes loading the model into Edge TPU memory.
298.33 ms
58.56 ms
58.85 ms
58.44 ms
59.03 ms
-------RESULTS--------
0
  object:     person
  score:  0.8046875
  bbox:   BBox(xmin=295, ymin=46, xmax=634, ymax=493)
71
  object:     tv
  score:  0.72265625
  bbox:   BBox(xmin=149, ymin=343, xmax=327, ymax=464)
Human Detected and Smoke Detected
Fire Detected!!
mendel@indigo-valet:~/test$
```

Figure 10: Terminal output

The above image shows the output when no object is detected as well as on object, i.e., TV in our case, is detected. In the below image, it can be noticed that a red box is placed around the TV and a green box around human.

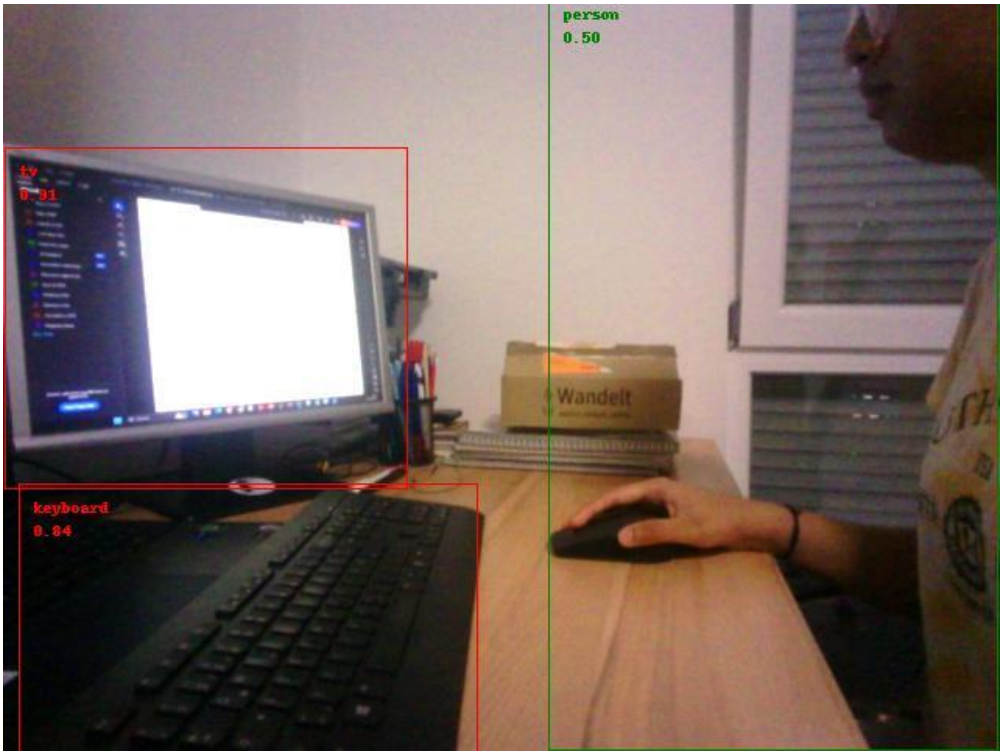


Figure 11: Captured image

12 Applications of the system

The integration of the Coral Mini Board with a camera and COx sensors to detect humans and monitor COx levels presents several highly relevant applications. These applications leverage the system's capabilities for enhanced safety and security in various domains. Key applications include:

1. Fire safety and prevention: The combined monitoring of COx levels and human presence provides a robust early warning system for fire hazards. This application is particularly valuable in environments where early detection of fire can save lives and prevent significant property damage. Eg; Home safety system, Industries, Office buildings, Schools, Hospitals etc.
2. Agricultural and Forest Monitoring:
    - Forest Fire Detection: The system can be used in forests to detect early signs of fire through COx monitoring, helping in rapid response to prevent widespread damage.
    - Farm Security: On farms, the system can help monitor for intruders and ensure the safety of livestock by detecting human presence and potential fire hazards.
3. Environmental Monitoring:
    - Industrial Plants: The system can be used in factories and industrial plants to monitor COx levels, ensuring that the environment remains safe for workers and preventing potential fire outbreaks.
    - Mining Operations: It can be used in mining operations to detect dangerous levels of gases and monitor the presence of workers in hazardous areas.

13 Lessons Learnt

Throughout the duration of the project, several key insights emerged, providing valuable lessons for future initiatives.

1. During the project, we observed significant differences in performance and compatibility between Windows and Linux environments when using the Mendel Development Tool (MDT). Linux provided a more stable and seamless experience, particularly when managing dependencies and system configurations. Utilizing a Virtual Machine (VM) on Windows helped mitigate some issues, but native Linux was preferable for optimal performance.
   Note: One of the steps we tried was to get a Linux OS running on Windows 11 Home as a dual boot. We noticed that the Home version has bitlocker encryption enabled on the memory due to which dual boot was not possible. It works fine if the bitlocker encryption is disabled which is feature only for Windows Pro users.
2. Ensuring proper USB configuration in a virtual machine setup becomes crucial. For our project, we used Oracle VM VirtualBox Manager (Setup source: Ubuntu 22.04 In VirtualBox - The Complete Guide (youtube.com)) with an Ubuntu 22.04.4. Below settings were used in the USB settings of the OS:

Figure 13: USB setup in Virtual Machine

3. When transitioning the development board between different computers, the necessity of a .key file to be transferred to the new computer. This file ensures consistent and secure access to the board and hence it is important to maintain and transfer authentication keys securely. (Source: Mendel Development Tool (mdt) | Coral)
4. For MDT key, if other SSH tools are preferred, simply generate your own key and push it to the device using “mdt pushkey”. Recovering the MDT access is also possible by restarting the mdt-keymaster service on the board. (Source: Mendel Development Tool (mdt) | Coral)
5. Encountering situations where the board became unresponsive ("bricking") emphasized the importance of having a reliable reset procedure. The reset guide provided. The forced fastboot of the board has to be executed on a Linux OS.
6. Ensuring not to update the board after force-fastboot. It is a recommended step in the Google Coral Mini Official website.

7. A safety shut down of the board has to be performed always. Once the tasks are done, keep the power connected to the board. Unplug the USB OTG cable and press the power button on the board.
8. Proper documentation of pin instantiation proved critical. Early in the project, inconsistencies and errors in pin assignments caused numerous debugging challenges. Maintaining comprehensive and up-to-date documentation of pin configurations facilitated smoother development and quicker troubleshooting.
9. Sticking to a regular network without any VPNs ensured a more stable and reliable internet connection for development tasks.
10. Support of the environment board to the Google Coral Mini wouldn't be planned by the Google Dev Team.

14 Future Work

During this whole project we build within code already provided for different sources, such as the official Google Coral (Mini) Board webpage for all the set-up, communication and troubleshooting of the Camera and the Board, alongside with a Github with a project already done to process an image with the COCO SSD model.

14.1 Environment board development

During the first phase of this project the usage of the "Coral Environmental Sensor Board" was intended to be used but after some research it was confirmed that the Environmental Board is only compatible with the Coral Board, there is no foreseeable plan to make it compatible in the near future with the Coral Mini Board.

Creating an external system that functions similarly to the Coral Environmental Sensor Board is the first possible path for this project's future development. The ability to sense environmental light, pressure, temperature, and humidity would all be replicated by this device. In doing so, it would give the Coral Mini board the environmental data it needs for precise fire monitoring and detection. For this method to continue having strong fire detection capabilities, appropriate sensors would need to be designed, integrated, and their connection with the Coral Mini board would need to be flawless.

14.2 Hardware Acceleration

Future work on optimizing real-time processing using the Google Coral Mini USB Accelerator or Mini PCIe Accelerator can significantly enhance capabilities in critical applications like fire detection and security monitoring. Leveraging the Edge TPU for hardware acceleration involves using the Edge TPU Compiler to optimize TensorFlow Lite models, converting them into a format that the Edge TPU can process more efficiently through quantization. Additionally, exploiting the TPU's parallel processing abilities allows multiple inferences to be handled simultaneously, crucial for monitoring multiple camera feeds in real-time.

Implementing lightweight and optimized neural network architectures, such as MobileNet (used in our current project) and EfficientNet, is another key area. These models are designed to be efficient and require fewer computational resources, making them ideal for edge devices. Techniques like quantization and pruning further reduce the model size and inference time, enhancing performance without significantly compromising accuracy.

Efficient system integration is essential for minimizing latency. This includes optimizing data preprocessing (e.g., resizing images, normalizing pixel values) and postprocessing routines to ensure that inputs and outputs are handled swiftly. Emphasizing local edge computing reduces the need for data transfer to central servers, thereby cutting down response times. Additionally, regular updates to firmware and software, along with performance monitoring, ensure that the system remains up-to-date and operates efficiently. These combined strategies ensure rapid, reliable responses in time-sensitive scenarios, enhancing overall system performance.

14.3 Customizable alert and notification systems

Future work on developing advanced notification systems for fire detection can focus on creating a more sophisticated and customizable alert system. This system should be capable of sending notifications based on detection results and user-defined criteria through multiple channels. Users could receive email notifications via an SMTP server, SMS notifications using services like Twilio, and push notifications via a dedicated mobile app or web service. By providing multiple notification options, the system ensures that users are promptly alerted through their preferred communication method. Additionally, the notification system should allow users to set specific criteria for different types of alerts, such as varying detection thresholds, monitored zones, and times of day when alerts should be active.

14.4 Key phrase detection

Furthermore, we used the Coral Mini to develop a keyphrase detection system, which may be linked into the fire detection system to improve functionality. Through this integration, the system may be programmed to react to voice commands for a variety of functions, such as monitoring and controlling fire detection processes, activating alarms, or querying sensor data.

Some snapshots of the keyphrase detection system:

```
mendel@indigo-valet:~$ cd voice_demo
mendel@indigo-valet:~/voice_demo$ ls
project-keyword-spotter
mendel@indigo-valet:~/voice_demo$ cd project-keyword-spotter/
mendel@indigo-valet:~/voice_demo/project-keyword-spotter$ ls
audio_recorder.py  hearing_snake_metadata.json  media           models         README.md             run_snake.sh
config             install_requirements.sh      mel_features.py  __pycache__    run_hearing_snake.py  run_yt_voice_control.py
CONTRIBUTING.md    LICENSE                      model.py         pygame_images  run_model.py          run_yt_voice_control.sh
mendel@indigo-valet:~/voice_demo/project-keyword-spotter$ python3 run_model.py
```

Figure 14: How to run the Keyphrase detection system

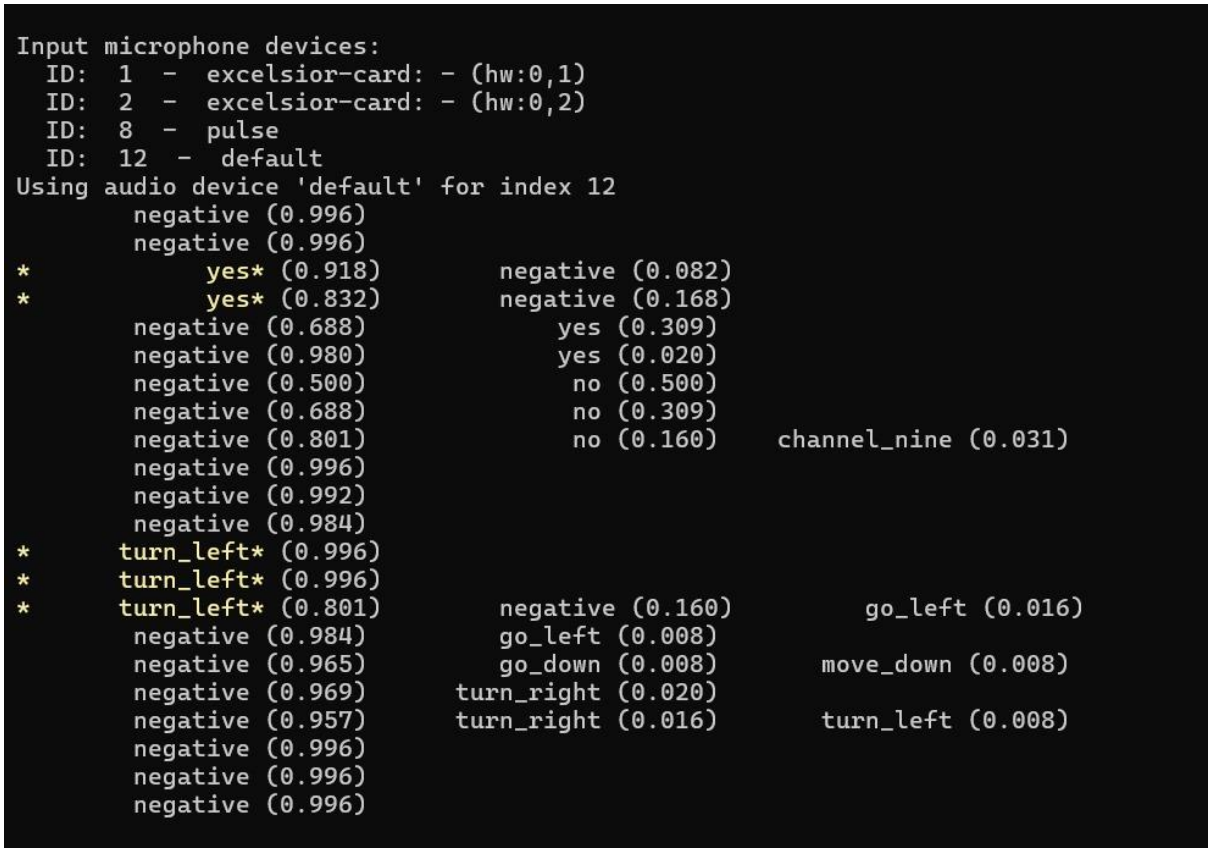


Figure 15: Results of the Keyphrase detection system

15 Appendix

15.1 Force-boot into fastboot mode (for bricked board scenario)

These following steps have to be performed on a Linux OS as the shell scripts have Linux OS dependencies.

1. Install the fastboot tool. On Linux, you can install as follows:

```
sudo apt-get install fastboot
```

2. Reload the udev rules installed by fastboot above (actually from the (android-sdk-platform-tools-common (package dependency):

```
sudo udevadm control --reload-rules && sudo udevadm trigger
```

Also make sure your Linux user account is in the plugdev and dialout system groups by running this command:

```
sudo usermod -aG plugdev,dialout <username>
```

3. Then reboot your computer for the new groups to take effect.
4. Do not power the board. If connected, unplug all cables from the board now. However, connect your USB data cable to your computer, and have it ready to connect to the board later.
5. On your host computer, install these Python packages required by the boot script:

```
python3 -m pip install pyftdi pyserial
```

6. Download the latest system image on your host computer:

```
cd $HOME/Downloads

curl -O https://dl.google.com/coral/mendel/excelsior/excelsior-eagle-20201210233645.zip

unzip excelsior-eagle-20201210233645.zip
```

7. Run enable_lk_fastboot.sh to enable fastboot via the LK bootloader:

```
cd excelsior-eagle-20201210233645

bash enable_lk_fastboot.sh
```

   By design, this has no immediate effect; the script waits for the device to boot. So the console should print several lines, ending with this one:

```
INFO: Waiting to connect platform...
```

8. Use a wire to connect the KPCOL0 test point to ground, as shown in figure 2. Ideally, use a female-to-male jumper wire, with the female end attached to pin 39 on the GPIO header (ground), and then hold the male end firmly on the test point.

9. While maintaining a firm connection with the jumper wire, carefully connect the USB data cable to the board's USB OTG port. As the board boots, the enable_lk_fastboot.sh script you ran above identifies the board, enables the LK bootloader, and initiates fastboot mode. That prompt will return to you after it prints this:

```
INFO: Loading file: lk.bin
INFO: Send lk.bin
INFO: Jump da
```

   If you don't see this, repeat from step 7.

10. When the prompt returns to you, you can release the jumper wire. Verify the board is in fastboot mode using "fastboot devices" command. The device will be listed like this:

```
0123456789ABCDEF  fastboot
```

11. Now you can flash the board using “bash flash.sh” command.

15.2 Connect to the board's shell via Serial console

After the mdt shell command, you are unable to connect to the board, you can directly connect to it via the serial console.

Additional requirements:

- USB-to-TTL serial cable (it must support 3.3 V logic levels, such as this one by Adafruit)
- 2 A / 5 V USB-C power supply

To connect to the Dev Board Mini's serial console from a Windows 10 computer:

1. Connect the USB-to-TTL serial cable to your computer and the board as shown in the image:

- Pin 6: Ground
- Pin 8: UART TX
- Pin 10: UART RX

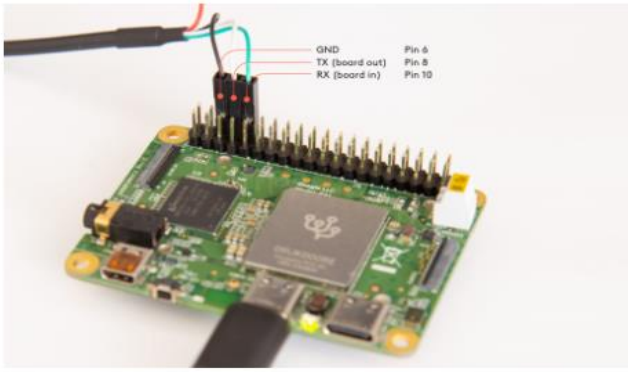


Figure 1. The ground and UART pins connected to 40-pin header

Warning: Do not connect the power wire if provided. This can cause power flow issues and potential hardware damage.

2. Power the board:

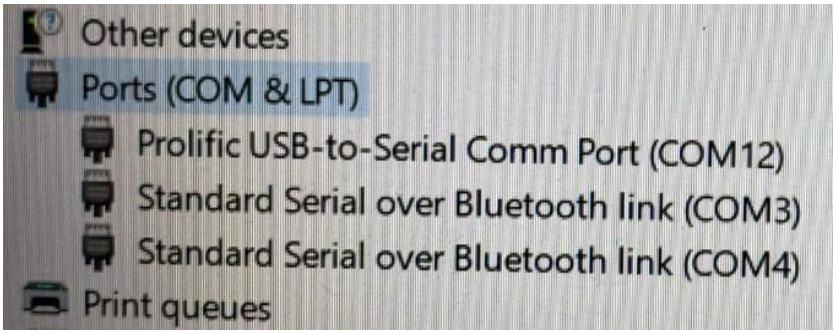
- Connect the board to power using the USB power port.
- Press the power button firmly to boot the board.

3. Find the COM port:

- Open Device Manager on your Windows computer.

- Within a minute, Windows should automatically install the necessary driver. Expand "Ports (COM & LPT)" to find your serial cable (e.g., "Silicon Labs CP210x USB to UART Bridge" or "Prolific USB-to-Serial Comm Port").

- Note the COM port (e.g., "COM12" as below).



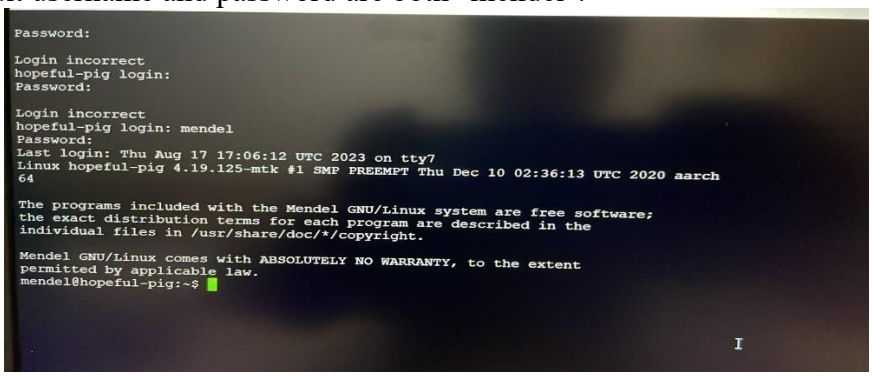
If Windows cannot identify the USB cable, it will be listed under "Other devices". Right-click the device and select "Update driver" to find the appropriate driver.

4. Open a serial console:
- Use PuTTY or another serial console app to start a serial console connection with the noted COM port, using a baud rate of 115200.

For PuTTY:
- Select "Session" in the left pane.
- For "Connection type", select "Serial".
- Enter the COM port (e.g., "COM12") for "Serial line" and "115200" for "Speed".
- Click "Open".

If the terminal screen is blank, press Enter to see the login prompt.

5. Login:
- The default username and password are both "mendel".



15.3 Update the Mendel software

To ensure you have the latest software updates:

1. Run the following commands in the board's shell terminal:

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo reboot now
```

Note: Rebooting is required for some kernel changes to take effect.

2. If you encounter a certificate verification error, it may be due to an incorrect system date. You can manually set the date with:

```
sudo date +%Y%m%d -s "20210121"
```

Then try to upgrade again.

3. When the board's LED turns green after reboot, it's ready for login. Reconnect from your computer with MDT:

```
mdt shell
```

Now, you're prepared to run TensorFlow Lite models on the Edge TPU!
Note: The `dist-upgrade` command updates all system packages for your current Mendel version. To upgrade to a newer Mendel version, you'll need to flash a new system image.

15.4 Relevant Links

1. https://coral.ai/docs/dev-board-mini/get-started/
2. https://coral.ai/docs/edgetpu/models-intro/
3. https://coral.ai/docs/edgetpu/retrain-detection/