The [Raspberry](https://botland.store/399-raspberry-pi) Pi AI Camera, based on the Sony IMX500 12MPx intelligent vision sensor, is an innovative solution for projects requiring real-time image processing. With an integrated inference engine, [the camera](https://botland.store/945-kamery-do-raspberry-pi) supports on-board processing for popular neural network models such as MobileNet, which significantly reduces the load on the [Raspberry Pi](https://botland.store/399-raspberry-pi) processor. The camera offers a 76° viewing angle and manual focus adjustment, providing high flexibility in adapting to various applications.  
**Raspberry Pi AI Camera Specifications**

* Sensor: Sony IMX500
* Matrix resolution: 12.3 MPx
* Viewing angle: 76°, 66°±3° / 52.3°±3°
* Focus setting: manual
* Focus range: 20 cm to infinity
* Inference engine: integrated, supporting popular neural network models (MobileNet preloaded)
* Sensor size: 7.857 mm (type 1/2.3)
* Pixel size: 1.55 x 1.55 μm
* Resolution: 4056 x 3040 px
* IR cut filter: integrated(can be removed [Camera - Raspberry Pi Filter Removal Documentation](https://www.raspberrypi.com/documentation/accessories/camera.html#filter-removal))
* Focal length: 4.74 mm
* Aperture: F1.79
* Infrared sensitivity: none
* Output: Image (Bayer RAW10), ISP (YUV/RGB), Roit, Metadata
* Tensor input size: 640 x 640 px
* Input data type: 'in8' or 'uint8'
* Memory Size: 8388480 bytes
* Frames per second:
  + 2x2 binned: 2028 x 1520 px 10-bit 30 fps
  + full resolution: 4056 x 3040 px 10-bit 10 fps
* Dimensions: 25 x 24 x 11.9 mm
* Tape length: 200 mm
* Ribbon Connector: 15 x 11mm FPC or 22 x 0.5mm FPC
* Compatibility: All Raspberry Pi models

## **Model deployment**

To deploy a new neural network model to the Raspberry Pi AI Camera, complete the following steps:

1. Provide a neural network model.
2. Quantise and compress the model so that it can run using the resources available on the IMX500 camera module.
3. Convert the compressed model to IMX500 format.
4. Package the model into a firmware file that can be loaded at runtime onto the camera.

**The first three steps will normally be performed on a more powerful computer such as a desktop or server. You must run the final packaging step on a Raspberry Pi.**

## **Step 1: Model Preparation (Offline - Desktop/Server)**

### 1. Provide a Neural Network Model

Start with a trained neural network model. You can either:

Reuse pre-trained models (e.g., MobileNet, YOLO, ResNet) or train a custom model using a framework like:TensorFlow /Keras or PyTorch (to export as .onnx)

### **Step 2 : Quantisation and Compression**

The IMX500 has **limited computational resources(**, so your model must be optimized.

Use **Sony’s Model Compression Toolkit** to:

* **Quantize**: Reduce precision (e.g., float32 → int8) to speed up inference.
* **Compress**: Minimize model size while preserving accuracy.

#### **🔧 Installation:**

bash

pip install model\_compression\_toolkit

This toolkit outputs a compressed model in either:

* **Keras format** (.h5) – from TensorFlow
* **ONNX format** – from PyTorch

🔍 Why quantisation matters: IMX500 uses a fixed-function AI accelerator; running full-precision models is infeasible due to memory/compute limitations.

### **Step 3:-Conversion**

To convert a model, first install the converter tools:

For Tensorflow :-

pip install imx500-converter[tf]

\*\*Always use the same version of TensorFlow you used to compress your model.

For Pythorch:-

pip install imx500-converter[pt]

If you need to install both packages, use two separate Python virtual environments. This prevents TensorFlow and PyTorch from causing conflicts with one another.

### **Convert the Model**

#### **For TensorFlow:**

bash

imxconv-tf -i model.h5 -o output\_folder

#### **For PyTorch:**

bash

imxconv-onnx -i model.onnx -o output\_folder

Each command produces:

* packerOut.zip: Binary representation of the model
* memory\_report.html: Report of RAM/compute usage.

## **Step 4: Model Packaging (On Raspberry Pi)**

This step **must** be run on the **Raspberry Pi**, as it prepares the model for deployment in a format the IMX500 can use at runtime.

### **Install Required Tools:**

bash

sudo apt install imx500-tools

### **Package the Model:**

bash

imx500-package -i path/to/packerOut.zip -o output\_folder

This generates:

* network.rpk: The final **runtime package** that will be loaded into the IMX500 camera.

You can now reference this file when launching AI applications on the Raspberry Pi that interface with the camera.

#### **Object detection**

The MobileNet SSD neural network performs basic object detection, providing bounding boxes and confidence values for each object found. imx500\_mobilenet\_ssd.json contains the configuration parameters for the IMX500 object detection post-processing stage using the MobileNet SSD neural network.

imx500\_mobilenet\_ssd.json declares a post-processing pipeline that contains two stages:

1. imx500\_object\_detection, which picks out bounding boxes and confidence values generated by the neural network in the output tensor.
2. object\_detect\_draw\_cv, which draws bounding boxes and labels on the image.

The MobileNet SSD tensor requires no significant post-processing on your Raspberry Pi to generate the final output of bounding boxes. All object detection runs directly on the AI Camera.

The following command runs rpicam-hello with object detection post-processing :

rpicam-hello -t 0s --post-process-file /usr/share/rpi-camera-assets/imx500\_mobilenet\_ssd.json --viewfinder-width 1920 --viewfinder-height 1080 --framerate 30

After running the command, you should see a viewfinder that overlays bounding boxes on objects recognised by the neural network:

To record video with object detection overlays, use rpicam-vid instead:

You can configure the imx500\_object\_detection stage in many ways.

For example, max\_detections defines the maximum number of objects that the pipeline will detect at any given time. threshold defines the minimum confidence value required for the pipeline to consider any input as an object.

The raw inference output data of this network can be quite noisy, so this stage also performs some temporal filtering and applies hysteresis. To disable this filtering, remove the temporal\_filter config block.