ChatGPT

Elixir/Nx Inference Pipeline on Raspberry Pi CM5

Deploying real-time object detection on a Raspberry Pi (Compute Module 5) involves setting up an Elixir process that captures video frames, runs a neural network inferences, and overlays or emits annotations. In practice, you can use Nx (Elixir's tensor library) with a compiled model to perform inference on each frame. For example, one can convert each frame (via Evision or `Nx.Image`) into an `Nx.Tensor`, then feed it into a pre-loaded neural network (using **Axon** or the **Bumblebee** high-level API). This produces bounding-box predictions for objects of interest. The annotated image can be generated by drawing rectangles and labels on the frame (e.g. with Evision/OpenCV commands like `Evision.rectangle`/`putText`, or by using a library such as Image to compose shapes). Output options include streaming the annotated video to an operator *and* separately emitting the raw box coordinates (as JSON or message) for onboard navigation.

Example: A YOLO model detecting and labeling objects ("person", "car") in a Raspberry Pi video frame (image from LearnOpenCV).

Choosing an Object-Detection Model

For "military interesting objects" and other drones, one-stage detectors like YOLO (You Only Look Once) are recommended due to their speed/accuracy balance on edge devices. The latest YOLO variants (e.g. YOLOv8 or the newly released **YOLO11** series) have *Nano* and *Small* models optimized for low-power hardware. For instance, Ultralytics notes that YOLO11n and YOLO11s (Nano/Small) run efficiently on Pi-class devices [1]. These models are pretrained on COCO (people, vehicles, etc.), so they will detect generic targets like cars, aircraft, or persons out-of-the-box; custom classes (e.g. specific drone models or military equipment) would require fine-tuning with additional data. YOLO networks can be exported to formats like ONNX or **NCNN** (a mobile/edge inference library by Tencent) for faster ARM inference. In fact, converting to NCNN + quantizing can slash inference time by ~62%, achieving low-millisecond latency [2]. Official Raspberry Pi guidance highlights that Ultralytics models can be exported to NCNN to leverage ARM64 optimizations (e.g. NEON) and 8-bit quantization [3]. In short, a Tiny/Small YOLO (e.g. `yolov8n` or `yolov11n`) is a strong choice. Alternative lightweight detectors include **MobileNet-SSD**, **EfficientDet-lite**, or other "lite" models, but YOLO-nano variants tend to achieve higher FPS on the Pi's CPU.

> **Citing sources:** YOLO11/YOLOv8 are explicitly designed for Pi: Nano/Small variants give best speed on low-power hardware [1]. Exporting YOLO to NCNN (with quantization) yields extremely fast inference on ARM (e.g. 25+ FPS at 240×240) [4] [3].

Running the Model with Elixir Nx

In Elixir, you can load or import a pretrained model and run inference via Nx/Axon. Two main approaches are: (1) **Bumblebee** (high-level HF/Onnx integration) or (2) **Axon + AxonOnnx/AxonInterp** (direct ONNX import). For example, Bumblebee's vision API allows loading Hugging Face models into an `Nx.Serving` struct with minimal code [5]. Although Bumblebee's examples often show image classification (e.g.

ResNet-50) [5], the same pattern applies to detection models if available in HF or ONNX form. If a YOLO ONNX file is used, you can import it with AxonOnnx:

```
{model, params} = AxonOnnx.import("yolo_small.onnx")
prediction_fn = Axon.build(model, compiler: EXLA)
```

Then each frame (converted to an `Nx.Tensor` of shape `{1,3,H,W}`) is passed through the compiled `prediction_fn`, returning raw box coordinates and scores. (Alternatively, the AxonInterp library can wrap an ONNX YOLO directly; its docs show a YOLOv4 example using `use AxonInterp, model: "yolov4.onnx", inputs: [{1,3,608,608}], outputs: [...]` [6].) In any case, apply post-processing (confidence thresholding and **non-max suppression**) to filter detections. For instance, the `PostDNN.non_max_suppression_multi_class/4` function (used in the AxonInterp example [7]) can combine boxes and scores into final labeled boxes.

System libraries: Use Evision (OpenCV) or NxImage to handle frame I/O and conversion to tensors. In one example, the first video frame was captured with `Evision.VideoCapture.read/1` and converted via `Evision.Mat.to_nx() |> Nx.backend_transfer()` [8] [5]. Ensure Nx's backend is set to EXLA (`config :nx, default_backend: EXLA.Backend`) for optimized compiled execution [9]. Combine these with the existing Membrane RTSP pipeline: have each processed frame pushed into the Nx/Axon flow, then the resulting annotated frame passed to the RTSP encoder.

> **Citing sources:** The Elixir "Bumblebee" tutorial shows loading a pre-trained vision model and running inference on video frames [5]. The AxonInterp docs give an example of defining a YOLO model from an ONNX file and drawing boxes on the image [6] [7]. The AxonOnnx library enables importing ONNX models into Axon [10].

Overlaying Annotations on Video

After inference, draw the bounding boxes and labels on the frame before streaming. In Elixir, you can use Evision (OpenCV bindings) to render shapes/text on an `Evision.Mat`. For each detected box `[x1,y1,x2,y2]`, call `Evision.rectangle(frame, {x1, y1}, {x2, y2}, color)` and `Evision.putText(frame, label, ...)`. Alternatively, the Image library can create shape overlays: e.g. `{:ok, box_img} = Image.Shape.rect(width, height, fill_color: :none, stroke_color: color)` produces a rectangle image [11], which you can then composite over the original frame. The annotated frame (image or JPEG) is then sent via the RTSP pipeline.

In addition to visual overlay, the system should make box coordinates available (e.g. as JSON or Elixir messages) for navigation logic. The original detections from Nx (class IDs, scores, and box coordinates) can be packaged and emitted to the Elixir process controlling the UAV. This separation allows using either the composited video for the human operator or the raw box data for an autonomous controller.

> **Citing sources:** The Elixir/Vix `Image.Shape.rect/3` function can draw bounding-box rectangles (returning an image) [11]. The AxonInterp example uses a color palette and draws each box onto the image with `CImg.fill_rect` [12] (Evision has equivalent functions).

Resolution, Performance, and Latency

Real-time detection imposes limits on resolution and FPS. On a Pi CM5 (quad-core Cortex-A76, 2.4 GHz, 8 GB), a full 1080p NN inference each frame is very heavy. In practice, it's common to downscale frames before inference. For example, YOLOv8 Nano on Pi4 achieves ~10–12 FPS at 480p and ~6–8 FPS at 720p [13]. The LearnOpenCV YOLO11 benchmark reports **25+ FPS at 240×240** with NCNN/quantization [4]. Thus, a typical approach is: capture high-res frames for streaming, but feed a smaller (e.g. 416×416 or 640×640) version into the NN. The detected box coordinates are then scaled back to the original image size. This preserves low-latency detection (essential for navigation) while still sending high-res video to the operator.

Other speedups include model quantization (8-bit) and using hardware acceleration. The Pi CM5's VideoCore GPU might offer some OpenCL acceleration, and one can plug in a Google Coral USB TPU for further speed (though that departs from pure Nx). Critically, compiling Nx with EXLA and turning on hardware-aware optimizations (as used in YOLO11, see **NCNN quantization** [2]) is needed to approach real-time on the Pi. In summary: expect to run at sub-10 FPS at 1080p, so targeting 720p or less for NN input is advisable.

> **Citing sources:** On Raspberry Pi hardware, downsampling greatly improves FPS. For example, YOLOv8n on Pi (via Python) achieves ~10 FPS at 640×480 and ~6–8 FPS at 720p [13]. Using NCNN with YOLO11 yielded over 25 FPS at just 240×240 frames [4], illustrating the trade-off.

References: We followed Elixir-Nx best practices and Pi-AI literature to design this setup [5] [1] [4] [6] [10] [11]. These sources cover deploying YOLO models on Pi, Elixir/ExLA inference, and drawing detection overlays.

---

[1] [3] Deploying Ultralytics YOLO models on Raspberry Pi devices - Raspberry Pi
https://www.raspberrypi.com/news/deploying-ultralytics-yolo-models-on-raspberry-pi-devices/

[2] [4] YOLO11 on Raspberry Pi: Optimizing Object Detection for Edge
https://learnopencv.com/yolo11-on-raspberry-pi/

[5] [8] [9] Video object detection in Elixir using Nx and Bumblebee | Culttt
https://culttt.com/2023/01/26/video-object-detection-elixir-nx-bumblebee

[6] [7] [12] Object Detection: YOLOv4 — axon_interp v0.1.0
https://hexdocs.pm/axon_interp/yolov4.html

[10] AxonOnnx — AxonOnnx v0.4.0
https://hexdocs.pm/axon_onnx/AxonOnnx.html

[11] Image.Shape — image v0.61.0
https://hexdocs.pm/image/Image.Shape.html

[13] Real-Time and Image-Based Object Detection on Raspberry Pi with YOLOv8 | SOSLab
https://soslab.net/articles/object-detection-on-raspberry-pi-with-yolov8-picamera2