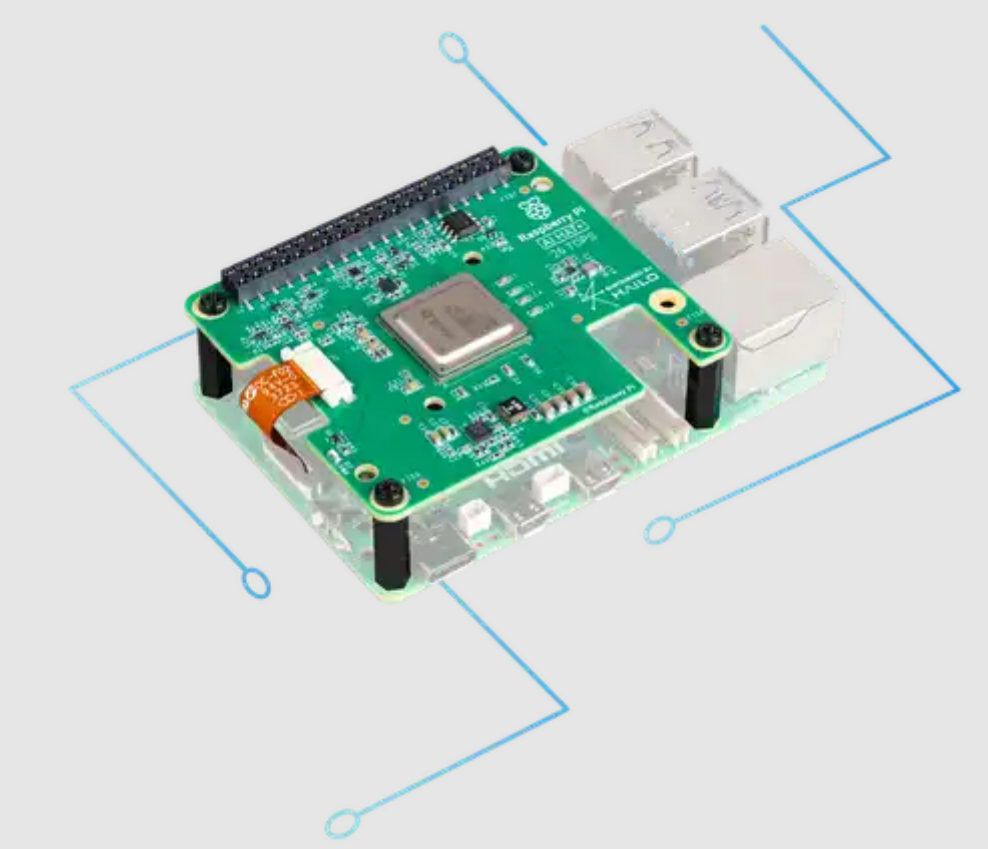
AI HAT+ y AI Camera

> Este proyecto nace de la intención de montar en mi centro una herramienta de IA de uso local que no dependa (y no envíe) datos a internet. Spoiler: No sale bien.

IA HAT+

Qué es



Es una "placa" o tarjeta de expansión que se encaja encima de una Raspberry Pi (HAT significa Hardware Attached on Top). Añade potencia extra para procesar Inteligencia Artificial localmente (sin depender de internet). Permite que la computadora reconozca objetos, caras o voces de forma mucho más rápida y eficiente de lo que podría hacerlo sola. Es, básicamente, darle un "cerebro especializado" a una computadora pequeña para que pueda realizar tareas de IA sin trabarse.

Enlaces de interés

- AI HAT+ (la que he usado para las pruebas documentadas mas abajo)
  - Página oficial: https://www.raspberrypi.com/products/ai-hat/
  - Documentación oficial: https://www.raspberrypi.com/documentation/accessories/ai-hat-plus.html
  - Resumen: https://pip-assets.raspberrypi.com/categories/1081-raspberry-pi-ai-hat/documents/RP-008133-DS-1-raspberry-pi-ai-hat-plus-product-brief.pdf?disposition=inline
- AI HAT+ 2
  - Página oficial https://www.raspberrypi.com/products/ai-hat-plus-2/
  - Proyectos de la empresa fabricante https://github.com/hailo-ai/hailo-apps

Características principales

1. Potencia de cálculo (NPU)
   Su corazón es un acelerador Hailo-8L. Dependiendo del modelo, ofrece un rendimiento de 13 o 26 TOPS (Tera Operaciones por Segundo). Esto permite procesar redes neuronales complejas (como detección de objetos en tiempo real o segmentación de imágenes) de forma fluida.
2. Conexión de alta velocidad
   Se conecta directamente al puerto PCIe 2.0 de la Raspberry Pi 5. Esto es clave porque permite que los datos viajen mucho más rápido entre el procesador principal y el chip de IA, evitando los "cuellos de botella" que tenían los modelos anteriores por USB.
3. Integración total (Plug & Play)
   Al ser un accesorio oficial, el sistema operativo Raspberry Pi OS la reconoce automáticamente. No necesitas instalar controladores complicados; las librerías de cámara estándar (como rpicam-apps) ya vienen preparadas para usar este hardware.
4. Eficiencia energética y térmica
   Está diseñada para consumir poca energía mientras trabaja a máxima potencia. Además, su formato permite que el ventilador oficial de la Raspberry Pi 5 siga funcionando debajo de ella, manteniendo todo el sistema fresco.
5. Formato HAT+
   Sigue el nuevo estándar HAT+, lo que significa que es más delgada, permite el apilamiento de otras placas y tiene una gestión de energía más inteligente que los HAT antiguos.

Versiones

En este momento, según la documentación de la web de la Raspberry nos encontramos con las siguientes versiones hardware:

Versión	Chip	Rendimiento	RAM
AI HAT+ 13 TOPS	Hailo-8L	13 TOPS	Comparte la de la Raspberry Pi.
AI HAT+ 26 TOPS	Hailo-8L	26 TOPS	Comparte la de la Raspberry Pi.
AI HAT+2	Hailo-10	40 TOPS	Tiene 8 GB dedicados en la propia placa

AI Camera

> En la documentación de este producto hacen especial hincapié en no manejarlo si tenemos riesgo de pasarle electricidad estática pues el producto es tremendamente sensible a este tipo de corrientes.

Qué es

Es un módulo de cámara oficial que combina un sensor de imagen de alta calidad (Sony IMX500) con un procesador de IA (NPU) **integrado en el mismo hardware**. Sirve para dotar de visión artificial a cualquier modelo de Raspberry Pi (desde la Zero hasta la 5) sin sobrecargar el procesador principal. Como la cámara hace el trabajo sucio de la IA, el procesador de tu Raspberry Pi queda libre para otras tareas. Puede ejecutar modelos de detección de objetos a una velocidad de fotogramas muy alta y con muy baja latencia

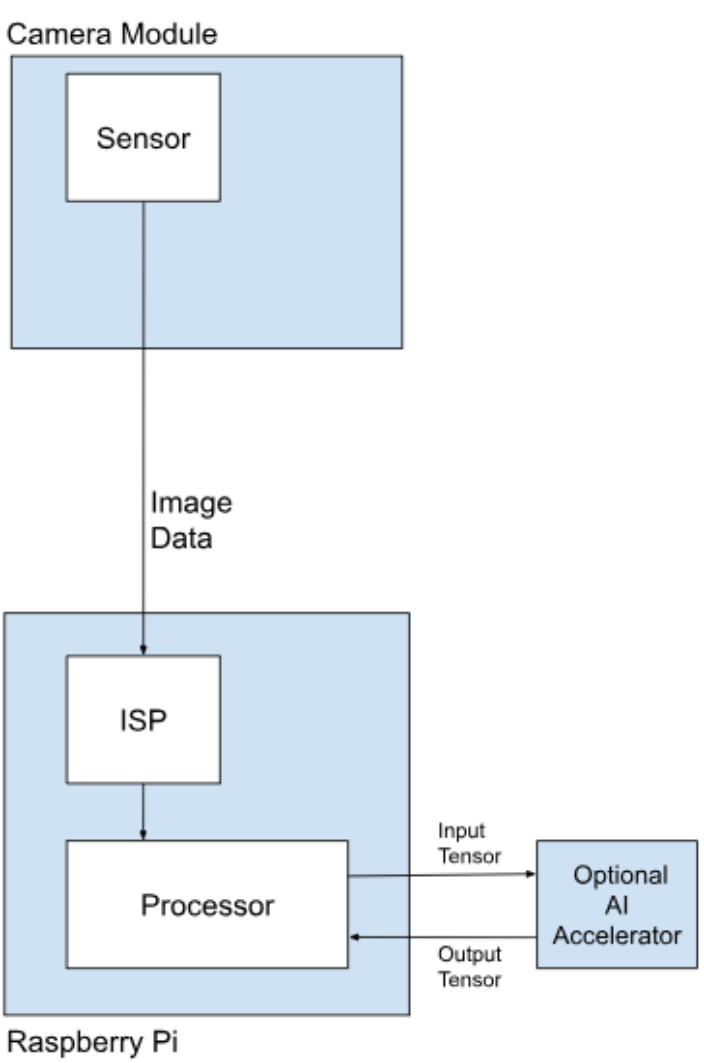
Enlaces de interés

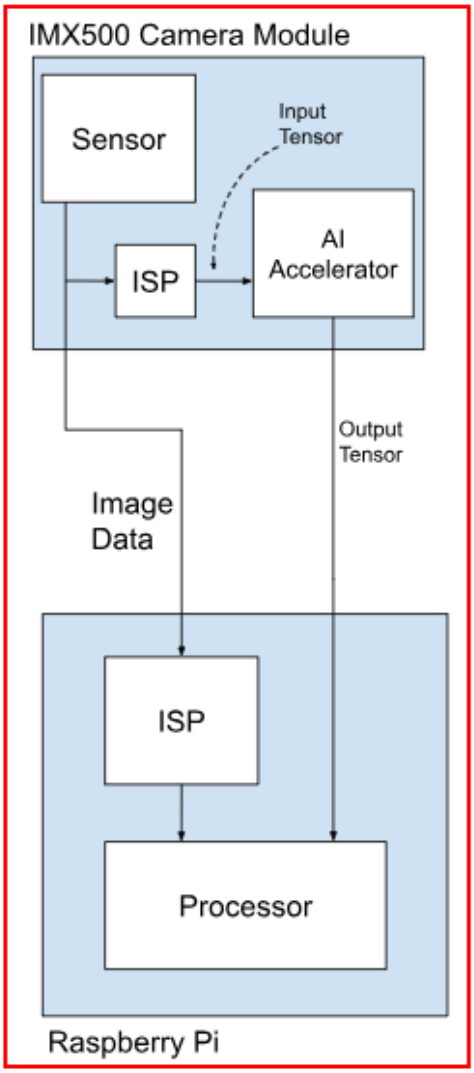
- Página oficial: https://www.raspberrypi.com/products/ai-camera/
- Documentación oficial https://www.raspberrypi.com/documentation/accessories/ai-camera.html

- Resumen: https://pip-assets.raspberrypi.com/categories/1044-raspberry-pi-ai-camera/documents/RP-008147-DS-1-ai-camera-product-brief.pdf?disposition=inline

Características principales

1. Sensor de imagen inteligente (Sony IMX500): Es el primer sensor de visión con procesamiento de IA integrado. Combina un sensor de 12.3 megapíxeles con un acelerador lógico (NPU) en el mismo chip. Esto permite capturar la imagen y analizarla en microsegundos.
2. Procesamiento "On-board" (NPU integrada)
3. Potencia: Ofrece un rendimiento de 2.3 TOPS (Tera Operaciones por Segundo): La Raspberry Pi no tiene que "pensar" para reconocer objetos; la cámara le envía directamente los metadatos (por ejemplo: "Hay una persona en estas coordenadas con un 95% de certeza").
4. Compatibilidad universal
   A diferencia de los AI HAT+ (que solo funcionan con la Raspberry Pi 5), la AI Camera se conecta mediante el puerto CSI (cámara) estándar. Esto la hace compatible con:
   Raspberry Pi 4 y 5.
   Raspberry Pi Zero / Zero 2 W.
   Modelos antiguos (3B+, etc.).
5. Resolución: 4056 x 3040 píxeles.
6. Posee enfoque **manual** ajustable, lo que permite nitidez tanto en objetos cercanos como lejanos.
7. Campo de visión (FoV): 76 grados, ideal para vigilancia y robótica estándar.
8. Baja latencia: Al procesar la IA en el sensor, no hay retrasos por el envío de datos pesados a través de cables.
9. Ahorro de energía: Consume mucho menos que un HAT de IA externo, lo que la hace perfecta para proyectos que funcionan con baterías.
10. Software "Plug & Play": Está totalmente integrada en el ecosistema oficial. Utiliza las librerías rpicam-apps, por lo que puedes ejecutar modelos de detección de objetos o segmentación con comandos muy sencillos, sin necesidad de instalar complejos entornos de programación.





Versiones

En este momento no existen versiones de este hardware. Hay que tener en cuenta que si existen otras cámaras compatibles con la raspberry pi pero estas no incorporan chips de IA y, por tanto, el trabajo deberá hacerlo la raspberry pi o una placa de expansión específica con lo que perderemos la ventaja de este chip.

Montaje

El montaje de la cámara en la Raspberry Pi 5 ha resultado muy sencillo. Se ha utilizado una pulsera anti estática para evitar los riesgos indicados por el fabricante. En la Raspberry Pi se dispone de 2 bahías dónde conectarla (la 0 y la 1). En mi caso la he conectado a la ranura 0. En la caja venían 2 cables para conectar la cámara. Uno mas ancho que otro. He tenido que utilizar el estrecho. En el

proyecto indico como verificar que está correctamente instalada.

Proyectos

Los proyectos que a continuación se indican se han realizado con el siguiente hardware:

- Raspberry Pi 5 de 16 GB de RAM
- Cargador oficial de Raspberry Pi 5
- Tarjeta microSD oficial de 128 GB (en previsión de descarga de modelo IA de gran tamaño)
- Raspberry Pi IA HAT+ en su versión de 26 TOPs
  - el término TOPs es una unidad de medida que se utiliza para cuantificar el rendimiento de los procesadores diseñados para ejecutar tareas de IA. Sus siglas significan Tera Operations Per Second (Billones de operaciones por segundo)
- Raspberry Pi IA Camera

No nos ha sido posible encontrar en el mercado la versión AI HAT+2 (40 TOPs y memoria dedicada). Probablemente llevándose el desarrollo de la práctica principal a ese hardware se hubiera cumplido el objetivo inicial.

Todo ensablado queda como se ve en la siguiente imagen:

(TO DO)

Creación de un LLM de uso local sin dependencia de internet

Como ya se adelantó anteriormente, la idea era montar una IA generativa en local que nos asegurase el no envío de nuestros datos de trabajo al exterior. A continuación enumero los diferentes pasos realizados. Comenzamos:

1. Con Raspberry Pi imager cargo en la SD la imagen del sistema operativo para la raspberry pi 5
2. Actualizamos sistema y borramos paquetes innecesarios
   1. Ejecutamos en terminal `sudo apt update && sudo apt upgrade -y && sudo apt autoremove -y && sudo apt clean`
3. Instalamos Ollama
   1. Nos va a permitir gestionar modelos de lenguaje
   2. Web de Ollama con información https://ollama.com/

3. Ejecutamos en terminal `curl -fsSL https://ollama.com/install.sh | sh`
4. Tras instalarlo merece la pena fijarse en 3 líneas que arroja el instalador:
    1. `The Ollama API is now available at 127.0.0.1:11434` Nos indica dónde ha montado por defecto un servidor al cual lanzarle peticiones
    2. `Install complete. Run "ollama" from the command line.` Nos indica cual es el comando que deberamos usar para trabajar con Ollama
    3. `WARNING: No NVIDIA/AMD GPU detected. Ollama will run in CPU-only mode.` Aquí ya nos advierte de que el funcionamiento de Ollama no va a ser óptimo
4. Descargamos un modelo para trabajar con él
    1. Enumero los que en este momento (abril de 2026) corren en el hw con el que contamos:
        1. llama3.2:3b (rápido y ligero)
        2. mistral:7b (más capaz, mas lento)
        3. phi3:mini (muy eficiente para su tamaño)
        4. gemma2:2b (excelente relación calidad/velocidad)
    2. Instalamos 1 de ellos. Los demás se haría de modo análogo
        1. Ejecutamos en el terminal `ollama pull phi3:mini`
5. Lo probamos a través de una llamada al API REST (documentación del endpoint que vamos a usar https://docs.ollama.com/api/generate)
    1. Escribimos en el terminal:
    2. `curl http://127.0.0.1:11434/api/generate -d '{ "model": "phi3:mini", "prompt": "Explica qué es una raspberry pi", "stream": false }'`
    3. Y esperamos por aproximadamete **4 minutos** para ver el resultado. Luego con este modelo, no es funcional
6. Vamos a descargar otro modelo mas ligero y repetir la consulta para ese modelo:
    1. `ollama pull llama3.2:3b`
    2. `curl http://127.0.0.1:11434/api/generate -d '{ "model": "llama3.2:3b", "prompt": "Explica qué es una raspberry pi", "stream": false }'` en esta ocasión la respuesta se ha demorado **2 minutos** lo cual sigue haciendo, bajo mi punto de vista, al modelo poco eficiente como sustituto de las IAs comerciales.
7. Vamos a preparar una interfaz web para quienes no quieran usar el terminal.
    1. Uaremos docker, si no lo tienes instalado, tines que instalarlo:
        1. Consulta la documentación en web oficial para hacerlo. Yo te dejo a continuación lo que en la fecha actual he hecho yo para instalarlo
        2. `sudo apt remove $(dpkg --get-selections docker.io docker-compose docker-doc podman-docker containerd runc | cut -f1)`
        3.

```
# Add Docker's official GPG key:
sudo apt update
sudo apt install ca-certificates curl
```

```
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
sudo tee /etc/apt/sources.list.d/docker.sources <<EOF
Types: deb
URIs: https://download.docker.com/linux/debian
Suites: $(. /etc/os-release && echo "$VERSION_CODENAME")
Components: stable
Architectures: $(dpkg --print-architecture)
Signed-By: /etc/apt/keyrings/docker.asc
EOF

sudo apt update
```

4. `sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin`
5. `sudo systemctl status docker`
6. `sudo groupadd docker` quizás te diga que ya existe
7. `sudo usermod -aG docker $USER`
8. `sudo groupadd docker`
9. `sudo systemctl enable containerd.service`
10. Reiniciamos

2. Ahora ejecutaremos en el terminal `docker run -d -p 3000:8080 --add-host=host.docker.internal:host-gateway -e OLLAMA_BASE_URL=http://host.docker.internal:11434 ghcr.io/open-webui/open-webui:main`

3. Si ahora accedemos en el navegador a http://localhost:3000 tendremos acceso a una interface web para interactuar con la IA que hemos preparado y/o con openIA

En las llamadas a curl hemos utilizado 3 parámetros:

- model: Obligatorio. Nombre del modelo a usar.
- promt: Obligatorio. Texto de entrada.
- stream: `true` devuelve tokens uno a uno, `false` espera respuesta completa

Puede resultar interesante conocer estos otros:

- system: Define el comportamiento/personalidad del modelo
- options
  - temperature: valores entre 0.0 y 1.0. Recomendado 0.7. Determinista y precio 0.0. Creativo y variado 1.0
  - top_p: diversidad del vocabulario. Recomendado 0.9
  - top_k: limita el número de tokens candidatos en cada paso
  - num_predict: máximo de tokens a generar en la respuesta (-1: sin límite)
  - num_ctx: tamaño de la ventana de contexto (memoria del modelo). Cuanto mas alto mas conversación recuerda pero uso mas RAM.
  - repeat_penalty: penaliza repetir las mismas palabras (Sin penalización 1.0. Penalización alta 1.3)
  - seed: fija la semilla aleatoria. Útil para reproducir exactamente la misma respuesta
- format: formato de salida. "jason" obliga al modelo a responder en un JSON válido
- keep_alive: tiempo que el modelo permanece cargado en RAM tras la llamada (5 minutos 5m. Descarga inmediata 0) Mas permanencia, mas consumo de RAM.

Aquí un ejemplo completo:

```
{
  "model": "llama3.2:3b",
  "prompt": "Explica qué es una Raspberry Pi",
  "stream": false,

  "system": "Eres un asistente experto en electrónica",

  "options": {
    "temperature": 0.7,
    "top_p": 0.9,
    "top_k": 40,
    "num_predict": 512,
    "num_ctx": 2048,
    "repeat_penalty": 1.1,
    "seed": 42
  },

  "format": "json",
  "keep_alive": "5m"
}
```

La documentación completa de la API la tenemos disponible en https://docs.ollama.com/api/introduction Además de /api/generate puede resultarte interesante probar /api/chat

Tratando de optimizar el funcionamiento

Tras buscar información en internet he encontrado que al hacer uso de ollama para correr un LLM no estamos realmente haciendo uso del HAT y que el HAT en si (esta versión del HAT) no está pensada para correr LLMs sino para ejecutar modelos relacionados con reconocimiento de imagen y audio así que descarto esa función y vamos a focalizarnos en los 2 siguientes

AI HAT+ sin cámara: procesamiento de vídeo

Vamos a tratar de detectar elementos de un vídeo que descarguemos de internet

> Para resumir lo hecho en 8 pasos he realizado multitud de pruebas e interacciones con la IA para conseguir resultados

1. Actualizamos el sistema `sudo apt update && sudo apt upgrade -y`
2. Instalamos las bibliotecas del HAT `sudo apt install hailo-all -y`
3. Reiniciamos `sudo reboot`
4. Verificamos instalación `python3 -c "from hailo_platform import VDevice; print('HailoRT OK')"`
5. Instalamos dependencias: `pip3 install opencv-python numpy --break-system-packages`
6. Creamos el script `nano ~/detector_trafico.py`
    1. ```
       import cv2
       import numpy as np
       from hailo_platform import VDevice, HEF, HailoStreamInterface, InferVStreams,
       ConfigureParams, InputVStreamParams, OutputVStreamParams, FormatType
       import time

       HEF_PATH = "/usr/share/hailo-models/yolov8s_h8.hef"
       VIDEO_PATH = "/home/pi/trafico2.mp4"
       OUTPUT_PATH = "/home/pi/trafico2_out.mp4"

       COCO_LABELS = [
       ```
       ```

```
    "person","bicycle","car","motorcycle","airplane","bus","train","truck",
    "boat","traffic light","fire hydrant","stop sign","parking meter","bench",

"bird","cat","dog","horse","sheep","cow","elephant","bear","zebra","giraffe",

"backpack","umbrella","handbag","tie","suitcase","frisbee","skis","snowboard",
    "sports ball","kite","baseball bat","baseball
glove","skateboard","surfboard",
    "tennis racket","bottle","wine glass","cup","fork","knife","spoon","bowl",
    "banana","apple","sandwich","orange","broccoli","carrot","hot dog","pizza",
    "donut","cake","chair","couch","potted plant","bed","dining table","toilet",
    "tv","laptop","mouse","remote","keyboard","cell phone","microwave","oven",
    "toaster","sink","refrigerator","book","clock","vase","scissors","teddy
bear",
    "hair drier","toothbrush"
]

COLORS = [(0,255,0),(0,0,255),(255,0,0),(0,255,255),(255,0,255),(255,255,0)]

def letterbox(img, new_shape=(640, 640)):
    h, w = img.shape[:2]
    scale = min(new_shape[0]/h, new_shape[1]/w)
    new_w, new_h = int(w * scale), int(h * scale)
    resized = cv2.resize(img, (new_w, new_h))
    top = (new_shape[0] - new_h) // 2
    bottom = new_shape[0] - new_h - top
    left = (new_shape[1] - new_w) // 2
    right = new_shape[1] - new_w - left
    padded = cv2.copyMakeBorder(resized, top, bottom, left, right,
cv2.BORDER_CONSTANT, value=(114,114,114))
    return padded, scale, left, top

def run_detection():
    cap = cv2.VideoCapture(VIDEO_PATH)
    if not cap.isOpened():
```

```
        print(f"Error: No se pudo abrir {VIDEO_PATH}")
        return

    fps = cap.get(cv2.CAP_PROP_FPS) or 30
    w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    out = cv2.VideoWriter(OUTPUT_PATH, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w,
h))
    print(f"Procesando vídeo: {w}x{h} @ {fps} FPS")

    with VDevice() as target:
        hef = HEF(HEF_PATH)
        configure_params = ConfigureParams.create_from_hef(hef,
interface=HailoStreamInterface.PCIe)
        network_groups = target.configure(hef, configure_params)
        network_group = network_groups[0]
        network_group_params = network_group.create_params()

        input_info = hef.get_input_vstream_infos()[0]
        input_h, input_w = input_info.shape[0], input_info.shape[1]

        input_vstreams_params = InputVStreamParams.make(network_group,
format_type=FormatType.UINT8)
        output_vstreams_params = OutputVStreamParams.make(network_group,
format_type=FormatType.FLOAT32)

        frame_count = 0
        t_start = time.time()

        with InferVStreams(network_group, input_vstreams_params,
output_vstreams_params) as infer_pipeline:
            with network_group.activate(network_group_params):
                while cap.isOpened():
                    ret, frame = cap.read()
                    if not ret:
```

```
                        break

                    lb, scale, pad_left, pad_top = letterbox(frame, (input_h,
input_w))
                    rgb = cv2.cvtColor(lb, cv2.COLOR_BGR2RGB)
                    input_data = {input_info.name:
np.expand_dims(rgb.astype(np.uint8), axis=0)}

                    raw_detections = infer_pipeline.infer(input_data)

                    for output_name, val in raw_detections.items():
                        batch0 = val[0]
                        for cls_id, class_dets in enumerate(batch0):
                            dets = np.array(class_dets)
                            if dets.ndim == 2 and len(dets) > 0:
                                for d in dets:
                                    conf = float(d[4])
                                    if conf > 0.4:
                                        # Formato: [y1, x1, y2, x2] normalizado
                                        x1 = int((d[1] * input_w - pad_left) /
scale)
                                        y1 = int((d[0] * input_h - pad_top) /
scale)
                                        x2 = int((d[3] * input_w - pad_left) /
scale)
                                        y2 = int((d[2] * input_h - pad_top) /
scale)
                                        # Clamp para evitar coordenadas fuera de
imagen
                                        x1, x2 = max(0, x1), min(w, x2)
                                        y1, y2 = max(0, y1), min(h, y2)
                                        label = COCO_LABELS[cls_id] if cls_id <
len(COCO_LABELS) else f"ID:{cls_id}"
                                        color = COLORS[cls_id % len(COLORS)]
                                        cv2.rectangle(frame, (x1, y1), (x2, y2),
```

```
color, 2)
                                        cv2.putText(frame, f"{label}
{conf:.2f}", (x1, y1 - 5),
                                                    cv2.FONT_HERSHEY_SIMPLEX,
0.5, color, 2)

                    out.write(frame)
                    frame_count += 1
                    if frame_count % 30 == 0:
                        elapsed = time.time() - t_start
                        print(f"Frame {frame_count} | FPS:
{frame_count/elapsed:.1f}")

    cap.release()
    out.release()
    elapsed = time.time() - t_start
    print(f"\n COMPLETADO")
    print(f"Frames: {frame_count} | Tiempo: {elapsed:.1f}s | FPS:
{frame_count/elapsed:.1f}")
    print(f"Guardado en: {OUTPUT_PATH}")

if __name__ == "__main__":
    run_detection()
```

7. Ejecutamos python3 ~/detector_trafico.py
   1. Necesitarás tener un vídeo llamado trafico.mp4 en /home/pi o cambiar la ruta
   2. El resultado lo verás en trafico_out.mp4

Abro simultáneamente el vídeo original y el generado con la detección de objetos y este es el resultado:

En la imagen se puede ver que detecta coches y la probabilidad con la que "cree" que son coches. En el vídeo también aparecen personas, motocicletas, camiones,... y todas ellas las va detectando mientras varía el porcentaje de seguridad (1: implica seguridad plena y 0: nada seguridad)

> El proceso de poner este proyecto en marcha me ha llevado aproximadamente 5h y ha sido poco intuitivo y por momentos parecía orientado al fracaso.

> En estas 5h en que la raspberry pi ha estado trabajando junto con el hat ninguna de las placas se ha sobrecalentado mas allá de lo razonable y la sensación de fluided del dispositivo ha sido total.

AI HAT+ y AI cámara

Lo primero ha sido verificar que la cámara estaba correctamente instalada, para ello:

```
sudo apt update
sudo apt install libcamera-apps
rpicam-hello --list-cameras
```

Lo que en mi caso ha devuelto:

```
Available cameras
-----------------
0 : imx500 [4056x3040 10-bit RGGB] (/base/axi/pcie@1000120000/rp1/i2c@88000/imx500@1a)
    Modes: 'SRGGB10_CSI2P' : 2028x1520 [30.02 fps - (0, 0)/4056x3040 crop]
                             4056x3040 [10.00 fps - (0, 0)/4056x3040 crop]
```

Lo que significa que ha encontrado la cámara conectada, su tipo y sus características.

Si quieres ver qué se está viendo por la cámara dispones del comando `rpicam-hello -t 0` mientras que si quieres hacer una foto puedes usar `rpicam-still -o foto.jpg`

Ahora comenzaremos con la **parte de IA** de la cámara:

```
sudo apt update
sudo apt install imx500-all
```

En `/usr/share/rpi-camera-assets/` se habrán guardado los ficheros .json de demostración que incorpora el paquete.

Podemos probar diferentes modos de funcionamiento con `rpicam-hello -t 0 --post-process-file /usr/share/rpi-camera-assets/imx500_mobilenet_ssd.json` y con `rpicam-hello -t 0 --post-process-file /usr/share/rpi-camera-assets/imx500_posenet.json` Los comandos anteriores funcionarán respectivamente para la detección de objetos y la detección de esqueleto (brazos, piernas y tronco)

Revision #20
Created 2026-03-28 09:30:53 CET by Pablo Ruiz
Updated 2026-04-14 16:44:05 CEST by Pablo Ruiz