INFORME

PRACTICA PROFESIONAL SUPERVISADA

Estudiante: Bautista Garcia

Supervision: Cesar Estrebou

Inferencia Optimizada en Microcontroladores con NPU: Caso RV1103

Universidad Nacional de La Plata
Facultad de Informatica e Ingenieria

Universidad Nacional de La Plata
Facultad de Informatica e Ingenieria

Contents

1 Analisis de Placa de Desarrollo

El Luckfox Pico Mini es un módulo de cómputo de bajo costo diseñado para tareas de inteligencia artificial en el borde (edge computing), especialmente en aplicaciones donde el tamaño, el consumo energético y la latencia son factores determinantes. En el mismo se encuentra el SoC (System on Chip) RV1103, este SoC cuenta con: un procesador ARM Cortex-A7, una unidad de procesamiento neuronal (NPU), un coprocesador RISC-V de ultra bajo consumo y un conjunto de interfaces de entrada/salida diseñadas para aplicaciones embebidas.

Processor	Cortex A7@1.2GHz + RISC-V
NPU	0.5TOPS, supports int4, int8 and int16
ISP	Input 4M @30fps (Max)
Memory	64MB DDR2
USB	USB 2.0 Host/Device
Camera	MIPI CSI 2-lane
GPIO	24 × GPIO pins
Ethernet port	10/100M Ethernet controller and embedded PHY
Default Storage	TF card (Not included)

Figure 1: Especificaciones

1.1 Arquitectura

El procesador principal, basado en la arquitectura ARMv7-A de 32 bits y opera a una frecuencia de aproximadamente 1.2 GHz. Esta CPU está equipada con unidades SIMD NEON y una FPU (unidad de punto flotante) que permiten ejecutar cálculos vectorizados de manera más eficiente que en microcontroladores tradicionales. Asimismo al tratarse de una arquitectura ARMv7 se reduce la compatibilidad con librerías modernas, ya que muchos frameworks actuales como PyTorch ya no mantienen versiones precompiladas para armhf (linux-armv7l). En casos como **TF-Lite** es el usuario el que debe realizar el build de las librerias para su uso en ARMv7 (lo que lleva mayor tiempo de desarrollo).

1.2 NPU

El modulo NPU (Neural Processing Unit), es una unidad dedicada exclusivamente a acelerar operaciones de inferencia típicas en modelos de aprendizaje profundo. Esta NPU, perteneciente a la cuarta generación de aceleradores de Rockchip, es capaz de alcanzar un rendimiento de hasta 0.5 TOPS (INT8) o incluso 1 TOPS utilizando cuantización más agresiva como INT4 (poco precisa). La arquitectura de la NPU está diseñada para operar en paralelo, delegando a la CPU preprocesamiento y postprocesamiento de los datos mientras que la NPU se encarga de la cuantizacion y la inferencia en el modelo.

Para facilitar el desarrollo y despliegue de modelos en esta plataforma, Rockchip provee el RKNN Toolkit 2, un conjunto de herramientas que permite convertir modelos desde frameworks populares como TFLite, ONNX o PyTorch al formato **`.rknn`** compatible con la NPU, realizar cuantizaciones a las versiones con soporte (INT4 e INT8) y herramientas de debugging y profiling. Este toolkit también incluye APIs en C y Python que permiten realizar la inferencia a partir del modelo `.rknn` como fuente.

1.3 Entrada y Salida

En términos de entrada/salida, el Luckfox Pico incluye interfaces como MIPI-CSI para conexión de cámaras (soporta sensores de hasta 4 MP a 30 fps), puertos USB 2.0 OTG, GPIOs y opciones de conectividad como Ethernet en algunas variantes. Además, el SoC cuenta con un procesador de señales de imagen (ISP 3.2) integrado, que permite realizar operaciones como HDR, reducción de ruido o balance de blancos directamente en hardware, util para tareas de visión artificial embebida para no delegar esas tareas a la CPU.

1.4 Memoria

La placa cuenta con 64 MB de memoria DDR2, la cual es volatil. Esto significa que para disponer de un OS con herramientas basicas (librerias y entornos de runtime) debemos contar con una memoria micro-SD adicional con los elementos de booting.

En resumen, el análisis del Luckfox Pico demuestra que, si bien existen desafíos técnicos asociados al entorno de desarrollo embebido y a la arquitectura ARMv7, estos pueden ser superados con configuraciones cuidadosas y uso eficiente de las herramientas provistas. Esto habilita una amplia gama de aplicaciones en campos como visión embebida, IoT

inteligente, monitoreo ambiental y dispositivos portátiles, donde el procesamiento local con IA es cada vez más demandado.

2 Buildroot en RV1103

El desarrollo de aplicaciones para sistemas embebidos requiere habitualmente la configuración de un entorno personalizado que se adapte a las limitaciones de hardware y a las necesidades específicas de cada plataforma. En este proyecto se utiliza Buildroot, este permite compilar el código fuente un sistema Linux completamente adaptado a la arquitectura y capacidades del hardware **`rv1103`** incluyendo bibliotecas de runtime y herramientas de desarrollo. Este ya incluye los drivers necesarios para hacer target a la **NPU**, las bibliotecas de **C runtime** y herramientas de **profiling**.

Se opto por buildroot en lugar de las versiones reducidas de Ubunutu, ya que tiene un tiempo de booting mas rapido y cuenta con una menor cantidad de archivos base del OS, lo cual da mas espacio a los pesos de los modelos a cargar (suelen ocupar bastante memoria).

2.1 Instalación en la Luckfox Pico RV1103

Como ya se menciono, la placa de desarrollo no cuenta con memoria flash interna (SPI-NAND), por lo que el sistema operativo debe ser provisto mediante una tarjeta micro-SD.

El primer paso consiste en la configuración del entorno de desarrollo, lo cual se recomienda realizar dentro de un contenedor Docker. Esto asegura la compatibilidad con el SDK oficial, el cual ha sido probado principalmente sobre sistemas basados en Ubuntu 22.04. En entornos Windows se recomienda utilizar WSL2 para una mayor compatibilidad. El siguiente fragmento muestra cómo preparar el entorno de desarrollo:

NOTA: *El procedimiento completo junto con algunos bugs y errores que se fueron encontrando al preparar dicha tarjeta pueden consultarse en detalle en la* ***documentación adicional*** *incluida en el proyecto.*

```
# Iniciar contenedor con imagen oficial que cuenta con los paquetes
preinstalados.
docker pull luckfoxtech/luckfox_pico:1.0
docker run -it --name luckfox luckfoxtech/luckfox_pico:1.0 /bin/bash

cd /home
git clone https://github.com/LuckfoxTECH/luckfox-pico.git
cd luckfox-pico
```

Una vez dentro del contenedor, se puede proceder a compilar Buildroot. Aqui se deben elegir las caracteristicas correspondientes a la placa a utilizar.

```
# Exportar entorno
source tools/linux/toolchain/arm-rockchip830-linux-uclibcgnueabihf/
env_install_toolchain.sh

# (Opcional) Limpiar builds previos
./build.sh clean

# Seleccionar configuración (ej.: SD-CARD, RV1103)
./build.sh lunch

# Iniciar compilación
./build.sh
```

Finalizada la compilación, se obtienen imágenes de sistema (.img) listas para ser grabadas en una tarjeta micro-SD. Para ello, se utiliza la herramienta **Soc Toolkit**, provista también por el fabricante. Es importante tener en cuenta que esta herramienta no reconoce lectores de tarjetas internos; se requiere el uso de un adaptador USB ↔ micro-SD. Además, es necesario ejecutarla en modo administrador para garantizar el acceso a bajo nivel al dispositivo de almacenamiento. Estos pasos están descritos en detalle en la documentacion mencionada.

2.2 Conexion a la placa

Una vez grabada la imagen, la tarjeta puede insertarse en la placa y se puede establecer una conexión con el dispositivo a través de un cable USB-C con soporte Ethernet virtual. Esto permite conectarse vía SSH a la dirección IP asignada por defecto, utilizando las credenciales predeterminadas:

```
ssh root@172.32.0.93
# o alternativamente:
ssh pico@172.32.0.70
# Contraseña: luckfox
```

Este entorno base proporciona una plataforma funcional sobre la cual se pueden desplegar modelos optimizados para la NPU y realizar las pruebas de rendimiento necesarias. La

correcta construcción del sistema mediante Buildroot garantiza una integración limpia, mínima y eficiente, alineada con los objetivos del presente proyecto.

3 Conversion de modelos

La NPU del RV1103 sólo ejecuta modelos compilados en su formato propietario `.rknn`. Por esta razón, todo modelo entrenado usando librerias de alto nivel: PyTorch, TensorFlow, TensorFlow-Lite u ONNX debe atravesar un proceso de conversion que adapte su grafo de ejecucion y sus tensores a los requisitos del compilador de Rockchip. El flujo se realiza usando la herramienta **RKNN-Toolkit 2**, que ofrece una API de Python **`rknn.api`** para cargar el modelo original, configurar sus parámetros de entrada, cuantizar los pesos y compilar el modelo a un formato ejecutable por la NPU.

3.1 Cuantizacion

El procedimiento parte de un modelo en punto flotante. Mediante las funciónes **`rknn.load_onnx()`**o **`rknn.load_pytorch()`** (para mas informacion sobre librerias disponibles ver **documentacion oficial**) se analiza la red, se resuelven operadores y se aplica un layout de tensores compatible con la arquitectura del hardware. A continuación se invoca **`rknn.config()`**, donde se especifica la plataforma destino (rv1103) y parametros como la media y varianza del set de entrenamiento para usar en la cuantizacion. La NPU del RV1103 es más eficiente cuando opera con enteros de 8 bits (**`int8`**) y alcanza su mayor rendimiento con 4 bits (**int4**); en ambos casos los datos se representan en escalas enteras y ceros de calibración (asymmetric_quantized-8, dynamic_fixed_point-4). Mas alla de que con **int4** se logre el mejor rendimiento, *hasta 1 TOPS (*10^{12} *operaciones x segundo)*, existe una perdida muy grande de precision. Es por esto que en los modelos probados se opto por la cuantizacion **int8**.

Para minimizar este error se emplea cuantización post-entrenamiento basada en calibración. El compilador recorre la red con un conjunto representativo de entradas y registra los rangos dinámicos de cada activación. Con esa estadística construye las tablas de escala y zero-points que mapearán los valores flotantes al espacio entero. *Para el modelo probado (Mobilenet) se realizo un script que arma un dataset de calibracion con 100 imagenes de diversas fuentes (Unsplash, wikimedia, …) para realizar la cuantizacion post-entrenamiento.* La cuantizacion del modelo se realiza con la funcion **`rknn.build(do_quantization=True, dataset=calibracion.txt)`** (en calibracion.txt se deben encontrar las rutas relativas al dataset de calibracion).

Por ultimo utilizando la funcion **`rknn.export_rknn(args.output)`** crea el `.rknn` aceptado por la `rknn C runtime API`. Este va a ser uno de los archivos que se deben enviar a la placa de desarrollo (ej: usando scp).

4 Inferencia con RV1103 (NPU)

La inferencia sobre la NPU del RV1103 se realiza a través de un script en C que emplea la API de bajo nivel del RKNN Toolkit. Este script está dividido en tres etapas principales:
1. Inicialización del runtime y prealocacion de buffers.
2. Inferencia.
3. Postprocesamiento e interpretacion de los resultados.

En la **etapa de inicialización**, se carga el modelo previamente convertido al formato `.rknn` y se consulta su estructura de entrada y salida utilizando `rknn_query`. Esta etapa también incluye la configuración de memoria mediante la API de zero-copy (`rknn_create_mem` y `rknn_set_io_mem`).

En la **etapa de ejecución** se invoca a la función `rknn_run`, que ejecuta el modelo directamente sobre la NPU utilizando los tensores de entrada previamente cargados. Gracias a la arquitectura especializada del RV1103 y al uso de cuantización (int8 o uint8), esta etapa se completa en apenas unos milisegundos en comparacion a su modelo analogo ejecutado en CPU.

La etapa de post-procesamiento consiste en convertir los tensores de salida (que pueden venir en formatos como NC1HWC2 cuantizados en int8) a un arreglo en punto flotante para su análisis. Esto requiere aplicar la desnormalización (con scale y zero-point) y una operación de softmax, seguida de una ordenación para obtener las clases más probables (top-k).

4.1 Ventajas de Zero-Copy

Esta interfaz elimina la necesidad de copiar datos desde la memoria del usuario hacia la memoria interna de la NPU durante cada ejecución: en cambio, utiliza memoria preasignada —ya sea proporcionada por el usuario con dirección física o por un subsistema como DRM—, lo que reduce significativamente el uso de CPU, el ancho de banda de memoria y la latencia total de inferencia.

Ademas, en este modelo de ejecución, todo el preprocesamiento de las entradas (cuantizacion, normalizacion, etc) se realiza directamente en la NPU, siempre que se cumplan ciertos alineamientos.

Esto se hace a traves de funciones como `rknn_create_mem` y `rknn_set_io_mem`, que permiten gestionar buffers de entrada y salida compartidos con la NPU de forma eficiente, cumpliendo con los requisitos del flujo de datos optimizado para RV1103.

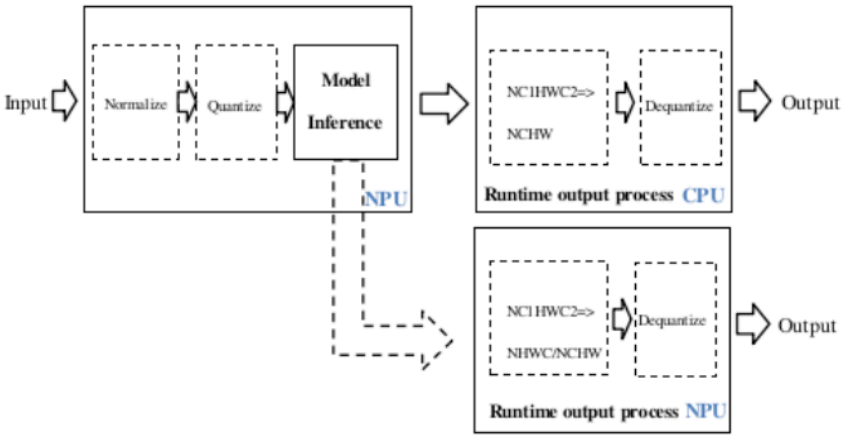

Figure 2: Datos en Zero copy API

4.2 Optimización y Limitaciones

Si bien el rendimiento de la inferencia en la NPU es muy superior al de una CPU, existen ciertas limitaciones impuestas por el hardware y el soporte de la API. Algunas operaciones avanzadas como **flash-attention** (usada en modelos SOTA) o la posibilidad de usar la API de `matmul` (que si estan soportadas para la gama de procesadores `rk` mas nuevos) no estan soportadas en el rv1103, lo que nos quita la posibilidad de optimizar aun mas los modelos y adaptarlos exclusivamente a las operaciones utilizadas.

Optimizaciones Futuras:

- Fusión de operaciones para reducir la cantidad de kernels lanzados a la NPU.
- Cuantizacion usando `int4` con la posibilidad de lograr **1 TOPS** teorico mencionado en la datasheet.

5 Inferencia con RV1103 (NEON ARMv7)

El RV1103 cuenta con un núcleo ARM Cortex-A7 compatible con el conjunto de instrucciones NEON, lo cual permite realizar inferencia directamente sobre CPU utilizando bibliotecas optimizadas como TensorFlow Lite (TFLite). Esta alternativa resulta útil cuando el modelo aún no es compatible con la NPU, durante etapas de validación o debugging, o simplemente para comparar rendimiento entre diferentes targets de ejecución (motivo propio de uso).

Para ello se utilizó TensorFlow Lite en su versión C (tensorflow/lite/c/c_api.h), compilado específicamente para la arquitectura armv7hf mediante un cross-compiler proporcionado por Rockchip. Este proceso de compilación cruzada permitió generar una versión estatica de la librería `.a`, sin dependencias adicionales.

El modelo utilizado fue MobileNetV2 en formato `.tflite` con pesos en float32, cargado directamente en memoria a través de la función `TfLiteModelCreateFromFile`. La entrada esperada es una imagen RGB cruda de 224 × 224 píxeles, codificada en uint8. Esta imagen es cargada desde disco, normalizada al rango $[-1.0, 1.0]$ y convertida a float32 en formato NHWC (Height × Width × Channels), compatible con la entrada del modelo.

La configuración al compilar fue **Single-Threaded**, adecuado para un procesador mononúcleo como el Cortex-A7.

- `TfLiteInterpreterOptionsSetNumThreads(options, 1)`

Esta implementación demuestra que incluso sin aceleración por hardware, es posible realizar inferencia eficiente en dispositivos embebidos, aprovechando extensiones SIMD como NEON y una versión optimizada de TensorFlow Lite. Además, ofrece una base sólida para pruebas funcionales de modelos y algoritmos en etapa temprana, sin depender de las herramientas de conversión al entorno de la NPU.

6 Benchmarks & Profiling

Para evaluar el rendimiento real de la placa RV1103 en tareas de inferencia, se ejecutó el modelo MobileNetV2 —con aproximadamente 300 millones de operaciones MAC por inferencia tanto en la CPU (Cortex-A7 con soporte NEON) como en la NPU integrada. Se realizaron cinco ejecuciones consecutivas en cada caso, manteniendo constante el modelo, la imagen de entrada y la lógica de pre/postprocesamiento.

6.1 Resultados

Run	CPU Total (µs)	CPU Inference (µs)	NPU Total (µs)	NPU Inference (µs)
1	1853192	1510135	67459	14865
2	1777080	1506761	67302	14999
3	1681110	1446135	67627	15087
4	1632595	1450338	68003	14940
5	1707828	1492757	67402	14931
Promedio	1738361	1481225	67558	14964

Los resultados obtenidos muestran una diferencia de rendimiento muy marcada entre ambos entornos de ejecución. La CPU, utilizando una versión `float32` del modelo y la API C de TensorFlow Lite, alcanzó un tiempo promedio de inferencia de 1.48 segundos por imagen, mientras que la NPU completó la misma tarea en apenas 14.9 milisegundos en promedio. Esta diferencia representa una aceleración de más de 99% en favor de la NPU, lo que la convierte en la opción preferida para despliegues en tiempo real.

Al calcularse los TOPS efectivos (Tera Operaciones Por Segundo), los valores medidos distan significativamente de los 0.5 TOPS teóricos que se publicitan para la NPU del RV1103. En estas pruebas, se alcanzaron apenas 0.0401 TOPS reales, es decir, alrededor del 8% del rendimiento pico declarado. Este desfasaje puede explicarse por múltiples factores: en primer lugar, la eficiencia del uso de hardware depende fuertemente del diseño interno del modelo y su compatibilidad con las operaciones nativamente soportadas por

la NPU. En segundo lugar, la ausencia de acceso público a los controladores internos de la NPU limita las posibilidades de realizar optimizaciones finas que permitan llegar a un 100% de performance.

$$\text{GOPS} = \frac{\text{MAC} * 2}{\text{Inference Time} * (10^9)}$$

$$\text{GFLOPS}_{\text{CPU}} = 0.405$$

$$\text{GOPS}_{\text{NPU}} = 40.1$$

En contrapartida, la ejecución en CPU ofreció una ventaja en términos de precisión numérica, ya que se utilizó un modelo en punto flotante de 32 bits (FP32) sin cuantización, lo que puede resultar relevante en aplicaciones donde los detalles finos de las predicciones son críticos. Sin embargo, esto se logra a costa de un rendimiento sustancialmente inferior y un mayor consumo energético por inferencia.

Por último, si bien el framework RKNN ofrece funciones de profiling detallado como `rknn_eval_perf()` que permiten analizar los tiempos de cada capa del modelo, dicha funcionalidad no está disponible para la serie RV1103, lo que impidió realizar un análisis detallado del desempeño interno de cada operador sobre la NPU. Esto limita el alcance del diagnóstico y obliga a depender únicamente de métricas externas (tiempos globales y consumo) para evaluar el rendimiento.

7 Conclusion

El presente estudio permitió comparar de forma cuantitativa el rendimiento y comportamiento de inferencia de un modelo MobileNetV2 sobre las dos unidades de cómputo disponibles en la placa RV1103: la CPU ARM Cortex-A7 y la NPU dedicada. Los resultados demostraron una ventaja abrumadora de la NPU en términos de velocidad, con tiempos de inferencia más de 90 veces menores y consumos estimados significativamente reducidos.

A pesar de esto, los TOPS reales alcanzados por la NPU distan del valor teórico publicitado (0.5 TOPS), lo que evidencia oportunidades de mejora en la eficiencia del stack software y en el acceso a herramientas de optimización más avanzadas. Por otro lado, la ejecución sobre CPU permitió validar la funcionalidad del modelo sin cuantización, aprovechando mayor precisión numérica a costa de una notable pérdida de rendimiento.

En conjunto, los resultados validan el uso de la NPU como motor de inferencia principal para despliegues en tiempo real, y del entorno CPU como herramienta auxiliar para validación, debug o ejecución de modelos no compatibles con el ecosistema RKNN.

8 Bibliografia

1. Rockchip RKNN Model Zoo. Modelos de referencia optimizados para la NPU de Rockchip: **https://github.com/airockchip/rknn_model_zoo/tree/main**
2. Rockchip RKNN Toolkit 2 - Documentación oficial. Guía de uso del SDK para inferencia acelerada en dispositivos Rockchip: **https://github.com/airockchip/rknn-toolkit2/tree/master/doc**
3. Keras Applications - MobileNetV2. Descripción y especificaciones del modelo MobileNetV2 utilizado como base: **https://keras.io/api/applications/mobilenet/**
4. Luckfox Pico - Guía de inicio rápido. Documentación general de la placa de desarrollo utilizada: **https://wiki.luckfox.com/luckfox-pico/luckfox-pico-quick-start/**
5. TensorFlow Lite C API. API oficial de TensorFlow Lite para ejecución de modelos en C: **https://www.tensorflow.org/lite/guide/inference#c_api**
6. ARM Cortex-A7 Technical Reference Manual. Arquitectura y capacidades de procesamiento del núcleo ARMv7-A: **https://developer.arm.com/documentation/ddi0464/latest/**