



ugr

Universidad  
de Granada

TRABAJO DE FIN DE MÁSTER  
MÁSTER OFICIAL EN CIENCIA DE DATOS E INGENIERÍA DE  
COMPUTADORES

---

# Aceleración por hardware de LLMs en sistemas empuotrados

**Autor**

Ángel Hurtado Flores

**Tutor**

Alberto Guillén Perales

---



E.T.S. DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

Granada, septiembre de 2025



# Aceleración por hardware de LLMs en sistemas empujados

Ángel Hurtado Flores

---

**Palabras clave:** LLMs, NPU, SBCs, Rockchip, IA, aceleración por hardware

## Resumen

Los avances recientes en electrónica, informática y aprendizaje automático han impulsado el desarrollo de tecnologías avanzadas en inteligencia artificial, incluyendo *modelos de lenguaje de gran escala* (LLMs). Estos modelos, tradicionalmente diseñados para ejecutarse en hardware de alto rendimiento, están siendo progresivamente adaptados para su implementación en sistemas de medio-bajo consumo energético, ampliando sus aplicaciones en dispositivos embebidos y plataformas edge.

En este contexto, la aceleración por hardware desempeña un papel clave para superar las limitaciones de cómputo en dispositivos con recursos restringidos. Este trabajo se centra en la optimización y despliegue de LLMs utilizando la *Unidad de Procesamiento Neural* (NPU) del chip RK3588. Como caso de estudio, se desarrolla un framework simplificado basado en el proyecto rknn-llm, diseñado para maximizar la eficiencia en sistemas embebidos que usen el chip mencionado.

El objetivo principal es explorar y validar estrategias que permitan la ejecución eficiente de LLMs en entornos de bajo consumo, incluyendo técnicas de conversión, cuantización y optimización específica para la NPU del RK3588. Los resultados obtenidos se integran en una plataforma experimental para evaluar su impacto en términos de rendimiento y usabilidad en aplicaciones del mundo real.



# Hardware-Accelerated LLMs in Embedded Systems

Ángel Hurtado Flores

---

**Keywords:** LLMs, NPU, SBCs, Rockchip, AI, hardware-acceleration

## Abstract

Recent advances in electronics, computer science, and machine learning have driven the development of cutting-edge technologies in artificial intelligence, including *Large Language Models* (LLMs). These models, traditionally designed for high-performance hardware, are increasingly being adapted for deployment on low-power and resource-constrained systems, broadening their applications in embedded devices and edge platforms.

In this context, hardware acceleration plays a key role in overcoming computational limitations in constrained devices. This work focuses on the optimization and deployment of LLMs utilizing the *Neural Processing Unit* (NPU) of the RK3588 chip. As a case study, a simplified framework based on the rknn-llm project, is developed to maximize efficiency in embedded systems.

The primary objective is to explore and validate strategies for efficient execution of LLMs in low-power environments, including techniques such as model conversion, quantization, and hardware-specific optimization for the RK3588 NPU. The results are integrated into an experimental platform to assess their performance and usability in real-world applications.



---

Yo, **Ángel Hurtado Flores**, alumno de la titulación Máster Oficial en Ciencia de Datos e Ingeniería de Computadores de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**, con DNI XXXXXXXXA autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

**Firmado:** Ángel Hurtado Flores

Granada a 1 de septiembre de 2025





---

D. **Alberto Guillén Perales**, Profesor del Departamento Ingeniería de Computadores, Automática y Robótica de la Universidad de Granada.

**Informa:**

Que el presente trabajo, titulado ***Aceleración por hardware de LLMs en sistemas empotrados***, ha sido realizado bajo su supervisión por **Ángel Hurtado Flores**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 1 de septiembre de 2025

**El director:**

**Alberto Guillén Perales**



# Agradecimientos

A mis amigos, por las “noches de durum”, “miércoles de pollo” y todos los buenos ratos que hemos pasado juntos, ya fuese en el fútbolín, jugando a cualquier juego, cocinando pollo con arroz los fines de semana o simplemente charlando en el comedor de la ETSIIT. En especial, a José por tantísimas cosas que hemos hecho juntos: trabajos, jugar, cocinar, el robot...

A mi familia, por el apoyo y paciencia durante los cuatro años del grado y ahora este año de máster. Por un lado, a mi padre, por enseñarme a “cacharrear”, a mi hermano, por todo lo que me ha enseñado de tecnología e ingeniería y a mi tío Alberto, por hacerme ver la tecnología desde el punto de vista del entretenimiento. Por otro lado, a mi madre, por enseñarme de tantas cosas distintas a la tecnología que siempre son útiles y por tantos otros consejos; y a mi hermana, por contagiarme su simpatía, ánimo y sonrisa que todos los días luce.

A Alberto, mi tutor, por el apoyo brindado durante la realización del TFM, especialmente bajo la situación tan complicada que he tenido durante el desarrollo de este. Desde luego, sin este apoyo no estaría donde estoy hoy en día. Oh, y por supuesto, también por las buenas clases que ha impartido por el máster (me sorprendieron las de Emprendimiento, a pesar de que no es uno de mis temas favoritos, fueron considerablemente entretenidas e interesantes, un muy buen cierre al máster)



# Siglas

**AI** Artificial Intelligence

**CPU** Central Processing Unit

**eMMC** embedded MultiMediaCard

**GPIO** General Purpose Input/Output

**GPU** Graphical Processing Unit

**HDMI** High-Definition Multimedia Interface

**IA** Inteligencia Artificial

**IoT** Internet of Things

**LLM** Large Language Model

**NPU** Neural Processing Unit

**ONNX** Open Neural Network Exchange

**RAM** Random Access Memory

**RK3588** Rockchip RK3588

**SBC** Single Board Computer

**SDK** Software Development Kit

**SoC** System on a Chip

**SSD** Solid State Drive

**USB** Universal Serial Bus



# ÍNDICE GENERAL

---

<b>1. Introducción</b>	<b>21</b>
1.1. Motivación . . . . .	21
1.2. Descripción general del sistema . . . . .	22
1.2.1. Hardware: Orange Pi 5 . . . . .	22
1.2.2. Software . . . . .	25
1.3. Objetivos . . . . .	26
1.4. Planificación . . . . .	27
1.5. Estado del arte . . . . .	27
<b>2. Fundamentos</b>	<b>29</b>
2.1. Fundamentos teóricos de LLMs . . . . .	29
2.1.1. Redes Neuronales en LLMs . . . . .	29
2.1.2. Transformers . . . . .	30
2.1.3. Optimización de LLMs . . . . .	31
2.2. Arquitectura del RK3588 . . . . .	34
2.2.1. Arquitectura de la NPU . . . . .	35
<b>3. Entorno y Herramientas</b>	<b>39</b>
3.1. rknn-toolkit2 . . . . .	39
3.2. rknn-llm . . . . .	41
3.3. ezrknpu . . . . .	43
<b>4. Implementación</b>	<b>45</b>
4.1. Actualización del driver de NPU . . . . .	45
4.2. Conversión automatizada de LLMs . . . . .	47
4.2.1. ezrkllm-toolkit . . . . .	47
4.2.2. Desarrollo del contenedor . . . . .	47
4.2.3. Flujo de conversión . . . . .	49
4.2.4. Detalles del proceso . . . . .	49
4.3. Automatización de la instalación . . . . .	50
4.3.1. Bibliotecas . . . . .	50

4.3.2.	Paquetes . . . . .	51
4.3.3.	Programa rkllm . . . . .	51
4.4.	Uso de LLMs . . . . .	52
4.4.1.	Uso básico desde CLI . . . . .	52
4.4.2.	Servidor web para uso desde navegador . . . . .	53
<b>5.</b>	<b>Resultados y Pruebas</b>	<b>55</b>
5.1.	Resultados . . . . .	55
5.1.1.	Funcionamiento conseguido . . . . .	55
5.1.2.	Facilidad de uso . . . . .	55
5.2.	Pruebas de rendimiento . . . . .	56
5.2.1.	Consumo energético . . . . .	56
5.2.2.	Comparativa contra CPU . . . . .	58
5.2.3.	Comparativa contra GPU . . . . .	59
5.3.	Limitaciones . . . . .	60
<b>6.</b>	<b>Conclusiones</b>	<b>63</b>
6.1.	Análisis de los objetivos y planificación . . . . .	63
6.2.	Aplicaciones y usos . . . . .	64
6.3.	Trabajos y mejoras futuras . . . . .	66
6.4.	Conclusión . . . . .	67
<b>A.</b>	<b>Instalación automatizada de ezrknpu</b>	<b>71</b>
A.1.	Requisitos previos . . . . .	71
A.2.	Instrucciones de instalación . . . . .	72
A.3.	Verificación de la instalación . . . . .	72
A.4.	Ejecución de un LLM . . . . .	73
<b>B.</b>	<b>Conversión y cuantización de LLMs</b>	<b>75</b>
B.1.	Conversión . . . . .	75
B.2.	ez-er-rkllm-toolkit . . . . .	77
<b>C.</b>	<b>Evaluación de métricas de la NPU</b>	<b>79</b>
C.1.	ntop.sh . . . . .	79
C.2.	rknputop . . . . .	80
	<b>Bibliografía</b>	<b>85</b>



## ÍNDICE DE FIGURAS

---

1.1. Orange Pi 5 . . . . .	23
1.2. Planificación de este trabajo . . . . .	27
2.1. Diagrama de la arquitectura interna de un transformer . . . . .	32
2.2. Comparación de Llama 2, 3 y 3.1 con diferentes niveles de cuantización [14] . . . . .	33
2.3. Diagrama de la arquitectura del SoC RK3588 [17] . . . . .	35
3.1. Diagrama de funcionamiento de RKNN-Toolkit 2 [5] . . . . .	40
3.2. Diagrama de funcionamiento de RKLLM [1] . . . . .	41
4.1. Ejemplo de ejecución de un modelo desde CLI . . . . .	53
5.1. Ejecución de modelo en CPU . . . . .	59
6.1. Resultados reales de la planificación . . . . .	64
A.1. Ejemplo simple de primera ejecución de un LLM y prompt . . . . .	73
A.2. Prompt “tell me a joke” en DeepSeek R1 . . . . .	74
C.1. Ejemplo de output de ntop.sh (derecha) mientras se ejecuta un LLM (izquierda) . . . . .	80
C.2. rknpustop . . . . .	81



## ÍNDICE DE TABLAS

---

5.1. Comparativa de consumo energético usando dos dispositivos de medida . . . . .	57
5.2. Comparativa de rendimiento entre NPU y CPU RK3588 . . .	58
5.3. Comparativa entre procesadores del RK3588 . . . . .	60



# Capítulo 1

## INTRODUCCIÓN

---

En este trabajo se desarrolla un framework simplificado basado en el proyecto rknn-llm [1], diseñado para maximizar la eficiencia de ejecución de Large Language Models (LLMs) en sistemas embebidos que usen la **Neural Processing Unit (NPU) encontrada en el chip Rockchip RK3588 (RK3588)**.

El objetivo principal es explorar y validar estrategias que permitan la **ejecución eficiente de LLMs en entornos de bajo consumo, incluyendo técnicas de conversión, cuantización y optimización específica para la NPU del RK3588**. Los resultados obtenidos se integran en una plataforma experimental para evaluar su impacto en términos de rendimiento y usabilidad en aplicaciones del mundo real.

### 1.1. Motivación

Desde la aparición de los *transformers* en 2017 con la publicación “*Attention is all you need*” [2] y la posterior popularización de ChatGPT a finales de 2022, los modelos de lenguaje de gran escala (LLMs) han revolucionado la inteligencia artificial.

No obstante, su implementación está limitada principalmente a hardware de alto rendimiento con ejecución en la nube, restringiendo su uso en dispositivos embebidos/empotrados y de bajo consumo, así como el uso y ejecución en local. Con la creciente necesidad de llevar IA avanzada a sistemas de *edge computing*, las *Neural Processment Unit* (NPUs), como la del chip RK3588, ofrecen una solución prometedora para romper la alta barrera de rendimiento.

Este trabajo busca explorar cómo **optimizar y ejecutar LLMs en**

hardware económico y eficiente, contribuyendo a democratizar el acceso a estas tecnologías avanzadas y potenciando su aplicabilidad en entornos reales.

Gracias a ello, se consiguen precios menores, reducción en el consumo de energía y el uso de LLMs de código abierto, fomentando la innovación, la colaboración comunitaria y la personalización de estos.

## 1.2. Descripción general del sistema

El sistema desarrollado en este trabajo tiene como objetivo optimizar y ejecutar modelos de lenguaje de gran escala (LLMs) en plataformas embebidas, utilizando como base el hardware de la **Orange Pi 5** [3], equipada con el chip RK3588 y su NPU. Este dispositivo, conocida alternativa a las Raspberry Pi 5, ofrece un equilibrio entre potencia de cálculo y bajo consumo energético, haciéndolo ideal para aplicaciones en sistemas embebidos y de edge computing.

En el ámbito del software, el sistema se fundamenta en el framework *open source* **rknn-llm**, que será adaptado, simplificado y mejorado. Este framework incluye herramientas para la conversión, cuantización y optimización de modelos LLM, maximizando su eficiencia en la NPU del RK3588. El sistema busca demostrar la capacidad de ejecutar LLMs de forma eficiente en hardware económico, promoviendo su uso en tareas como asistentes locales, análisis de texto y aplicaciones IoT, reduciendo la dependencia de infraestructuras de alto rendimiento y conectividad a la nube.

### 1.2.1. Hardware: Orange Pi 5

Como se ha mencionado, se trabajará concretamente con la **Orange Pi 5**, conocida alternativa entre los *Single Board Computers* (SBC), computadores embebidos de alta potencia que generalmente se utilizan para proyectos de IoT, electrónica, robótica, visión por computador, servidores, educación o combinaciones de estas.

El fabricante, como su nombre indica, es **Orange Pi** que además fabrica otros SBCs alternativos con el mismo chip (RK3588), como la Orange Pi 5 Pro, Orange Pi 5 Max, Orange Pi 5 Ultra u Orange Pi 5 Plus. Se ha decidido elegir la Orange Pi 5 al ser la primera en ser lanzada (Noviembre/Diciembre de 2022), así como la más simple y barata de ellas, con un PVP inicial de tan sólo US\$60 para la versión de 4 GB de RAM, la más barata de las opciones

de RAM [4].

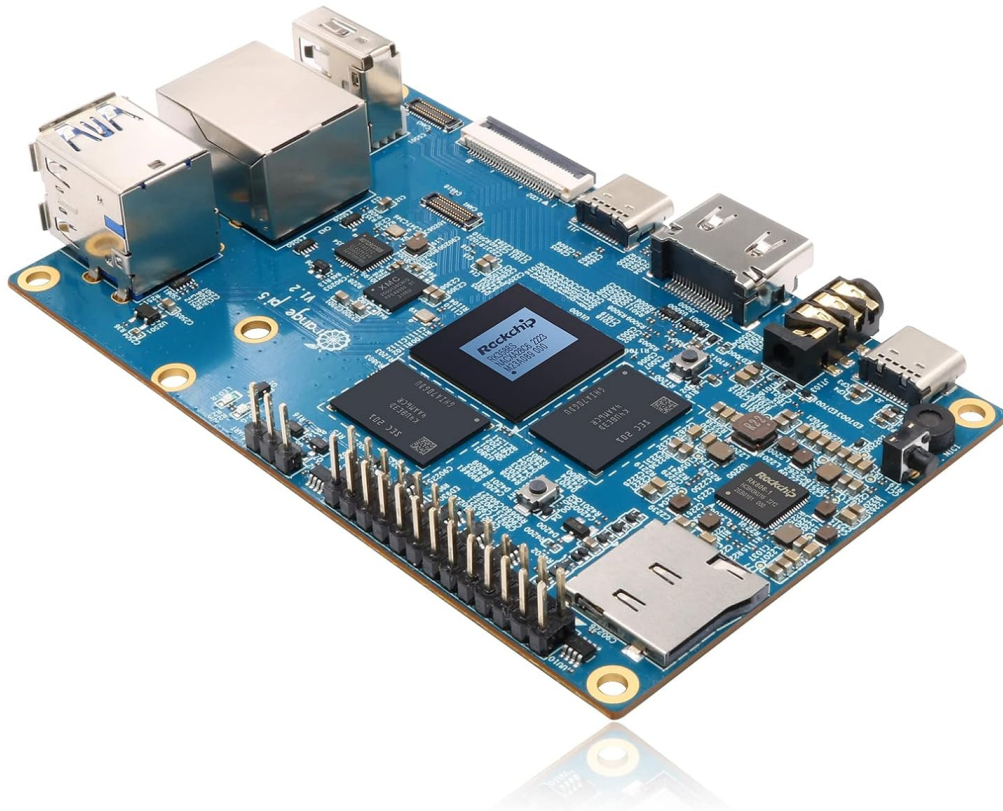


Figura 1.1: Orange Pi 5

Las especificaciones técnicas de la Orange Pi 5 son las siguientes:

- **Procesador:** Rockchip RK3588S, CPU de 8 núcleos (4x Cortex-A76 y 4x Cortex-A55).
- **GPU:** ARM Mali-G610 MP4 con soporte para gráficos avanzados.
- **NPU:** Unidad de Procesamiento Neural integrada con un rendimiento de hasta 6 TOPS.

- **Memoria RAM:** Opciones de 4 GB, 8 GB, 16 GB o 32 GB LPDDR4X.
- **Almacenamiento:**
  - Ranura para tarjeta microSD.
  - Almacenamiento eMMC opcional de hasta 256 GB.
  - Puerto M.2 PCIe 2.1 para unidades SSD NVMe.
- **Conectividad:**
  - Ethernet Gigabit.
  - Wi-Fi 6 y Bluetooth 5.0 (módulo adicional no incluido).
- **Puertos:**
  - 2 puertos USB 3.0 y 1 puerto USB 2.0.
  - HDMI 2.1 (hasta 8K@60fps) y DisplayPort.
  - Puerto GPIO de 26 pines compatible con Raspberry Pi.
  - Entrada de alimentación USB-C (PD o 5V/4A). **Consumo máximo teórico de tan sólo 20 W**
- **Sistema operativo:** Compatible con Android, Debian, Ubuntu y otras distribuciones de Linux.
- **Aplicaciones:** Ideal para IA, IoT, servidores multimedia, computación embebida y más.

Podemos observar dos detalles llamativos de la lista de especificaciones:

- **4 GB de RAM:** se ha destacado en negrita los 4 GB de RAM al ser el modelo que se va a usar, al ser el más barato. Para ciertos modelos es preferible usar el de 8 o incluso 16 GB de RAM (generalmente, los que necesiten más de 16 GB de RAM serán muy lentos, así que no es recomendable usar el de 32 GB si no se va a combinar un LLM con otras utilidades).
- **RK3588S:** Este chip es una variante del RK3588 que utiliza un bus PCIe 2.0 frente al 3.0 del modelo estándar para la conexión M.2 con SSDs. Esto permite abaratar el coste del chip y no es vital para el uso de LLMs en nuestro escenario. El resto de características del SoC se mantienen intactas. *Nota: a lo largo de este trabajo **se usará tanto RK3588 como RK3588S indistintamente**, al ser su única diferencia irrelevante para la correcta ejecución de este trabajo*



### 1.2.2. Software

El software que se usará es un framework llamado **rknn-llm** [1] diseñado para aprovechar las capacidades de las NPUs en los chips Rockchip, como el RK3588. Su objetivo es facilitar la implementación de modelos de lenguaje de gran escala (LLMs) en dispositivos embebidos mediante técnicas de optimización específicas para hardware, como la conversión de modelos desde formatos populares (ONNX y Hugging Face Transformers) y la cuantización a INT8 para mejorar la eficiencia energética.

Sin embargo, esta herramienta posee diversos defectos, por ejemplo, falta de documentación, dificultad para la conversión de modelos e incluso falta de implementación del driver necesario en el kernel de Linux. En otras palabras, este framework posee muchísima dificultad de uso para un usuario común e incluso levemente avanzado. Se requieren bastantes conocimientos de desarrollador en varios ámbitos para el uso de esta herramienta de base, incluyendo:

- Linux en general
- Manejo de la terminal y el shell
- Implementación de drivers en el kernel y posterior compilación
- Uso de la herramienta de conversión de modelos
- Computador con arquitectura x86 necesario para la conversión de modelos, con altas capacidades de CPU y RAM.
- Descarga de LLMs con Git LFS

Por ello, se ha decidido realizar un fork del repositorio original que tenga como principal objetivo simplificar el uso de esta herramienta, así como mejorarla en general.

Paralelamente, también existe **rknn-toolkit2** [5], el cual proporciona herramientas para convertir, optimizar y analizar modelos IA, ajustándolos para su ejecución eficiente en las NPUs de Rockchip, con un enfoque en redes neuronales más simples, como puede ser modelos para visión por computador. No obstante, no es el objetivo principal en este TFM.

## 1.3. Objetivos

En términos generales, se propone conseguir la **aceleración por hardware de LLMs en sistemas de bajo consumo**, en este caso, utilizando la **NPU encontrada en el chip RK3588S**, que se puede encontrar en SBCs como la Orange Pi 5 que se usará para este trabajo.

Además, se intentará que todo este proceso resulte lo más simple posible de cara a usuarios menos experimentados.

- **OBJ-1:** introducción, objetivos, planificación y preparación general. Preparación para la realización de este trabajo, incluyendo la realización de este capítulo, familiarización con el ámbito, preparar la plantilla de LaTeX...
- **Capítulo 2 (OBJ-2):** fundamentos. Se propone como objetivo entender los fundamentos teóricos de los LLMs modernos, así como un entendimiento más profundo del chip RK3588, en especial, en su NPU. Ello permitirá un mejor desarrollo y optimización de este trabajo.
- **Capítulo 3 (OBJ-3):** entorno y herramientas. Se trata de entender el entorno y herramientas que se van a usar para la ejecución de LLMs. Similar al anterior objetivo, se intenta profundizar en la tecnología antes del desarrollo.
- **Capítulo 4 (OBJ-4):** implementación. Se desarrollará el software que permitirá convertir y usar LLMs de la forma más simplificada posible en base a rknn-llm.
- **Capítulo 5 (OBJ-5):** resultados y pruebas. Se realizará un análisis y evaluación de los resultados, así como se realizarán pruebas para evaluar el rendimiento y calidad de los resultados de una manera sistemática.
- **Capítulo 6 (OBJ-6):** terminamos este TFM con las conclusiones extraídas, trabajos y mejoras futuras, análisis de la planificación y objetivos cumplidos.

Resumidamente, se trata de conseguir que LLMs funcionen eficientemente en el chip RK3588 de la forma más simple posible.

### 1.4. Planificación

Para cumplir los objetivos provistos, se ha realizado la siguiente planificación:

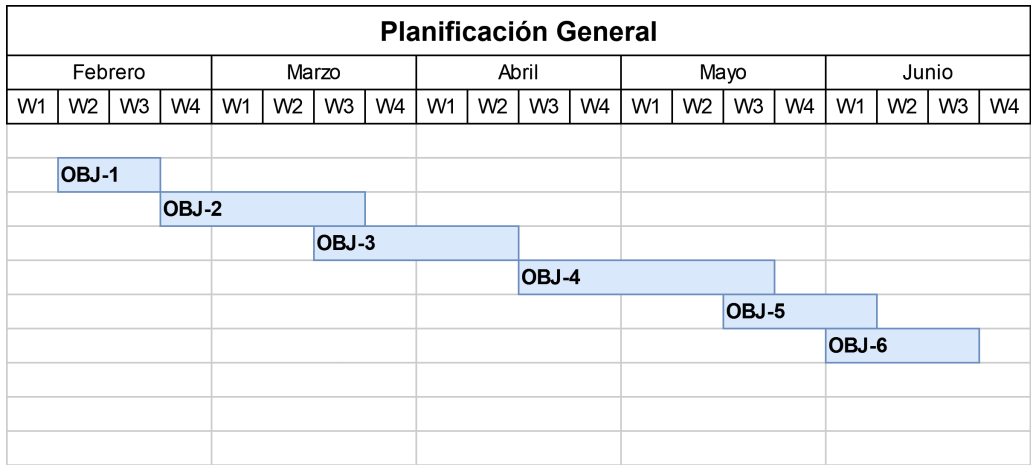


Figura 1.2: Planificación de este trabajo

Por supuesto, esta planificación es aproximada y flexible hasta cierto punto. Debido a la naturaleza de este trabajo, es difícil prever con exactitud el correcto desarrollo de la planificación, al requerir un estudio constante de una tecnología en desarrollo con multitud de opciones, así como la dependencia de un proyecto de ámbito *open source*, en constante evolución con los posibles problemas y contratiempos que ello puede ocasionar, por ejemplo, las actualizaciones de este.

En el último capítulo, se hará una comparación entre esta planificación y el resultado final de esta, con el objetivo de entender qué ha pasado durante el desarrollo de este trabajo y si el desarrollo ha ido como fue planeado.

### 1.5. Estado del arte

El estado del arte en el mundo de los LLMs varía considerablemente dependiendo de en qué área nos centremos. Por ejemplo, en cuanto a LLMs de texto destacan modelos como Claude 3.5 Sonnet y GPT o3. También destacan modelos como Midjourney o DALL·E 3 en cuanto a generación de imágenes.

No obstante, este TFM se centra en el uso de LLMs de bajo consumo,

como pueden ser Llama 3 de Meta (en sus variantes de menor cantidad de parámetros), Phi-3 Mini de Microsoft o Gemma de Google. Estos LLMs son considerados entre los mejores del ámbito *open source* y de bajo consumo de recursos. Por ejemplo, Llama 3 tiene una variante con 8B parámetros (que significa que tiene, aproximadamente, 8 billion, o sea, 8 mil millones de parámetros). Meta afirma que este modelo es comparable a GPT 3.5 Turbo [6], el cual se ejecuta en los servidores de OpenAI y tiene bastante mayor uso de parámetros y, por ende, consumo.

Por otro lado, es vital entender cuál es el estado en cuanto a aceleración por hardware de LLMs. Los casos más interesantes y conocidos hoy en día son el uso en local de LLMs en móviles, en concreto, Gemini de Google en dispositivos Pixel [7] y Apple Intelligence en dispositivos iPhone [8]. Ambos dispositivos utilizan sus respectivas NPUs para poder acelerar la ejecución de estos LLMs de forma local y con bajo consumo de recursos. En el caso de Google, su SoC Tensor G4 contiene lo que llaman una *TPU* (Tensor Processing Unit) y en el caso de Apple *Neural Engine*.

Si bien el caso anterior es interesante por la aceleración en dispositivos de tamaño contenido, el precio y potencia de estos no refleja totalmente lo que se conoce como “sistema empotrado” [11]. Por ello, es importante comparar con dispositivos que sí cumplan las características para ser llamados “empotrados”. Entre ellos, destacan la Raspberry Pi 5 (el referente en el mundo de los *Single Board Computers*) y la Nvidia Jetson Orin Nano Super. La primera, si bien no contiene una NPU, tiene un gran soporte de la comunidad *open source* y un coste por debajo de los 100 euros [9]. La segunda, aunque con mayor coste (\$249), contiene una GPU de Nvidia, líderes en aceleración de aplicaciones de IA, incluyendo LLMs [10].

# Capítulo 2

## FUNDAMENTOS

---

Tras una breve introducción a este trabajo, es crítico entender una pequeña base del funcionamiento de un LLM y las NPUs (o en este caso, NPU del RK3588) para poder comprender y desarrollar este trabajo de forma óptima.

### 2.1. Fundamentos teóricos de LLMs

Los **Large Language Models (LLMs, por sus siglas en inglés)** representan un avance significativo en el campo de la inteligencia artificial (IA) y el procesamiento del lenguaje natural (NLP, por sus siglas en inglés). Estos modelos son redes neuronales entrenadas con cantidades masivas de datos textuales para comprender, generar y manipular texto de forma coherente y contextual. En este apartado, se describen los conceptos clave necesarios para comprender su funcionamiento, con un enfoque en su uso práctico [12].

#### 2.1.1. Redes Neuronales en LLMs

Las redes neuronales son estructuras matemáticas inspiradas en el funcionamiento del cerebro humano. Están formadas por capas de nodos (neuronas), conectados por pesos que se ajustan durante el entrenamiento para aprender patrones complejos. En el contexto de los LLMs, estas redes tienen millones o incluso miles de millones de parámetros, lo que les permite modelar relaciones complejas en el lenguaje. Hoy en día, se cree que modelos más grandes de código cerrado como las versiones más recientes de ChatGPT o Gemini, andan en el orden de billones de parámetros.

- **Entrenamiento supervisado:** Consiste en ajustar los parámetros del modelo para minimizar el error entre las predicciones del modelo y los datos reales.
- **Capacidad de generalización:** Gracias a su arquitectura, los LLMs pueden generar texto coherente en múltiples idiomas y adaptarse a diferentes contextos y aplicaciones.

### 2.1.2. Transformers

Introducida en 2017 con el artículo “*Attention is All You Need*” [2], la arquitectura de transformadores (*transformers* en inglés) revolucionó el campo del procesamiento del lenguaje natural (NLP) y ha sido la base para el desarrollo de modelos avanzados como GPT y BERT.

- **Mecanismo de atención:** Los transformadores usan un mecanismo de atención que evalúa la relevancia de cada palabra o token en una secuencia, permitiendo capturar relaciones a largo plazo y dependencias complejas entre palabras o fragmentos de texto. Esto supera las limitaciones de arquitecturas anteriores como las redes recurrentes (*RNNs*) y las redes convolucionales (*CNNs*) en tareas de secuencia.
- **Escalabilidad:** Gracias a su diseño basado en el mecanismo de atención y el uso de operaciones altamente paralelizables, los transformadores son especialmente adecuados para el entrenamiento de modelos grandes, como los LLMs. Esta escalabilidad ha sido clave para procesar grandes cantidades de datos y construir modelos con miles de millones de parámetros.
- **Componentes principales:**
  - **Encoders y decoders:** Los transformadores se componen de dos partes principales:
    - El **encoder**, que toma una entrada y genera representaciones internas codificadas del texto.
    - El **decoder**, que utiliza estas representaciones para generar salidas, como texto traducido o respuestas a preguntas.

Modelos como BERT se basan únicamente en el encoder, mientras que GPT utiliza principalmente el decoder.

- **Capa de atención auto-regresiva:** Utilizada en tareas de generación de texto, esta capa asegura que el modelo preste atención únicamente a los tokens previos en la secuencia al generar el siguiente token. Este enfoque permite generar texto fluido y coherente.
- **Feedforward y normalización:** Cada capa incluye subcapas de redes feedforward completamente conectadas, combinadas con mecanismos de normalización y capas residuales para mejorar la estabilidad del entrenamiento y la capacidad de aprendizaje.
- **Versatilidad:** Los transformadores no solo se aplican al texto, sino también a otras modalidades de datos, como imágenes, audio y genomas, gracias a su diseño generalista.

### 2.1.3. Optimización de LLMs

Una vez entrenados, los LLMs pueden ser extremadamente exigentes en términos de recursos computacionales. Para facilitar su uso en dispositivos de bajo consumo, se aplican técnicas de optimización como la cuantización (*quantization*, en inglés) [13].

### Pesos y Activaciones: Conceptos Fundamentales

En el contexto de las redes neuronales y LLMs, los pesos y las activaciones son conceptos clave que definen cómo opera un modelo:

- **Pesos:** Son los parámetros aprendidos por el modelo durante el entrenamiento. Representan la fuerza de las conexiones entre las neuronas y determinan cómo se procesan las entradas para generar las salidas.
- **Activaciones:** Son los valores intermedios generados al pasar una entrada por las capas del modelo. Representan cómo responden las neuronas al procesar la información en cada capa.

Ambos elementos tienen un impacto directo en el tamaño del modelo, la memoria utilizada y los recursos computacionales necesarios para la inferencia.

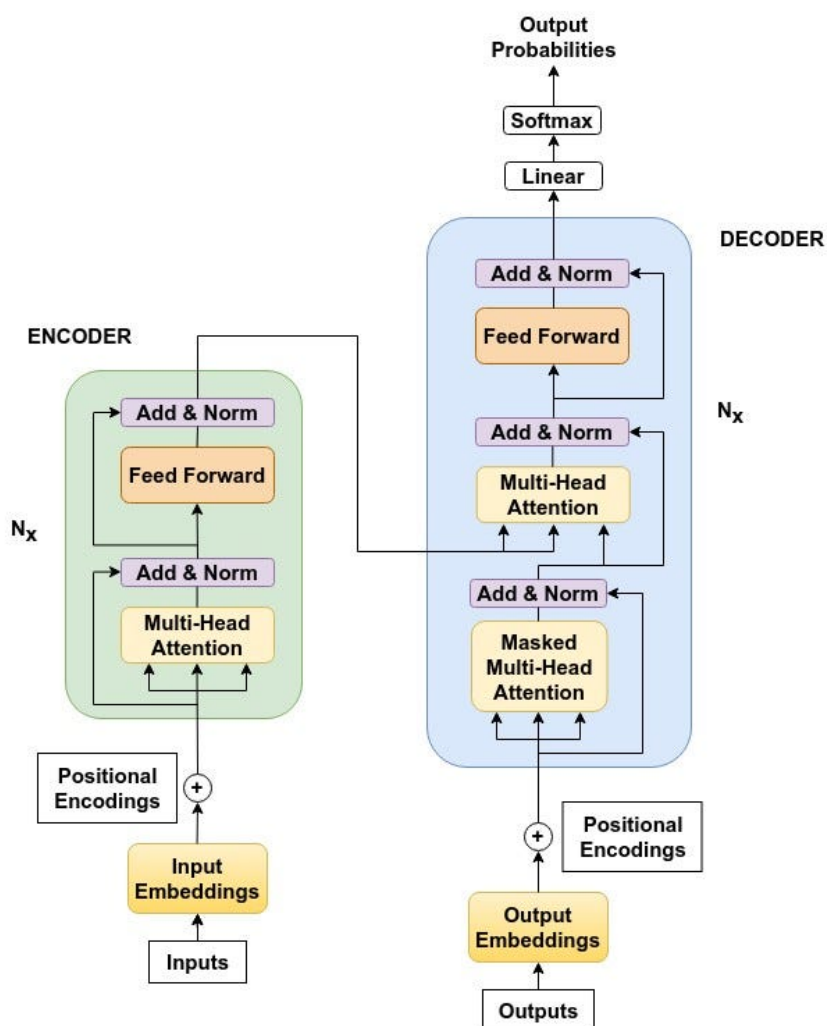


Figura 2.1: Diagrama de la arquitectura interna de un transformer

## Cuantización

La cuantización consiste en reducir la precisión de los valores numéricos del modelo (normalmente de 32 bits de coma flotante, FP32, a 8 bits enteros, INT8). Existen diversos métodos, como GGUF (GPT Graph Universal Format), SpQR (Sparse Quantized Representations), GPTQ (GPT Quantized)... [13]. Los puntos clave de la cuantización de cara a este trabajo son:

- **Beneficios:** Esta técnica reduce el tamaño del modelo, disminuye el consumo energético y aumenta la velocidad de inferencia, haciéndola especialmente útil en hardware embebido.



- **Impacto:** Aunque puede introducir una ligera pérdida de precisión a la hora del uso de un LLM, esto suele ser tolerable en la mayoría de las aplicaciones prácticas.
- **Cuantización en RK3588:**
  - La NPU del RK3588 no es suficientemente potente, por tanto, necesita cuantización.
  - Trabajamos con cuantización **w8a8**, es decir, *weights* a 8 bits y *activations* también a 8 bits.

En la siguiente figura podemos observar cómo afecta la cuantización GGUF a los modelos Llama 2, 3 y 3.1 con diferentes números de bits. Como referencia, estos modelos parten de flotantes de 16 bits en sus versiones estándar.

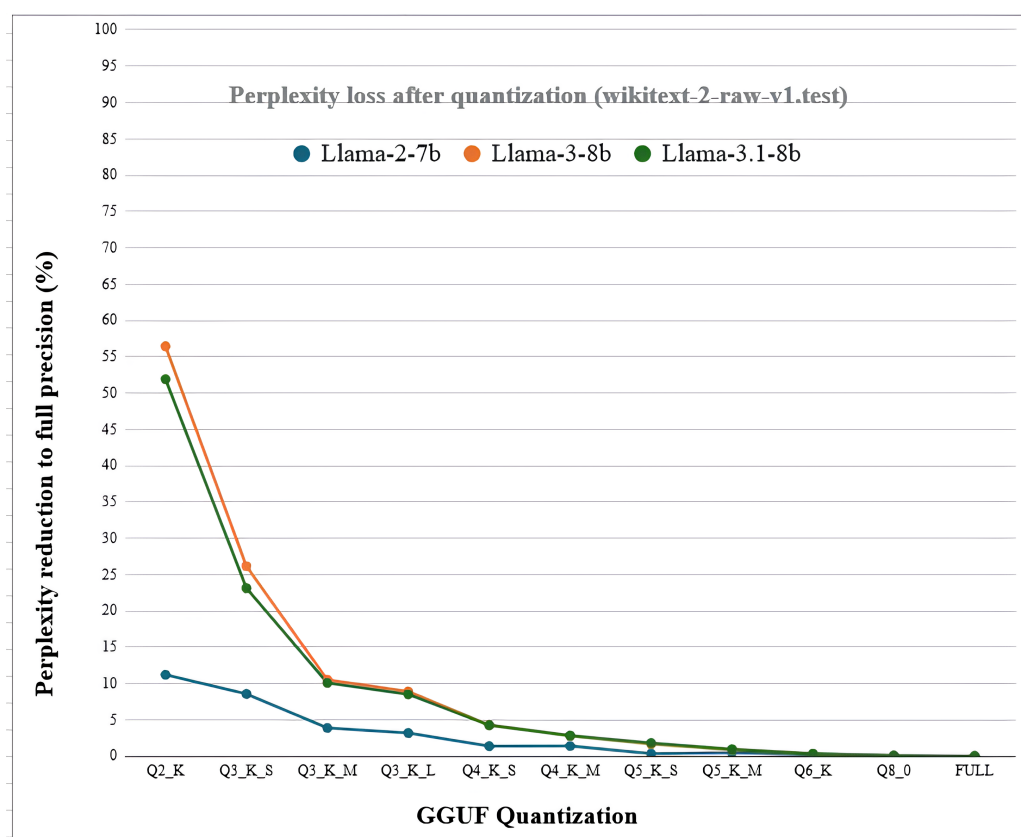


Figura 2.2: Comparación de Llama 2, 3 y 3.1 con diferentes niveles de cuantización [14]

Podemos observar que la cuantización de 4 a 8 bits no tiene ni un 5 % de pérdida de precisión respecto al modelo completo. No obstante, a partir de 3 bits y menos, la cuantización tiene mucha pérdida. Llama la atención el hecho de que Llama 3 (modelo superior a Llama 2) cuantiza peor.

## Distilación de LLMs

Los modelos *distill* (destilados) son versiones comprimidas y optimizadas de modelos de lenguaje grandes (*LLMs*) entrenados mediante una técnica llamada *knowledge distillation*. Esta técnica consiste en transferir el conocimiento de un modelo grande y preciso (denominado *teacher*) a otro más pequeño y eficiente (*student*), que imita sus salidas [15].

El objetivo principal es mantener un rendimiento similar al del modelo original, pero con un tamaño mucho menor y una velocidad de inferencia significativamente más rápida. Esto hace que los modelos destilados sean especialmente útiles en entornos con recursos limitados, como los dispositivos embebidos usados en este TFM.

Por ejemplo, modelos como **DeepSeek R1 Distill Qwen** son variantes más ligeras de modelos grandes, optimizadas para tareas específicas (como razonamiento o generación de código) con un menor coste computacional. Estas versiones son ideales para su ejecución en NPUs o plataformas edge como RK3588, ya que requieren menos memoria y energía sin sacrificar excesivamente el rendimiento.

Es posible además, hacer destilación de LLMs distintos. Como bien se ha indicado de ejemplo, se puede destilar DeepSeek R1 (LLM de 685B [16], extremadamente potente) sobre Qwen (creado por Alibaba, no DeepSeek, y de tamaños muy variables, pocos miles de millones de parámetros hasta cerca de 1 billón).

Uno de los ejemplos de LLMs que se van a usar para este TFM es un LLM destilado; por ello, se menciona este pequeño apartado de optimización.

## 2.2. Arquitectura del RK3588

Para la correcta implementación y optimización de LLMs, es interesante entender la arquitectura del chip con el que trabajamos. En la siguiente figura, podemos observar la arquitectura interna del RK3588. Recordemos que el RK3588S es equivalente, cambiando el bus PCIe 3.0 por uno 2.0:

Desgraciadamente, no tenemos mucha información respecto a la NPU en

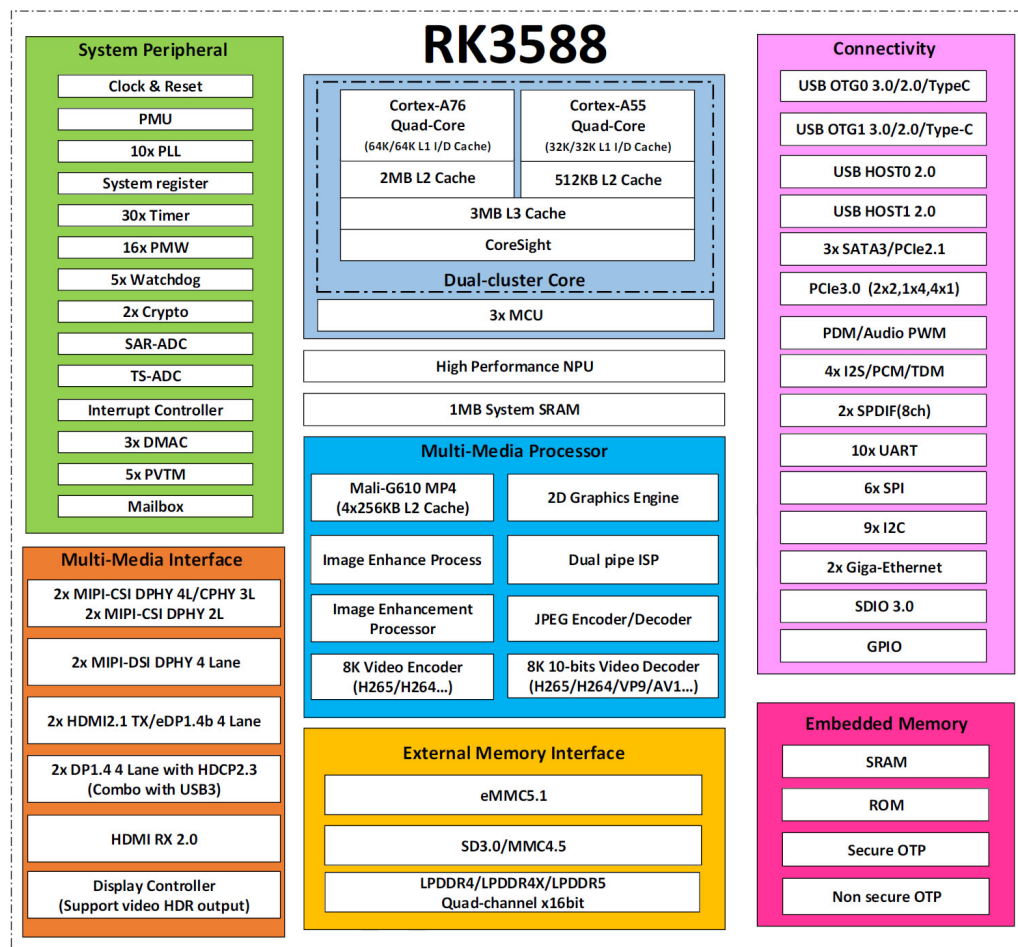


Figura 2.3: Diagrama de la arquitectura del SoC RK3588 [17]

el anterior diagrama. En la hoja de especificaciones o *datasheet* sí tenemos algo más de información. En el siguiente apartado veremos algo más de información de la NPU.

### 2.2.1. Arquitectura de la NPU

Según la *datasheet* del RK3588 [17], tenemos las siguientes características de la NPU:

- Neural network acceleration engine with processing performance up to 6 TOPS.
- Includes a triple-core NPU, supporting triple-core co-work, dual-core

co-work, and independent operation.

- Supports operations in INT4 , INT8 , INT16 , FP16 , BF16 , and TF32 .
- Embedded internal buffer of 384KB per core (three cores in total).
- Multi-tasking and multi-scenario parallel execution.
- Supports deep learning frameworks: TensorFlow, Caffe, TFLite, PyTorch, ONNX NN, Android NN, etc.
- One isolated voltage domain to support *Dynamic Voltage and Frequency Scaling* (DVFS).

Los puntos más interesantes, en general, son:

- **3 núcleos:** podemos ejecutar hasta 3 LLMs en paralelo sin problema. No obstante, es preferible usar los 3 núcleos para un solo LLM, debido a la relativa baja potencia de cada núcleo.
- **Soporte para distintos frameworks:** para utilizarse en más aplicaciones de redes neuronales, no solo LLMs
- **Distintas opciones de memoria:** podemos usar memorias de hasta 32 GB, de tipo LPDDR4, LPDDR4X e incluso LPDDR5, permitiendo opciones más baratas o de mayor rendimiento, ya que en estos casos de uso la cantidad y velocidad de memoria son críticos.

## NPU vs GPU

Ya que no hay información exacta de la NPU en el RK3588, podemos hacer una breve comparación entre GPUs y NPUs (recordemos que son similares otros procesadores de IA con otros nombres, como las TPUs, Neural Engine...) para entender levemente mejor sus diferencias:

### GPU:

- **Arquitectura:** Diseñadas para el procesamiento paralelo masivo, las GPUs cuentan con numerosos núcleos que permiten ejecutar múltiples operaciones simultáneamente.
- **Versatilidad:** Utilizan conjuntos de instrucciones generales que las hacen adecuadas para una amplia gama de aplicaciones, desde el renderizado de gráficos hasta el procesamiento de datos para inteligencia artificial.

- **Consumo:** Aunque ofrecen alto rendimiento, las GPUs pueden consumir más energía, especialmente durante operaciones intensivas.
- **Casos de Uso:** Ampliamente utilizadas en videojuegos, diseño gráfico, simulaciones científicas y, más recientemente, en el entrenamiento de modelos de aprendizaje automático debido a su capacidad de procesamiento paralelo.

#### NPU:

- **Arquitectura:** Especializadas en acelerar operaciones de redes neuronales, las NPUs están optimizadas para ejecutar cálculos específicos de inteligencia artificial de manera eficiente.
- **Rendimiento:** Diseñadas para ejecutar tareas de aprendizaje automático más rápido que las CPUs o GPUs generalizadas, las NPUs pueden ofrecer mejoras significativas en latencia y eficiencia energética en aplicaciones de IA.
- **Consumo:** Al estar dedicadas a operaciones específicas de IA, las NPUs consumen menos energía, lo que las hace ideales para dispositivos móviles y aplicaciones sensibles al consumo energético.
- **Casos de Uso:** Comúnmente integradas en dispositivos móviles y sistemas embebidos para tareas de IA que requieren eficiencia energética y baja latencia, como el reconocimiento de voz, procesamiento de imágenes y otras aplicaciones de inteligencia artificial en el borde de la red.
- **Integración:** Pueden integrarse en CPUs o sistemas en chip (SoC), permitiendo que dispositivos como smartphones y PCs realicen tareas de IA localmente, mejorando la eficiencia y reduciendo la dependencia de la nube.

Además, se recomienda leer el siguiente artículo [18], donde se expone una comparativa exhaustiva de rendimiento entre CPU, GPU y NPU de un Intel AIPC.

El artículo concluye que en plataformas heterogéneas con CPU, GPU y NPU, cada unidad de procesamiento sobresale en tareas específicas. La NPU es más rápida en multiplicación de matrices y LLMs, mientras que la GPU destaca en multiplicaciones de matrices grandes y redes LSTM. Se subraya la importancia de elegir la unidad adecuada para aplicaciones de IA y se sugiere investigar soluciones de cómputo dual y la optimización del consumo de energía.



# Capítulo 3

## ENTORNO Y HERRAMIENTAS

---

Dado un conocimiento básico de LLMs y funcionamiento de la NPU en un RK3588, este capítulo se centrará en las bases sobre las que se trabajará. Como es lógico, existen ya ciertas bases de software para poder usar la NPU de este dispositivo, así como ejecución de LLMs en este, aparte de otras tareas de IA.

Se explicará el funcionamiento de las herramientas provistas por Rockchip para poder acelerar aplicaciones de IA en sus SoCs, así como las aportaciones del autor de este TFM.

### 3.1. rknn-toolkit2

**rknn-toolkit2** es un Software Development Kit (SDK) proporcionado por Rockchip que permite la conversión, inferencia y evaluación de rendimiento de modelos de inteligencia artificial (IA) en plataformas con NPU de Rockchip, como el RK3588. Este toolkit es especialmente útil para desplegar modelos de visión por computador en dispositivos embebidos, aprovechando la aceleración por hardware que ofrecen las NPUs integradas [5].

El proceso típico de uso de **rknn-toolkit2** comienza con la conversión de modelos entrenados en formatos populares como Open Neural Network Exchange (ONNX), TensorFlow, PyTorch o Caffe al formato propietario **.rknn**. Esta conversión se realiza en un PC con arquitectura x86 y sistema operativo Linux (Ubuntu 20.04, 22.04 o 24.04), utilizando Python 3.6 a 3.12, según la versión del toolkit.

Una vez convertido el modelo, se puede realizar inferencia y evaluación de rendimiento directamente en el PC o desplegar el modelo en una placa de

desarrollo con NPU de Rockchip, como la Orange Pi 5. Para facilitar esta tarea, Rockchip proporciona **rknn-toolkit-lite2**, una versión ligera del toolkit que ofrece interfaces de programación en Python para ejecutar modelos **.rknn** en dispositivos con NPU.

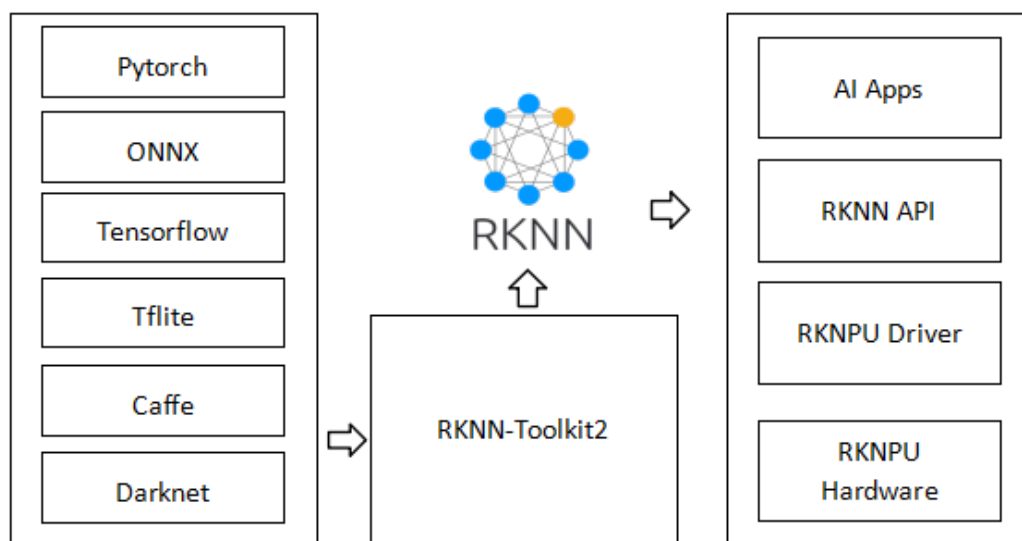


Figura 3.1: Diagrama de funcionamiento de RKNN-Toolkit 2 [5]

Entre las funcionalidades clave de **rknn-toolkit2** se incluyen:

- **Conversión de modelos:** soporta la conversión de modelos desde formatos como ONNX, TensorFlow, PyTorch, Caffe y Darknet al formato **.rknn**.
- **Cuantización:** permite convertir modelos de precisión flotante a modelos cuantizados (INT8), utilizando métodos de cuantización asimétrica y cuantización híbrida.
- **Inferencia:** posibilita la ejecución de modelos **.rknn** tanto en el PC como en dispositivos con NPU, proporcionando resultados de inferencia y evaluaciones de rendimiento.
- **Evaluación de rendimiento y memoria:** ofrece herramientas para medir el consumo de memoria y el rendimiento de los modelos durante la inferencia en dispositivos con NPU.

Este toolkit es ampliamente utilizado en aplicaciones de visión por computador, como detección de objetos, reconocimiento de imágenes y análisis de



vídeo en tiempo real. Por ejemplo, se ha utilizado para convertir y desplegar modelos YOLOv8 [19] o YOLOv11 optimizados para plataformas Rockchip, mejorando significativamente el rendimiento en tareas de detección de objetos en dispositivos embebidos.

En el contexto de este trabajo, aunque el enfoque principal es la ejecución de modelos de lenguaje de gran escala (LLMs) en dispositivos de bajo consumo, **rknn-toolkit2** representa una herramienta valiosa para explorar aplicaciones adicionales de la NPU en tareas generales de IA, especialmente de visión por computador.

### 3.2. rknn-llm

El framework **rknn-llm** (a veces conocido como **RKLLM**) es una solución desarrollada por Rockchip para facilitar la implementación eficiente de LLMs en sus plataformas con NPU, como el chip RK3588. Actualmente, solo este chip y el RK3576 son compatibles con este framework, a diferencia de **rknn-toolkit2**, que es compatible con diversos System on a Chip (SoC) de Rockchip (mientras incluyan NPU) [1].

Este framework está diseñado para maximizar el rendimiento y reducir el consumo energético en dispositivos embebidos utilizando la NPU, permitiendo la ejecución local de modelos sin depender de infraestructura en la nube.

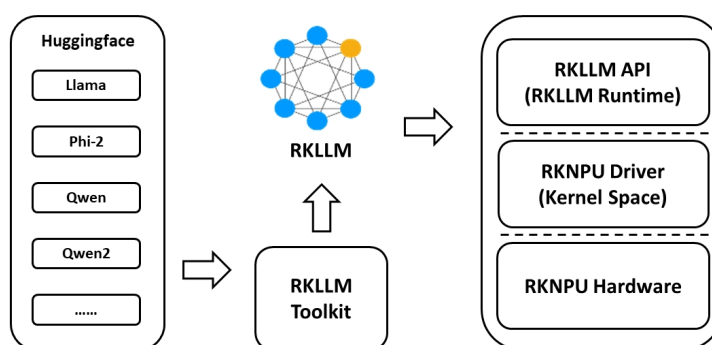


Figura 3.2: Diagrama de funcionamiento de RKLLM [1]

## Componentes principales

El ecosistema de **rknn-llm** se compone de tres elementos fundamentales:

- **rkllm-toolkit**: Herramienta de desarrollo que permite la conversión y cuantización de modelos LLM en formato Hugging Face al formato optimizado para las NPUs de Rockchip. Soporta modelos como **LLaMA** (de Meta), **Qwen** (de Alibaba), **Phi** (de Microsoft), **ChatGLM3**, **Gemma** (de Google), **InternLM2**, entre otros.
- **rkllm-runtime**: Proporciona interfaces de programación en C/C++ para desplegar los modelos convertidos en dispositivos con NPU de Rockchip, facilitando la inferencia eficiente.
- **Driver de la NPU para Linux**: Responsable de la interacción con el hardware de la NPU. Este controlador ha sido liberado como código abierto y se encuentra disponible en el código de rknn-llm. No está implementado en el kernel por defecto.

En el capítulo 4, se comentará más al respecto de cada apartado y lo que se mejorará.

## Funcionalidades destacadas

- **Conversión de modelos**: **rkllm-toolkit** permite convertir modelos en formato Hugging Face al formato RKLLM, optimizado para la ejecución en las NPUs de Rockchip.
- **Cuantización**: Soporta técnicas como **w4a16** y **w8a8**, que reducen el tamaño del modelo y mejoran la eficiencia energética sin comprometer significativamente la precisión. En el caso del RK3588, solo cuantiza a **w8a8**.
- **Optimización del rendimiento**: Incluye características como reutilización automática de caché de prompts, soporte para diálogos de múltiples turnos y expansión del contexto máximo hasta 16K tokens.

## Consideraciones prácticas

Aunque **rknn-llm** ofrece una solución potente para la implementación de LLMs en dispositivos embebidos, su uso puede presentar desafíos, especialmente para usuarios sin experiencia previa en desarrollo de drivers o

compilación de kernels. La documentación puede ser limitada y se requiere una configuración cuidadosa del entorno para garantizar una implementación exitosa. En ocasiones, incluso hay falta de documentación o documentación en chino, complicando bastante al usuario promedio, e incluso avanzado.

Sin embargo, el propósito principal de este TFM es llevar a cabo la simplificación y automatización del uso de este framework, desde la documentación, hasta la instalación y uso de LLMs en Single Board Computer (SBC) como la Orange Pi 5.

En resumen, `rknn-llm` representa una herramienta clave para llevar la potencia de los modelos de lenguaje de gran escala a dispositivos de bajo consumo, abriendo nuevas posibilidades en aplicaciones de Inteligencia Artificial (IA) en el borde, pero con el limitante de facilidad de uso, que aborda este TFM.

### 3.3. ezrknpu

El proyecto **ezrknpu** es una iniciativa de código abierto desarrollada por el autor de este TFM, cuyo objetivo es simplificar y automatizar la instalación y uso de las NPUs de Rockchip, ya sea LLMs u otro tipo de uso de NPUs, aunque el enfoque principal son los LLMs [20].

Este proyecto facilita la implementación de modelos de lenguaje de gran escala (LLMs) y modelos de visión por computador en dispositivos embebidos, eliminando la necesidad de configuraciones complejas y conocimientos avanzados en compilación de kernel de Linux, desarrollo de drivers, instalación de librerías manual o compilación de código C/C++.

El repositorio principal de **ezrknpu** integra y automatiza la configuración de dos submódulos clave. Estos submódulos son forks de los repositorios previamente mencionados (`rknn-toolkit2` y `rknn-llm`), llamados **ezrknn-toolkit2** y **ezrknn-llm**, un pequeño juego de palabras con la palabra informal *ez*, proveniente del inglés informal *easy*:

- **ezrknn-llm**: basado en la versión 1.2.1 de `rknn-llm`, facilita la conversión de LLMs y ejecución de estos, así como documentación extendida de cara al usuario y no al desarrollador.
- **ezrknn-toolkit2**: derivado de la versión 2.3.2 de `rknn-toolkit2`. Actualmente solo facilita la instalación de librerías necesarias para usar la NPU con modelos convertidos al formato de Rockchip.

Ambas versiones son las últimas versiones del repositorio oficial y original de Rockchip, a fecha de junio de 2025.

Además, **ezrknpu** incluye una pequeña herramienta adicional, `ntop.sh`, un script que permite monitorear en tiempo real la carga de trabajo de la NPU, proporcionando una idea similar a la del comando `top` en sistemas Linux, pero simplificada.

## Modelos preconvertidos

Para facilitar aún más el proceso de implementación, se ha creado el repositorio `ezrkllm-collection` [29] en Hugging Face, donde se encuentran disponibles modelos LLM ya convertidos al formato RKLLM y listos para ser ejecutados en dispositivos con NPU de Rockchip. Entre los modelos disponibles se incluyen:

- DeepSeek R1 Distill Qwen 1.5B
- Qwen Chat 1.8B
- Microsoft Phi-2 2.7B
- Llama 2 7B y 13B
- TinyLlama v1 1.1B

Estos modelos han sido cuantizados utilizando el esquema `w8a8`, compatible con el RK3588. Es importante saber que algunos modelos son solo compatibles con versiones previas de RKLLM y, por tanto, incompatibles si se usa la última versión.

## Instalación y uso

La instalación de **ezrknpu** está diseñada para ser lo más sencilla posible, permitiendo a los usuarios configurar y ejecutar modelos LLM en menos de 10 comandos. El proceso automatiza la instalación de dependencias, la configuración del entorno y la descarga de modelos preconvertidos, eliminando la necesidad de realizar conversiones manuales o compilaciones complejas.

Este enfoque facilita la adopción de soluciones de inteligencia artificial en el borde, permitiendo a desarrolladores y entusiastas experimentar con modelos avanzados de lenguaje y visión por computador en dispositivos de bajo consumo y costo reducido.

# Capítulo 4

## IMPLEMENTACIÓN

---

Una vez entendido el entorno de desarrollo y el objetivo que se tiene en este TFM, se va a proceder a explicar cómo se han implementado las mejoras y desarrollos necesarios para la correcta, y en especial, simple y fácil, ejecución de LLMs en la Orange Pi 5.

### 4.1. Actualización del driver de NPU

Durante las fases iniciales del proyecto, fue necesario actualizar manualmente el *driver* de la NPU del SoC RK3588, ya que las imágenes de Armbian disponibles en ese momento no incluían una versión suficientemente reciente del controlador `rknpu2`. Esta actualización era esencial para poder utilizar versiones más modernas del framework de Rockchip, especialmente `rknn-llm`.

Para llevar a cabo esta actualización, se optó por compilar una imagen personalizada de Armbian, adaptando el sistema desde cero e integrando un kernel modificado con soporte actualizado para la NPU. El procedimiento se basó en el uso del framework de compilación oficial de Armbian, disponible públicamente en GitHub [23]. También disponen de un kernel propio de Linux adaptado para Rockchip, necesario para actualizar los drivers [25]. Se utilizaron dos *forks* de ambos repositorios para dejar los controladores actualizados:

- `armbian-build-rknpu-updates` — framework de compilación de Armbian modificado [24]. Utiliza el kernel modificado por mi con los controladores actualizados.

- `linux-rockchip-npu-updates` — repositorio con los controladores actualizados en sus versiones 0.9.6 y 0.9.8 [26].

Cabe destacar que los drivers actualizados son provistos por Rockchip en su repositorio `text rknn-llm`, pero ellos no implementan en ningún kernel ni distribución de Linux estos drivers; por ello, se tuvo que hacer este paso [1].

El proceso completo de compilación se realizó de la siguiente manera:

1. Clonar el repositorio modificado:

```
1 git clone https://github.com/Pelochus/armbian-build-rknpu-updates
2 cd armbian-build-rknpu-updates
```

2. Cambiar el repositorio del kernel si fuese necesario. Por defecto, el fork de este trabajo utiliza el kernel con el driver actualizado. En caso de querer usar otro repo con otro kernel, modificar el archivo `userpatches/lib.config`, específicamente las variables `KERNELSOURCE` y `KERNELBRANCH`
3. Ejecutar el script principal de compilación:

```
1 ./compile.sh BOARD=orangepi5 BRANCH=current RELEASE=jammy
  BUILD_MINIMAL=no BUILD_DESKTOP=no KERNEL_CONFIGURE=no
```

Este comando generará una imagen completa de Armbian para la placa especificada (en este caso, para la Orange Pi 5) utilizando el kernel modificado. También se puede usar una configuración interactiva activando `KERNEL_CONFIGURE=yes`, pero no se recomienda.

4. Una vez generada la imagen, se flashea en una tarjeta SD o en el almacenamiento eMMC, y se arranca el sistema.

Durante las primeras pruebas se utilizó el **driver 0.9.6**, que ofrecía soporte básico y permitió validar el correcto funcionamiento de la NPU con modelos de prueba para las primeras versiones de `rknn-llm`. Posteriormente, Rockchip actualizó y, consecuentemente, se integró la versión **0.9.8**, que presentaba mejoras de compatibilidad y estabilidad, así como nuevas funciones, para las nuevas versiones de `rknn-llm`.

Cabe destacar que, a fecha actual, las imágenes oficiales de Armbian para dispositivos con RK3588 ya incluyen el driver `rknpu2` preinstalado en versiones actualizadas. Gracias a esto, la compilación personalizada ya no

es necesaria para usos comunes o demostraciones, y se ha optado por utilizar imágenes oficiales en las fases finales y pruebas del proyecto, además de la correcta estabilidad que trae usar el OS oficial.

No obstante, esta etapa de compilación fue fundamental durante las etapas iniciales de prueba e investigación, ya que permitió explorar el funcionamiento básico de LLMs en la placa y hacer pruebas iniciales, para determinar que efectivamente, este trabajo era viable y prometedor.

## 4.2. Conversión automatizada de LLMs

Convertir modelos LLM en formato compatible con la NPU del RK3588 no es un proceso trivial: requiere instalar múltiples dependencias (Python 3.8–3.12, `git-lfs`, bibliotecas de cuantización, compiladores Cython, y ciertos Python Wheels del propio `rknn-llm`), configurar el entorno y asegurar la compatibilidad entre versiones.

Para simplificar este proceso y evitar errores, se decidió encapsular todo en un contenedor Docker, garantizando entornos reproducibles y aislados.

### 4.2.1. ezrkllm-toolkit

El contenedor Docker `ezrkllm-toolkit` ha sido creado por el autor de este TFM para facilitar la conversión de LLMs al formato de Rockchip. Está disponible en Docker Hub [28], con diferentes tags por versión de `ezrknn-llm`. Este contiene:

- Instancia de `ezrknn-llm (v1.2.1)`, con todos los componentes: `data_quant.py`, `export_rkllm.py`.
- Python (3.8–3.12), `git-lfs`, Cython y dependencias C++ ya instaladas.
- Dependencias y librerías propias de Rockchip para la conversión.

### 4.2.2. Desarrollo del contenedor

El directorio `docker/` del repositorio `ezrknn-llm` contiene los archivos necesarios para construir un contenedor Docker reproducible y optimizado para la conversión de LLMs. Entre ellos destacan:

- **Dockerfile**: define la imagen base (Ubuntu 24 LTS) y copia los siguientes archivos al contenedor.
- **setup.sh**: script que instala dependencias como Python (3.8–3.12, en función de la versión de `ezrknn-llm`), `git-lfs`, `curl`, compiladores y librerías necesarias para `ezrknn-llm`. También clona la versión 1.2.1 de `ezrknn-llm`.
- **entrypoint.sh**: script que se copia dentro de la imagen para establecer el punto de entrada. Tan solo actualiza repo y paquetes, para evitar actualizaciones completas de imagen de Docker.

En los siguientes bloques de código podemos ver los scripts. Primero `setup.sh`:

```

1 #!/bin/bash
2
3 # Dependencies
4 apt update
5 apt install -y python3 pip git curl wget nano sudo apt-utils
6
7 # Follow the branch corresponding to the Docker tag. Updated manually.
8 git clone --branch 1.2.1 --single-branch https://github.com/Pelochus/ezrknn-llm/
9 cd ezrknn-llm
10 # Optional:
11 # git checkout dev-in-progress
12 bash install.sh
13
14 # For converting
15 export BUILD_CUDA_EXT=0
16 pip install /ezrknn-llm/rkllm-toolkit/packages/rkllm_toolkit-1.2.1b1-cp312-cp312-linux_x86_64.whl --break-system-packages
17
18 # Clone LLM
19 cd /ezrknn-llm/examples/DeepSeek-R1-Distill-Qwen-1.5B-Demo/ export
20 git clone https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
21
22 # Done here to avoid cloning full repository for the Docker image
23 apt install -y git-lfs
24
25 # Needed
26 DEBIAN_FRONTEND=noninteractive apt install -y python3-tk
27
28 # cd DeepSeek-R1-Distill-Qwen-1.5B
29 # git lfs pull

```

Aquí `entrypoint.sh`:

```

1 #!/bin/bash
2
3 # Always useful
4 apt update && apt full-upgrade -y

```



```
5
6 # Update repo without updating image
7 cd /ezrknn-llm
8 git pull
9
10 bash
```

Este diseño modular permite separar:

1. Preparación de ambiente limpio ( `Dockerfile` ).
2. Proceso de instalación ( `setup.sh` ).
3. Actualizaciones básicas sin actualizar imagen y contenedor con ( `entrypoint.sh` ).
4. Por último, el contenedor es publicado en Docker Hub para mayor facilidad de uso, con diferentes tags por versión

### 4.2.3. Flujo de conversión

Aquí un ejemplo corto de flujo de conversión de un LLM. Se asumen los prerequisites (PC Linux x86, Docker instalado, suficiente RAM...)

```
1 docker run -it --rm pelochus/ezrkllm-toolkit:latest bash
2
3 # Dentro del contenedor:
4 cd /ezrknn-llm/examples/DeepSeek-R1-Distill-Qwen-1.5B_Demo/export
5 GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/Pelochus/deepseek-R1-
  distill-qwen-1.5B
6 cd deepseek-R1-distill-qwen-1.5B && git lfs pull
7
8 # Convertir
9 python3 data_quant.py -m ./
10 python3 export_rkllm.py # Modificar dentro de este archivo la ruta al LLM
11
12 # Copiar el archivo rkllm al SBC. Hay varias formas: docker cp, scp, SFTP...
13
14 # Por ultimo, ejecutar!
15 rkllm DeepSeek-R1-Distill-Qwen-1.5B_W8A8_RK3588.rkllm 1024 1024
```

Toda esta funcionalidad se ampliará y detallará en el Apéndice B, donde se explica con más detalle todo esto, incluidos prerequisites.

### 4.2.4. Detalles del proceso

- **Ejecución de contenedor:** Se ejecuta el contenedor, se descarga la imagen y se accede a la carpeta donde se realiza la conversión.

- **Cuantización primero:** `data_quant.py` genera archivos de calibración basados en ejemplos reales.
- **Conversión final:** `export_rkllm.py` produce un `.rkllm` cuantizado en formato W8A8, compatible con RK3588.
- **Ejecución del modelo:** `rkllm` permite testear el modelo, pero deberá ser en la placa correspondiente.
- **Requisitos de RAM:** se recomienda disponer de al menos 3–4× el tamaño del modelo en memoria, por los picos de consumo durante cuantización y conversión (por ejemplo, 16 RAM para modelos de 4 GB). Alternativamente, se puede usar swap, pero ralentiza el proceso y podría dañar el almacenamiento.

Este enfoque basado en Docker ha permitido acelerar el desarrollo, reducir errores de entorno y asegurar reproducibilidad al convertir modelos LLM para su ejecución en la NPU del RK3588.

## 4.3. Automatización de la instalación

Para evitar la complejidad de configurar manualmente entornos con múltiples dependencias, se desarrolló un sistema de instalación automatizada usando scripts Bash. Todo el flujo se encuentra centralizado en el script `install.sh` del repositorio `ezrknpu`, que a su vez invoca otros scripts de los repositorios submódulo.

### 4.3.1. Bibliotecas

Las bibliotecas compartidas (`.so`) y paquetes Python (`.whl`) necesarios para RKNN Toolkit 2 y RKLLM se gestionan automáticamente:

- Cada submódulo se gestiona sus propias bibliotecas con sus respectivos `install.sh`
- También instalan con `pip` sus wheels correspondientes
- El archivo principal de instalación se encuentra en `ezrknpu`, que llamará automáticamente al resto

- Estas bibliotecas son necesarias para usar los LLMs o aplicaciones de CV desde C/C++ o Python, según aplicación y biblioteca usada, así como interfaces para el driver del kernel.

### 4.3.2. Paquetes

Similarmente, estos scripts se ocupan de la instalación de paquetes con `apt` y `pip`:

- Usa `apt install` para paquetes esenciales (`git`, `python3-pip`, `cmake`, `git-lfs` ...) en una sola llamada, minimizando prompts. - Instala dependencias Python definidas en `requirements.txt`, que incluyen `onnx`, `numpy`, y herramientas de cuantización específicas.

Esto permite el uso fácil y transparente de cara al usuario menos experto, facilitando instalación y actualización, así como centralización de cara a, por ejemplo, instalación en clústeres de SBCs.

### 4.3.3. Programa rkllm

El ejecutable `rkllm` (parte de `ezrknn-llm`) se compila automáticamente dentro del flujo de instalación:

- El código de este ejecutable es parte del repositorio original, no obstante, presenta las siguientes desventajas:
  - Lenguaje y comentarios en idioma chino
  - No compila automáticamente, requiere descargar y usar cross-compilers (desde x86 u otras arquitecturas)
  - Es básico y apenas aporta información, el código fuente es mejorable
- Por ello, el fork de este TFM aporta lo siguiente:
  - Código fuente mejorado, con optimizaciones leves y traducciones de comentarios al inglés
  - Mejoras de usabilidad del programa
  - Compilación automática con la versión de `gcc` del sistema en la Orange Pi 5, evitando instalación de cross-compilers

- Instalación automática a `/usr/bin` para poder ser ejecutado en cualquier momento de manera fácil como `rkllm`
- Breve ejemplo de uso en la documentación principal de `ezrknpu`, para facilitar el primer uso
- Medida de rendimiento a través del programa, concretamente, medida de tiempo de cada respuesta, número de tokens y, gracias a ambos, token/s. Esta medida es crítica de cara al Capítulo 5, donde se hace benchmarking y comparativas contra CPU y GPU.

## 4.4. Uso de LLMs

El ejecutable `rkllm` permite llevar modelos LLM cuantizados (formato `.rkllm`) al dispositivo con NPU, ofreciendo una interfaz sencilla para ejecutar inferencia desde la línea de comandos o integrarlos en aplicaciones.

Aunque la interfaz base está pensada para terminal, existen varias herramientas adicionales para facilitar su uso, incluyendo un servidor web y demostraciones interactivas.

### 4.4.1. Uso básico desde CLI

El uso más directo de `rkllm` es de la siguiente forma. Previamente, se recomienda seguir el Apéndice A para más información:

```
1 rkllm <modelo>.rkllm <longitud_contexto> <longitud_salida>
```

Por ejemplo:

```
1 rkllm DeepSeek-R1-Distill-Qwen-1.5B_W8A8_RK3588.rkllm 1024 256
```

Esto carga el modelo, inicializa el context window y muestra la salida generada. Internamente se utiliza el runtime C/C++ vinculado a la NPU, gestionando memoria y ejecución optimizada para RK3588. En el código de C++ de este programa, se incluyen más parámetros avanzados ajustables, de cara a los usuarios más avanzados.

Una vez hayamos ejecutado el LLM, veremos lo siguiente desde la terminal:

```
FW-RP-1 Root Root 991 May 28 17:58 README.md
orangePi5:deepseek-R1-distill-qwen-1.5B:# rkllm DeepSeek-R1-Distill-Qwen-1.5B_W8A8_RK3588.rkllm 1024 1024 <main >
RKLLM starting, please wait...
I rkllm: rkllm-runtime version: 1.2.1b1, rknpu driver version: 0.9.8, platform: RK3588
I rkllm: loading rkllm model from DeepSeek-R1-Distill-Qwen-1.5B_W8A8_RK3588.rkllm
I rkllm: rkllm-toolkit version: 1.2.1b1, max_context_limit: 4096, npu_core_num: 3, target_platform: RK3588
I rkllm: Enabled cpus: [4, 5, 6, 7]
I rkllm: Enabled cpus num: 4
I rkllm: system_prompt:
I rkllm: prompt_prefix: <| User | >
I rkllm: prompt_postfix: <| Assistant | ><think>\n
rkllm init success

***** Pelochus' ezrkllm runtime *****

[0] Welcome to ezrkllm! This is an adaptation of Rockchip's rknn-llm repo (see github.com/airockchip/rknn-llm) for running LL
Ms on its SoCs' NPUs.

[1]
To exit the model, enter either exit or quit

[2]
More information here: https://github.com/Pelochus/ezrknpu
[3]
Detailed information for devs here: https://github.com/Pelochus/ezrknn-llm

*****

I rkllm: reset chat template:
I rkllm: system_prompt:
I rkllm: prompt_prefix: <| User | >
I rkllm: prompt_postfix: <| Assistant | >

You: |
```

Figura 4.1: Ejemplo de ejecución de un modelo desde CLI

#### 4.4.2. Servidor web para uso desde navegador

El repositorio oficial de `rknn-llm` viene con un ejemplo en la carpeta

`examples/rkllm_server_demo`

que implementa un servidor HTTP usando `Flask`, y puede integrarse con `Gradio` para ofrecer interfaces de chat.

Esto inicia un endpoint REST y una interfaz web accesible vía navegador en `http://0.0.0.0:8080`, facilitando prompts desde el navegador sin SSH.

Es importante saber que esto es parte del repositorio original y no es parte de este TFM, se ha mantenido sin ningún tipo de problema la compatibilidad.

Desafortunadamente, debido a la naturaleza de este proyecto original por Rockchip, no existe compatibilidad directa con soluciones más extendidas como `Ollama` o `RamaLama`.

Más información en el siguiente enlace:

`github.com/airockchip/rknn-llm/tree/main/examples/rkllm_server_demo`



# Capítulo 5

## RESULTADOS Y PRUEBAS

---

Tras el trabajo de integración y desarrollo realizado, se ha conseguido ejecutar modelos LLM acelerados por la NPU del RK3588 en una Orange Pi 5. Este capítulo presenta los resultados obtenidos, el rendimiento observado en diferentes configuraciones, comparativas contra otros entornos de ejecución y las limitaciones del sistema.

### 5.1. Resultados

#### 5.1.1. Funcionamiento conseguido

Se ha logrado ejecutar correctamente modelos de lenguaje (LLMs) cuantizados y optimizados sobre la NPU del RK3588. Mediante las modificaciones realizadas al framework original de Rockchip, se ha conseguido ejecutar LLMs de manera considerablemente sencilla en hardware especializado.

Los modelos usados se han cargado sin errores críticos, las inferencias se completan con éxito y las respuestas generadas son coherentes, dentro de las limitaciones del tamaño del modelo y de la precisión reducida de la NPU.

#### 5.1.2. Facilidad de uso

Uno de los objetivos del proyecto era reducir la barrera de entrada para ejecutar LLMs en dispositivos de bajo consumo. Se ha logrado que el proceso completo, desde una instalación limpia del sistema operativo hasta la ejecución de un LLM por NPU, pueda completarse con los siguientes pasos principales:

- Instalar una imagen estándar de un sistema operativo basado en Ubuntu o Debian (por ejemplo, Armbian) en una Orange Pi 5 o dispositivo similar.
- Ejecutar un script que automatiza la instalación de bibliotecas, drivers y programas necesarios, con un único comando.
- Descargar un modelo LLM con un solo comando.
- Ejecutar el binario `rkllm` utilizando el modelo previamente descargado, también con un único comando.

Este proceso, que como explicado a lo largo de este trabajo, implicaba utilizar SOs compilados por uno mismo (kernel incluido), entender la documentación escasa, compilar el binario, transformar modelos (requiere grandes cantidades de procesamiento y memoria para ello, y otro PC)... se ha reducido a tan solo ejecutar (asumiendo que no hay errores, versiones incompatibles del SO...) **3 comandos** y esperar un poco en cada uno, tras el cual ya se puede charlar con el LLM descargado.

Cómo es lógico, previamente se necesitaría de la mayoría del esfuerzo dedicado en este trabajo para conseguir el mismo resultado desde 0, es decir, decenas si no cientos de comandos, así como investigación y unos conocimientos mínimos, tanto de software, como Linux, driver development, o LLMs, así como recursos para la conversión.

## 5.2. Pruebas de rendimiento

Se han realizado pruebas de rendimiento centradas en los tokens/s, el uso de recursos (CPU, RAM y NPU) y el consumo energético del dispositivo.

Para estas pruebas se utiliza el modelo `DeepSeek-R1-Distill-Qwen-1.5B`, sin ningún otro software ejecutándose en la Orange Pi, aparte de demonios en background como `sshd` y los mínimos de un sistema operativo Linux CLI.

### 5.2.1. Consumo energético

Uno de los puntos fuertes del uso de una Orange Pi 5 con NPU para inferencia es su bajo consumo energético.

Para ello, se usaron dos dispositivos de medición de consumo:



- **Medidor USB (KWS-066C):** mide directamente desde el USB-C de la Orange Pi 5, incluyendo voltaje, amperaje... la Orange Pi 5 consume 5V, así que nos interesa más el amperaje y consumo en vatios.
- **Medidor de enchufe pared:** mide directamente desde la PSU. Se utiliza en este trabajo como contraste de menor precisión y exactitud al medidor USB.

Pueden comprobarse los resultados en los siguientes vídeos de YouTube:

- Consumo con 2 LLMs:

<https://youtube.com/shorts/glEB7v1KDEY?feature=share>

- Consumo con 2 LLMs (incluyendo enchufe pared):

<https://youtube.com/shorts/uZVooBSn7X4?feature=share>

En las pruebas realizadas se observaron los siguientes valores:

Situación	USB	Enchufe
Consumo en reposo (idle, after boot)	~ 1.75 W	~ 2.5 W
Consumo durante inferencia por NPU	~ 5.7 W	~ 7.5 W
Picos máximos observados	~ 5.9 W	~ 7.8 W
Inferencia por NPU (2 LLMs)	~ 7 W	~ 8.8 W
Picos máximos observados (2 LLMs)	~ 7.2 W	~ 9 W

**Tabla 5.1:** Comparativa de consumo energético usando dos dispositivos de medida

Se puede ver además, dos filas indicando el consumo al ejecutar 2 LLMs simultáneamente. El rendimiento en este caso bajaba a unos 7-8 token/s (ver Tabla 5.3) cada LLM, mayor en conjunto respecto a un solo LLM.

Los consumos son considerablemente bajos, como podemos ver. Cabe destacar que la diferencia entre el medidor de enchufe y USB es considerable debido a las pérdidas del adaptador AC-DC principalmente.

Los consumos tan bajos convierten al sistema en una opción muy competitiva frente a soluciones que requieren CPUs o GPUs de escritorio, especialmente en contextos donde el consumo energético es crítico, como sistemas embebidos, IoT o edge computing. Además, permite uso elevado de LLMs en local sin afectar gravemente a la factura de electricidad de un usuario común

que quiera usar LLMs *on a budget* como se suele decir en inglés, es decir, con bajo presupuesto.

Asumiendo peor caso, digamos, 10 W (0.01 kW) constantes y un precio de 0,25 €/kWh (mayor al precio consultado en cualquier tramo horario en <https://tarifaluzhora.es/> el 5 de agosto de 2025), el precio total de electricidad sería de:

$$\text{Coste diario} = 0.01 \text{ kW} \times 24 \text{ h} \times 0.25 \frac{\text{€}}{\text{kWh}} = 0.06 \text{ €} \quad (5.1)$$

Lo cual da un consumo de tan solo **22 euros** al año. Una cifra impresionante, especialmente teniendo en cuenta que se está asumiendo un escenario incluso peor de lo normal. Podemos asumir tranquilamente **un precio inferior a 10 euros al año en situaciones normales**, al menos si se mantiene el sistema conectado en todo momento, para tener disponibilidad inmediata del LLM.

### 5.2.2. Comparativa contra CPU

Se comparó el tiempo medio de inferencia por token de un modelo equivalente ejecutado por CPU (usando `ollama`) frente a la ejecución acelerada por NPU. El modelo utilizado es el mismo, pero **cuantizado a 4 bits**:

<https://ollama.com/erwan2/DeepSeek-R1-Distill-Qwen-1.5B>

Los resultados fueron los siguientes:

Plataforma	Token/s	Uso CPU (%)
NPU	~ 15 token/s	15–35
NPU, 2 LLMs	~ 6.5-9 token/s por LLM (16-18 ambos)	25–45
CPU	~ 9 token/s	99–100

**Tabla 5.2:** Comparativa de rendimiento entre NPU y CPU RK3588

En adelante, una captura de la ejecución por CPU. Podemos ver un uso máximo de todos los núcleos de la CPU:

Como puede observarse, la aceleración por NPU consigue una **mejora superior a x1.5 en velocidad y reduce considerablemente la carga de la CPU**, permitiendo así el uso de recursos para otros procesos en paralelo. Todo esto, además, con una cuantización mayor y, por ende, peor rendimiento del modelo (tan solo hay que ver la respuesta al primer prompt en la Figura 5.1 respecto a Figura A.2).

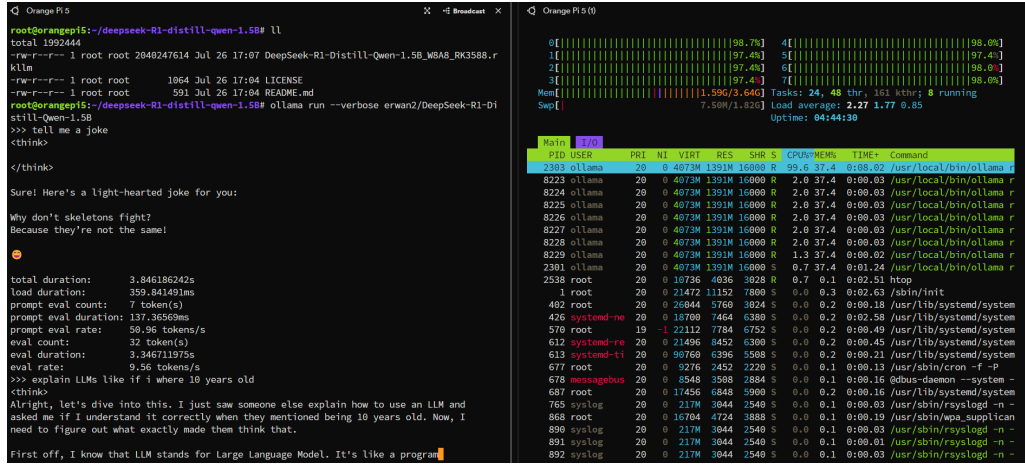


Figura 5.1: Ejecución de modelo en CPU

### 5.2.3. Comparativa contra GPU

Si bien este trabajo no ha implementado pruebas con GPU directamente, se puede recurrir a datos publicados por el equipo de MLC [30] para contextualizar el rendimiento de la GPU del RK3588 en Llama 3. Los resultados obtenidos fueron los siguientes:

- Llama-2-7B-chat-q4f16: ~2.5 tokens/s
- Llama-3-8B-Instruct-q4f16: ~2.3 tokens/s
- RedPajama-3B-chat-q4f16: ~5 tokens/s
- Llama-2-13B-q4f16 (en Orange Pi 5+): ~1.5 tokens/s

En adelante, una comparativa del rendimiento de cada procesador de la Orange Pi 5 medido en token/s por cada mil millones (1B) parámetros. Calculado como resultado de token/s multiplicado por B parámetros del modelo, de otro modo, siguiendo esta fórmula:

$$\text{Rendimiento por 1B parámetros} = \frac{\text{tokens} \cdot \text{parámetros}}{\text{s} \cdot 1\text{B}} \quad (5.2)$$

Donde 1B representa 1,000,000,000 (mil millones) de parámetros, o sea, *billion* en el idioma inglés.

Procesador	Tokens/s por 1B parámetros
GPU	15–18.4
NPU	22.5–25
CPU	~ 14 (4 bits quant)

**Tabla 5.3:** Comparativa entre procesadores del RK3588

Los valores de rendimiento muestran una clara ventaja a favor de la NPU frente a la GPU Mali y la CPU del RK3588. La GPU se puede utilizar como segundo acelerador, para ejecutar modelos secundarios más pequeños como en el siguiente ejemplo:

El rendimiento en términos de tokens por segundo por cada billón de parámetros es razonable ver como la NPU supera al resto, aunque levemente a la GPU. La CPU, como es esperable, es inferior, aunque sorprendentemente no por demasiado. Esto quizá se deba a que el test que se hizo con CPU fue con una cuantización mayor, permitiendo mejor rendimiento.

Esta diferencia es especialmente notable al escalar el número de instancias de inferencia en paralelo: la CPU rápidamente alcanza su máximo de utilización, mientras que la NPU puede seguir ejecutando múltiples modelos sin afectar dramáticamente el rendimiento individual ni el consumo eléctrico.

### 5.3. Limitaciones

Pese a los resultados obtenidos, existen limitaciones claras en el uso de la NPU del RK3588 para la ejecución de LLMs:

- **Modelos soportados:** Sólo es posible usar modelos previamente convertidos a un formato compatible con las bibliotecas propietarias de Rockchip (RKNN). El proceso de conversión requiere una máquina x86 con drivers y herramientas específicas.
- **Tamaño de modelo limitado:** Los modelos deben caber en la RAM del sistema, y además estar cuantizados a 8 bits, lo que puede afectar la calidad de las respuestas.
- **Documentación escasa:** El soporte por parte del fabricante es limitado, y las herramientas disponibles no siempre son estables o bien documentadas.

- **Compatibilidad con otro software:** La naturaleza semi-abierta del repositorio original de Rockchip, especialmente en cuanto a APIs se refiere, fuerza a no haber una gran compatibilidad con otros software como puede ser Ollama, llama.cpp, LangChain...

Aun así, estas limitaciones se compensan con el bajo coste del hardware, la facilidad de despliegue (gracias a este trabajo) y el ahorro energético.

Los resultados obtenidos demuestran que es posible ejecutar modelos de lenguaje de tamaño reducido en dispositivos embebidos gracias a aceleradores como la NPU del RK3588. Si bien hay limitaciones de potencia y compatibilidad, se logra una ejecución eficiente en cuanto a recursos y consumo, haciendo viable su aplicación en sistemas edge, domótica o agentes embebidos.



# Capítulo 6

## CONCLUSIONES

---

En este último capítulo, se extraen conclusiones en base a los resultados obtenidos. Se incluye, por supuesto, un análisis de los objetivos y planificación, así como posibles aplicaciones, para poder extraer conclusiones desde un punto de vista de la ingeniería en general, además de los resultados más tangibles y observables a simple vista.

Se incluye un ejemplo de aplicación real por otro autor (basado en este trabajo), así como diversas mejoras y trabajos futuros.

### 6.1. Análisis de los objetivos y planificación

Es importante observar hasta qué punto se cumplieron las metas establecidas. Por ello, a continuación se muestra un pequeño análisis de si se cumplió cada objetivo, y hasta qué punto:

- **OBJ-1:** introducción, objetivos, planificación y preparación general. Es evidente que se cumplió este objetivo, al ser introductorio y de carácter anterior al resto de objetivos.
- **Capítulo 2 (OBJ-2):** fundamentos. Se cumplió sin problemas tras una buena base de referencias e información en internet.
- **Capítulo 3 (OBJ-3):** entorno y herramientas. También se completó de forma satisfactoria. Se analizaron los frameworks oficiales (rknn-toolkit2, rknn-llm) y se sentaron las bases para el posterior desarrollo.
- **Capítulo 4 (OBJ-4):** implementación. El objetivo se cumplió de forma destacada. Se implementaron scripts, instaladores automáticos, herramientas propias, así como un contenedor Docker funcional y versátil





- **Edge computing e IoT:** se abre la posibilidad de integrar capacidades avanzadas de lenguaje en dispositivos con limitaciones energéticas, como hubs domésticos, routers inteligentes, dispositivos de voz o robots personales.
- **Aplicaciones educativas:** el sistema puede ser usado como entorno de formación en inteligencia artificial embebida, permitiendo a estudiantes y makers experimentar con modelos LLM sin necesidad de GPUs o acceso a la nube.
- **Entornos offline o de baja conectividad:** la ejecución local permite desplegar soluciones en entornos aislados (rurales, militares, emergencias), donde no se dispone de infraestructura externa fiable.
- **Potenciación del open source:** este trabajo focaliza considerablemente en conseguir ejecución de LLMs open source, facilitando y fomentando la creación de LLMs con conocimiento abierto, para el mayor y mejor avance de la ciencia.
- **Fomento de la aceleración por HW de IA:** utilizar NPUs en este trabajo demuestra que, efectivamente, la aceleración por hardware es considerablemente útil para conseguir mejores rendimientos y consumos cuando otras formas de hardware o software son incapaces.
- **Bajo coste:** el hecho de que se puedan ejecutar LLMs en un hardware inferior a 100 euros con consumos de electricidad bajísimos es impresionante.

Además, el sistema puede combinarse con otros modelos, al poseer tres procesadores, es posible ejecutar varios modelos simultáneamente sin sobrecargar la NPU.

Un ejemplo considerablemente interesante de aplicación de este trabajo, donde un usuario de reddit y GitHub consiguió ejecutar un asistente de voz utilizando los tres procesadores, donde:

- **NPU:** Ejecutaba el LLM que generaba el texto. Esta basado en el proyecto de este TFM, cuando aun no estaba completo (versiones previas).
- **GPU:** El text-to-speech lo ejecutaba este procesador.
- **CPU:** Finalmente este ejecutaba el ASR (Audio Speech Recognition)

En este caso, en vez de usar una Orange Pi 5, se utilizó una Rock 5B, que usa el mismo SoC. La variante era de 8GB de RAM, ya que con 4 no es suficiente (según el creador los 8 GBs van muy justos para ejecutarlo todo). Cabe destacar que esto se consiguió en 2024, con hardware de 2022, por tanto, como autor de este TFM, confío en que se pueda conseguir mucho más en tiempos recientes con la mejora de LLMs, hardware...

Recomiendo encarecidamente echar un vistazo al video en el post original:

[https://www.reddit.com/r/LocalLLaMA/comments/1hryfs6/%C2%B5localglados\\_offline\\_personality\\_core/](https://www.reddit.com/r/LocalLLaMA/comments/1hryfs6/%C2%B5localglados_offline_personality_core/)

Más información en el repositorio de GitHub [27].

## 6.3. Trabajos y mejoras futuras

Aunque el sistema actual es funcional y ha demostrado su utilidad, hay múltiples líneas de mejora y ampliación:

- **Mejor compatibilidad con modelos recientes:** debido a los rápidos avances en LLMs, sería interesante adaptar y probar nuevos modelos open source más recientes (especialmente, el recién anunciado GPT-OSS).
- **Automatización completa de la conversión y despliegue:** la integración de herramientas como un frontend web o CLI más avanzada podría facilitar aún más el proceso a usuarios menos técnicos.
- **Benchmarking en otras plataformas:** comparar el rendimiento del sistema en otras placas SBC (como Jetson Nano, Jetson Orin, Raspberry Pi 5 o RK3576) ayudaría a caracterizar mejor el mercado y guiar futuras elecciones de hardware.
- **Soporte para tareas más complejas:** explorar el uso del sistema en aplicaciones reales, como generación de código, análisis semántico o chat multi-turno, evaluando los límites funcionales de cada modelo y su viabilidad en la NPU.
- **Convertir más LLMs:** debido al alto coste de convertir modelos, para este TFM solo se ha convertido uno, y de bajo consumo. Existen más LLMs disponibles que usar.

- **Otros tipos de LLMs:** como generación de imágenes, reconocimiento de objetos... Actualmente el repo original implemento recientemente VLLMs.
- **Otros procesadores de Rockchip:** que son compatibles pero se han obviado para este trabajo.
- **Mejora de la facilidad de uso:** por ejemplo, utilizando servidor web, permitiendo el uso sin utilizar la terminal (instalación desde, por ejemplo, la app store de Ubuntu), entre otras muchas formas. Incluso utilizar un OS modificado en base a Armbian para que venga todo preinstalado.

En una línea más ambiciosa, podrían explorarse también:

- **Implementaciones multimodales:** combinación de texto con imagen o audio, por ejemplo, con modelos como CLIP, Whisper o LLaVA adaptados al entorno embebido. Existen ciertos ejemplos
- **Proyectos colaborativos:** integración del sistema en entornos maker, educativos o de software libre, aprovechando su código abierto y baja barrera de entrada.
- **Integración con otras herramientas:** como Ollama, VSCode para programar con un LLM local...
- **Integración con MCP servers:** para poder dar más contexto. Esto, por la naturaleza del coste del contexto, requerirá mejores NPUs y mayores cantidades de RAM.
- **Clústers de SBCs/LLMs:** con varios SBCs, se podría hacer un clúster que pueda utilizar varios LLMs de manera coordinada a demanda, similar a un ChatGPT para varias personas en local.

## 6.4. Conclusión

En conclusión, este trabajo ha demostrado que es posible ejecutar modelos de lenguaje de gran tamaño (LLMs) de manera eficiente en dispositivos embebidos como la Orange Pi 5, gracias al uso de la NPU integrada en el SoC RK3588.

No solo se ha logrado ejecutar estos modelos, sino que además se ha construido un sistema accesible y automatizado, que reduce significativamente las barreras de entrada para usuarios sin conocimientos avanzados en kernel, drivers o frameworks de IA.

El consumo energético, la facilidad de uso y el rendimiento alcanzado en inferencias locales muestran que la ejecución de modelos LLM en dispositivos edge ya no es una utopía, sino una realidad viable en ciertos escenarios. Es especialmente cierto esto, cuando tenemos en cuenta que se está utilizando hardware salido al mercado al mismo tiempo que la primera versión de ChatGPT que revolucionó la IA, es decir, sin tener en cuenta el diseño y potencia de NPUs para el uso de LLMs. Hoy en día, la combinación de ello más la tecnología más avanzada, permitirá resultados mucho más prometedores.

Finalmente, cabe destacar que el trabajo ha tenido cierta repercusión práctica: el proyecto se ha publicado en GitHub bajo el nombre **ezrknpu** con un impacto notable (+170 estrellas en GitHub, miles de visualizaciones del perfil del autor de este trabajo).

Con modelos preconvertidos disponibles en Hugging Face para el uso facilitado de este, tutoriales en medios especializados del uso desde 0 de esta herramienta e incluso se han compartido videos demostrativos en plataformas como Reddit (incluso se ha creado un subreddit dedicado, con alrededor de 1000 miembros) y YouTube, donde ha sido bien recibido por la comunidad técnica interesada en IA y sistemas embebidos.

Para más inri, existen ejemplos reales de aplicaciones de este proyecto, como la previamente mencionada en la que se usa tanto CPU como GPU y NPU para un asistente de voz completamente en local, pero definitivamente muy capaz.

Se espera que esta contribución sirva de base para futuros desarrollos más ambiciosos, acercando la inteligencia artificial generativa a todo tipo de dispositivos, y promoviendo una adopción descentralizada y accesible de estas tecnologías, tanto desde el punto de vista de software open source, como de hardware económico y accesible.

Incluso más allá, se espera motivar el uso de LLMs en local, sin dependencia de la nube, con menores consumos y mayor optimización que permitirán, como efecto alternativo pero definitivamente deseado, un menor consumo de energía en un mundo donde esta escasea.

En definitiva, este trabajo ha conseguido avanzar el panorama de los LLMs desde un punto de vista de la ingeniería, es decir, desde la aplicabilidad real y el coste, consiguiendo reducir costes, aplicar IA en hardware que pocos años atrás era totalmente incapaz de ejecutar, democratizar el uso de LLMs

sin depender de la nube, suscripciones, modelos cerrados o dificultad alta de instalación y uso, e incluso conseguir una reducción del uso de energía, debido a una eficiencia altísima.



# Capítulo A

## INSTALACIÓN AUTOMATIZADA DE EZRKN-PU

---

En adelante, una breve guía de cómo instalar ezrknpu y ejecutar un primer LLM. Existe una guía similar en el `README` del repositorio principal [20].

### A.1. Requisitos previos

- **Sistema operativo:** Linux basado en Debian o Ubuntu, con el driver 0.9.8 de la NPU. Se recomienda utilizar **Armbian** con soporte actualizado para RK3588, ya que incluye drivers NPU recientes (versión mínima de kernel 6.10 o superior)
- **Placas compatibles:** Cualquier placa con SoC RK3588 e, idealmente, pueden ejecutar Armbian con soporte completo. En caso contrario, deberá encontrar un OS con el driver 0.9.8 o más reciente si existe. Algunas placas compatibles y recomendadas con soporte para Armbian:
  - Orange Pi 5 y variantes (Plus, Pro...)
  - Radxa Rock 5 y variantes (5A, 5B...)
  - FriendlyElec NanoPi R6S/R6C
- **Python 3.12:** Idealmente, el sistema instalará Python 3.12. Si se usa un sistema basado en Debian 12 o Ubuntu 24.04 LTS, esto será así por defecto. En caso contrario, el usuario deberá instalar Python 3.12 manualmente. Las últimas versiones de Armbian instalan Python 3.12 por defecto.
- **Acceso de superusuario** (`sudo`) y conexión a Internet.

- **Hardware adicional:** opcionalmente, una unidad SSD NVMe o tarjeta microSD de alta velocidad podrían ayudar a la carga inicial de los LLMs, así como al almacenamiento de varios de estos.

## A.2. Instrucciones de instalación

Para instalar, solo hay que ejecutar el comando:

```
1 curl https://raw.githubusercontent.com/Pelochus/ezrknpu/main/install.sh |  
  sudo bash
```

Si se tiene la placa y sistema operativo correcto, no debería fallar. Como mucho, se deberá de instalar `curl`, en caso de que el sistema operativo sea mínimo y no lo incluya por defecto. Ejecutar `sudo apt install curl`.

El comando de instalación realiza:

- Instalación de paquetes esenciales, Python 3.12, compiladores y librerías del sistema.
- Clonar el repositorio `ezrknpu`. Incluye `ntop.sh`, para poder ver el uso de la NPU. Ver Apéndice C.
- Descarga e instalación de `rknn-toolkit2` y `rknn-llm`.

Por último, opcional pero recomendado, reiniciar con `sudo reboot`

## A.3. Verificación de la instalación

La forma más simple de verificar la instalación es ejecutando el comando `rkllm` y comprobar que salga el mismo output:

```
1 orangepi5:~: $ rkllm  
2 Usage: rkllm model_path max_new_tokens max_context_len
```

Esto significa que el ejecutable para usar los LLMs se ha compilado correctamente, lo que implica que las librerías también se han instalado correctamente.



## A.4. Ejecución de un LLM

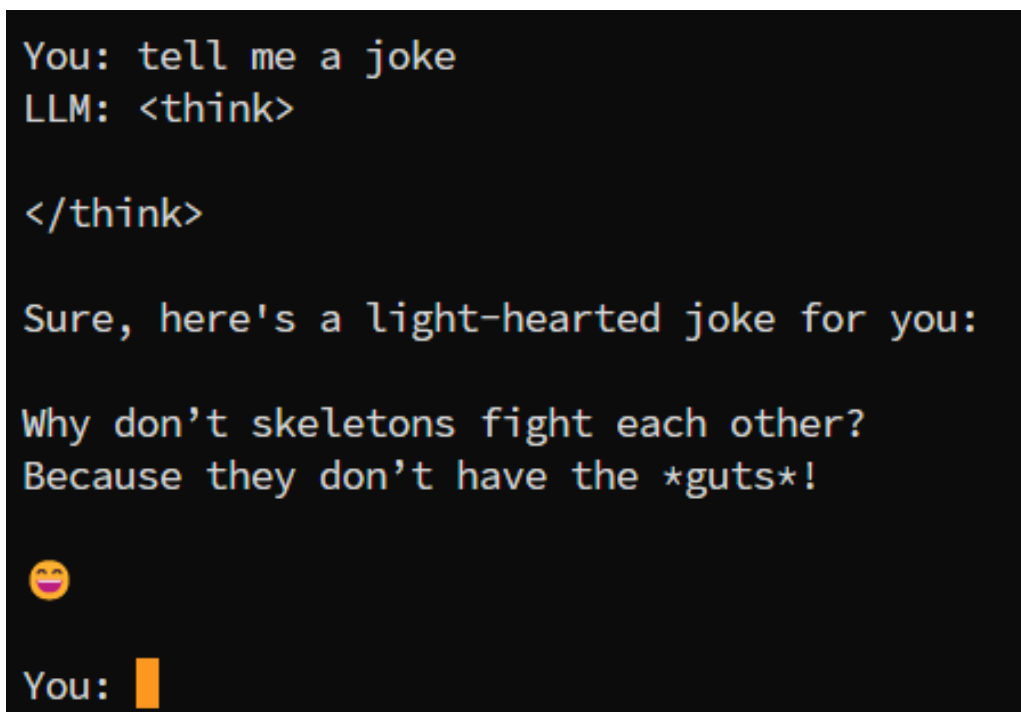
Una vez instalado todo lo necesario, ejecutar un LLM es trivial. Se asume desde esta guía que se usa la última versión de `ezrknn-llm` y de Armbian a junio de 2025, que son 1.2.1 y 25, respectivamente. En caso contrario, puede que todo este apéndice y guía este desactualizado, especialmente en cuanto a LLMs convertidos se trata.

```
1 GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/Pelochus/deepseek-R1-  
  distill-qwen-1.5B # Running git lfs pull after is usually better  
2 cd deepseek-R1-distill-qwen-1.5B && git lfs pull # Pull model  
3 rkllm DeepSeek-R1-Distill-Qwen-1.5B_W8A8_RK3588.rkllm 1024 1024 # Run!
```

Con los comandos anteriores, se descarga y ejecuta una versión básica de DeepSeek R1, destilada sobre Qwen 1.5B.

Este ejemplo funciona en cualquier SBC con chip RK3588 y con 4GB de RAM o superior, siempre que se cumplan los requisitos anteriores de instalación, sistema operativo...

Ya que este LLM es simple, sus respuestas son deterministas. Un ejemplo simple puede ser el siguiente:



```
You: tell me a joke  
LLM: <think>  
  
</think>  
  
Sure, here's a light-hearted joke for you:  
  
Why don't skeletons fight each other?  
Because they don't have the *guts*!  
  
😄  
  
You: █
```

Figura A.1: Ejemplo simple de primera ejecución de un LLM y prompt

Como curiosidad, aquí está el resultado del mismo prompt a DeepSeek. Esto fue probado el 6 de junio de 2025 en ambos modelos, tanto el DeepSeek R1 oficial desde su página web como el modelo convertido para este trabajo:

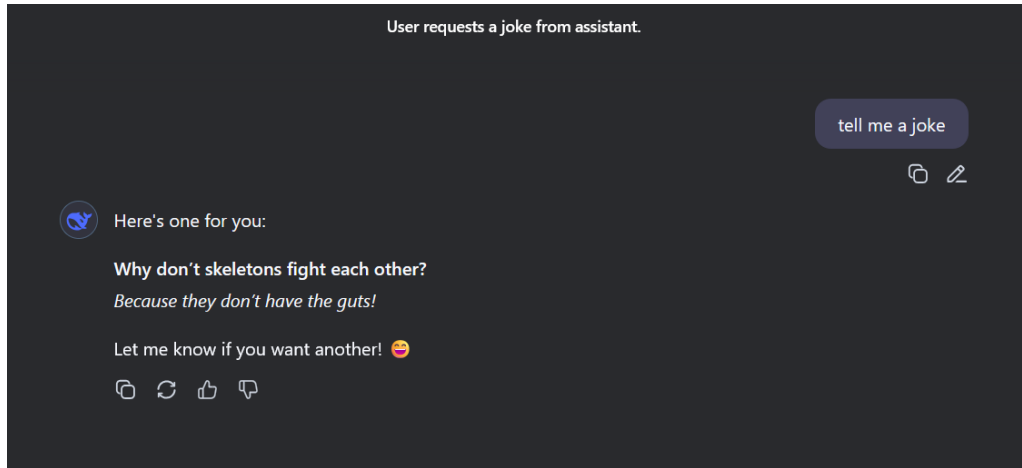


Figura A.2: Prompt “tell me a joke” en DeepSeek R1

# Capítulo B

## CONVERSIÓN Y CUANTIZACIÓN DE LLMs

---

La conversión de modelos LLM para su ejecución en NPUs de Rockchip se realiza típicamente en un PC con arquitectura x86, aprovechando el contenedor de Docker desarrollado en el proyecto `ezrknn-llm`, una herramienta mantenida el autor de este TFM basada en la versión 1.2.1 de `rknn-llm` [1, 21].

### B.1. Conversión

1. **Entorno x86 con Docker:** se utiliza un contenedor Docker llamado `ezrkllm-toolkit` que incluye la versión 1.2.1 de `rknn-llm`. Este incluye todos los prerequisites para simplemente convertir en los menores pasos posibles [21].
2. **Clonación del modelo:** dentro del contenedor se clona el repositorio de un modelo compatible (por ejemplo, DeepSeek-R1-Distill-Qwen-1.5B), descargando con `git-lfs`, también incluido en el Docker por defecto.

3. **Cuantización previa:** se ejecuta

```
python3 generate_data_quant.py -m ruta/al/modelo
```

que genera archivos de calibración (JSON) necesarios para la conversión.

4. **Conversión al formato RKLLM:** tras cambiar la ruta al modelo en el siguiente archivo, se ejecuta

```
python3 export_rkllm.py
```

produciendo un archivo `Nombre\_del\_Modelo\_W8A8\_RK3588.rk1lm` cuantizado y compatible con la NPU [21]. Este archivo se puede modificar para, por ejemplo, utilizar una GPU de Nvidia, no obstante, por defecto debería de funcionarle a cualquiera que cumpla los requisitos.

5. **Requisitos de RAM:** durante la conversión se recomienda tener al menos entre 3 y 4 veces (dato obtenido de forma aproximada) el tamaño del modelo en memoria RAM, debido al proceso de carga y cuantización. Se puede usar memoria virtual (swap memory), pero el proceso sera más lento y con mucha carga sobre almacenamiento del PC.
6. **Despliegue en el SBC:** el archivo `.rk1lm` generado se extrae del contenedor y se transfiere al dispositivo con NPU (por SSH, Hugging Face, etc.), listo para su ejecución con el runtime de `rknn-llm`.

En definitiva, los pasos traducidos a comandos son los siguientes:

```

1 # 1. Ejecutar el contenedor Docker preparado
2 docker run -it pelochus/ezrk1lm-toolkit:latest bash
3 # Dentro del contenedor:
4 cd /ezrknn-llm/examples/DeepSeek-R1-Distill-Qwen-1.5B-Demo/export
5
6 # 2. Clonar el modelo con Git LFS
7 GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/Pelochus/deepseek-R1-
  distill-qwen-1.5B
8 cd deepseek-R1-distill-qwen-1.5B && git lfs pull
9
10 # 3. Ejecutar cuantizacion
11 python3 generate_data_quant.py -m ./
12
13 # 4. Convertir el modelo. Cambiar dentro de este archivo la ruta al modelo
14 python3 export_rk1lm.py
15
16 # 5. El paso 5 varia segun como se quiera mover el modelo al SBC
   correspondiente

```

## Compatibilidad y optimización

Este esquema soporta una amplia variedad de modelos como LLaMA, Qwen, Phi, ChatGLM3-6B, Gemma, InternLM2, MiniCPM, TeleChat, DeepSeek-R1-Distill, entre otros [1]. La elección de la cuantización W8A8 busca un equilibrio entre eficiencia energética y precisión en NPUs como la del RK3588, así como ser la única compatible en la NPU.

## B.2. ez-er-rkllm-toolkit

Aprovechando este apéndice, si bien esta herramienta no ha sido desarrollada por el autor de este TFM, es una extensión del trabajo creado por un usuario de la comunidad de NPUs de Rockchip, por ello, se menciona como herramienta alternativa y mejorada, así como ejemplo de contribución a la comunidad open source de Rockchip, sus NPUs y el RK3588.

Script de Python automatizado desarrollado por c0zaut (basado en el Docker realizado para este TFM), diseñado para convertir modelos como el Docker de este TFM.

El repositorio puede ejecutarse tanto de forma interactiva como en modo batch, facilitando la conversión, generación de modelos card con metadatos e incluso la subida directa a Hugging Face mediante autenticación automática [31].

- **Automatización avanzada:** permite conversiones múltiples sin intervención manual tras la configuración inicial.
- **Gestión de metadatos:** extrae archivos ‘.json’ y genera un modelo card enriquecido con detalles del modelo original.
- **Compatibilidad de versiones:** funciona para las versiones 1.1.x de `rknn-llm` y `ezrknn-llm`. No se ha probado si funciona para la 1.2.x.
- **Interacción con Hugging Face:** soporta subida automática de modelos convertidos con autenticación mediante token.



# Capítulo C

## EVALUACIÓN DE MÉTRICAS DE LA NPU

---

Como es lógico, debemos analizar el uso de NPU, RAM y CPU para poder hacer una correcta evaluación de rendimiento del RK3588.

A continuación, se describen dos herramientas para evaluar el uso de la NPU:

### C.1. ntop.sh

El script `ntop.sh` es una herramienta desarrollada por el autor de este TFM dentro del proyecto `ezrknpu`. Ofrece una visualización en tiempo real del uso de la NPU, similar al comando `top` en sistemas Linux.

Este script lee el archivo `/sys/kernel/debug/rknpu/load`, que proporciona información sobre la carga de cada núcleo de la NPU, y actualiza la pantalla cada 0.5 segundos. Además, incluye una opción para limpiar la salida en cada actualización, mejorando la legibilidad [20].

En adelante, un ejemplo de ejecución y output:

```
1 > bash ntop.sh
2 NPU load: Core0: 0%, Core1: 0%, Core2: 0%,
3 NPU load: Core0: 0%, Core1: 0%, Core2: 0%,
4 # Asumamos que en este punto se ejecuta un prompt en un LLM
5 NPU load: Core0: 10%, Core1: 90%, Core2: 80%,
6 NPU load: Core0: 98%, Core1: 92%, Core2: 85%,
7 NPU load: Core0: 40%, Core1: 16%, Core2: 84%
```

Como se puede ver en la siguiente imagen:





```

4 `curl https://raw.githubusercontent.com/ramonbroox/rknputop/main/install.sh
  | sudo bash`
5
6 ## Running
7
8 If you followed quick install, just enter `sudo rknputop` on your shell
9
10 $ rknputop -h
11 Usage: Show different NPU/CPU stats
12
13 Options:
14   -h, --help      show this help message and exit
15   -n, --npu-only  Only show the NPU load
16   -b, --npu-bars  Show the NPU with bars instead of lines

```

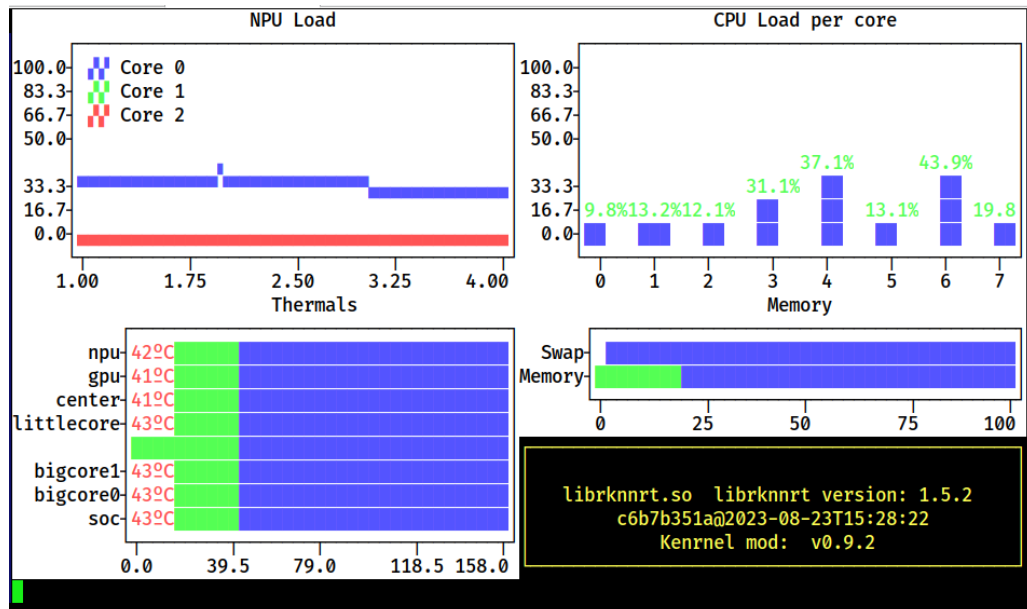


Figura C.2: rknputop



## BIBLIOGRAFÍA

---

- [1] Rockchip, “airockchip/rknn-llm.” <https://github.com/airockchip/rknn-llm>.
- [2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2023.
- [3] Orange Pi, “Orange Pi 5.” <http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-5.html>.
- [4] Alex Alderson, “Orange Pi 5 lands for US\$60 with Rockchip RK3588S and 4 GB of LPDDR4X RAM.” <https://www.notebookcheck.net/Orange-Pi-5-lands-for-US-60-with-Rockchip-RK3588S-and-4-GB-of-LPDDR4X-RAM-668640.0.html>.
- [5] Rockchip, “airockchip/rknn-toolkit2.” <https://github.com/airockchip/rknn-toolkit2>.
- [6] Meta, “Introducing Llama 3.” <https://ai.meta.com/blog/meta-llama-3/>.
- [7] Google, “Pixel 9 AI features.” <https://blog.google/products/pixel/google-pixel-9-new-ai-features/>.
- [8] Wikipedia, “Apple Intelligence.” [https://en.wikipedia.org/wiki/Apple\\_Intelligence](https://en.wikipedia.org/wiki/Apple_Intelligence).
- [9] Raspberry Pi Foundation, “Introducing Raspberry Pi 5.” <https://www.raspberrypi.com/news/introducing-raspberry-pi-5/>.
- [10] Nvidia, “Nvidia Jetson Orin Nano Super Developer Kit.” <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/nano-super-developer-kit/>.

- [11] M. Barr and A. Massa, “Programming Embedded Systems: With C and GNU Development Tools,” 2006. Definición: Un sistema empotrado es un sistema de computación diseñado para realizar una función específica, a menudo en tiempo real, dentro de un sistema más grande.
- [12] Geeks for Geeks, “What is a Large Language Model.” <https://www.geeksforgeeks.org/large-language-model-llm/>.
- [13] Karthikeyan Dhanakotti, “Exploring Quantization in LLMs: Concepts and Techniques.” <https://medium.com/data-science-at-microsoft/exploring-quantization-in-large-language-models-llms-concepts-and-techniques-4e>
- [14] Ricciuti Federico, “LLM Quantization evaluation - leave no bits behind.” <https://ricciuti-federico.medium.com/llm-quantization-evaluation-leave-not-bits-behind-0777e23b9f91>.
- [15] Stanislav Karzhev, “LLM Distillation Explained: Applications, Implementation & More.” <https://www.datacamp.com/blog/distillation-llm>.
- [16] DeepSeek, “DeepSeek R1 on Hugging Face.” <https://huggingface.co/deepseek-ai/DeepSeek-R1>.
- [17] Rockchip, “RK3588 Datasheet.” <https://www.cnx-software.com/pdf/Rockchip%C2%A0RK3588%C2%A0Datasheet%C2%A0V0.1-20210727.pdf>.
- [18] R. Jayanth, N. Gupta, and V. Prasanna, “Benchmarking Edge AI Platforms for High-Performance ML Inference,” 2024.
- [19] Chatree Sen, “Python YOLOv8 On RK3588 NPU.” [https://www.youtube.com/watch?v=waDHaCa\\_Kss](https://www.youtube.com/watch?v=waDHaCa_Kss).
- [20] Pelochus, “Pelochus/ezrknpu.” <https://github.com/Pelochus/ezrknpu>.
- [21] Pelochus, “Pelochus/ezrknn-llm.” <https://github.com/Pelochus/ezrknn-llm>.
- [22] Pelochus, “Pelochus/ezrknn-toolkit2.” <https://github.com/Pelochus/ezrknn-toolkit2>.
- [23] Armbian, “armbian/build.” <https://github.com/armbian/build>.

- 
- [24] Pelochus, “Pelochus/armbian-build-rknpu-updates.” <https://github.com/Pelochus/armbian-build-rknpu-updates>.
  - [25] Armbian, “armbian/linux-rockchip.” <https://github.com/armbian/linux-rockchip>.
  - [26] Pelochus, “Pelochus/linux-rockchip-npu-updates.” <https://github.com/Pelochus/linux-rockchip-npu-updates>.
  - [27] dnhkng, “dnhkng/GLaDOS.” <https://github.com/dnhkng/GLaDOS>.
  - [28] Pelochus, “Pelochus/ezrkllm-toolkit.” <https://hub.docker.com/repository/docker/pelochus/ezrkllm-toolkit/general>.
  - [29] Pelochus, “Pelochus/ezrkllm-collection.” <https://huggingface.co/Pelochus/ezrkllm-collection>.
  - [30] MLC, “GPU-Accelerated LLM on a \$100 Orange Pi.” <https://blog.mlc.ai/2024/04/20/GPU-Accelerated-LLM-on-Orange-Pi>.
  - [31] c0zaut, “ez-er-rkllm-toolkit.” <https://github.com/c0zaut/ez-er-rkllm-toolkit>.
  - [32] ramonbroox, “rknputop.” <https://github.com/ramonbroox/rknputop>.