Informe técnico — Jetson AGX Orin

**1) Identificación del sistema**

* **Modelo**: NVIDIA **Jetson AGX Orin Developer Kit** (aarch64).
* **SO/L4T/JetPack**: **L4T r36.4.4** (JetPack **6.2.1**), fecha 2025-06-16.
* **CUDA**: **12.6** (detectado por autotag).
* **Ubuntu**: 22.04 (Jammy).
* **Docker**: runtime por defecto **nvidia** (nvidia-container-toolkit configurado).

**2) Memoria (RAM, swap y “VRAM”)**

* **RAM física**: **64 GB** (MemTotal ≈ 64,349,016 kB).  
  Linux reporta utilizable ≈ **61 GiB** (resto reservado para GPU/firmware).
* **Swap zram**: ~**30 GiB** (12 dispositivos zram de ~2.6 GiB).
* **VRAM dedicada**: **no existe**. Jetson usa **memoria unificada (UMA)**: CPU y GPU comparten la **RAM del sistema**.
  + En la práctica, la “VRAM disponible” ≈ **RAM libre** que ves en free -h/tegrastats.
  + Ejemplo (tegrastats): RAM 5001/62841MB ⇒ ~**57.8 GB libres** (eso es lo que la GPU podría usar si hiciera falta).

**3) Almacenamiento**

* **eMMC** interna: ~**59.2 GB** totales.
  + Partición raíz / (ext4): ~**57.8 GB** (era lo que llenaba con imágenes/modelos).
  + EFI /boot/efi: 64 MB (vfat).
* **NVMe/M.2**: **no detectado** actualmente.
  + lspci solo muestra la **Wi-Fi Realtek RTL8822CE**; parted /dev/nvme0n1 falla y el dmesg indica “Phy link never came up” en otros puentes PCIe → no hay NVMe presente o no enlaza.

# Por qué Jetson “usa RAM” y no VRAM?

Es un **SoC ARM** con **GPU integrada**; no hay tarjeta gráfica separada con GDDR propia.

La **LPDDR del sistema** sirve tanto a CPU como a GPU (memoria unificada).

Para **LLMs**:

* **Pesos del modelo** + **KV-cache** residen en **RAM**.
* La **GPU** acelera el cómputo leyendo esa RAM por el bus de memoria de alto ancho de banda.
* Si la RAM se agota, entra la **swap zram** (más lenta) → baja el rendimiento.

1**) Arquitectura y aceleración (ARM + GPU integrada)**

* **CPU**: ARM **aarch64** (64-bit).
* **GPU**: integrada en el SoC (NVIDIA), comparte RAM del sistema (**UMA**).
* Lecturas típicas:
  + GR3D\_FREQ: frecuencia de GPU (p.ej. **0%@[1300,1300]** cuando está ociosa).
  + EMC\_FREQ: controlador de memoria (p.ej. **0%@3199** → ~3.2 GHz).
  + Temperaturas/potencia (CPU/GPU) también salen en tegrastats.

**2) Implicaciones prácticas para LLM en tu Jetson (64 GB)**

* **Tamaños de modelo** recomendados:
  + **4–8B** parámetros en **INT4/FP8** → cómodos en 64 GB UMA.
  + **13B** puede ir, pero más justo (RAM y ancho de banda).
* **Contexto** (tokens) aumenta mucho la **KV-cache** ⇒ más RAM.
* **Espacio en disco**: además de la RAM, necesitas **varios GB** para:
  + Pesos (HF\_HOME), compilar/optimizar (MLC\_CACHE\_DIR), y audios (Piper/Riva).
* **Rendimiento**: evita caer en swap; si free -h baja demasiado, reduce tamaño del modelo o el contexto.

# Comandos útiles:

# RAM / swap / disponibilidad

free -h

sudo tegrastats --interval 1000

# Almacenamiento

df -h

du -xh --max-depth=1 ~ | sort -h

# PCIe / NVMe

sudo lspci -nn

sudo dmesg | egrep -i 'pcie|nvme|m\.2' | tail -n 200

Text (LLM) - Ollama

# Informe de prueba — Jetson + Ollama (llama3.2:3b)

* **Resultado clave:** ~**44.8 tokens/seg** en decodificación (media de 5 ejecuciones) con uso intensivo de GPU y temperaturas normales (~60 °C).
* **Modelo:** llama3.2:3b (~**2.0 GB** en disco).
* **Telemetría:** GPU al **90–99% @ ~1.3 GHz** durante inferencia, RAM de sistema con amplio margen, **sin swap**.

**Equipo y SO**

* **Dispositivo:** NVIDIA **Jetson AGX Orin Developer Kit**
* **L4T / JetPack:** **R36.4.4 / 6.2.1**
* **CUDA:** **12.6**
* **RAM del sistema:** ~**64 GB** (MemTotal ≈ 61–62 GiB utilizable)
* **Almacenamiento:** eMMC ~**57 GB** (liberado y organizado antes de la prueba)

Nota: En Jetson **no hay VRAM dedicada**. La GPU usa **memoria unificada (UMA)**: comparte la **RAM del sistema**. Por eso no verás “VRAM” separada; todo sale de RAM.

**Preparación**

ollama run llama3.2:3b # (descarga ~2.0 GB la primera vez)

**Entorno de prueba** (aislado)

python3 -m venv .venv && source .venv/bin/activate

echo 'requests>=2.31' > requirements.txt

pip install -r requirements.txt

**Script de benchmark**  
Archivo: test\_ollama\_llama3.2:3b\_jetson.py

* Se conecta a http://localhost:11434
* Hace **warmup**
* Ejecuta N carreras con medición de:
  + **prefill\_tokens** / **decode\_tokens**
  + **prefill\_tps** / **decode\_tps**
  + **wall time**

**Metodología de prueba**

Comando principal:

python 'test\_ollama\_llama3.2:3b\_jetson.py' --model 'llama3.2:3b' -n 5 --out metrics.jsonl

**Resultados (5 carreras)**

| **Run** | **wall (s)** | **prefill tok** | **prefill t/s** | **decode tok** | **decode t/s** |
| --- | --- | --- | --- | --- | --- |
| 1 | 2.48 | 45 | 5 199 | 107 | **44.4** |
| 2 | 2.46 | 45 | 11 687 | 108 | **45.0** |
| 3 | 2.54 | 45 | 11 580 | 111 | **44.9** |
| 4 | 2.68 | 45 | 11 712 | 117 | **44.8** |
| 5 | 2.68 | 45 | 11 435 | 117 | **44.8** |

**Media decode\_tps:** **44.8 tok/s** (5 runs)

Observación: el **prefill\_tps** es muy alto (prefill es más paralelo); lo que miramos para velocidad de conversación es **decode\_tps** (~45 t/s).

**Telemetría (tegrastats) durante inferencia**

* **GPU (GR3D\_FREQ):** **90–99%** @ **~1.30 GHz**
* **EMC (memoria):** **~45%** @ 3199 (no cuello de botella)
* **RAM:** ~**5.6 / 61 GiB** usada (mucho margen), **swap 0**
* **Temperaturas:** **~60–61 °C** (estables)
* **Potencia (VDD\_GPU\_SOC):** **~31–35 W** en carga; ~**5 W** en reposo

**Conclusiones**

* El Jetson AGX Orin maneja **llama3.2:3b** con **~45 tok/s** de decodificación y **uso efectivo de GPU**.
* **Temperatura y consumo** dentro de rangos normales; **RAM suficiente** sin tocar swap.
* La arquitectura **UMA** simplifica memoria (no VRAM aparte) y funciona bien para LLMs con tamaños moderados.

Comparativo — RTX Ada 2000 vs Jetson AGX Orin  
Modelo: llama3.2:3b (Ollama)

# 1) Resumen

Se comparó el rendimiento de generación (decodificación de tokens) del modelo llama3.2:3b ejecutado con Ollama en dos entornos: un portátil con GPU NVIDIA RTX Ada 2000 y un Jetson AGX Orin. Las métricas provienen de la API de Ollama (prompt\_eval\_count, eval\_count, \*\_duration en ns), convertidas a tokens por segundo (t/s). El indicador principal es decode\_tps.

Resultado clave: la RTX Ada 2000 alcanzó una media global de 74.65 t/s frente a 44.8 t/s en Jetson, lo que supone un factor de aceleración ≈ 1.67×.

# 2) Metodología

• Modelo idéntico: llama3.2:3b.  
• Servidor: Ollama local (HTTP 11434).  
• Parámetros clave: temperature=0, seed=42, num\_ctx 2048/4096 (según lote), num\_predict 256 y 512.  
• Métrica primaria: decode\_tps (eval\_count / eval\_duration). Cifras calculadas por ejecución y promediadas.  
• Runs RTX: 9 (6 con num\_predict=256 y 3 con 512). Runs Jetson: 5.  
• Limitación: en Jetson sólo se dispone de la media de decode\_tps (44.8 t/s) en el informe aportado.

# Comparativa de decode\_tps medio (t/s):

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

Resumen por lote (RTX) y referencia Jetson:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Equipo/Lote | num\_predict | runs | decode\_tps media | mediana | p90 | min | max | total\_s media | load\_s media | decode\_tokens media |
| RTX Ada 2000 | 256 | 6 | 73.10 | 72.56 | 78.46 | 67.50 | 79.53 | 1.394 | 0.044 | 97.8 |
| RTX Ada 2000 | 512 | 3 | 77.74 | 75.55 | 82.98 | 72.85 | 84.83 | 1.444 | 0.040 | 108.7 |
| Jetson AGX Orin | — | 5 | 44.80 | n/d | n/d | n/d | n/d | n/d | n/d | n/d |

Aceleración relativa de decode\_tps (RTX/Jetson): 1.63× (256), 1.74× (512), 1.67× (global).

# 4) Interpretación

• La RTX Ada 2000 ofrece entre ~1.6× y ~1.7× más velocidad de decodificación que Jetson con el mismo modelo, consistente entre lotes de 256 y 512 tokens previstos.  
• El tiempo de carga (`load\_s`) en RTX es bajo (~0.04 s), indicio de buen cacheo y servidor en caliente.  
• Diferencias de `num\_ctx` entre pruebas (2048 vs 4096) no alteran la conclusión principal; para una foto 1:1 se recomienda repetir ambos lados con los mismos flags finales.