

Tu copiloto de código, 100 % local

Capítulo 13 — Cuando algo falla: diagnosticar y reparar (extracto) · v2026.07 · quellm.com/copilote-local

Capítulo 13 — Cuando algo falla: diagnosticar y reparar

La semana 1 de una configuración local, el problema casi nunca es «el modelo es malo». Son cinco fallos, siempre los mismos. Hacen creer que lo local no funciona. En realidad, solo faltan dos líneas de configuración. Este capítulo los diagnostica **por síntoma**: partes de lo que ves (un mensaje de error, un rendimiento que se desploma, una autocompletación que se ralentiza), remontas hasta la causa y, después, aplicas la solución. Este es el capítulo de referencia para el diagnóstico. Los capítulos 4, 6 y 12 te remiten aquí en cuanto aparece un fallo.

Punto de referencia: estás ejecutando tu `monprojet` (la pequeña API web Flask/Python que sirve de hilo conductor desde el capítulo 7), backend Ollama, ancla de hardware RTX 5070 Ti de 16 GB. Cada vez te doy la versión «para TU VRAM». Una solución válida con 24 GB puede ser exactamente el reflejo equivocado con 8 GB.

✅ **PARA RECORDAR — La regla de oro de la depuración local.** El 90 % de los fallos de la semana 1 se deben a **un solo número**: `num_ctx`. El `num_ctx` es el tamaño de la memoria de trabajo del modelo. Lo has aumentado para que quepa tu repo y ha expulsado al modelo de la VRAM. Antes de sospechar otra cosa, comprueba dónde se ejecuta tu modelo (`ollama ps`) y cuánto has aumentado el contexto.

📖 En la guía completa, este capítulo empieza con un árbol de decisión (síntoma → sección → solución) que cubre las 5 averías típicas de la primera semana. Este extracto gratuito contiene la avería n.º 1 (OOM) y el comienzo de la n.º 2 (GPU sin usar).

Sección A — OOM de GPU: «out of memory» o rendimiento que se desploma

El síntoma

Tiene dos formas. La evidente: un mensaje `CUDA error: out of memory` (o su equivalente en ROCm/Metal), y el modelo se niega a responder. La engañosa, más frecuente y traicionera: **responde, pero a unos pocos tokens por segundo**. El modelo se ha quedado sin VRAM. Ollama ha trasladado parte de las capas a la CPU (es el offload), y tu GPU de 16 GB observa cómo la CPU se arrastra.

La causa

El presupuesto de VRAM es sencillo y puedes calcularlo tú mismo:

```
VRAM nécessaire ≈ (poids du modèle quantifié)
                  + (cache KV, qui grandit avec num_ctx)
                  + overhead
```

La caché KV es la memoria donde el modelo almacena el contexto ya procesado. En tus 16 GB, el modelo agentic predeterminado —Devstral Small 24B en `q4_K_M` (~14 GB)— cabe **mientras no aumentes `num_ctx`**. La trampa clásica: pasas `num_ctx` de 8192 a 32768 para hacer que quepa más de tu `monprojet`. La caché KV se duplica o triplica y superas la barrera de los 16 GB sin ningún mensaje de error: solo aparece el offload silencioso.

El diagnóstico en un comando

```
ollama ps
```

Lee la columna `PROCESSOR` (su interpretación detallada está en el **capítulo 4**). Es el instrumento de diagnóstico central de todo el capítulo. Así puedes usarla para el OOM:

Pantalla <code>ollama ps</code>	Interpretación	Acción
<code>100% GPU</code>	Todo está en la VRAM, es correcto	No hay que hacer nada por el OOM
<code>58%/42% CPU/GPU</code> (o cualquier %CPU distinto de cero)	Offload parcial → rendimiento desplomado	Reducir <code>num_ctx</code> o el tamaño del modelo
<code>100% CPU</code>	La GPU no se utiliza en absoluto	Ve a la Sección B (no es un OOM)

Tabla 13.1 — Lectura de `ollama ps` para diagnosticar el OOM.

La solución (por orden, de la menos destructiva a la más)

1. Reduce `num_ctx` primero. Es el recurso n.º 1 y el menos costoso. Vuelve a un valor que quepa (8192 es el valor predeterminado para chat/agent en 16 GB). Compruébalo con `ollama ps` y después súbelo gradualmente hasta justo antes de que aparezca el offload.

```
# Prueba directa: impone un contexto moderado y comprueba si volvemos al 100% de GPU
OLLAMA_CONTEXT_LENGTH=8192 ollama run devstral:24b
```

En el Modelfile (consulta el **capítulo 11** para ver la estructura completa), lo mismo de forma permanente:

```
FROM devstral:24b
PARAMETER num_ctx 8192
```

2. Baja un nivel de cuantización o de tamaño. Si se desborda incluso con un `num_ctx` razonable, el modelo es demasiado grande para tu tarjeta. Quédate en `q4_K_M` (el compromiso

predeterminado entre calidad y espacio), sin consumir más. El ajuste fino temp/top-p/ num_ctx se trata en el **capítulo 12**; la elección del modelo adecuado para tu VRAM, en el **capítulo 5**.

3. Limita explícitamente las capas de GPU (caso extremo). ¿Quieres forzar un reparto controlado en lugar de sufrir el offload automático? num_gpu fija el número de capas enviadas a la GPU:

```
FROM devstral:24b
PARAMETER num_ctx 8192
PARAMETER num_gpu 99 # "lo máximo posible"; reduce este número si quieres descargar
parte del trabajo a la CPU de forma voluntaria
```

⚠ ADVERTENCIA — num_gpu no es una fórmula mágica. Reducirlo no «libera» rendimiento. Al contrario: envía *más* capas a la CPU, así que ralentiza el proceso. Es una herramienta para los casos en los que quieres ejecutar voluntariamente un modelo demasiado grande, aceptando la lentitud. No es un ajuste para usar a diario. En un uso normal, deja que Ollama se encargue. Actúa mejor sobre num_ctx y el tamaño del modelo.

Para TU VRAM:

VRAM	Reacción principal ante un OOM
8 GB	El nivel agentic denso de 24B queda fuera de alcance; apunta a un modelo ligero (Qwen2.5-Coder 7B) y num_ctx ≤ 8192. Cualquier aumento del contexto se paga caro.
12 GB (RTX 5070, RTX 4070)	Devstral Small 24B NO cabe aquí: elige Qwen2.5-Coder 14B (~9 GB) o DeepSeek-Coder-V2-Lite 16B (~10 GB) en q4_K_M, num_ctx 4096. Por encima de eso, vigila ollama ps .
16 GB (ANCLA — RTX 5070 Ti / 4080 / 5080 / RX 9070 XT, Mac M4 24 GB)	Devstral Small 24B q4_K_M (~14 GB) + num_ctx 8192 = el punto ideal para agentic. También queda margen para Qwen2.5-Coder 14B en Q8.
24 GB (RTX 3090/4090, RX 7900 XTX, Mac M4 Pro 48 GB)	Puedes permitirte Qwen2.5-Coder 32B q4_K_M (~19 GB), Devstral en Q5/Q6 o un contexto largo sin offload; el OOM se vuelve poco frecuente.
Apple Silicon (Metal)	La VRAM se comparte con la RAM del sistema (memoria unificada). No hay un OOM brusco, sino presión de memoria que provoca swap: vigila la presión de memoria del sistema y cierra los procesos grandes antes de una sesión importante.

Tabla 13.2 — Reacción ante un OOM según el nivel de VRAM.

Sección B — La GPU no se utiliza en absoluto

El síntoma

`ollama ps` muestra `100% CPU`, el ventilador de la GPU está parado y la generación es lenta en todas partes: no se desploma como con un offload, simplemente es *lenta* en todas partes, desde el primer token. Es el caso más frustrante: el hardware está ahí, pero no se aprovecha.

La causa

Ollama no detecta tu acelerador. Hay tres causas habituales: - El runtime de GPU no se detecta (faltan los drivers de CUDA/ROCm o son demasiado antiguos). - En Linux, Ollama se ejecuta en un contexto (contenedor, servicio systemd) que no tiene acceso al dispositivo GPU. - En Windows/WSL, el passthrough de GPU no está configurado.

El diagnóstico

```
# NVIDIA (tu 5070 Ti): la GPU debe aparecer y listar un proceso ollama durante una
generación
nvidia-smi

# AMD ROCm
rocm-smi

# Y, sobre todo: lee los registros de Ollama al arrancar; allí anuncia lo que detecta
```

En Linux, los registros del servicio:

```
journalctl -u ollama --no-pager | grep -i -E "gpu|cuda|rocm|library"
```

Busca una línea en la que Ollama indique que ha encontrado una biblioteca de GPU (CUDA/ROCm). Si anuncia que cambia a la CPU, ya tienes la causa. (La instalación correcta del runtime según tu sistema operativo se explica en el **capítulo 4**; si esta sección te afecta, a menudo es porque se omitió el paso de GPU del capítulo 4).

El extracto termina aquí. El resto de la sección B y las secciones C (desbordamiento de contexto), D (etiqueta incorrecta) y E (autocompletado FIM lento), además de la tabla de referencia síntoma → causa → solución, están en la guía.

El resto está en la guía

25 capítulos (unas 175 páginas) y el paquete de configuraciones listas para pegar: Modelfiles ajustados, Cline, Aider y Tabby, tokens FIM correctos, chuleta VRAM → modelo → comando. Pago único de 24 €, actualizaciones de por vida.

<https://quellm.com/copilote-local/>