

Il tuo copilota di codice, 100% locale

Capitolo 13 — Quando si rompe: diagnosticare e riparare (estratto) · v2026.07 · qualellm.it/copilote-local

Capitolo 13 — Quando si rompe: diagnosticare e riparare

La settimana 1 di una configurazione locale, il problema non è quasi mai «il modello è scarso». Sono cinque guasti, sempre gli stessi. Fanno pensare che il locale non funzioni. In realtà, mancano solo due righe di configurazione. Questo capitolo li diagnostica **in base al sintomo**: parti da ciò che vedi (un messaggio di errore, una velocità che crolla, un completamento automatico che rallenta), risalisci alla causa e poi alla soluzione. È il capitolo di riferimento per la diagnosi. I capitoli 4, 6 e 12 ti rimandano qui non appena si verifica un guasto.

Punto di riferimento operativo: stai usando la tua `monprojet` (la piccola API web Flask/Python che fa da filo conduttore dal capitolo 7), backend Ollama, ancoraggio hardware RTX 5070 Ti da 16 GB. Ogni volta ti do la versione «per la TUA VRAM». Un fix valido con 24 GB può essere esattamente il riflesso sbagliato con 8 GB.

✅ **DA RICORDARE — La regola d'oro del debugging locale.** Il 90 % dei guasti della settimana 1 dipende da **un solo numero**: `num_ctx`. Il `num_ctx` è la dimensione della memoria di lavoro del modello. L'hai aumentato per farci entrare il tuo repo e il modello è finito fuori dalla VRAM. Prima di sospettare altro, controlla dove viene eseguito il modello (`ollama ps`) e di quanto hai gonfiato il contesto.

📖 Nella guida completa questo capitolo si apre con un albero decisionale (sintomo → sezione → soluzione) che copre i 5 guasti tipici della prima settimana. Questo estratto gratuito contiene il guasto n. 1 (OOM) e l'inizio del n. 2 (GPU non utilizzata).

Sezione A — OOM GPU: «out of memory» o throughput che crolla

Il sintomo

Si presenta in due forme. Quella evidente: un messaggio `CUDA error: out of memory` (o l'equivalente ROCm/Metal) e il modello rifiuta di rispondere. Quella subdola, più frequente e più infida: **risponde, ma a pochi token al secondo**. Il modello ha superato la VRAM. Ollama ha spostato parte dei layer sulla CPU (è l'offload), e la tua GPU da 16 GB guarda la CPU arrancare.

La causa

Il budget VRAM è semplice e puoi calcolarlo da solo:

```
VRAM necessaria ≈ (dimensioni del modello quantizzato)
                  + (cache KV, che cresce con num_ctx)
                  + overhead
```

La cache KV è la memoria in cui il modello archivia il contesto già elaborato. Sui tuoi 16 GB, il modello agentic predefinito — Devstral Small 24B in `q4_K_M` (~14 GB) — ci sta **finché non aumenti `num_ctx`**. La trappola classica: passi `num_ctx` da 8192 a 32768 per far entrare più elementi della tua `monprojet`. La cache KV raddoppia o triplica e superi la soglia dei 16 GB senza messaggi di errore: solo un offload silenzioso.

La diagnosi con un comando

```
ollama ps
```

Leggi la colonna `PROCESSOR` (la sua interpretazione dettagliata è nel **capitolo 4**). È lo strumento diagnostico centrale di tutto il capitolo. Ecco come usarla per l'OOM:

Visualizzazione <code>ollama ps</code>	Interpretazione	Azione
100% GPU	È tutto in VRAM, è corretto	Nessuna azione per l'OOM
58%/42% CPU/GPU (o qualsiasi %CPU diverso da zero)	Offload parziale → throughput crollato	Riduci <code>num_ctx</code> o la dimensione del modello
100% CPU	La GPU non viene utilizzata affatto	Vai alla Sezione B (non è un OOM)

Tabella 13.1 — Lettura di `ollama ps` per diagnosticare l'OOM.

Il fix (in ordine, dal meno distruttivo al più drastico)

1. Riduci prima `num_ctx`. È la leva n. 1 e quella meno costosa. Torna a un valore che ci stia (8192 è il valore predefinito per chat/agent su 16 GB). Verifica con `ollama ps`, poi aumenta a intervalli fino a poco prima dell'offload.

```
# Test diretto: imposta un contesto modesto e controlla se torni al 100% GPU
OLLAMA_CONTEXT_LENGTH=8192 ollama run devstral:24b
```

Nel Modelfile (vedi il **capitolo 11** per la struttura completa), la stessa cosa in modo permanente:

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

2. Scendi di un livello di quantizzazione o di dimensione. Se trabocca anche con un `num_ctx` ragionevole, il tuo modello è troppo grande per la tua scheda. Rimani in `q4_K_M` (il compromesso predefinito tra qualità e spazio), senza andare oltre. La regolazione fine temp/top-p/ `num_ctx` è trattata nel **capitolo 12**; la scelta del modello giusto per la tua VRAM nel **capitolo 5**.

3. Imposta esplicitamente un limite ai layer GPU (caso limite). Vuoi forzare una suddivisione controllata invece di subire l'offload automatico? `num_gpu` imposta il numero di layer inviati alla GPU:

```
FROM devstral:24b
PARAMETER num_ctx 8192
PARAMETER num_gpu 99 # "il più possibile"; riduci questo numero se vuoi scaricare
intenzionalmente parte del lavoro sulla CPU
```

⚠ AVVERTENZA — `num_gpu` non è una formula magica. Ridurlo non «libera» prestazioni. Al contrario: invia *più* layer sulla CPU, quindi rallenta. È uno strumento per i casi in cui vuoi eseguire intenzionalmente un modello troppo grande, accettando la lentezza. Non è un'impostazione da usare ogni giorno. Nell'uso normale, lascia che Ollama gestisca la situazione. Intervieni piuttosto su `num_ctx` e sulla dimensione del modello.

Per la TUA VRAM:

VRAM	Riflesso principale per l'OOM
8 GB	Il tier agentic denso da 24B è fuori portata; punta a un modello leggero (Qwen2.5-Coder 7B) e a <code>num_ctx</code> ≤ 8192. Ogni aumento del contesto si paga immediatamente.
12 GB (RTX 5070, RTX 4070)	Devstral Small 24B NON ci sta: scegli Qwen2.5-Coder 14B (~9 GB) oppure DeepSeek-Coder-V2-Lite 16B (~10 GB) in <code>q4_K_M</code> , <code>num_ctx</code> 4096. Oltre, controlla <code>ollama ps</code> .
16 GB (ANCORA — RTX 5070 Ti / 4080 / 5080 / RX 9070 XT, Mac M4 24 GB)	Devstral Small 24B <code>q4_K_M</code> (~14 GB) + <code>num_ctx</code> 8192 = punto ideale per l'agentic. C'è margine anche per Qwen2.5-Coder 14B in Q8.
24 GB (RTX 3090/4090, RX 7900 XTX, Mac M4 Pro 48 GB)	Puoi permetterti Qwen2.5-Coder 32B <code>q4_K_M</code> (~19 GB), Devstral in Q5/Q6 oppure un contesto lungo senza offload; l'OOM diventa raro.
Apple Silicon (Metal)	La VRAM è condivisa con la RAM di sistema (memoria unificata). Non c'è un OOM improvviso, ma una pressione sulla memoria che causa swap: controlla la pressione della memoria di sistema e chiudi i processi pesanti prima di una sessione impegnativa.

Tabella 13.2 — Riflesso anti-OOM per livello di VRAM.

Sezione B — La GPU non viene utilizzata affatto

Il sintomo

ollama ps mostra 100% CPU, la ventola della GPU è ferma e la generazione è lenta ovunque: non crolla come con un offload, è semplicemente lenta *dappertutto*, fin dal primo token. È il caso più frustrante: l'hardware c'è, ma non viene sfruttato.

La causa

Ollama non rileva il tuo acceleratore. Tre cause comuni: - Il runtime GPU non viene rilevato (driver CUDA / ROCm assenti o troppo vecchi). - Su Linux, Ollama viene eseguito in un contesto (container, servizio systemd) che non ha accesso al device GPU. - Su Windows/WSL, il passthrough GPU non è configurato.

La diagnosi

```
# NVIDIA (la tua 5070 Ti) – la GPU deve comparire e deve elencare un processo ollama
durante una generazione
nvidia-smi

# AMD ROCm
rocm-smi

# E soprattutto: leggi i log di Ollama all'avvio, quando comunica ciò che rileva
```

Su Linux, i log del servizio:

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

Cerca una riga in cui Ollama dichiara di aver trovato una libreria GPU (CUDA/ROCm). Se comunica che passa alla CPU, hai trovato la causa. (L'installazione corretta del runtime in base al tuo sistema operativo è trattata nel **capitolo 4**: se questa sezione ti riguarda, spesso significa che hai saltato il passaggio GPU del cap. 4.)

L'estratto finisce qui. Il seguito della sezione B e le sezioni C (contesto superato), D (tag sbagliato) ed E (autocompletamento FIM lento), oltre alla tabella di riferimento sintomo → causa → soluzione, sono nella guida.

Il seguito è nella guida

25 capitoli (circa 175 pagine) e il pacchetto di configurazioni pronte da incollare: Modelfile ottimizzati, Cline, Aider e Tabby, token FIM corretti, promemoria VRAM → modello → comando. Pagamento unico di 24 €, aggiornamenti a vita.

<https://qualellm.it/copilote-local/>