

Ihr Code-Copilot, 100 % lokal

Kapitel 13 — Wenn etwas kaputtgeht: diagnostizieren und reparieren (Auszug) · v2026.07 ·
welchesllm.de/copilote-local

Kapitel 13 — Wenn etwas kaputtgeht: diagnostizieren und reparieren

In Woche 1 eines lokalen Setups lautet das Problem fast nie: „Das Modell ist schlecht.“ Es sind fünf Fehler, immer dieselben. Sie erwecken den Eindruck, dass lokal nichts funktioniert. In Wirklichkeit fehlen nur zwei Konfigurationszeilen. Dieses Kapitel diagnostiziert sie **nach Symptomen**: Sie gehen von dem aus, was Sie sehen (eine Fehlermeldung, ein einbrechender Durchsatz, eine stockende Autovervollständigung), arbeiten sich zur Ursache vor und beheben sie dann. Das ist das Referenzkapitel für die Diagnose. Die Kapitel 4, 6 und 12 verweisen Sie hierher, sobald ein Fehler auftritt.

Arbeitsgrundlage: Sie arbeiten mit Ihrer `monprojet` (der kleinen Flask/Python-Web-API, die seit Kapitel 7 als roter Faden dient), Backend Ollama, Hardware-Referenz RTX 5070 Ti mit 16 GB. Jedes Mal gebe ich Ihnen die Version „für IHREN VRAM“. Ein Fix, der mit 24 GB funktioniert, kann bei 8 GB genau die falsche Maßnahme sein.

✓ **ZUM MERKEN — Die goldene Regel für lokales Debugging.** 90 % der Fehler in Woche 1 gehen auf **eine einzige Zahl** zurück: `num_ctx`. Der `num_ctx` ist die Größe des Arbeitsspeichers des Modells. Sie haben ihn erhöht, damit Ihr Repo hineinpasst, und dadurch das Modell aus dem VRAM gedrängt. Bevor Sie etwas anderes vermuten, prüfen Sie, wo Ihr Modell ausgeführt wird (`ollama ps`) und um wie viel Sie den Kontext vergrößert haben.

 Im vollständigen Leitfaden beginnt dieses Kapitel mit einem Entscheidungsbaum (Symptom → Abschnitt → Lösung) für die 5 typischen Pannen der ersten Woche. Dieser kostenlose Auszug enthält Panne Nr. 1 (OOM) und den Anfang von Nr. 2 (GPU wird nicht genutzt).

Abschnitt A — OOM GPU: „out of memory“ oder einbrechender Durchsatz

Das Symptom

Es tritt in zwei Formen auf. Die offensichtliche: eine Meldung `CUDA error: out of memory` (oder das ROCm/Metal-Äquivalent), und das Modell verweigert die Antwort. Die heimtückische, häufigere und tückischere: **es antwortet, aber mit wenigen Tokens pro Sekunde**. Das Modell

hat den VRAM überlaufen lassen. Ollama hat einen Teil der Layer auf die CPU ausgelagert (das ist Offloading), und Ihre GPU mit 16 GB sieht zu, wie die CPU sich abmüht.

Die Ursache

Das VRAM-Budget ist einfach, und Sie können es selbst aufstellen:

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

Der KV-Cache ist der Speicher, in dem das Modell den bereits verarbeiteten Kontext ablegt. Auf Ihren 16 GB passt das standardmäßige agentische Modell — Devstral Small 24B in `q4_K_M` (~14 GB) — **solange Sie `num_ctx` nicht vergrößern**. Die klassische Falle: Sie erhöhen `num_ctx` von 8192 auf 32768, damit mehr von Ihrem `monprojet` hineinpasst. Der KV-Cache verdoppelt oder verdreifacht sich, und Sie überschreiten die 16-GB-Grenze ohne Fehlermeldung — nur durch stilles Offloading.

Die Diagnose mit einem Befehl

```
ollama ps
```

Lesen Sie die Spalte `PROCESSOR` (die ausführliche Interpretation finden Sie in **Kapitel 4**). Sie ist das zentrale Diagnoseinstrument des gesamten Kapitels. So verwenden Sie sie für OOM:

Anzeige <code>ollama ps</code>	Interpretation	Maßnahme
100% GPU	Alles liegt im VRAM, das ist gesund	Für OOM nichts zu tun
58%/42% CPU/GPU (oder jeder Wert von %CPU ungleich null)	Teilweises Offloading → Durchsatz bricht ein	<code>num_ctx</code> oder die Modellgröße reduzieren
100% CPU	Die GPU wird überhaupt nicht verwendet	Gehen Sie zu Abschnitt B (das ist kein OOM)

Tabelle 13.1 — `ollama ps` zur OOM-Diagnose lesen.

Der Fix (in der Reihenfolge vom wenigsten zum stärksten Eingriff)

1. Senken Sie zuerst `num_ctx`. Das ist der Hebel Nr. 1 und der kostengünstigste. Setzen Sie den Wert wieder auf einen Wert zurück, der passt (8192 ist der Standard für Chat/Agent bei 16 GB). Überprüfen Sie dies mit `ollama ps` und erhöhen Sie den Wert dann schrittweise bis kurz vor dem Offload.

```
# Direkttest: einen moderaten Kontext erzwingen und prüfen, ob wieder 100% GPU erreicht werden
OLLAMA_CONTEXT_LENGTH=8192 ollama run devstral:24b
```

Im Modelfile (siehe **Kapitel 11** für die vollständige Struktur) gilt dasselbe fest eingetragen:

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

2. Eine Quantisierungs- oder Größenstufe heruntergehen. Wenn es selbst bei einem angemessenen `num_ctx` überläuft, ist Ihr Modell zu groß für Ihre Karte. Bleiben Sie bei `q4_K_M` (dem standardmäßigen Kompromiss zwischen Qualität und Platz), nicht bei einer anspruchsvolleren Variante. Die Feinabstimmung von temp/top-p/ `num_ctx` wird in **Kapitel 12** behandelt; die Wahl des richtigen Modells für Ihren VRAM in **Kapitel 5**.

3. Die GPU-Layer explizit begrenzen (Sonderfall). Sie möchten eine kontrollierte Aufteilung erzwingen, statt das automatische Offloading hinzunehmen? `num_gpu` legt die Anzahl der an die GPU gesendeten Layer fest:

```
FROM devstral:24b
PARAMETER num_ctx 8192
PARAMETER num_gpu 99 # "so viele wie möglich" ; verringern Sie diese Zahl, wenn Sie
absichtlich auf die CPU auslagern möchten
```

⚠️ WARNUNG — `num_gpu` ist keine Zauberformel. Wenn Sie den Wert verringern, „setzen“ Sie keine Leistung „frei“. Im Gegenteil: Dadurch werden *mehr* Layer an die CPU gesendet, und das System wird langsamer. Dieses Werkzeug ist für Fälle gedacht, in denen Sie absichtlich ein zu großes Modell ausführen und die Langsamkeit akzeptieren. Es ist keine Einstellung für den täglichen Gebrauch. Im Normalbetrieb lassen Sie Ollama die Verwaltung übernehmen. Verändern Sie stattdessen `num_ctx` und die Modellgröße.

Für IHREN VRAM:

VRAM	Wichtigste OOM-Maßnahme
8 GB	Die dichte agentische 24B-Klasse ist außer Reichweite; zielen Sie auf ein leichtes Modell (Qwen2.5-Coder 7B) und <code>num_ctx</code> $\leq$ 8192. Jede Erhöhung des Kontexts schlägt sofort zu Buche.
12 GB (RTX 5070, RTX 4070)	Devstral Small 24B passt hier NICHT hinein: Wählen Sie Qwen2.5-Coder 14B (~9 GB) oder DeepSeek-Coder-V2-Lite 16B (~10 GB) in <code>q4_K_M</code> , <code>num_ctx</code> 4096. Darüber hinaus behalten Sie <code>ollama ps</code> im Auge.
16 GB (REFERENZ — 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 = Sweet Spot für agentische Aufgaben. Auch Spielraum für Qwen2.5-Coder 14B in Q8.
24 GB (RTX 3090/4090, RX 7900 XTX, Mac M4)	Sie können sich Qwen2.5-Coder 32B <code>q4_K_M</code> (~19 GB), Devstral in Q5/Q6 oder langen Kontext ohne Offloading leisten; OOM wird

VRAM	Wichtigste OOM-Maßnahme
Pro 48 GB)	selten. Der VRAM wird mit dem System-RAM geteilt (Unified Memory). Es gibt kein abruptes OOM, aber Speicherdruck, der zum Swapping führt: Beobachten Sie den Speicherdruck des Systems und schließen Sie vor einer großen Sitzung ressourcenintensive Prozesse.
Apple Silicon (Metal)	

Tabelle 13.2 — OOM-Maßnahme nach VRAM-Stufe.

Abschnitt B — Die GPU wird überhaupt nicht verwendet

Das Symptom

`ollama ps` zeigt `100% CPU` an, der GPU-Lüfter steht still, und die Generierung ist überall langsam — nicht wie beim Offloading eingebrochen, sondern einfach *von Anfang an überall* langsam. Das ist der frustrierendste Fall: Die Hardware ist vorhanden, wird aber nicht genutzt.

Die Ursache

Ollama erkennt Ihren Beschleuniger nicht. Drei häufige Ursachen: - Die GPU-Laufzeitumgebung wird nicht erkannt (CUDA-/ROCm-Treiber fehlen oder sind zu alt). - Unter Linux läuft Ollama in einem Kontext (Container, systemd-Dienst), der keinen Zugriff auf das GPU-Gerät hat. - Unter Windows/WSL ist das GPU-Passthrough nicht konfiguriert.

Die Diagnose

```
# NVIDIA (Ihre 5070 Ti) – die GPU muss erscheinen und während einer Generierung einen
Ollama-Prozess auflisten
nvidia-smi

# AMD ROCm
rocm-smi

# Und vor allem: beim Start die Ollama-Logs lesen; dort wird angekündigt, was erkannt
wurde
```

Unter Linux lauten die Logs des Dienstes:

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

Suchen Sie nach einer Zeile, in der Ollama meldet, dass eine GPU-Bibliothek (CUDA/ROCm) gefunden wurde. Wenn angekündigt wird, dass auf die CPU gewechselt wird, haben Sie die Ursache gefunden. (Die saubere Installation der Laufzeitumgebung für Ihr Betriebssystem wird in **Kapitel 4** behandelt — wenn dieser Abschnitt auf Sie zutrifft, wurde der GPU-Schritt aus Kapitel 4 oft übersprungen.)

Der Auszug endet hier. Die Fortsetzung von Abschnitt B sowie die Abschnitte C (Kontextüberlauf), D (falscher Tag) und E (langsame FIM-Autovervollständigung) und die Referenztable Symptom → Ursache → Lösung stehen im Leitfaden.

Die Fortsetzung steht im Leitfaden

25 Kapitel (rund 175 Seiten) und das Paket mit einsatzbereiten Konfigurationen: abgestimmte Modelfiles, Cline, Aider und Tabby, korrekte FIM-Tokens, Spickzettel VRAM → Modell → Befehl. Einmalzahlung 24 €, Updates auf Lebenszeit.

<https://welchesllm.de/copilote-local/>