

あなたのコードコパイロットを、100% ローカルで

第13章 — 壊れたとき：診断して修復する (抜粋) ・ v2026.07 ・ local-llm-navi.com/copilote-local

第13章 — 壊れたとき：診断して修復する

ローカルセットアップの最初の1週間に起きる問題は、ほとんどの場合「モデルがダメ」なのではありません。原因は、いつも同じ5つの障害です。そのせいで、ローカル環境が動かないと思い込んでしまいます。実際には、設定が2行足りないだけです。この章では、**症状ごとに診断**します。目に見えているもの（エラーメッセージ、急落するスループット、もたつく自動補完）から出発し、原因を突き止めて修正します。ここは診断のリファレンスとなる章です。障害が発生すると、第4章、第6章、第12章からここへ案内されます。

作業時の前提：実行しているのは `monprojet`（第7章から主軸として使っている小さなFlask/Python Web API）、バックエンドは Ollama、ハードウェアの基準は RTX 5070 Ti 16 GBです。そのたびに「あなたのVRAM向け」のバージョンを示します。24 GBで有効な修正が、8 GBではまったく逆効果になることがあります。

✓ **覚えておくこと — ローカルデバッグの黄金則。** 1週目の障害の90 %は、**1つの数字**が原因です：
`num_ctx`。 `num_ctx` は、モデルの作業メモリの容量です。リポジトリを収めるためにこれを増やした結果、モデルがVRAMから押し出されました。ほかの原因を疑う前に、モデルがどこで実行されているか（`ollama ps`）と、コンテキストをどれだけ増やしたかを確認してください。

📖 完全版ガイドでは、この章は最初の1週間に起こりやすい5つのトラブルを網羅する判断ツリー（症状 → セクション → 対処法）から始まります。この無料サンプルには、トラブル1（OOM）とトラブル2（GPU が使われない）の冒頭が含まれています。

セクションA — GPU OOM：「out of memory」またはスループットの急落

症状

症状は2種類あります。明白なものは、`CUDA error: out of memory`（またはROCm/Metalの同等メッセージ）が表示され、モデルが応答を拒否するケースです。より巧妙で、頻度も高く厄介なのは、**応答はするものの、毎秒数トークンしか生成できない**ケースです。モデルがVRAMからはみ出しています。Ollamaが一部のレイヤーをCPUへ移しました（これがオフロードです）。16 GBのGPUが、もたつくCPUを眺める状態になります。

原因

VRAMの予算は単純で、自分で次のように整理できます。

```
必要なVRAM ≈ (量子化済みモデルの重み)
                + (num_ctx に応じて増加するKVキャッシュ)
                + オーバーヘッド
```

KVキャッシュは、モデルが処理済みのコンテキストを保存するメモリです。16 GBの環境では、デフォルトのエージェント向けモデル — Devstral Small 24B (q4_K_M、約14 GB) — は、**num_ctx** を増やさな
い限り収まります。典型的な落とし穴は、より多くの monprojet を収めるために、num_ctx を8192から
32768へ変更することです。KVキャッシュが2倍、3倍に増え、エラーメッセージがないまま16 GBの上限
を超えます — 起こるのは、単に静かなオフロードです。

1コマンドで診断する

```
ollama ps
```

PROCESSOR 列を確認してください (詳しい解釈は第4章にあります)。これは、この章全体の中心的な診断
手段です。OOMの診断では、次のように使います。

表示	ollama ps	解釈	対処
100% GPU		すべてVRAM上にあり、正常 です	OOM側で行うことはありません
58%/42% CPU/GPU (または0で ないCPU %)		部分的なオフロード → スルー プットが急落	num_ctx またはモデルサイズを減らし ます
100% CPU		GPUがまったく使われていま せん	セクションBへ進んでください (OOM ではありません)

表13.1 — OOM診断のための ollama ps の読み方。

修正 (影響が小さい順)

1. まず **num_ctx** を下げます。これは最も重要で、最もコストの低い調整項目です。収まる値に戻します
(16 GB環境でのチャット/エージェントのデフォルトは8192です)。ollama ps で確認し、オフロード
の直前まで段階的に上げます。

```
# 直接テスト：控えめなコンテキストを強制し、GPU使用率が100%に戻るか確認
OLLAMA_CONTEXT_LENGTH=8192 ollama run devstral:24b
```

Modelfile側では (完全な構成については第11章を参照)、同じ設定を固定できます。

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

2. 量子化またはサイズを1段階下げます。num_ctx を妥当な値にしてもあふれる場合、そのモデルはグラ
フィックカードに対して大きすぎます。デフォルトで品質と容量のバランスが取れている q4_K_M にとど
め、それ以上メモリを消費する設定にはしないでください。temp/top-p/ num_ctx の微調整については第
12章で、VRAMに合ったモデルの選び方については第5章で扱います。

3. GPUレイヤー数に明示的な上限を設けます（特殊なケース）。自動オフロードを受け入れるのではなく、制御された分担を強制したい場合は、 `num_gpu` でGPUへ送るレイヤー数を固定します。

```
FROM devstral:24b
PARAMETER num_ctx 8192
PARAMETER num_gpu 99 # 「可能な限り多く」；CPUへ意図的に負荷を逃がしたい場合はこの数値を下げる
```

⚠ 警告 — `num_gpu` は魔法の公式ではありません。これを下げても性能が「解放」されるわけではありません。逆に、より多くのレイヤーをCPUへ送るため、遅くなります。これは、意図的に大きすぎるモデルを、遅さを受け入れたうえで動かしたい場合の手段です。日常的に使う設定ではありません。通常の利用では、Ollamaに任せてください。調整するなら、まず `num_ctx` とモデルサイズを変更します。

あなたのVRAM向け：

VRAM	OOMへの主な対処
8 GB	高密度な24Bエージェントモデルは手が届きません。軽量モデル（Qwen2.5-Coder 7B）を選び、 <code>num_ctx ≤ 8192</code> を目安にしてください。コンテキストを増やすと、その分が直接コストになります。 Devstral Small 24Bはここには収まりません。Qwen2.5-Coder 14B（約9 GB）またはDeepSeek-Coder-V2-Lite 16B（約10 GB）を <code>q4_K_M</code> 、 <code>num_ctx 4096</code> で選んでください。それ以上では、 <code>ollama ps</code> を監視します。
12 GB (RTX 5070、RTX 4070)	
16 GB (基準 — RTX 5070 Ti / 4080 / 5080 / RX 9070 XT、Mac M4 24 GB)	<code>q4_K_M</code> （約14 GB）のDevstral Small 24B + <code>num_ctx 8192</code> が、エージェント用途のスイートスポットです。Qwen2.5-Coder 14Bを Q8で動かす余裕もあります。
24 GB (RTX 3090/4090、RX 7900 XTX、Mac M4 Pro 48 GB)	Qwen2.5-Coder 32B <code>q4_K_M</code> （約19 GB）、Q5/Q6のDevstral、またはオフロードなしの長いコンテキストを選べます。OOMはまれになります。 VRAMはシステムRAMと共有されています（ユニファイドメモリ）。突然のOOMではなく、スワップを伴うメモリ圧迫が起きます。システムのメモリプレッシャーを監視し、大きなセッションの前に大きなプロセスを終了してください。
Apple Silicon (Metal)	

表13.2 — VRAM容量別のOOMへの対処。

セクションB — GPUがまったく使われていない

症状

`ollama ps` に 100% CPU と表示され、GPUのファンは眠ったままです。生成はどこでも遅くなります。オフロードのように急激に落ち込むのではなく、最初のトークンから全体的に遅い状態です。最もいら立たしいケースです。ハードウェアはあるのに、使われていません。

原因

Ollamaがアクセラレーターを認識していません。よくある原因は3つです。 - GPUランタイムが検出されていない (CUDA / ROCmドライバーがない、または古すぎる)。 - Linuxでは、OllamaがGPUデバイスにアクセスできない環境 (コンテナ、systemdサービスなど) で動作している。 - Windows/WSLでは、GPUパススルーが設定されていない。

診断

```
# NVIDIA (あなたの5070 Ti) - GPUが表示され、生成中にollamaプロセスが一覧に出る必要がある
nvidia-smi

# AMD ROCm
rocm-smi

# そして何より、起動時のOllamaログを読む。検出内容がそこで報告される
```

Linuxでは、サービスのログを確認します。

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

OllamaがGPUライブラリ (CUDA/ROCm) を検出したと示す行を探します。CPUへ切り替えたと報告されていれば、原因は特定できました。OSに応じたランタイムの正しいインストールは**第4章**で扱っています。このセクションに該当する場合、第4章のGPU手順を飛ばしていることが多いです。

サンプルはここまでです。セクションBの続き、セクションC (コンテキスト超過)、D (タグの誤り)、E (FIM 補完が遅い)、そして症状 → 原因 → 対処法の早見表はガイド本編に収録されています。

続きはガイド本編で

全25章 (約175ページ) と、そのまま貼り付けられる設定パック: 調整済み Modelfile、Cline・Aider・Tabby、正しい FIM トークン、VRAM → モデル → コマンドの早見表。24 € の一回払い、生涯アップデート付き。

<https://local-llm-navi.com/copilote-local/>