Ollama Troubleshooting Guide: Every Common Problem and Fix

January 31, 2026 · by Mark Bartlett

Download this guide as PDF

Quick Answer: Most Ollama problems fall into three categories: GPU not being used (check with 'ollama ps' — if it says CPU, your drivers need attention), out of memory (reduce num_ctx or use a smaller model), or connection issues (make sure the server is running with 'ollama serve'). Enable debug logging with OLLAMA_DEBUG=1 to see exactly what's happening. For slow performance, the #1 cause is the model silently falling back to CPU — verify with 'ollama ps' and check the Processor column. Running Qwen 3.5? Update to v0.17.7 — earlier versions had broken tool calling and a crash bug when splitting across GPU/CPU.

📚 **More on this topic:** Run Your First Local LLM · Ollama vs LM Studio · Open WebUI Setup · llama.cpp vs Ollama vs vLLM · Qwen 3.5 9B Setup Guide · Planning Tool

Ollama is the easiest way to run local LLMs, right up until it isn't. The installation is one command, but when something goes wrong — GPU not detected, model won't load, painfully slow responses — the error messages aren't always helpful.

This guide covers every common Ollama problem with exact commands to diagnose and fix it. Bookmark this for when things break.

What's New in Ollama v0.17.x (March 2026)

If you updated recently and something broke — or you're confused by new features — here's the short version. The current stable release is **v0.17.7** (March 5, 2026).

Auto-update broken on v0.17.1: If you're on v0.17.1, automatic updates won't bring you to v0.17.4 or later. You need to manually re-download from ollama.com/download or re-run the install script. This is a one-time issue — once you're on v0.17.4+, future updates work normally.

Context compaction with `ollama launch` (v0.17.7): When using `ollama launch` with OpenClaw or other integrations, Ollama now supports context length for compaction, which lets long agent sessions stay within the model's context window instead of crashing when the conversation grows too large.

Thinking levels in the API (v0.17.7): Values like `"medium"` for thinking intensity now work correctly for all thinking-capable models. Before v0.17.7, these were silently ignored for some model families.

Dynamic context scaling (v0.17.0): Context length now automatically scales based on available VRAM. This means fewer surprise OOM crashes when you load a model with a long default context on a smaller GPU. You can still override with `OLLAMA_CONTEXT_LENGTH` or `num_ctx`.

Qwen 3.5 tool calling fixed (v0.17.3, v0.17.6): Earlier versions routed Qwen 3.5 through the wrong tool-call pipeline. v0.17.3 fixed parsing during thinking mode, and v0.17.6 fixed the remaining parsing issues. If you had broken tool calling with Qwen 3.5, update to v0.17.7.

Qwen 3.5 GPU/CPU split crash fixed (v0.17.5): Models that split across GPU and CPU would crash on Qwen 3.5. Fixed in v0.17.5.

Peak memory in `--verbose` (v0.17.5): `ollama run --verbose` now shows peak memory usage, which is useful for diagnosing whether your model barely fits or has headroom.

Web search for tool-capable models (v0.17.1): Models that support tool calling can now use web search natively.

OpenClaw auto-setup (v0.17.0): `ollama launch openclaw` handles the full connection between Ollama and OpenClaw without manual config.

`ollama launch` command (v0.16.0+): Start coding integrations like Claude Code, Cline, and OpenClaw directly from the CLI. Run `ollama launch claude` or `ollama launch cline` and it sets up the model connection for you.

Cloud models and privacy (v0.16.2): Ollama added cloud model support (`:cloud` tag) for running larger models on remote hardware. If you don't want any data leaving your machine, go to Ollama app settings and enable "disable cloud models for sensitive and private tasks."

Qwen 3.5 and Gated DeltaNet (v0.17.4+): Qwen 3.5 uses a hybrid attention architecture called Gated DeltaNet that older Ollama versions can't run. If you see "unsupported model architecture" or the model refuses to pull, update to v0.17.4 or newer. This is a new architecture, not just a new model, so the version requirement is hard. You need v0.16.0+ to pull the manifest, v0.17.4+ for the architecture, and v0.17.5+ for GPU/CPU split stability. If you're getting `412` errors when pulling Qwen 3.5 models, see the Qwen 3.5 section below.

How to Check What's Going Wrong

Before fixing anything, gather information. These three commands tell you most of what you need to know:

```
# What models are loaded and where they're running (GPU vs CPU)
ollama ps

# Is the GPU visible to the system?
nvidia-smi          # NVIDIA
rocm-smi            # AMD

# Enable debug logging for detailed diagnostics
OLLAMA_DEBUG=1 ollama serve
```

The `ollama ps` command is your most important diagnostic tool. The **Processor** column shows whether the model is running on GPU, CPU, or a split:

Processor Output	Meaning
`100% GPU`	Fully on GPU — good
`100% CPU`	Entirely on CPU — slow
`48%/52% CPU/GPU`	Split between CPU and GPU — slower than full GPU

If you expected GPU and see CPU, that's your problem. Read the GPU section below.

Where to Find Logs

OS	Location
Linux (systemd)	`journalctl -u ollama --no-pager -f`
Linux (manual)	Terminal output from `ollama serve`
macOS	`~/.ollama/logs/server.log`
Windows	`%LOCALAPPDATA%\Ollama\server.log`
Docker	`docker logs -f ollama`

GPU Not Detected / Running on CPU

This is the #1 problem people hit. The model loads but runs on CPU at 2-8 tok/s instead of GPU at 40-100+ tok/s.

NVIDIA: Diagnosing the Problem

```
# Step 1: Can the OS see your GPU?
nvidia-smi

# If nvidia-smi fails: drivers aren't installed or are broken
# If it works: check the driver version (top of output)
```

Minimum driver version for Ollama: 531+. If yours is older, update.

```
# Step 2: Check what Ollama sees
OLLAMA_DEBUG=1 ollama serve
# Look for lines about GPU detection, CUDA version, VRAM
```

```
# Step 3: Check what's actually running
ollama ps
# Look at the Processor column
```

Common NVIDIA Fixes

Driver issue after update: Certain driver versions (notably 555.85) break Ollama's GPU detection. Downgrade to a known-working version or update to the latest.

After suspend/resume on Linux: The GPU can disappear after waking from sleep.

```
sudo rmmod nvidia_uvm && sudo modprobe nvidia_uvm
```

Force CUDA version: If auto-detection picks the wrong library:

```
OLLAMA_LLM_LIBRARY=cuda_v12 ollama serve
```

Permissions (Linux): Your user needs GPU access:

```
sudo usermod -aG video,render $USER
# Log out and back in
```

Docker: The `--gpus=all` flag is required:

```
docker run -d --gpus=all -v ollama:/root/.ollama -p 11434:11434 ollama/ollama
```

Test GPU passthrough first: `docker run --gpus all ubuntu nvidia-smi`

AMD ROCm: Diagnosing the Problem

```
# Step 1: Can ROCm see your GPU?
rocm-smi
rocminfo

# Step 2: Check kernel messages
sudo dmesg | grep -i amdgpu
```

Common AMD Fixes

Unsupported GPU architecture: Most AMD GPU issues come down to architecture mismatch. Override the GFX version:

```
# Common overrides:
# RX 6600/6600 XT (gfx1032) → set to gfx1030
HSA_OVERRIDE_GFX_VERSION=10.3.0 ollama serve

# RX 7600 (gfx1102) → set to gfx1100
HSA_OVERRIDE_GFX_VERSION=11.0.0 ollama serve
```

For the systemd service (persistent):

```
sudo systemctl edit ollama.service
```

Add:

```
[Service]
Environment="HSA_OVERRIDE_GFX_VERSION=10.3.0"
```

Then:

```
sudo systemctl daemon-reload
sudo systemctl restart ollama
```

Permissions (Linux):

```
sudo usermod -aG render,video $USER
```

iGPU conflict: If your CPU has integrated graphics and ROCm picks the wrong GPU, disable the iGPU in BIOS or set:

```
ROCR_VISIBLE_DEVICES=1 ollama serve    # Use second GPU (check rocminfo for correct ID)
```

Docker with AMD:

```
docker run -d --device /dev/kfd --device /dev/dri \
  -v ollama:/root/.ollama -p 11434:11434 \
  -e HSA_OVERRIDE_GFX_VERSION="10.3.0" \
  ollama/ollama:rocm
```

Verifying the Fix

After any GPU fix, verify with:

```
ollama run llama3.2 --verbose
# Check the eval rate in the output — GPU should give 40+ tok/s for a 3B model
# Then:
ollama ps
# Should show 100% GPU
```

Also watch VRAM usage in real-time:

```
watch -n 1 nvidia-smi     # NVIDIA
watch -n 1 rocm-smi       # AMD
```

VRAM usage should spike when the model loads.

Apple M5 Pro / M5 Max: First Steps

If you just upgraded to a MacBook Pro with M5 Pro or M5 Max and Ollama feels slow or isn't using the GPU properly:

Update Ollama first. Older versions don't know about the M5's Metal 4 GPU architecture or the Neural Accelerator. Run:

```
curl -fsSL https://ollama.com/install.sh | sh
```

Or download the latest `.dmg` from ollama.com. You want v0.17.5 or newer for the best M5 support and MLX stability.

Check that the GPU is actually being used:

```
ollama run qwen3.5:9b --verbose
# Look for the eval rate — M5 Pro should push 50+ tok/s on a 9B model
```

```
ollama ps
# Should show 100% GPU
```

If you're seeing CPU fallback on M5, the most common cause is running an Ollama version from before M5 shipped. The MLX runner improvements in v0.16.3+ expanded architecture support, and v0.17.5 improved MLX stability further. Update and restart.

Unified memory advantage: The M5 Pro's 36GB and M5 Max's 128GB of unified memory mean you can load larger models than discrete GPU users. A 70B Q4 model fits entirely in memory on the M5 Max without any CPU offloading. See our Mac M-series guide for model-to-chip recommendations.

Out of Memory Errors

The error: `llama runner exited, you may not have enough memory to run the model`

This means the model weights + KV cache + overhead exceed your available VRAM (and possibly RAM).

Why It Happens

Total VRAM needed = **Model Weights** + **KV Cache** + **~500MB-1GB overhead**

The KV cache is the part people forget. It scales with context length:

Model Size	2K Context	4K Context	8K Context	32K Context
8B params	~0.3 GB	~0.6 GB	~1.2 GB	~5 GB
14B params	~0.5 GB	~1.0 GB	~2.0 GB	~8 GB
32B params	~1.0 GB	~2.0 GB	~4.0 GB	~16 GB

A 14B model at Q4 takes about 8GB for weights. Add 2GB of KV cache at 8K context plus overhead, and you're at ~11GB. That fits in 12GB VRAM but barely. Bump context to 16K and it won't.

Fixes (In Order of Impact)

1. Reduce context length — the fastest fix:

```
ollama run llama3.2 /set parameter num_ctx 2048
```

Or set it globally:

```
export OLLAMA_CONTEXT_LENGTH=4096
```

Note: As of v0.17.0, Ollama dynamically scales context length based on available VRAM, which reduces surprise OOM crashes. But if you've set an explicit `num_ctx` in a Modelfile or via the API, that override still applies and can still blow your VRAM budget.

2. Enable KV cache quantization — halves KV cache memory:

```
export OLLAMA_FLASH_ATTENTION=1
export OLLAMA_KV_CACHE_TYPE=q8_0
ollama serve
```

Q8 KV cache has negligible quality loss. For even more savings, use `q4_0` (roughly 1/3 the size of f16).

3. Use a smaller model or lower quantization:

- Switch from Q6 to Q4_K_M (saves ~30% VRAM)
- Switch from 14B to 8B (saves ~50% VRAM)
- See our VRAM requirements guide for what fits where

4. Unload other models:

```
ollama stop <model-name>
```

Or limit concurrent models:

```
export OLLAMA_MAX_LOADED_MODELS=1
```

5. Watch out for parallel requests: Setting `OLLAMA_NUM_PARALLEL=4` with `num_ctx=2048` allocates KV cache for an effective 8192 tokens. This alone can push a model off GPU.

The Partial Offload Trap

When a model doesn't fully fit in VRAM, Ollama splits layers between GPU and CPU. This works but performance drops dramatically — from 50+ tok/s to 5-10 tok/s. You might not even notice it's happening unless you check `ollama ps`.

If you see a CPU/GPU split, your model is too large for full GPU loading. Either reduce context, use a smaller model, or upgrade your GPU.

Slow Performance

If Ollama is running but painfully slow, work through these causes:

1. Model Running on CPU (Most Common)

```
ollama ps
```

If the Processor column shows CPU or a CPU/GPU split, that's your answer. See the GPU section above, or reduce model size to fit fully in VRAM.

Expected speeds (full GPU):

Model	RTX 3060 12GB	RTX 3090 24GB	RTX 4090 24GB
8B Q4	~35-45 tok/s	~80-112 tok/s	~95-140 tok/s
14B Q4	~20-25 tok/s	~40-55 tok/s	~55-75 tok/s
32B Q4	Too large	~25-35 tok/s	~34-50 tok/s

If you're seeing 2-8 tok/s on any of these, the model is on CPU.

2. Context Length Too High

Every token of context costs VRAM. Ollama's default is 4096, but some model configs request much higher. Check what your model is using:

```
ollama show <model-name>
# Look for num_ctx in the parameters
```

Lower it if needed:

```
ollama run llama3.2 /set parameter num_ctx 4096
```

3. Multiple Models Loaded

Ollama keeps models in memory by default (up to 3 per GPU). If you've been testing several models, they're all competing for VRAM.

```
ollama ps              # See what's loaded
ollama stop <model>    # Unload specific models
```

Or set auto-unload:

```
export OLLAMA_KEEP_ALIVE=5m    # Unload after 5 minutes idle
```

4. Enable Flash Attention

Free performance improvement, especially for longer contexts:

```
export OLLAMA_FLASH_ATTENTION=1
```

5. Request Serialization With Multiple Models (Known Issue)

If you have two models loaded and send a request to model B while model A is busy with a long generation, model B's request can queue for 50+ seconds even though it's already in VRAM. This is a known concurrency issue as of March 2026. Workaround: run separate Ollama instances on different ports for latency-sensitive multi-model setups.

6. Measuring Performance

```
ollama run llama3.2 --verbose
```

After each response, this prints timing stats including the eval rate (tokens per second). As of v0.17.5, it also shows **peak memory usage** — useful for seeing how close you are to your VRAM ceiling. Use this to verify whether changes actually help.

Installation & Startup Issues

Ollama Won't Start

"command not found: ollama" Ollama isn't installed or isn't in PATH.

```
# Install (Linux)
curl -fsSL https://ollama.com/install.sh | sh

# Verify
which ollama    # Should return /usr/local/bin/ollama
```

"bind: address already in use" Port 11434 is occupied by another process (or a zombie Ollama session).

```
# Find what's using the port
sudo lsof -i :11434          # Linux/Mac
netstat -aon | findstr :11434 # Windows

# Kill the process, then start Ollama
```

Or change the port:

```
OLLAMA_HOST=0.0.0.0:11435 ollama serve
```

Service Management

Linux:

```
sudo systemctl start ollama
sudo systemctl stop ollama
sudo systemctl restart ollama
sudo systemctl status ollama
```

macOS: Launch the Ollama app from Applications, or:

```
ollama serve    # Run manually in terminal
```

Windows: Find “Ollama” in Start Menu or the system tray. For service control, open Services (`services.msc`) and find “Ollama”.

Pinning a Specific Version

If an update breaks something:

```
curl -fsSL https://ollama.com/install.sh | OLLAMA_VERSION=0.17.5 sh
```

Model Download & Pull Issues

Failed or Stuck Downloads

Check network connectivity:

```
curl -I https://registry.ollama.ai/v2/
```

If this fails, it’s a network issue (firewall, VPN, DNS).

Check disk space: Models range from 2GB (small 3B) to 45GB+ (70B). Make sure you have enough free space.

Clear corrupted downloads:

```
sudo systemctl stop ollama      # Stop the service first
rm -rf ~/.ollama/models/*       # Nuclear option: remove all models
rm -rf ~/.ollama/cache/*        # Clear download cache
sudo systemctl start ollama
ollama pull llama3.2            # Re-download
```

Windows equivalent: delete contents of `%HOMEPATH%\.ollama\models` and `%HOMEPATH%\.ollama\cache`.

Proxy issues: If you're behind a corporate proxy:

```
HTTPS_PROXY=https://proxy.example.com ollama pull llama3.2
```

Do **not** set `HTTP_PROXY` — it can break Ollama's internal client-server communication.

Model Name Errors

Model names are exact. Common mistakes:

```
ollama pull llama-3.2     # Wrong — no hyphen
ollama pull llama3.2      # Correct

ollama pull llama3.2:7b   # Wrong — it's 3b or 1b for 3.2
ollama pull llama3.2:3b   # Correct
```

Check available tags on ollama.com/library.

Changing Model Storage Location

Models are stored at:

OS	Default Path
Linux (service)	`/usr/share/ollama/.ollama/models`
Linux (user)	`~/.ollama/models`
macOS	`~/.ollama/models`
Windows	`C:\Users\%username%\.ollama\models`

Move them to a larger drive:

```
export OLLAMA_MODELS=/mnt/large-drive/ollama-models
```

Set this permanently in your systemd override or shell profile.

Qwen 3.5 Model Issues

Qwen 3.5 dropped in late February 2026 and immediately became one of the most popular model families on Ollama. It's also generating a fresh wave of troubleshooting questions. Here's what people are hitting.

"pull model manifest: 412" Error

This is the most common Qwen 3.5 error. It means your Ollama version is too old to understand the model manifest format that Qwen 3.5 uses.

Fix: Update Ollama.

```
curl -fsSL https://ollama.com/install.sh | sh
```

On macOS, download the latest `.dmg` from ollama.com. On Windows, run the installer from the same page. You need v0.16.0 or newer to pull Qwen 3.5, and v0.17.7 for full tool calling, stability, and thinking level support.

After updating, the pull should work:

```
ollama pull qwen3.5:9b
```

Broken Tool Calling

Tool calling with Qwen 3.5 had problems on two fronts, both now fixed:

GGUF conversion issue (35B, 27B): Unsloth flagged broken GGUF conversions that produced malformed tool call responses. Fixed March 2, 2026. Re-pull:

```
ollama pull qwen3.5:35b    # Re-pulls the fixed version
ollama pull qwen3.5:27b
```

Ollama parsing issue (all Qwen 3.5 sizes): Ollama was routing Qwen 3.5 tool calls through the wrong parsing pipeline (Hermes-style JSON instead of the Qwen3-Coder XML format the model was trained on). v0.17.3 fixed parsing during thinking mode. v0.17.6 fixed the remaining cases. **Update to v0.17.6 or newer.**

If you read elsewhere that Qwen 3.5 tool calling is "broken in Ollama" — that was true before v0.17.6. It's fixed now.

Thinking Mode Disabled by Default on Small Models

This catches people off guard. Qwen 3.5 supports "thinking mode" (chain-of-thought reasoning inside `<think>` tags), but **it's disabled by default on the smaller sizes: 0.8B, 2B, 4B, and 9B.**

If you're running `qwen3.5:9b` and expecting chain-of-thought reasoning like you saw in benchmarks or demos, you're not getting it unless you explicitly enable it.

Enable thinking mode in Ollama:

Create a Modelfile:

```
FROM qwen3.5:9b
PARAMETER num_ctx 8192
TEMPLATE """{{- if .System }}<|im_start|>system
{{ .System }}<|im_end|>
{{ end }}<|im_start|>user
{{ .Prompt }}<|im_end|>
<|im_start|>assistant
```

```
<think>
"""
```

Then:

```
ollama create qwen3.5-think -f Modelfile
ollama run qwen3.5-think
```

Or use the `/think` toggle if your client supports it (Open WebUI does).

When to bother: Thinking mode helps on math, logic puzzles, and multi-step coding problems. For plain conversation and simple Q&A, leave it off — it adds latency and the quality difference is marginal. See our Qwen 3.5 9B setup guide for detailed benchmarks with and without thinking.

Vision and Multimodal on Small Models

The Qwen 3.5 small models (0.8B through 9B) are natively multimodal — they handle images without a separate vision adapter. But you need to know how to actually send images.

Via the Ollama CLI:

```
ollama run qwen3.5:9b "Describe this image" --images ./photo.jpg
```

Via the Ollama API:

```
curl http://localhost:11434/api/generate -d '{
  "model": "qwen3.5:9b",
  "prompt": "What do you see in this image?",
  "images": ["'"$(base64 -w 0 photo.jpg)"'"]
}'
```

The image gets base64-encoded and passed in the `images` array. This works with any Qwen 3.5 small model, no extra setup needed.

Common gotcha: If you're using an older Ollama version, the vision capabilities might not work even if the model loads fine. Update to v0.16.0+ to be safe.

For a full walkthrough of what these models can do with images, see our Qwen 3.5 small models overview.

Crash When Splitting Across GPU and CPU

If you're running a Qwen 3.5 model that doesn't fully fit in VRAM and Ollama crashes (not just slow — actually crashes), update to v0.17.5 or newer. Earlier versions had a bug specific to Qwen 3.5 when splitting layers between GPU and CPU. Fixed.

Repetition / Looping Output

Some users reported Qwen 3.5 producing excessively long chain-of-thought loops that never resolved into a final answer, or repeating itself endlessly. This was caused by a missing presence penalty. Fixed in v0.17.5. Update and re-run.

Connection & API Issues

"Could Not Connect to Ollama"

```
# Is the server running?
curl http://localhost:11434
# Should return "Ollama is running"
```

If not:

```
ollama serve                          # Start manually
# or
sudo systemctl start ollama           # Start the service
```

Remote Access (Binding to 0.0.0.0)

Security warning: Researchers have found 175,000+ publicly exposed Ollama instances across 130 countries, many with no authentication. Ollama has no built-in auth. If you bind to `0.0.0.0` and your firewall is open, anyone on the internet can use your GPU. Put a reverse proxy with auth in front, or use a VPN/Cloudflare Tunnel for remote access. Only bind to `0.0.0.0` on a trusted LAN.

By default, Ollama only listens on localhost. To access from other machines:

Linux (systemd):

```
sudo systemctl edit ollama.service
```

Add:

```
[Service]
Environment="OLLAMA_HOST=0.0.0.0:11434"
```

```
sudo systemctl daemon-reload
sudo systemctl restart ollama
```

macOS:

```
launchctl setenv OLLAMA_HOST "0.0.0.0:11434"
# Restart the Ollama app
```

Windows: Set `OLLAMA_HOST` to `0.0.0.0:11434` in System Environment Variables, then restart Ollama from the taskbar.

Open the firewall:

```
sudo ufw allow 11434/tcp    # Linux
```

Docker Networking (Open WebUI)

The most common Docker problem: Open WebUI can't reach Ollama because `localhost` inside the container doesn't point to the host.

Option A — Host networking (Linux, simplest):

```
docker run -d --network=host \
  -v open-webui:/app/backend/data \
  -e OLLAMA_BASE_URL=http://127.0.0.1:11434 \
  --name open-webui ghcr.io/open-webui/open-webui:main
```

Option B — host.docker.internal (Mac/Windows):

```
docker run -d -p 3000:8080 \
  --add-host=host.docker.internal:host-gateway \
  -v open-webui:/app/backend/data \
  --name open-webui ghcr.io/open-webui/open-webui:main
```

Set `OLLAMA_BASE_URL=http://host.docker.internal:11434` in Open WebUI settings.

Option C — Docker Compose (both containerized):

```
services:
  ollama:
    image: ollama/ollama
    environment:
      - OLLAMA_HOST=0.0.0.0:11434
    ports:
      - "11434:11434"
  open-webui:
    image: ghcr.io/open-webui/open-webui:main
    environment:
      - OLLAMA_BASE_URL=http://ollama:11434
    ports:
      - "3000:8080"
```

CORS Issues

If a web app can’t connect to Ollama’s API:

```
export OLLAMA_ORIGINS=http://localhost:3000,http://your-server-ip:3000
```

Set this as an environment variable the same way as `OLLAMA_HOST` (systemd edit, launchctl, or Windows env vars).

Common Error Messages: Quick Reference

Error	Cause	Fix
`could not connect to ollama app`	Server not running	`ollama serve` or `sudo systemctl start ollama`
`bind: address already in use`	Port 11434 occupied	Kill the other process or change port
`llama runner exited, not enough memory`	Model + context exceeds VRAM/RAM	Reduce `num_ctx`, use smaller model, enable KV cache quant
`pull model manifest: 412`	Ollama too old for this model format	Update to v0.17.7: `curl -fsSL https://ollama.com/install.sh | sh`
`model not found`	Typo or model not pulled	Check name, run `ollama pull <model>`
`command not found: ollama`	Not installed or not in PATH	Install with `curl -fsSL https://ollama.com/install.sh | sh`
`connection refused` (Docker)	Container can't reach host	Use `host.docker.internal` or `--network=host`
`Max retries exceeded` (Python)	API server unreachable	Check server is running, check firewall
GPU error code 3	GPU not initialized	Reinstall drivers, check `nvidia-smi`
GPU error code 100	No GPU device found	Driver issue or GPU not connected
`cudaMalloc failed: out of memory`	VRAM exhausted mid-generation	Restart Ollama, reduce concurrent models

---

Environment Variables Reference

The most useful ones to know:

Variable	Default	What It Does
`OLLAMA_HOST`	`127.0.0.1:11434`	Bind address (set to `0.0.0.0` for remote access)
`OLLAMA_MODELS`	OS-specific	Model storage path
`OLLAMA_DEBUG`	`0`	Set to `1` for verbose logging
`OLLAMA_FLASH_ATTENTION`	`false`	Enable Flash Attention (faster, less memory)
`OLLAMA_KV_CACHE_TYPE`	`f16`	KV cache quant: `f16`, `q8_0`, `q4_0`
`OLLAMA_CONTEXT_LENGTH`	`4096`	Default context window
`OLLAMA_NUM_PARALLEL`	`1`	Concurrent requests per model
`OLLAMA_MAX_LOADED_MODELS`	`3 * GPU count`	Max models in memory
`OLLAMA_KEEP_ALIVE`	`5m`	Time before idle model unloads (`-1` = never)
`OLLAMA_ORIGINS`	localhost	Allowed CORS origins
`OLLAMA_LLM_LIBRARY`	auto	Force backend: `cuda_v12`, `rocm`, `cpu`, etc.
`CUDA_VISIBLE_DEVICES`	all	Select specific NVIDIA GPUs (e.g., `0,1`)
`HSA_OVERRIDE_GFX_VERSION`	auto	Override AMD GPU architecture version

How to set them permanently:

- **Linux (systemd):** `sudo systemctl edit ollama.service` → add `Environment="VAR=value"` under `[Service]` → `sudo systemctl daemon-reload && sudo systemctl restart ollama`
- **macOS:** `launchctl setenv VAR "value"` → restart Ollama app
- **Windows:** System Settings → Environment Variables → add/edit → restart Ollama from taskbar
- **Docker:** `-e VAR=value` in the `docker run` command

When to Reinstall vs When to Debug

Debug first if:

- The problem started after a specific change (driver update, new model, config edit)
- `ollama ps` and `nvidia-smi` / `rocm-smi` give useful output
- Logs show a specific error message you can search for

Reinstall if:

- `ollama serve` crashes immediately with no useful output
- Driver issues that can't be resolved with version changes
- Corrupted installation (missing binaries, broken symlinks)

How to clean reinstall:

```
# Linux
sudo systemctl stop ollama
sudo rm /usr/local/bin/ollama
sudo rm -rf /usr/share/ollama      # Removes service user data
rm -rf ~/.ollama                    # Removes your models and config
curl -fsSL https://ollama.com/install.sh | sh
```

Your models will need to be re-downloaded after a clean install. If you just want to reinstall the binary without losing models, skip the `rm -rf ~/.ollama` step.

The Bottom Line

Most Ollama problems come down to three things:

1. **GPU not being used** — check with `ollama ps`, fix drivers or permissions
2. **Not enough memory** — reduce `num_ctx`, enable KV cache quantization, use a smaller model
3. **Server not reachable** — make sure `ollama serve` is running, check the port, configure Docker networking correctly

When in doubt: `OLLAMA_DEBUG=1 ollama serve` tells you everything Ollama is doing. Read the output, search for the error message, and you'll find your fix.

Source: https://insiderllm.com/guides/ollama-troubleshooting-guide/