CUDA Out of Memory: What It Means and How to Fix It

February 18, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Your model plus its context window plus CUDA overhead exceeds your GPU's VRAM. Fix it in order: (1) close other GPU apps, (2) drop quantization from Q8 to Q4_K_M, (3) reduce context length, (4) use a smaller model, (5) offload layers to CPU with --n-gpu-layers. Check your current VRAM usage with nvidia-smi. If nothing works, you need more VRAM — a used RTX 3090 (24GB, ~$700-900) is the most cost-effective upgrade.

📚 **More on this topic:** VRAM Requirements · Quantization Explained · What Can You Run on 8GB VRAM · GPU Buying Guide · Planning Tool

You loaded a model. It crashed. The error says something like:

```
CUDA error: out of memory
```

Or in Ollama:

```
Error: model requires more system memory (X GiB) than is available (Y GiB)
```

You're not doing anything wrong. Your model simply doesn't fit in your GPU's video memory. Here's how to fix it — fast fixes first, explanations after.

Fix It (Ranked by Effort)

1. Close Other GPU Apps

Your GPU's VRAM is shared. A browser with hardware acceleration, Discord, a game running in the background — these eat VRAM silently. Check what's using it right now:

```
nvidia-smi
```

Look at the "Memory-Usage" column. If your 8GB card shows 2GB used before you even load a model, that's your problem. Close Chrome, Discord, and any game. On Linux, `nvtop` gives a real-time view.

Expected savings: 0.5-3GB depending on what's running.

2. Use a Smaller Quantization

Quantization compresses model weights. A 7B model at Q8 needs ~8GB. The same model at Q4_K_M needs ~5GB. Same model, less VRAM, minor quality loss.

Quantization	7B Model	14B Model	32B Model
Q8_0	~8 GB	~15 GB	~34 GB
Q6_K	~6 GB	~12 GB	~26 GB
Q4_K_M	~5 GB	~9 GB	~20 GB
Q3_K_S	~3.5 GB	~7 GB	~16 GB

In Ollama, you can't pick quantization directly — it uses Q4_K_M by default. If that doesn't fit, try a smaller model size. In llama.cpp or LM Studio, you download specific GGUF files at the quant level you want.

Expected savings: 2-15GB depending on what you drop to.

3. Reduce Context Length

Context length is the invisible VRAM eater. The KV cache — where the model stores conversation history — grows with context size. An 8B model with 32K context needs ~4.5GB just for the cache. Cut it to 4K and that drops to ~0.6GB.

```
# llama.cpp: set context to 2048 instead of default
llama-cli -m model.gguf -ngl 99 -c 2048

# Ollama: set in Modelfile or via API
ollama run qwen3:8b /set parameter num_ctx 2048
```

The tradeoff: shorter context means the model "forgets" earlier parts of your conversation sooner. For quick Q&A, 2048 is fine. For long documents or multi-turn conversations, you'll feel the limit.

Expected savings: 1-6GB depending on original vs reduced context.

4. Use a Smaller Model

Sometimes the answer is just "that model is too big for your card." Here's what fits at Q4_K_M with room for a reasonable context window:

Your VRAM	Largest Comfortable Model	Examples
4GB	3B	Llama 3.2 3B, Phi-3.5 Mini
6GB	7B (tight)	Mistral 7B, Qwen3-4B
8GB	8B	Llama 3.1 8B, Qwen3-8B
12GB	14B	Qwen3-14B, DeepSeek-R1-14B
16GB	32B (tight)	Qwen3-32B at Q3
24GB	32B (comfortable)	Qwen3-32B at Q4-Q6

If you're on 8GB trying to run a 14B model — it won't fit. Drop to 8B. The quality jump from 3B to 8B is massive; the jump from 8B to 14B is noticeable but not night-and-day. For a full breakdown, see VRAM requirements by model size.

5. Enable CPU Offloading

If you're almost there — model needs 10GB and you have 8GB — you can offload some layers to system RAM. It's slower but it works.

```
# llama.cpp: offload 20 layers to GPU, rest goes to CPU
llama-cli -m model.gguf -ngl 20 -c 4096

# Start low and increase -ngl until you hit OOM, then back off by 2
```

In Ollama, offloading happens automatically when a model exceeds VRAM. You'll see it in the logs — generation will be noticeably slower (5-15 tok/s instead of 30-50 tok/s) because system RAM bandwidth is 10-15x slower than VRAM.

LM Studio: check "GPU Offload" in the model settings and set the number of layers.

The tradeoff is speed. Every layer on the CPU runs at RAM speed (~50 GB/s) instead of VRAM speed (~936 GB/s on RTX 3090). A few layers offloaded = minor slowdown. Half the model offloaded = painful.

6. Buy More VRAM

If you're constantly fighting the VRAM ceiling, the real fix is hardware:

Upgrade	VRAM	Price	What It Unlocks
RTX 3060 12GB	12GB	$170-220 used	14B models comfortably
RTX 4060 Ti 16GB	16GB	~$400 new	32B at Q3, 14B at Q8
RTX 3090	24GB	$700-900 used	32B at Q6, 70B at Q3

The used RTX 3090 at 24GB is the best value upgrade in local AI. It costs less than a new 12GB card and gives you double the VRAM.

Why This Happens: The Math

VRAM usage isn't just the model. Three things compete for space:

Total VRAM needed = Model weights + KV cache + Overhead

- **Model weights:** (Parameters × Bits per weight) / 8. A 7B model at Q4 = ~3.5GB of raw weights.
- **KV cache:** Grows with context length and model architecture. At 8K context on a 7B model, expect ~1-1.5GB. At 32K, expect ~4-5GB.
- **Overhead:** CUDA runtime, framework buffers, activation memory. Budget 0.5-1.5GB.

A 7B model at Q4_K_M with 8K context: ~3.5 + 1.5 + 1.0 = **~6GB**. That fits on an 8GB card with some room. Push context to 32K and it's ~3.5 + 4.5 + 1.0 = **~9GB**. Now it doesn't fit.

This is why you can load a model fine but crash when the conversation gets long. The KV cache grows as you talk.

Common Traps

Ollama keeps models loaded. After you run a model, Ollama keeps it in VRAM for 5 minutes (default). If you switch models, both might try to coexist. Run `ollama stop` or `ollama ps` to see what's loaded.

```
# See what's loaded
ollama ps

# Stop a specific model
ollama stop qwen3:8b
```

VRAM fragmentation after swapping models. Loading and unloading multiple models can fragment VRAM. If you get OOM on a model that should fit, restart your inference server or reboot. This clears fragmented allocations.

Desktop environment eats VRAM. On Linux with a compositor, your desktop might use 200-500MB of VRAM. On Windows with Aero, similar. If you're on the edge, a headless server setup reclaims that.

Monitoring tools use VRAM too. Running `nvtop` or GPU monitoring widgets adds a small overhead. Ironic, but real when you're fighting for the last 200MB.

Bottom Line

CUDA out of memory means your model + context + overhead > your VRAM. Work through the fixes in order:

1. Close GPU-hungry apps
2. Drop quantization (Q8 → Q4_K_M)
3. Reduce context length
4. Use a smaller model
5. Offload layers to CPU
6. Buy more VRAM

Most people land on fix #2 or #4. If you keep hitting the wall, a used RTX 3090 at $700-900 solves VRAM problems for everything under 70B.

Source: https://insiderllm.com/guides/cuda-out-of-memory-fix/