Why Your Local LLM Is Slow: The num_ctx VRAM Overflow Nobody Warns You About

March 3, 2026

Download this guide as PDF

> **Quick Answer:** If your local LLM is running way slower than expected, check your num_ctx (context window) setting. A 14B model at Q4 uses ~8GB for weights, but a 16K context window adds another 4GB+ of KV cache. On a 12GB card, that overflows into system RAM and tanks speed from 35 tok/s to under 5 tok/s. Fix: set num_ctx=4096 or 8192 to keep everything in VRAM. Check with 'ollama ps' and 'nvidia-smi' while the model is running.

More on this topic: VRAM Requirements for Every LLM | Ollama Troubleshooting Guide | Context Length Explained | Quantization Explained

I spent hours debugging a slow inference problem last week. DeepSeek-R1 14B on an RTX 3060 12GB was running at 4.8 tokens per second. It should have been doing 35. Same model that was fast two days earlier, same GPU, same drivers. Nothing had changed except a config parameter I didn't think to check.

The fix took about ten seconds once I found it. Getting there took most of an afternoon.

The symptom: fast GPU, slow inference

Here's what this looks like in practice. You pull a model that should run fine on your GPU. The specs check out, the VRAM math looks right, `nvidia-smi` shows the GPU is being used. But the actual token generation speed is 5-10x slower than what other people report for the same hardware.

You run `ollama ps` and it says the model is on the GPU. You check the driver version. You restart the service. Nothing helps.

This is what VRAM overflow looks like, and nothing in Ollama's output tells you it's happening.

The wrong diagnoses (I tried them all)

We were running DeepSeek-R1 14B as part of an automated extraction pipeline. The model had been fast, then one day the pipeline started timing out. Steps that used to take 64 seconds were taking 173 seconds. Total pipeline time ballooned from 10 minutes to over 12.

First thing I suspected was thinking mode. DeepSeek-R1 uses `<think>` tags by default, and I figured the chain-of-thought overhead was eating all the time. I disabled it with a custom Modelfile. Still slow.

Then I went deeper. I traced the entire code path from the application daemon through the worker process down to the Ollama API call, looking for a dropped parameter or a misconfigured timeout. Every layer had the correct settings. Every request was formatted right. I added logging at every stage. The bottleneck was clearly in the LLM inference step itself, not in the surrounding code.

I even checked for GPU driver regressions, thermal throttling, and whether another process was eating VRAM. All clean.

After a few hours of this, I did what I should have done first: I tested the model with different context window sizes.

The actual problem: num_ctx overflows VRAM into system RAM

Here's the test I ran:

```
# Test 1: small context window
curl http://localhost:11434/api/generate -d '{
  "model": "deepseek-r1:14b",
  "prompt": "What is 2+2?",
  "options": {"num_ctx": 4096}
}'
# Result: 35.3 tok/s

# Test 2: large context window
curl http://localhost:11434/api/generate -d '{
  "model": "deepseek-r1:14b",
  "prompt": "What is 2+2?",
  "options": {"num_ctx": 16384}
```

```
}'
# Result: 4.8 tok/s (or timeout on longer prompts)
```

Same model. Same GPU. Same prompt. 7x speed difference from one parameter.

The problem is that `num_ctx` controls how much KV cache Ollama pre-allocates, and that KV cache lives in VRAM right alongside the model weights. When the total exceeds your card's VRAM, the excess spills into system RAM. And system RAM is slow.

How slow? DDR4-3200 system RAM runs at about 25.6 GB/s. The GDDR6X on an RTX 3060 runs at 936 GB/s. That's a 37x bandwidth difference. Every model layer that touches the offloaded KV cache runs at RAM speed instead of VRAM speed. The GPU sits there waiting for data from your motherboard like it's sipping through a coffee straw.

The math: why 14B + 16K context doesn't fit in 12GB

The arithmetic is simple once you know to look for it:

- **Model weights** (DeepSeek-R1 14B at Q4): ~8GB
- **KV cache at 16K context**: ~4GB+
- **Framework overhead** (CUDA, Ollama runtime): ~0.5-1GB
- **Total**: ~12.5-13GB

An RTX 3060 has 12GB of VRAM. The model doesn't fit. And Ollama won't tell you. No error, no warning. It just quietly offloads the overflow to system RAM and keeps going at a fraction of the speed.

Drop the context to 4096 and the math changes:

- **Model weights**: ~8GB
- **KV cache at 4K context**: ~1GB
- **Overhead**: ~0.5-1GB
- **Total**: ~9.5-10GB

That fits in 12GB with room to spare. Full GPU speed restored.

At 8192 context, we measured 27.2 tok/s, which is a good compromise if you need more context than 4K but can't afford the VRAM hit of 16K.

How to check if this is your problem

Run these while the model is actively generating:

```
# Watch VRAM usage in real time
watch -n 1 nvidia-smi

# Check if Ollama shows GPU or CPU/GPU split
ollama ps
```

If `nvidia-smi` shows VRAM usage at or near 100% of your card's capacity during inference, you're likely overflowing. If `ollama ps` shows a CPU/GPU split in the Processor column, the overflow is confirmed. See our Ollama troubleshooting guide for more on reading these outputs.

You can also test directly:

```
# Test with a small context window
ollama run deepseek-r1:14b --verbose /set parameter num_ctx 4096
# Note the eval rate

# Test with a large context window
ollama run deepseek-r1:14b --verbose /set parameter num_ctx 16384
# Compare the eval rate
```

If the speed difference is more than 2x, you're overflowing.

The fix: match num_ctx to your available VRAM

Available for KV cache = Total VRAM - Model Weights - 1GB overhead

For a 12GB card running a 14B Q4 model: 12 - 8 - 1 = 3GB left for KV cache. That supports roughly 4K-6K context. 8K is possible with KV cache quantization enabled (`OLLAMA_KV_CACHE_TYPE=q8_0`), which cuts the cache size roughly in half.

You have three ways to set `num_ctx`. In the API call directly:

```
curl http://localhost:11434/api/generate -d '{
  "model": "deepseek-r1:14b",
  "options": {"num_ctx": 4096},
  "prompt": "your prompt here"
}'
```

As a global environment variable:

```
export OLLAMA_CONTEXT_LENGTH=4096
```

Or baked into a Modelfile so you don't have to think about it again:

```
FROM deepseek-r1:14b
PARAMETER num_ctx 4096
```

Then `ollama create my-fast-r1 -f Modelfile`.

Here's a rough guide for safe `num_ctx` values, assuming Q4 quantization and leaving 1GB headroom:

GPU VRAM	7-9B Model	14B Model	32B Model
8GB	8K	2K (tight)	Won't fit
12GB	16K	4-6K	Won't fit
16GB	32K+	8-12K	2-4K
24GB	32K+	32K+	8-12K

These are conservative estimates. You can push higher with KV cache quantization or flash attention enabled (`OLLAMA_FLASH_ATTENTION=1`). But if you're seeing slow speeds, start with these values and work up.

The broader lesson

Ollama's default `num_ctx` isn't optimized for your specific GPU. Depending on the model and your Ollama version, the default might be 4096, or it might be whatever the model's config file

specifies, which for some models is 32K, 64K, or higher. The Ollama documentation on GitHub mentions this, but it's buried.

What bugs me is how silent this failure mode is. No log message that says "hey, your KV cache just spilled into system RAM and you're about to lose 80% of your inference speed." The model just runs slow, and you're left blaming your GPU.

After I fixed the `num_ctx` parameter, our pipeline went from 742 seconds (12 minutes 22 seconds) to 643 seconds (10 minutes 43 seconds). The extraction step alone dropped from 173 seconds to 64 seconds. Almost a 3x improvement on the bottleneck step, from changing one number.

Check your `num_ctx`. Seriously. If you're running a 12GB or 16GB card and your model is slower than it should be, this is probably why. For more on how VRAM, model size, and context interact, see our VRAM requirements guide.
