Context Length Exceeded: What To Do When Your Model Runs Out of Space

February 18, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Context length is the model's working memory — measured in tokens, not words. When it fills, oldest messages get silently truncated and the model forgets. Simplest fix: start a new conversation. For RAG: retrieve fewer, smaller chunks. For coding: send relevant sections, not the whole file. Increasing context with --ctx-size costs VRAM and most models degrade past 8-16K regardless of their advertised limit. Smart retrieval at 4K beats brute-forcing 128K.

📚 **More on this topic:** Context Length Explained · VRAM Requirements · Local RAG Guide · Planning Tool

Your model was answering well. Then it started contradicting itself, forgetting what you said three messages ago, or throwing errors about context limits. The conversation got too long for the model's working memory.

Here's what's happening and how to handle it.

What Context Length Actually Is

Context length is the maximum number of **tokens** the model can process at once. Tokens are not words — they're chunks that the model's tokenizer splits text into. Rough conversion: 1 token is about 0.75 English words, or 4 characters.

The context window holds **everything**:

- System prompt
- Every user message in the conversation
- Every assistant response in the conversation
- Any injected RAG context
- The response currently being generated

A model with 8K context can hold roughly 6,000 words total. That sounds like a lot until you're five messages deep in a technical conversation with code blocks.

Common default context sizes:

Model	Trained Context	Typical Default
Llama 3.1 / 3.3	128K	4K-8K (Ollama default)
Qwen 2.5 / Qwen 3	32K-128K	4K-8K
Mistral / Mistral Nemo	32K-128K	4K-8K
Phi-4	16K	4K
Gemma 2 / 3	8K	8K

Notice the gap: models are trained on 32K-128K, but inference tools default to much less. That's intentional — larger context costs more VRAM.

What Happens When Context Fills Up

This is where people get confused, because the failure is silent.

Ollama silently truncates the oldest messages. Your system prompt stays, your most recent messages stay, but the middle of the conversation vanishes. The model doesn't warn you — it just "forgets" earlier context. You'll notice when it contradicts something it said earlier or asks a question you already answered.

llama.cpp depends on the flag. By default, it may error with `context length exceeded` or silently truncate like Ollama. With `--ctx-size` set below the input length, you get an explicit error.

LM Studio shows a warning when you're approaching the limit and truncates automatically.

The result is the same: the model loses access to old information and its responses degrade. It's not hallucinating (at least, not for this reason) — it literally cannot see the earlier messages anymore.

Fixes by Situation

Casual Chat: Start Fresh

The simplest fix. If you're having a general conversation and quality drops, start a new chat. The model gets a clean context window and responds normally again.

In Ollama: `/bye` then start a new session. In LM Studio: click "New Chat." In code: clear the message array.

If there's important context from the old conversation, summarize it in 2-3 sentences and paste it as the first user message in the new chat. This gives the model the essential context without the bloat.

RAG: Reduce What You Inject

RAG pipelines fill context fast. If you're retrieving 10 chunks of 500 tokens each, that's 5,000 tokens of context consumed before the user even asks a question.

Fixes:

- **Retrieve fewer chunks.** 3-5 relevant chunks usually beat 10+ mediocre ones. Better retrieval > more retrieval.
- **Reduce chunk size.** 200-300 tokens per chunk is often enough. Larger chunks carry more noise.
- **Summarize before injecting.** Run retrieved chunks through a summarizer first, then inject the summary. Uses fewer tokens for the same information.
- **Use a reranker.** Score retrieved chunks by relevance and only inject the top 3. This is the highest-impact improvement for RAG quality.

Coding: Send Only What's Relevant

Pasting an entire 2,000-line file into context is the fastest way to hit the limit. The model doesn't need your entire codebase — it needs the function you're asking about and its immediate dependencies.

Fixes:

- Send the specific function or class, not the whole file
- Include type signatures and import statements for context
- If referencing multiple files, send excerpts with file paths as comments

- For large refactors, work in stages — one file per conversation

Long Documents: Map-Reduce

Processing a 50-page document in a single prompt won't work, even with 128K context. Quality degrades long before you hit the hard limit.

The map-reduce pattern:

1. **Split** the document into sections that each fit in context
2. **Map**: process each section independently (summarize, extract, analyze)
3. **Reduce**: combine the results in a final pass

This works for summarization, data extraction, and analysis. It's how every production RAG system handles long documents.

Increasing Context Length

You can give the model a bigger context window. It costs VRAM.

How to Set It

```
# llama.cpp
llama-cli -m model.gguf --ctx-size 16384

# Ollama (in Modelfile)
PARAMETER num_ctx 16384

# Ollama (at runtime)
curl http://localhost:11434/api/generate -d '{
  "model": "qwen3:8b",
  "prompt": "Hello",
  "options": { "num_ctx": 16384 }
}'
```

The VRAM Cost

Context length directly controls the KV cache size. The KV cache stores attention states for every token in context and grows linearly with context length.

Context Size	Approximate KV Cache (7B model)	KV Cache (14B model)
4K	~0.5 GB	~1 GB
8K	~1 GB	~2 GB
16K	~2 GB	~4 GB
32K	~4 GB	~8 GB
128K	~16 GB	~32 GB

This is on top of the model weights. A 14B Q4_K_M model takes ~9 GB for weights. At 32K context, the KV cache adds another ~8 GB. You now need ~17 GB of VRAM for a model that "only needs 9 GB."

This is why inference tools default to 4K-8K context — it's the sweet spot between usable conversation length and VRAM consumption.

Flash attention reduces KV cache memory by 2-4x. If your engine supports it (`--flash-attn` in llama.cpp), enable it before increasing context.

The Quality Problem

Here's what most guides won't tell you: advertising "128K context" doesn't mean the model works well at 128K.

Most models perform best in the range they were primarily trained on. Llama 3.1 was trained with 128K context but performs best under 8-16K. At 64K+, accuracy on information in the middle of the context drops measurably — the "lost in the middle" problem.

Practical guidelines:

- **Under 8K:** Most models perform at full quality
- **8K-16K:** Slight degradation, still very usable
- **16K-32K:** Noticeable drop on recall tasks, fine for generation
- **32K+:** Use only if you specifically need it and test your use case
- **128K:** Works for needle-in-haystack but don't expect consistent quality across the full window

RoPE Scaling

Some models can extend their context beyond training length using RoPE (Rotary Position Embedding) scaling. In llama.cpp, this is `--rope-freq-scale` or `--rope-freq-base`. In Ollama, it's `rope_frequency_base` and `rope_frequency_scale` parameters.

The tradeoff: extended context works, but quality degrades faster than natively trained long context. A model trained on 8K and extended to 32K via RoPE will perform worse at 32K than a model natively trained on 32K.

The Honest Truth

4K-8K context with good retrieval beats 128K context with everything stuffed in. Every production AI system — ChatGPT, Claude, Gemini — uses retrieval and summarization behind the scenes. They don't just dump 100K tokens into the model and hope for the best.

For local AI:

- **Default context (4K-8K)** handles most conversations and simple tasks
- **16K context** is the practical ceiling for quality with affordable VRAM
- **Beyond 16K**, invest in better retrieval instead of bigger context

The model doesn't need to see everything at once. It needs to see the right things.

Bottom Line

Context length is a finite resource. When it fills, the model forgets — silently and without warning. The fix depends on your use case: start fresh for chat, trim injections for RAG, send excerpts for coding. Increasing context is possible but costs VRAM and quality.

The best approach is not a bigger window — it's better retrieval that puts the right information in a smaller window.

Source: https://insiderllm.com/guides/context-length-exceeded-fix/