Best Local LLMs for RAG in 2026

February 6, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** For RAG on 8GB VRAM, run Qwen 2.5 7B (Q4_K_M, 128K context). For 12-16GB, Qwen 3 14B or Gemma 3 12B are the best instruction followers. For 24GB, Command R 35B is the only local model with built-in citation grounding — it returns inline source references instead of hallucinating them. The sleeper pick is Qwen 3 30B-A3B — a mixture-of-experts model with only 3B active parameters that fits in 16GB at Q4 but performs like a much larger model. For embeddings, use nomic-embed-text (768d, 8K context) for most setups or bge-m3 for multilingual documents. Pair with Open WebUI + Ollama for the easiest RAG pipeline.

📚 **More on this topic:** Local RAG Setup Guide · Context Length Explained · VRAM Requirements · Open WebUI Setup

RAG — retrieval-augmented generation — lets your local LLM answer questions about your own documents by searching them first and feeding relevant chunks as context. Instead of retraining the model, you give it the right information at query time.

The model you pick for RAG matters more than the model you pick for general chat. A chat model just needs to sound coherent. A RAG model needs to follow instructions precisely, stick to the retrieved context instead of making things up, and handle long inputs without losing information in the middle. Most local LLMs are mediocre at this. Some are good. One is built specifically for it.

This guide covers which models to run at each VRAM tier, which embedding models to pair them with, and the failure modes that bad model choices cause.

What Makes a Good RAG Model

Not every good LLM is a good RAG model. The traits that matter:

Context window size. Your prompt includes the query, system instructions, and every retrieved chunk. A 4K context model can fit maybe 2-3 chunks. A 128K context model can fit dozens. Bigger isn't always better (models lose accuracy in the middle of long contexts), but anything under 32K is limiting.

Instruction following. RAG prompts say "answer based only on the provided context." Weak instruction followers ignore this and mix in their training data, which is where hallucinations come from. The model needs to respect the boundary between "context says X" and "I think X."

Grounded generation. Only one local model — Command R — is specifically trained to cite its sources with inline references. Every other model needs prompt engineering to produce citations, and most do it unreliably.

Context utilization. Research on Llama 3.1 and GPT-4 shows performance degrades when you use too much of the context window. The sweet spot is 40-70% utilization. A 128K model performs best with 50-90K tokens of actual content, not 128K crammed full.

Best Models by VRAM Tier

8GB VRAM (RTX 3060, RTX 4060, Arc B580)

Model	Context	VRAM (Q4_K_M)	RAG Strength
Qwen 2.5 7B	128K	~4.7 GB	Best instruction following at this size
Llama 3.1 8B	128K	~6 GB	Strong general quality, large community
Mistral 7B v0.3	32K	~4.5 GB	Fastest, but 32K context limits chunk count
Qwen 3 4B	128K	~3 GB	Fits anywhere, hybrid thinking mode

Pick Qwen 2.5 7B. At 128K context and ~4.7GB at Q4, it leaves enough VRAM for the embedding model and a reasonable KV cache. Instruction following is the best in the 7B class — it respects "answer only from context" prompts more reliably than Llama 3.1 8B.

Mistral 7B v0.3 is the fastest option but its 32K context window limits you to 5-7 retrieved chunks comfortably. If your documents are short and queries are simple, it works. For anything involving long documents or many chunks, the 128K models are better.

Qwen 3 4B is the new wildcard — hybrid thinking mode (it can reason step-by-step or answer directly) in just 3GB at Q4. Good for constrained hardware but weaker on complex multi-document queries than the 7B models.

```
# Ollama setup for 8GB VRAM RAG
ollama pull qwen2.5:7b
ollama pull nomic-embed-text
```

12-16GB VRAM (RTX 3060 12GB, RTX 4060 Ti 16GB, RTX 4070)

Model	Context	VRAM (Q4_K_M)	RAG Strength
Qwen 3 14B	128K	~9 GB	Best overall at this tier, thinking mode
Qwen 2.5 14B	128K	~9 GB	Proven, well-tested
Gemma 3 12B	128K	~7 GB	QAT models preserve near-FP16 quality
Qwen 3 30B-A3B	128K	~15 GB	MoE sleeper — 30B total, 3B active

Pick Qwen 3 14B for 16GB VRAM. Pick Gemma 3 12B for 12GB.

Qwen 3 14B is the best RAG model in this range. 128K context, hybrid thinking mode (useful for multi-step reasoning over retrieved documents), and strong instruction following. At ~9GB Q4, it fits on 12GB cards but leaves limited room for KV cache — 16GB is more comfortable.

Gemma 3 12B is the best fit for exactly 12GB VRAM. Google's QAT (Quantization-Aware Trained) versions preserve near-BF16 quality at int4 sizes, which matters for RAG accuracy. Ollama supports the QAT variants natively.

The sleeper pick: **Qwen 3 30B-A3B**. This MoE model has 30 billion total parameters with 128 experts, but only activates 8 experts (~3B parameters) per token. At Q4, it needs ~15-17GB VRAM. The trick: you get 30B-level knowledge with 3B-level inference cost. On CPU, it runs at 12-15 tok/s with 32GB RAM. The catch is higher disk usage — the full model weights are 30B even though only 3B are active.

```
# 16GB VRAM setup
ollama pull qwen3:14b
ollama pull nomic-embed-text

# Or the MoE sleeper
ollama pull qwen3:30b-a3b
```

24GB VRAM (RTX 3090, RTX 4090)

Model	Context	VRAM (Q4_K_M)	RAG Strength
Command R 35B	128K	~19 GB	Only model with built-in citations
Gemma 3 27B	128K	~14 GB	Room for large KV cache
Qwen 3 14B (Q8)	128K	~15 GB	Higher quant = better accuracy

Pick Command R 35B if you need citations. Pick Gemma 3 27B if you need headroom.

Command R is the standout for RAG at 24GB. Cohere built it specifically for retrieval-augmented generation with grounded generation — the model accepts document snippets and returns responses with inline citations tracing information back to source documents. No other local model does this natively. Every other model requires prompt engineering to produce citations, and they hallucinate sources 17-33% of the time (based on legal RAG research).

At ~19GB Q4, Command R fits on a 3090/4090 with room for a reasonable context window. The trade-off: CC-BY-NC license means non-commercial use only.

If you don't need citations, Gemma 3 27B at ~14GB Q4 leaves 10GB free for KV cache — enough to use a large portion of that 128K context window without running out of memory. Or run Qwen 3 14B at Q8 quantization for better accuracy than Q4 with the same VRAM headroom.

```
# 24GB citation-focused RAG
ollama pull command-r:35b
ollama pull nomic-embed-text
```

Beyond 24GB (Multi-GPU, GB10, Mac)

Llama 3.3 70B and Qwen 2.5 72B are excellent RAG models — they match or exceed 405B-class quality on instruction following and context utilization. At Q4, they need ~42-48GB VRAM (two 24GB GPUs or a Mac with 64GB+ unified memory). If you have the hardware, these are the best local RAG models that exist. But for most people, Command R 35B on a single 24GB GPU is 90% of the quality at half the cost.

Embedding Models — What Converts Your Documents to Vectors

The embedding model runs separately from your chat model. It converts text chunks into numerical vectors for similarity search. You need one. Here's what to use.

Model	Params	Dimensions	Max Tokens	MTEB Score	Best For
nomic-embed-text	137M	768	8,192	62.4	Default choice, good balance
bge-m3	568M	1,024	8,192	63.0	Multilingual, hybrid retrieval
mxbai-embed-large	335M	1,024	512	64.7	Highest accuracy (English)
snowflake-arctic-embed2	303M	1,024	8,192	—	Multilingual, Matryoshka
stella_en_1.5B	1.5B	1,024	512	71.2	Maximum retrieval quality
Qwen3-Embedding	0.6-8B	up to 4,096	32,000	70.6	Multilingual, longest context

The Default: nomic-embed-text

Use this unless you have a specific reason not to. 137M parameters (tiny — runs alongside any chat model), 8,192 token context (handles large chunks), 768 dimensions (good balance of accuracy and storage). Fully open source — code, data, and weights. Available in Ollama.

```
ollama pull nomic-embed-text
```

For Multilingual Documents: bge-m3

If your documents include non-English text, bge-m3 supports 100+ languages and offers three retrieval methods in one model: dense, sparse (keyword-like), and ColBERT (multi-vector). The 8K context window handles large chunks. 568M parameters — still small enough to run alongside any chat model.

For Maximum English Accuracy: mxbai-embed-large or stella_en_1.5B

mxbai-embed-large scores 64.7 on MTEB (outperforms OpenAI's text-embedding-3-large) in just 335M parameters. The catch: 512-token max context means you need smaller chunks.

stella_en_1.5B scores 71.2 on MTEB — the highest of any open model at its size — but at 1.5B parameters it needs ~3GB, which starts competing with your chat model for VRAM. Worth it if retrieval quality is your bottleneck and you have 16GB+ VRAM.

The New Option: Qwen3-Embedding

Released June 2025. The 0.6B variant is practical for local use. The 8B variant scored #1 on MTEB Multilingual but needs significant VRAM. The standout feature: 32K token context, which means you can embed entire documents as single chunks if your retrieval strategy benefits from that.

What Not to Use

Don't use your chat model for embeddings. Ollama supports dedicated embedding models for a reason — they're smaller, faster, and purpose-built for vector similarity. A 7B chat model produces embeddings that are both slower and less accurate than a 137M embedding model.

RAG Stack Options

You need three components: an LLM for generation, an embedding model for vectorization, and software to wire them together.

Easiest: Ollama + Open WebUI

Open WebUI has built-in RAG. Upload documents, query with `#`, done. It handles chunking, embedding, vector storage (ChromaDB by default), and retrieval automatically.

Critical setting: Ollama defaults to 2,048 token context. For RAG, you must increase this to at least 8,192 — ideally 16,000+ — in Open WebUI's admin panel under Models > Advanced Parameters. Without this, retrieved context gets silently truncated and the model answers from memory instead of your documents.

Default chunk size is 500 tokens with configurable overlap. Set overlap to 10-20% for best results.

For the full setup walkthrough, see our Open WebUI setup guide and Local RAG guide.

More Control: AnythingLLM

AnythingLLM gives you workspaces (separate document collections), supports Ollama and LM Studio as backends, and bundles LanceDB for vector storage. The desktop app is single-user and works out of the box. Docker deployment supports multi-user access.

Better than Open WebUI when you need to organize documents into separate knowledge bases — one workspace for legal docs, another for technical specs, another for meeting notes.

Full Customization: Python + LangChain

For developers who want control over every component — custom chunking strategies, reranking, hybrid search, multi-step retrieval chains. The setup is ~30-50 lines of Python connecting document loaders, a text splitter, an embedding model, a vector store (ChromaDB or FAISS), and an LLM via Ollama.

Use this when Open WebUI and AnythingLLM don't support your retrieval strategy. For most personal RAG setups, the GUI tools are faster to configure and produce equivalent results.

Why Your RAG Isn't Working — Common Failures

Most RAG problems aren't retrieval problems. They're model problems.

Hallucinated Citations

You ask for sources and the model invents plausible-looking references that don't exist in your documents. Legal RAG systems hallucinate citations 17-33% of the time. Local models without grounding training are worse.

Fix: Use Command R (built-in citation grounding) or add explicit instructions: "Only cite documents that appear in the provided context. If the context doesn't contain the answer, say so." Even with good prompting, expect some hallucinated citations from non-Command-R models. Verify critical citations manually.

Ignoring Retrieved Context

The model answers from its training data instead of the chunks you retrieved. This happens when the system prompt is vague ("answer the question") instead of explicit ("answer based ONLY on the following documents").

Fix: Use a system prompt that forces grounding:

```
You are a document assistant. Answer questions using ONLY the provided context.
If the context doesn't contain enough information, say "I don't have enough
information in the provided documents to answer this."
Do not use any knowledge from your training data.
```

Stronger instruction-following models (Qwen 2.5, Command R) respect this more reliably than weaker ones.

Lost in the Middle

Models attend more to the beginning and end of long contexts, missing information in the middle. If your answer is in chunk #4 out of 8, the model might miss it.

Fix: Limit retrieval to 5-7 chunks. Research on Llama 3.1 70B found optimal performance at 7-9 chunks using 40-70% of the context window. More chunks doesn't mean better answers — it means more noise. Also consider reranking: retrieve 20 chunks, rerank by relevance, pass only the top 5-7 to the model.

Summarizing Instead of Answering

You ask a specific question and the model gives you a summary of the retrieved documents instead. This is a prompt engineering problem — the model defaults to "describe what these documents say" instead of "extract the specific answer."

Fix: Be explicit in your prompt: "Answer the question directly and concisely. Do not summarize the documents." Also helps to rephrase queries as specific questions rather than topics. "What was Q3 revenue?" works better than "Tell me about Q3 financials."

Chunking and Context Window Interaction

How you split documents interacts directly with which model you run.

The Optimal Chunk Size

256-512 tokens with 10-20% overlap is the starting point for most setups.

- **Too small (100 tokens):** Fragments context. The embedding captures a sentence fragment, and the model gets orphaned snippets that lack enough information to answer from.
- **Too large (2,000+ tokens):** Dilutes the embedding. The vector represents a mix of topics, retrieval precision drops, and the model gets noise alongside signal.

The Chunking Formula

Your total prompt needs to fit in the model's context window:

```
(chunk_size × num_chunks) + system_prompt + query < 70% of context window
```

For a 128K model, that means staying under ~90K tokens of total input. For a 32K model (Mistral 7B), you're limited to ~22K tokens. At 500-token chunks, that's ~180 chunks on a 128K model vs ~44 chunks on a 32K model.

In practice, you rarely need more than 5-10 chunks. The diminishing returns hit fast — chunk #8 is usually less relevant than chunk #3, and it pushes useful information into the "lost in the middle" zone.

Embedding Context vs Chunk Size

Your embedding model's max token limit constrains your chunk size. nomic-embed-text handles 8,192 tokens per chunk. mxbai-embed-large only handles 512. If your chunks are 1,000 tokens and your embedding model caps at 512, the embeddings only represent the first half of each chunk — retrieval accuracy suffers silently.

Match your chunk size to your embedding model's capacity. If using mxbai-embed-large (512 max), keep chunks under 500 tokens. If using nomic-embed-text (8,192 max), you can experiment with larger chunks for documents where context spans multiple paragraphs.

The Recommendation

Your Setup	Generation Model	Embedding Model
8GB VRAM, getting started	Qwen 2.5 7B (Q4)	nomic-embed-text

Your Setup	Generation Model	Embedding Model
12GB VRAM	Gemma 3 12B (QAT)	nomic-embed-text
16GB VRAM	Qwen 3 14B (Q4)	nomic-embed-text
16GB, want a bigger model	Qwen 3 30B-A3B (Q4)	nomic-embed-text
24GB, need citations	Command R 35B (Q4)	nomic-embed-text
24GB, need headroom	Gemma 3 27B (Q4)	bge-m3
Multilingual documents	Any above	bge-m3
Maximum retrieval quality	Any above	stella_en_1.5B

→ Check what fits your hardware with our Planning Tool.

For the software stack, start with Ollama + Open WebUI. It works in 10 minutes and handles 80% of RAG use cases. Move to AnythingLLM if you need workspace separation, or Python + LangChain if you need custom retrieval logic.

Set Ollama's context to 16,000+ tokens. Use 500-token chunks with 10% overlap. Retrieve 5-7 chunks per query. And use a system prompt that explicitly forces the model to answer from context only.

Related Guides

- Local RAG: Search Your Documents with Private AI
- VRAM Requirements for Local LLMs
- Context Length Explained
- Open WebUI Setup Guide
- Best Local Models for OpenClaw

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/best-local-llms-rag/