RWKV-7: Infinite Context, Zero KV Cache — The Local-First Architecture

February 25, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** RWKV-7 is an RNN-based architecture with no KV cache and constant memory usage regardless of context length. A 7B RWKV model uses the same VRAM at 100 tokens as it does at 100,000 tokens, while a 7B transformer's memory grows with every token generated. The 2.9B model matches Qwen2.5-3B on English benchmarks (71.5% vs 71.4% average) despite training on 3x fewer tokens. It runs on llama.cpp and Ollama with GGUF support. The trade-off: smaller community, fewer fine-tunes, and slightly weaker on tasks that need precise recall of specific details buried in long contexts. But for long document processing, edge deployment, and always-on local assistants where stable memory matters more than peak benchmark scores, RWKV-7 is the better architecture.

Related: Memory Leak in Long Conversations: Causes and Fixes · Context Length Explained · VRAM Requirements · Running AI Offline · Planning Tool

The number one complaint in local AI: “my model ran out of VRAM during a long conversation.” You start chatting, everything’s fast, and 30 minutes later your GPU is thrashing or the process crashes. The culprit is the KV cache, a data structure that every transformer builds during inference. It grows with every token in the conversation. More context, more memory, until something breaks.

RWKV-7 doesn’t have a KV cache. It processes each token with a fixed-size state that never grows. A 7B RWKV model uses the same memory at token 100 as it does at token 100,000. Context length has zero effect on VRAM usage. Read that again, because if you’ve spent any time managing memory on local hardware, that sentence should feel like a relief.

The catch: it’s not a transformer, which means the ecosystem is smaller, the community is thinner, and raw benchmark quality is slightly behind on some tasks. But for specific workloads, the no-KV-cache architecture is the better choice.

How RWKV works (without the math)

RWKV (pronounced “RwaKuv”) is an RNN at inference time and a transformer at training time. This hybrid approach means it trains efficiently on GPUs using parallelism (like a transformer) but runs inference sequentially with fixed memory (like an RNN).

Here’s how it differs from a transformer:

A transformer processes your entire conversation at once. Every token can “look at” every other token through the attention mechanism. This is powerful but expensive: the attention computation scales quadratically with sequence length (O(T^2) time, O(T^2) memory). The KV cache stores the keys and values for every previous token so they don’t need to be recomputed, but that cache grows linearly with conversation length.

RWKV processes tokens one at a time and maintains a compressed “state” that summarizes everything it’s seen. This state has a fixed size regardless of how many tokens have passed through it. New tokens update the state, old information gets naturally compressed. Time complexity is O(T), memory is O(1) per token.

RWKV-7 (“Goose”) is the latest version. It introduces “Dynamic State Evolution,” which lets the model update its internal state more expressively than previous versions. The paper (March 2025, arXiv:2503.14456) shows it can do things that fixed-depth attention provably cannot, like state tracking and recognizing all regular languages.

In practice, RWKV trades the ability to look back at any arbitrary token in the conversation (what transformers do with attention) for constant memory and constant speed. It compensates by being smarter about what it puts into its state.

The VRAM comparison that matters

This is the table you came here for. Transformer VRAM grows with context length because of the KV cache. RWKV VRAM stays flat.

Approximate memory at different context lengths for 7B-class models (Q4 quantization, single GPU):

Context length	Transformer 7B (Llama/Qwen)	RWKV-7 7B
2K tokens	~5.5 GB	~4.5 GB
8K tokens	~7 GB	~4.5 GB

Context length	Transformer 7B (Llama/Qwen)	RWKV-7 7B
16K tokens	~9 GB	~4.5 GB
32K tokens	~12 GB	~4.5 GB
64K tokens	~18 GB	~4.5 GB
128K tokens	~30+ GB	~4.5 GB

The transformer column keeps climbing. The RWKV column doesn't move. At 2K tokens the difference is small. At 32K it's 2.5x. At 128K the transformer needs a multi-GPU setup or CPU offloading while RWKV runs comfortably on a single 8GB card.

This is why RWKV matters for local AI. Not because it beats transformers on MMLU. Because it solves the memory growth problem that kills long conversations on consumer hardware.

An int8 RWKV 14B model can run on sequences of any length with about 3GB of VRAM. Try that with a 14B transformer at 32K context.

Available models

RWKV-7 ships in two series. G1 is the current recommendation:

Model	Parameters	Training tokens	Notes
RWKV-7-G1 0.4B	400M	~3T	Tiny, good for experimentation
RWKV-7-G1 1.5B	1.5B	~5T	Runs on phones and Pi
RWKV-7-G1 2.9B	2.9B	5.6T	Best benchmarked size, 3B SoTA on multilingual
RWKV-7 7B	7B	Training	Comparable to Llama 3.2 3B quality
RWKV-7 14B	14B	Training	Largest available

The 2.9B model is the star. It matches Qwen2.5-3B on English benchmarks (71.5% average vs 71.4%) while training on 5.6 trillion tokens compared to Qwen's 18 trillion. That's the same quality from 3x less training data, which speaks well for the architecture's efficiency.

On the Hacker News thread discussing these results, the claim is RWKV-7 averages 72.8% across standard benchmarks versus Llama 3.2-3B's 69.7%. That was measured with fewer than a third of Llama's training tokens.

All models are released under Apache 2.0. The RWKV team also released their 3.1 trillion token multilingual training corpus, which is unusual and appreciated.

GGUF files are available from the RWKV GGUF collection on HuggingFace, with quantizations from Q4_K_M through FP16. The RWKV wiki recommends Q8_0 or higher, noting that lower quantizations degrade quality more than they do for transformers.

Running RWKV-7 locally

Ollama (easiest)

```
ollama run mollysama/rwkv-7-g1:2.9b
```

That's it. Ollama downloads the Q8_0 GGUF and starts a chat session. The G1 models support thinking mode by default (chain-of-thought traces before answering). Toggle it off with `/set nothink` if you want direct answers.

Other sizes available:

```
ollama run mollysama/rwkv-7-g1:1.5b
ollama run mollysama/rwkv-7-g1:2.9b-q6_k
```

llama.cpp

Download a GGUF from the RWKV collection, then:

```
./llama-cli -m models/rwkv-7-world-2.9b-Q8_0.gguf \
  -p "You are a helpful assistant" \
  -cnv -t 8 -ngl 99 -n 500
```

For a web interface:

```
./llama-server -m models/rwkv-7-world-2.9b-Q8_0.gguf -ngl 99
```

This gives you an OpenAI-compatible API at `http://127.0.0.1:8080`.

Other options

RWKV also runs on KoboldCpp, Text Generation WebUI, and SillyTavern. There's a dedicated `rwkv` Python package for direct integration, and ChatRWKV for a standalone chat interface. The native rwkv.cpp engine exists too, though llama.cpp has become the more practical choice since it now handles RWKV natively.

Where RWKV wins

Long document processing

Lawyers, researchers, anyone who feeds long documents into a model. A 50-page contract is roughly 25K tokens. With a transformer, that burns 25K tokens of KV cache before you've asked a single question. With RWKV, the document streams through the fixed-size state. VRAM doesn't care how long the document is.

Edge deployment

RWKV-7 on an ARM Cortex-A76 (Raspberry Pi class) generates 16.39 tok/s. For comparison, Llama 2-7B with INT4 on a Raspberry Pi 4 manages 0.11 tok/s. That's not a typo. RWKV is 150x faster on the same class of hardware because the constant-memory architecture maps naturally to devices with limited RAM and no GPU.

On a desktop NVIDIA RTX 5090, the 7B model at FP16 pushes 10,250+ tok/s in batched inference. Different hardware, same architectural advantage.

Always-on assistants

If you run a local AI assistant that stays active all day, memory stability matters. Transformer-based assistants slowly eat more memory as conversations grow, eventually needing a context reset or restart. RWKV-based assistants use the same memory at hour 8 as they did at hour 1. No memory leaks from the model side, no gradual slowdown, no surprise OOM crashes.

RAG with long contexts

RAG pipelines retrieve chunks of documents and stuff them into the context window. More chunks, more KV cache, more memory. With RWKV, you can feed in as many retrieved chunks as you want without worrying about VRAM limits. The bottleneck moves from memory to quality of the retrieved content, which is where it should be.

Offline and air-gapped systems

RWKV's predictable resource usage makes it ideal for air-gapped deployments where you can't afford surprises. You know exactly how much memory the model will use, regardless of what users throw at it. No edge cases that suddenly double VRAM consumption.

Where transformers still win

Raw quality per parameter

At the same parameter count, transformers generally produce more accurate and nuanced outputs on short-context tasks. The attention mechanism's ability to look back at any token is genuinely useful for complex reasoning where specific earlier details matter. RWKV's compressed state loses some of that granularity.

The gap has narrowed significantly with RWKV-7 (matching 3B transformers on English benchmarks), but on tasks like creative writing, complex multi-step reasoning, and precise information retrieval from context, transformers still have an edge.

Ecosystem maturity

Every major tool is transformer-first. Ollama, LM Studio, vLLM, Open WebUI. They all support RWKV now (llama.cpp merged RWKV support), but the experience is less polished. You won't find as many pre-made Modelfiles, fewer community guides, and the model selection is smaller.

Fine-tuning is the biggest gap. There are thousands of Llama and Qwen fine-tunes on HuggingFace. RWKV fine-tunes are scarce. If you need a model customized for your specific domain, transformers offer far more options.

Tool calling and structured output

Function calling, JSON mode, structured output — these features are well-tested on transformer models and less battle-tested on RWKV. If your application depends on reliable structured output, stick with a transformer until RWKV tooling catches up.

Community size

The RWKV Discord is active but small compared to r/LocalLLaMA's transformer-focused community. When you hit a problem, there are fewer people who've solved it before you. The HuggingFace download counts tell the story: RWKV-7 models have hundreds of downloads. Comparable Llama and Qwen models have millions.

When to use which

Your situation	Use this
Long conversations that crash from OOM	RWKV
Raspberry Pi or phone deployment	RWKV
Processing 50+ page documents	RWKV
Always-on assistant with stable memory	RWKV
Best quality on short tasks	Transformer
Need specific fine-tunes	Transformer
Tool calling and structured output	Transformer
First local AI setup (ecosystem support)	Transformer
RAG with large context windows	RWKV
Air-gapped/offline with predictable resources	RWKV

The bigger picture

RWKV, Mamba, and LFM2 are all attacking the same problem from different angles: transformer inference is expensive, and KV cache growth is the specific bottleneck that hurts local users most. Each offers a different trade-off between quality and efficiency.

RWKV is the most mature of the three for local deployment. It has llama.cpp and Ollama support, GGUF files ready to download, and a 2.9B model that competes with 3B transformers on benchmarks. The 7B and 14B models push quality higher for users with more hardware.

The architecture doesn't need to replace transformers everywhere. It needs to be good enough for the specific cases where constant memory matters more than peak quality. For long documents, edge devices, always-on assistants, and memory-constrained hardware, it already is.

Start with the 2.9B on Ollama. If the quality works for your use case, you just found a model that will never run out of VRAM no matter how long the conversation goes.

> Check if your GPU can handle it with the VRAM Calculator. For more on why transformers eat memory over time, read Memory Leak in Long Conversations. And for the broader view on post-transformer architectures, see Beyond Transformers: 5 Architectures.

Related guides

- Memory Leak in Long Conversations: Causes and Fixes
- Context Length Explained
- VRAM Requirements for Local LLMs
- Running AI Offline: Complete Guide
- What Can You Run on 8GB VRAM
- LiquidAI LFM2: The Non-Transformer Model Worth Running Locally
- Beyond Transformers: 5 Architectures for Your $50 Mini PC
