RWKV-7 Local Guide: Infinite Context, Zero KV Cache, Runs on Anything

February 23, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** RWKV-7 'Goose' is an RNN architecture that matches transformer quality at the 3B scale while using constant memory regardless of context length. No KV cache means VRAM doesn't grow as conversations get longer. The 2.9B model scores 71.5% on English benchmarks (matching Qwen2.5-3B's 71.4% with one-third the training data). Run it via Ollama: `ollama run mollysama/rwkv-7-g1:2.9b`. Available sizes: 0.1B through 13.3B. The G1 series adds chain-of-thought reasoning. Apache 2.0 license, Linux Foundation project.

📚 **More on this topic:** Beyond Transformers: 5 Architectures · Quantization Explained · CPU-Only LLMs · Planning Tool

We already benchmarked RWKV-7 against gemma3 on a $50 mini PC. The transformer crashed at turn 6 when its KV cache ate all 8GB of RAM. RWKV-7 held a flat 4.7 tok/s from turn 1 through turn 10 without flinching.

That wasn't a fluke. It's the architecture.

RWKV-7 "Goose" is an RNN that processes each token with constant memory, no matter how long the conversation gets. A transformer at 7B with 128K context might need 20+ GB. RWKV-7 at the same size stays at ~7GB whether you're at token 100 or token 100,000.

This guide covers what RWKV-7 is, how to run it locally, what sizes are available, and where it actually falls short.

What RWKV-7 Actually Is

RWKV (pronounced "RwaKuv") stands for Receptance-Weighted Key-Value. It's a pure RNN. No attention mechanism at all. But unlike traditional RNNs, it can be trained in parallel like a transformer, then runs inference sequentially like an RNN.

Each RWKV version gets an animal codename. v5 was "Eagle," v6 was "Finch," v7 is "Goose."

Property	Transformers	RWKV-7
Memory per token	O(n), grows with context	O(1), constant
KV cache	Yes, grows linearly	None
Time per sequence	O(n^2)	O(n)
Context limit	Fixed by VRAM budget	Theoretically unlimited
Training	Parallelizable	Also parallelizable

The v7 breakthrough is the “Generalized Delta Rule,” a new way of updating the hidden state. Instead of a single learning rate for state updates, RWKV-7 uses vector-valued gating that lets it selectively update different channels and independently decide what to forget versus what to write. It also dynamically adjusts update aggressiveness per token.

The authors describe this as “test-time training its state on the context via in-context gradient descent at every token.” In practice, it means the model gets better at using its limited state as it processes more of the conversation.

Available Models

Base Models (World v3 Dataset)

Model	Parameters	Training Data
RWKV7-World3-0.1B	0.1B	3.1T tokens
RWKV7-World3-0.4B	0.4B	3.1T tokens
RWKV7-World3-1.5B	1.52B	3.1T tokens
RWKV7-World3-2.9B	2.9B	3.1T tokens

G1 Reasoning Models (World v3.5 Dataset)

Model	Parameters	Training Data	Notes
RWKV7-G1 0.1B	0.1B	5.16T tokens	Thinking mode
RWKV7-G1 0.4B	0.4B	5.16T tokens	Thinking mode
RWKV7-G1 1.5B	1.5B	5.16T tokens	Thinking mode

Model	Parameters	Training Data	Notes
RWKV7-G1 2.9B	2.9B	5.16T tokens	Thinking mode
RWKV7-G1 7.2B	7.2B	5.16T tokens	Thinking mode
RWKV7-G1 13.3B	13.3B	5.16T tokens	Thinking mode

The G1 series adds chain-of-thought reasoning. You can toggle it with `/set think` and `/set nothink` in Ollama.

The “World” in the name means multilingual training: 80% English, 10% multilingual, 10% code. All models are Apache 2.0 licensed under the Linux Foundation.

VRAM Requirements

Here’s where RWKV gets interesting. These numbers stay constant regardless of context length.

Model	FP16	Q8_0	Q5_1	Q4_0
0.1B	~0.2 GB	~0.1 GB	~0.08 GB	~0.06 GB
0.4B	~0.8 GB	~0.4 GB	~0.3 GB	~0.2 GB
1.5B	~3.0 GB	~1.5 GB	~1.1 GB	~0.8 GB
2.9B	~5.8 GB	~2.9 GB	~2.1 GB	~1.5 GB
7.2B	~14.4 GB	~7.2 GB	~5.2 GB	~3.6 GB
13.3B	~26.6 GB	~13.3 GB	~9.6 GB	~6.7 GB

Compare this to a transformer 7B at 128K context, which can need 20+ GB total once the KV cache fills up. RWKV-7 7.2B at Q8_0 stays at ~7.2GB whether you’re processing 1K or 128K tokens.

The RWKV wiki recommends FP16 > Q8_0 > Q5_K_M > Q4_K_M. Lower quantizations (Q4_0, Q4_1) have been reported to cause quality issues with RWKV due to weight/activation outliers. Stick to Q5 or above if possible.

Benchmarks

RWKV-7 2.9B vs Competitors

RWKV-7 2.9B scores **71.5%** average across English downstream benchmarks (LAMBADA, HellaSwag, PIQA, ARC-E, ARC-C, GLUE, WinoGrande). For context:

Model	Avg English Score	Training Data
RWKV-7 2.9B	71.5%	5.6T tokens
Qwen2.5 3B	71.4%	18T tokens
Llama 3.2 3B	~71-73%	Significantly more

The efficiency story matters: RWKV-7 matches Qwen2.5-3B with roughly one-third the training data. It also claims a new 3B state-of-the-art on multilingual benchmarks.

MMLU specifically: RWKV-7 2.9B hits 54.56%. The jump from v6 to v7 is dramatic at smaller sizes, with the 1.5B going from 25.1% to 43.3% on MMLU.

Speed: The $50 PC Benchmark

We ran this on a Lenovo M710Q (i7-6700T, 8GB DDR4, no GPU):

Turn	RWKV-7 2.9B (Q8_0)	gemma3:4b
1	4.7 tok/s	5.7 tok/s
3	4.7 tok/s	5.3 tok/s
6	4.7 tok/s	CRASHED
10	4.7 tok/s	–

The transformer starts faster but dies when its KV cache fills memory. RWKV-7 holds a flat line because there's no cache to fill.

How to Run RWKV-7 Locally

Option 1: Ollama (Easiest)

```
# G1 reasoning model (recommended)
ollama run mollysama/rwkv-7-g1:2.9b

# Other sizes
ollama run mollysama/rwkv-7-g1:0.1b
ollama run mollysama/rwkv-7-g1:7.2b
ollama run mollysama/rwkv-7-g1:13.3b

# Toggle thinking mode
/set nothink    # disable chain-of-thought
/set think      # enable chain-of-thought
```

RWKV-7 support in Ollama comes from community contributor MollySophia. Models are hosted under the `mollysama` namespace.

Option 2: llama.cpp

RWKV-7 is supported in llama.cpp (merged via PR #12412). Pre-made GGUF files are available on HuggingFace.

```
# Download a GGUF model from HuggingFace
# Example: Mungert/rwkv7-2.9B-world-GGUF

# Run with llama.cpp
./llama-cli -m rwkv-7-world-2.9b-Q8_0.gguf -p "Your prompt here"
```

Option 3: rwkv.cpp (Native Runtime)

The dedicated RWKV inference engine supports v4 through v7.

```
git clone --recursive https://github.com/RWKV/rwkv.cpp.git

# Convert from PyTorch
python python/convert_pytorch_to_ggml.py input.pth output.bin FP16
```

```
# Quantize
python python/quantize.py model.bin model-Q5_1.bin Q5_1

# Run
python python/chat_with_bot.py model-Q5_1.bin
```

Option 4: Other Tools

- **Ai00 Server** – Int8 and NF4 quantization support
- **Text Generation WebUI** – has RWKV support built in
- **RWKV-Runner** – GUI launcher by josStorer

Where RWKV-7 Falls Short

The constant-memory trade-off is real. You get predictable VRAM, but you lose some things.

Needle-in-a-haystack retrieval is weaker. All information passes through a fixed-size hidden state (64x64 per head). Specific details can fade or blur over long contexts. Transformers with full attention can theoretically attend to any point in context with equal strength. RWKV-7 scores 72.93% on the single needle-in-a-haystack benchmark, decent but not transformer-level.

Prompt order matters more. Transformers treat input as a set where any token can attend to any other. RWKV processes strictly left-to-right. Reordering the same facts in your prompt can change output quality more than you'd expect.

The fixed-size state also means the model makes "tough decisions about what to keep and what goes" as context grows. Transformers pay O(n^2) to keep everything accessible. RWKV pays O(1) but compresses. That's the fundamental trade-off.

The ecosystem is smaller too. While Ollama and llama.cpp now support RWKV-7, most inference tools, fine-tuning frameworks, and RAG pipelines are optimized for transformers first. Expect occasional compatibility gaps. The biggest RWKV-7 is 13.3B (G1 series), so there's no 70B or 405B equivalent to compete with Llama 3.1 at the top end. And the RWKV-7 kernel is roughly 2x slower for 0.1B/0.4B models, only reaching competitive speed at 7B+.

Who Should Care

If you're running on 8GB RAM with no GPU, RWKV-7 won't crash mid-conversation like a transformer. The M710Q benchmark proved this.

If your use case involves processing long documents or extended conversations, constant VRAM matters. A transformer's memory footprint grows with every turn. RWKV's doesn't.

For edge deployment, RWKV-7 has been benchmarked at 16.39 tok/s on an ARM Cortex-A76 (via the RWKV-Lite compression framework). Transformers manage single-digit tok/s on the same hardware.

Even if none of those apply, try it. `ollama run mollysama/rwkv-7-g1:2.9b` takes two minutes. If it works for your use case, you just freed yourself from the KV cache tax.

Bottom Line

RWKV-7 matches transformer quality at the 3B scale with one-third the training data and uses constant memory regardless of context length. It runs on hardware that makes transformers crash. The trade-off is weaker needle-in-a-haystack retrieval and a smaller ecosystem.

For budget hardware, long conversations, and edge deployment, it's already the better choice. For everything else, it's worth testing alongside your current models. The architecture is sound, the models are Apache 2.0, and Ollama support means the setup is trivial.

The project is a Linux Foundation initiative led by Bo Peng (BlinkDL), with a growing community, active Discord, and models on HuggingFace.

📚 **Architecture:** Beyond Transformers: 5 Architectures · Context Length Explained

📚 **Setup:** Run Your First Local LLM · Ollama vs LM Studio · llama.cpp vs Ollama vs vLLM

📚 **Hardware:** CPU-Only LLMs · VRAM Requirements · Planning Tool

---

Source: https://insiderllm.com/guides/rwkv-7-local-guide/

Free guides for running AI locally