Beyond Transformers: 5 Architectures for Your $50 Mini PC

February 13, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** RWKV-7 2.9B held a flat 4.7 tok/s across 10 conversation turns on our $50 Lenovo M710Q (8GB RAM, no GPU). Gemma3 4B started faster at 5.7 tok/s but crashed at turn 6 when its KV cache ate all 8GB. If you're running AI on tight hardware, transformer alternatives like RWKV-7 already work today via Ollama. Mamba hybrids and looped architectures are coming fast.

📚 **More on this topic:** CPU-Only LLMs · Quantization Explained · Context Length Explained · Planning Tool

We ran two models on a $50 Lenovo M710Q. One crashed. The other didn't even flinch.

The machine: an i7-6700T, 8GB DDR4, no GPU. The kind of thing you find on eBay between listings for broken printers. We pulled gemma3:4b (a standard transformer) and RWKV-7 2.9B (an RNN that trains like a transformer) and ran a 10-turn conversation on each.

Here's what happened.

The $50 Benchmark That Changes Everything

Turn	RWKV-7 2.9B (Q8_0)	gemma3:4b (transformer)
1	4.7 tok/s	5.7 tok/s
2	4.7 tok/s	5.4 tok/s
3	4.7 tok/s	5.3 tok/s
5	4.7 tok/s	5.2 tok/s
6	4.7 tok/s	**CRASHED**
10	4.7 tok/s	—

The transformer was faster until it wasn't. The RNN never flinched.

Gemma3 started strong, but every turn added to its KV cache. By turn 5 it was swapping to disk. Turn 6: the kernel killed it. Meanwhile RWKV-7 held a dead-flat 4.7 tok/s from turn 1 through turn 10, because it doesn't have a KV cache. It processes each token with constant memory regardless of conversation length.

That flat line isn't a fluke. It's the architecture.

Why Transformers Struggle on Cheap Hardware

Transformers keep a running tab of every token they've seen. This is the KV cache, and it grows with every turn of conversation. A 4B transformer at 4K context can need 2-3GB of KV cache alone, on top of the model weights.

On a machine with 24GB of VRAM, you don't notice. On a machine with 8GB of total RAM, it's a death sentence.

Three problems compound:

KV cache growth. Every new token adds to the cache. In a multi-turn conversation, the cache grows with each exchange. There's no ceiling except your available memory.

Quadratic attention scaling. Standard self-attention scales with the square of the sequence length. Double your context, quadruple the compute. At 8K tokens that's tolerable. At 32K it's brutal on CPU.

Context rot. Even models advertising 128K or 1M context windows degrade well before those limits. Research consistently shows quality dropping around 32-100K tokens. You're paying the memory cost of long context without getting the quality you'd expect.

On budget hardware, these aren't theoretical concerns. They're the reason your model crashes mid-conversation.

---

5 Architectures Worth Watching

1. RWKV-7 "Goose" — The One You Can Run Today

RWKV-7 is an RNN that trains like a transformer. During inference it maintains a fixed-size hidden state instead of a growing KV cache. Constant memory, constant speed, regardless of conversation length. Our benchmark proves the claim.

The project is an incubation-stage project under the Linux Foundation AI & Data Foundation, Apache 2.0 licensed, and 100% attention-free. Sizes range from 0.1B to 7.2B, with larger models in training. The 2.9B Q8_0 fits comfortably in 8GB RAM.

GGUF files are available from Mungert's repos on HuggingFace, and llama.cpp merged RWKV-7 support in PR #12412. You can run it in Ollama today:

```
# Option A: community model (may be temporarily unavailable)
ollama run mollysama/rwkv-7-world:2.9b

# Option B: custom GGUF (any size/quant you want)
# 1. Download a GGUF from Mungert's repos
# 2. Create a Modelfile:
#    FROM ./rwkv7-2.9B-world-Q8_0.gguf
# 3. ollama create rwkv7:2.9b -f Modelfile
# 4. ollama run rwkv7:2.9b
```

The tradeoff: Initial tokens-per-second is slower than a similarly sized transformer. Gemma3 beat RWKV-7 on turn 1 by a full token per second. But RWKV-7 doesn't slow down and doesn't crash. For multi-turn chat on tight hardware, that consistency wins.

G1 reasoning models (RWKV-7 with chain-of-thought training) are also available at 0.1B through 2.9B, with larger G1 models in training.

2. Ouro / LoopLM — Depth Without Parameters

ByteDance's Ouro takes a different approach to efficiency. Instead of adding more layers, it loops the same 24-layer block 4 times, giving the model an effective depth of 96 layers with only 1.4B parameters. Think of it as a model that reads its own work and revises it three times before answering.

Ouro-1.4B matches 4B transformers on reasoning benchmarks. Ouro-2.6B-Thinking matches or exceeds 8B models on math and science tasks (GSM8K, MATH500, BBH). The paper is co-authored by Yoshua Bengio and describes loop depth as a third scaling axis alongside model size and data.

The trick: a learned exit gate lets the model bail out after fewer loops on easy questions and think harder on difficult ones. Simple fact retrieval might use 2 loops. A math proof gets all 4.

The catch: No GGUF. No Ollama. No llama.cpp support. Today you need HuggingFace Transformers or vLLM to run it, and vLLM doesn't support the adaptive exit feature yet. But the

architecture is Apache 2.0 and the weights are on HuggingFace. When someone writes the GGUF converter, this becomes a serious option for budget hardware.

3. Mamba / Mamba-2 — State Space Models

Mamba replaces attention with selective state spaces. Where a transformer remembers everything (KV cache), Mamba maintains a compressed hidden state and decides what to keep or forget per token. Linear time complexity instead of quadratic. Constant memory per step.

Mamba-2 formalized the connection between SSMs and a restricted form of attention, which made GPU implementations more efficient. Mamba-3 (2026) added second-order discretization and complex-valued states for better state tracking.

You can try small Mamba models on Ollama today (`Hudson/mamba-chat`), though the pure Mamba options are limited in size. The more practical path is hybrid models that combine Mamba with attention layers (see below).

The limitation: Pure Mamba models struggle with retrieval and in-context learning. A 2025 ablation study found that removing attention layers drops retrieval accuracy to zero. Mamba handles bulk sequence processing well, but it can't do the precise lookback that attention excels at. That's why the field is moving toward hybrids.

4. Hybrid Attention + SSM — Use Both, Waste Neither

This is where the serious investment is happening. Instead of choosing between attention (good at retrieval, expensive) and SSMs (efficient, weak at recall), hybrid models use both: a few attention layers for precision tasks and many SSM layers for the bulk of processing.

Already available:

- Nemotron-3 Nano (NVIDIA) — 23 Mamba-2 layers + 6 attention layers + MoE. 31.6B total params, 3.2B active. **On Ollama right now.**
- IBM Granite 4.0 — 9:1 ratio of Mamba-2 to attention, claims 70% less RAM for long inputs.

Coming soon:

- Qwen3-Next (Alibaba) — 80B total, 3B active, 75% linear attention layers, matches their flagship Qwen3-235B on benchmarks while running 10x faster.

The ratio keeps shifting. Jamba (AI21) used 1:7 attention-to-Mamba. Granite uses 1:9. The trend is clear: you need some attention, but far less than transformers currently use.

5. Diffusion LLMs — The Horizon

Standard LLMs generate one token at a time, left to right. Diffusion LLMs start with noise and iteratively refine an entire sequence in parallel, similar to how Stable Diffusion generates images.

Google's Gemini Diffusion reportedly hits 1,000-2,000 tok/s internally (5x their fastest autoregressive model). Mercury from Inception Labs demonstrated 1,100 tok/s for code generation on H100. LLaDA, an open-source 8B diffusion LLM, showed results competitive with Llama 3 8B.

This is not something you can run today. Ollama doesn't support diffusion decoding (open issue since March 2025). LLaDA requires custom PyTorch code and a GPU. Mercury is API-only. Gemini Diffusion isn't public.

But watch this space. If diffusion decoding matures, the entire bottleneck of autoregressive generation disappears. We'd be talking about a fundamentally different kind of machine.

What This Means for Your Homelab

The practical advice, right now:

If you have 8GB RAM: RWKV-7 2.9B Q8_0 is your best bet for multi-turn conversations. Transformers will beat it on single-shot queries, but any conversation longer than 5-6 turns on tight RAM will favor the RNN. Also try Nemotron-3 Nano if you can squeeze it in.

If you can upgrade to 16GB ($15 for a DDR4 stick): RWKV-7 7.2B Q4 becomes viable, and you get a meaningful quality jump. Hybrid models like Nemotron-3 Nano run comfortably.

Watch for Ouro GGUF conversions. Once someone adds looped-layer support to llama.cpp, a 1.4B model that reasons like a 4B model is a massive win for cheap hardware. The weights are open, the license is permissive. It's a matter of when, not if.

The next breakthrough in local AI won't come from a bigger model. It'll come from a smarter architecture running on hardware you already own.

How We Tested

- **Hardware:** Lenovo M710Q, Intel i7-6700T, 8GB DDR4, CPU-only, no GPU
- **OS:** Ubuntu 24

- **Runtime:** Ollama 0.16.1
- **RWKV-7:** Mungert/rwkv7-2.9B-world-GGUF Q8_0, custom Modelfile
- **gemma3:** `ollama pull gemma3:4b` (default quantization)
- **Test:** 10-turn conversation, tok/s measured per turn via Ollama API

Here's the script we used. Stdlib Python, zero dependencies. Point it at any model and see for yourself:

```
import urllib.request, time, json, sys

model = sys.argv[1] if len(sys.argv) > 1 else "rwkv7:2.9b"
turns = int(sys.argv[2]) if len(sys.argv) > 2 else 10

questions = [
    "What is beekeeping?",
    "What are Layens hives?",
    "How do you harvest honey?",
    "What is the newspaper method for combining colonies?",
    "How do you treat varroa mites naturally?",
    "What flowers are best for bees in California?",
    "How do you split a colony in spring?",
    "What is a swarm trap and how do you use one?",
    "How do you overwinter a hive in mild climates?",
    "Summarize everything we discussed about beekeeping.",
]

messages = []
print(f"\nModel: {model}")
print(f"{'Turn':<6} {'Tokens':<8} {'Time':>7} {'tok/s':>8}")
print("-" * 35)

for i in range(min(turns, len(questions))):
    messages.append({"role": "user", "content": questions[i]})
    data = json.dumps({"model": model, "messages": messages, "stream": False}).encode()
    req = urllib.request.Request("http://localhost:11434/api/chat",
        data=data, headers={"Content-Type": "application/json"})
    with urllib.request.urlopen(req, timeout=300) as resp:
        d = json.loads(resp.read())
    msg = d.get("message", {}).get("content", "")
    messages.append({"role": "assistant", "content": msg})

    ev = d.get("eval_count", 0)
    dur = d.get("eval_duration", 1) / 1e9
```

```
    tps = ev / dur if dur > 0 else 0
    print(f"{i+1:<6} {ev:<8} {dur:>6.1f}s {tps:>7.1f}")
```

Usage:

```
python3 bench_turns.py rwkv7:2.9b 10
python3 bench_turns.py gemma3:4b 10
```

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/beyond-transformers-5-architectures/