CPU-Only LLMs: What Actually Works

January 29, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** CPU-only inference is slower than GPU but genuinely usable. On a modern laptop or desktop (i7/Ryzen 7, DDR5), expect 10-15 tok/s on 7B models at Q4 — enough for interactive chat. For bigger models, a dual Xeon E5-2699 v4 server build ($1,000-1,200 used parts) gives you 128GB RAM and 8-channel memory bandwidth, enough to run 70B quantized at 3-5 tok/s. The key factors are RAM speed, channel count, and sticking to Q4 quantization. Start with Llama 3.2 3B or Qwen 2.5 7B at Q4_K_M using llama.cpp or Ollama.

📚 **More on this topic:** VRAM Requirements · GPU Buying Guide · Quantization Explained

No GPU? No problem — mostly.

CPU-only inference gets dismissed as unusable, and that's wrong. It's slower, yes. A 7B model that runs at 40+ tok/s on an RTX 4060 runs at 10-15 tok/s on a decent CPU. But 10-15 tok/s is still faster than you can read. For chat, quick coding questions, and summarization, that's enough.

What nobody tells you is that CPU inference has real advantages: it's stable (no CUDA driver nightmares), cheap (RAM costs a fraction of VRAM per gigabyte), and scales to enormous models. A ~$1,100 dual Xeon server with 128GB of RAM can run a 70B model that would need a $1,600 RTX 4090 — or two GPUs — to fit in VRAM.

This guide covers two tiers: what works on your existing laptop or desktop, and a researched budget server build for people who want to go bigger.

Who This Is For

- **Laptop users** with no discrete GPU (or a weak one) who want to try local AI
- **Desktop users** without a GPU, or with an older card that's not worth using
- **Budget builders** who'd rather spend ~$1,100 on a full server build than on a single high-end GPU
- **Quiet/stable operation** — no GPU fan noise, no driver updates breaking things
- **Privacy-first users** who want local inference on whatever hardware they have

If you own a GPU with 8GB+ VRAM, you should use it. GPU inference is 3-6x faster for the same model. This guide is for people who can't or don't want to go that route.

The Honest Truth About CPU Inference

CPU inference is memory-bandwidth limited. When generating text, the CPU reads the entire model's weights from RAM for every single token. The speed of that read — your memory bandwidth — determines your tokens per second, not your core count or clock speed.

This is why a 16-core CPU isn't twice as fast as an 8-core for inference. Both are waiting on the same memory bus. It's also why RAM speed and channel count matter far more than the CPU itself.

Here's the baseline you can expect:

Hardware	3B Model (Q4)	7B Model (Q4)	13B Model (Q4)	70B Model (Q4)
Laptop (i7, DDR4 dual-ch)	~20 tok/s	~8 tok/s	~4 tok/s	Won't fit
Desktop (Ryzen 7, DDR5 dual-ch)	~35 tok/s	~14 tok/s	~7 tok/s	~2 tok/s*
Dual Xeon (DDR4 8-ch, 128GB)	~50+ tok/s	~15-20 tok/s	~10-12 tok/s	~3-5 tok/s

*Requires 48GB+ RAM. Slow but functional.

At 10+ tok/s, responses feel like a fast typist. At 3-5 tok/s, you're watching words appear one by one — usable for batch work, not great for interactive chat. Below 2 tok/s, it's a screensaver.

The Advantages Nobody Mentions

GPU inference gets all the attention, but CPU-only has underrated strengths:

- **No driver hell.** No CUDA version mismatches, no ROCm compatibility matrix, no "works on 535.xx but crashes on 545.xx." llama.cpp on CPU just compiles and runs.
- **RAM is cheaper per gigabyte.** 128GB of DDR4 ECC RDIMM costs ~$800-1,000 used. 24GB of GDDR6X (an RTX 4090) costs $1,600+. You get 5x the memory capacity for half the price.
- **Stability.** CPU inference doesn't crash when VRAM fills up. Models memory-map from disk, context windows scale with available RAM, and the OS manages memory gracefully.
- **Silent operation.** No GPU fan spinning up to 80%. CPU coolers under inference load are barely audible.

- **Runs models GPUs can't.** A 70B model at Q4 needs ~40GB. That doesn't fit on any consumer GPU. On a server with 128GB RAM, it runs without tricks.

What Affects CPU Inference Speed

1. Memory Bandwidth (The Big One)

Token generation speed scales almost linearly with memory bandwidth. This is the single most important factor.

Configuration	Theoretical Bandwidth	~7B Q4 Speed
DDR4-2400, 2 channels (laptop)	~38 GB/s	~5-8 tok/s
DDR4-3200, 2 channels (desktop)	~51 GB/s	~8-10 tok/s
DDR5-4800, 2 channels (modern desktop)	~77 GB/s	~11-13 tok/s
DDR5-6000, 2 channels (fast desktop)	~96 GB/s	~14-16 tok/s
DDR4-2400, 8 channels (dual Xeon)	~154 GB/s	~15-22 tok/s

The takeaway: a dual Xeon with 8 channels of slow DDR4 beats a modern desktop with 2 channels of fast DDR5 — because total bandwidth wins over per-channel speed.

2. Channel Count

More RAM channels = more bandwidth = more tokens per second. This is why server hardware matters.

Platform	Channels	Why It Matters
Consumer desktop/laptop	2	Baseline. Adequate for 3B-7B
HEDT (Threadripper)	4	~2x bandwidth over consumer
Server (Xeon, EPYC)	8	~4x bandwidth over consumer
High-end server (EPYC Genoa)	12	~6x bandwidth over consumer

Populate all channels. One DIMM per channel beats two DIMMs on fewer channels. On a dual Xeon board with 8 DIMM slots, use 8 sticks — even if the total capacity is the same.

3. Core Count (Less Than You Think)

More cores help with prompt processing (the initial "thinking" phase) but barely affect token generation. In benchmarks, switching from 6 threads to 12 threads on the same CPU gave almost no improvement in generation speed (5.37 vs 5.33 tok/s).

The rule: Set thread count to your physical core count (not logical/hyperthreaded). On an i7-13700, that means 8 P-cores — ignore the E-cores and hyperthreading.

4. AVX-512 Support

AVX-512 dramatically accelerates prompt processing (2.8-10x faster) but doesn't help token generation because generation is memory-bound, not compute-bound.

CPU	AVX-512?	Notes
Intel 12th-14th Gen (consumer)	No*	Some have it disabled in BIOS
AMD Ryzen 7000/9000 (Zen 4/5)	Yes	Full AVX-512, big prompt speedup
Intel Xeon Scalable (Skylake+)	Yes	Also AMX on 4th Gen+
Xeon E5 v4 (Broadwell)	No	AVX2 only

*Some i9-12900K chips have AVX-512 accessible via BIOS, but Intel disabled it on later steppings.

If you're buying a CPU for inference, Zen 4+ with AVX-512 is the best consumer option. But remember: it mostly speeds up the prompt phase, not generation.

Tier 1: Laptop & Desktop (Consumer CPUs)

This is the starting point. You already have the hardware — the question is what to run on it.

What Fits

Model	RAM (Q4_K_M)	Speed (i7/ DDR5)	Speed (Ryzen 7/ DDR5)	Best For
Llama 3.2 3B	~2.5 GB	~30 tok/s	~35 tok/s	Quick Q&A, simple tasks
Phi-3 Mini (3.8B)	~2.8 GB	~28 tok/s	~32 tok/s	Reasoning, coding (compact)

Model	RAM (Q4_K_M)	Speed (i7/ DDR5)	Speed (Ryzen 7/ DDR5)	Best For
Qwen 2.5 7B	~5 GB	~11 tok/s	~14 tok/s	Multilingual, coding
Llama 3.1 8B	~5.5 GB	~10 tok/s	~13 tok/s	General assistant
Mistral 7B	~4.8 GB	~12 tok/s	~14 tok/s	Fast chat, summarization
DeepSeek R1 8B	~5.5 GB	~9 tok/s	~12 tok/s	Reasoning, math

The sweet spot: Llama 3.2 3B for speed, Qwen 2.5 7B for quality. If you have DDR5 and 16GB+ RAM, 7B models are comfortable. On DDR4 laptops with 8GB RAM, stick to 3B models.

What Won't Work on Consumer Hardware

- **13B models**: Technically runnable (~7GB RAM at Q4) but painfully slow at 4-7 tok/s on dual-channel DDR. Not a good experience.
- **30B+ models**: Need 20GB+ RAM at Q4. Even if your laptop has 32GB, the ~50 GB/s bandwidth means ~3 tok/s. That's a patience test.
- **Anything at FP16**: A 7B model at FP16 needs 14GB RAM and runs at half the speed of Q4. No reason to use FP16 on CPU.

Getting Started

```
# Install Ollama (works on Mac, Linux, Windows)
curl -fsSL https://ollama.com/install.sh | sh

# Run a 3B model (fast, low RAM)
ollama run llama3.2:3b

# Run a 7B model (better quality, needs 16GB+ RAM)
ollama run qwen2.5:7b
```

Ollama auto-detects CPU-only systems and runs inference on CPU. No configuration needed.

Tier 2: Budget Server Build (Dual Xeon)

This is where CPU-only gets interesting. Used server hardware is absurdly cheap, and a dual-socket Xeon build gives you 8-channel memory bandwidth and enough RAM to run models that don't fit on any consumer GPU.

Why Server Hardware

The Xeon E5-2699 v4 launched at $4,115 per chip in 2016. Today, a pair costs $180-300 on eBay. The same collapse happened to motherboards and coolers. You can build a system with 44 cores, 128GB RAM, and 154 GB/s memory bandwidth for roughly the cost of a used RTX 3090.

The secret: **memory bandwidth scales with channels, not speed.** Eight channels of DDR4-2400 (~154 GB/s) beats two channels of DDR5-6000 (~96 GB/s). For LLM inference, this is the only number that matters.

The Build: Dual Xeon E5-2699 v4

Component	Spec	Price (Jan 2026)	Source
2x Xeon E5-2699 v4	22C/44T each (44C/88T total), 2.2-3.6 GHz, AVX2	$180-300 (matched pair)	eBay
Motherboard	Supermicro X10DAi (16 DIMM slots, C612)	$110-250	eBay
RAM	8x 16GB DDR4-2400 ECC RDIMM (128GB total)	$800-1,000	eBay
CPU Coolers (x2)	Noctua NH-U12DX i4 (LGA 2011-3, both ILM types)	$130-150 (pair)	Amazon
Case	Phanteks Enthoo Pro 2 Server Edition (SSI-EEB)	$160-190	Amazon/ Newegg
PSU	850W 80+ Gold (needs 2x EPS 8-pin)	$100-140	Amazon
Storage	500GB SATA SSD (models load from disk)	$30-50	Amazon

Total: $1,510-2,080 (budget to mid-range)

RAM is the main cost bottleneck. Prices fluctuate — check current eBay listings for DDR4 ECC RDIMM before budgeting. Note: Xeon E5 v4 requires ECC RDIMM (Registered) memory — regular desktop DDR4 won't work.

Budget-optimized version (Chinese X99 dual board, generic coolers, used PSU):

Component	Budget Price
2x E5-2696 v4 (same chip as 2699)	$150-200
Chinese X99 dual-socket board	$80-120
8x 16GB DDR4-2133 ECC RDIMM	$800-1,000
2x Generic LGA 2011-3 coolers	$50-80
Mid-tower E-ATX case	$80-120
750W 80+ Bronze PSU	$60-80
500GB SATA SSD	$30-50

Budget total: $1,250-1,650

Expected Performance

Model	Quantization	RAM Used	Speed (est.)	Usability
Llama 3.2 3B	Q4_K_M	~2.5 GB	~50+ tok/s	Excellent — instant responses
Qwen 2.5 7B	Q4_K_M	~5 GB	~15-20 tok/s	Great — faster than reading
Qwen 2.5 14B	Q4_K_M	~9 GB	~10-15 tok/s	Good — interactive chat
Llama 2 30B	Q4_K_M	~20 GB	~8-12 tok/s	Usable — noticeable pauses
Llama 3.1 70B	Q4_K_M	~42 GB	~3-5 tok/s	Slow — batch work, not chat
DeepSeek R1 671B (MoE)	Q4	~100+ GB	~1-2 tok/s	Proof of concept only

The 7B-14B range is the sweet spot on this build. Fast enough for interactive use, with tons of RAM headroom for long context windows. The 70B tier works but at “patience required” speeds — useful for generating long documents, not for back-and-forth conversation.

Build Tips

- **Populate all 8 DIMM slots.** One stick per channel (4 per CPU) maximizes bandwidth. 8x16GB beats 4x32GB even though both are 128GB.
- **Use DDR4-2400 if you can.** DDR4-2133 works but leaves ~12% bandwidth on the table. The price difference is minimal.
- **ECC RDIMM is mandatory.** Xeon E5 v4 won’t boot with unbuffered or non-ECC RAM.

- **Check ILM type on your motherboard.** LGA 2011-3 has Square ILM (80x80mm) and Narrow ILM (56x94mm). The Noctua NH-U12DX i4 supports both. Most consumer tower coolers only support Square.
- **Power draw is real.** Two E5-2699 v4 at load pull 300-400W for the CPUs alone. Budget 500-600W total system draw. Your electricity bill will notice.
- **Disable Sub-NUMA Clustering (SNC)** in BIOS. Set 1 NUMA node per socket. Run inference with `numactl --interleave=all` for best performance.
- **E5-2696 v4 is the same silicon** as the E5-2699 v4 (same 22C/44T, same cache) but sold as an OEM part. Often $30-50 cheaper per chip. Most boards run it fine, but check BIOS compatibility.

Software: llama.cpp Is the Answer

For CPU-only inference, llama.cpp is the gold standard. It's written in C/C++ with hand-optimized CPU kernels, supports every quantization format, and runs on everything from Raspberry Pis to dual-socket servers.

Why llama.cpp

- **Native CPU optimization**: AVX2, AVX-512, ARM NEON — it uses whatever your CPU supports
- **Memory-mapped model loading**: Models load from disk on demand, no huge RAM spike at startup
- **Every quantization format**: Q2 through Q8, K-quants, IQ-quants — all supported
- **No Python dependency hell**: Compiles from source in minutes, no conda environments

CPU-Optimized Setup

```
# Clone and build with CPU optimizations
git clone https://github.com/ggml-org/llama.cpp
cd llama.cpp
cmake -B build -DGGML_NATIVE=ON
cmake --build build --config Release -j$(nproc)

# Run a model (set threads to physical cores)
./build/bin/llama-cli \
  -m models/qwen2.5-7b-q4_k_m.gguf \
  -t 8 \
  -c 4096 \
```

```
  --mlock \
  -ngl 0 \
  -p "Explain memory bandwidth in one paragraph."
```

Key flags:

Flag	What It Does	Recommended Value
`-t`	Thread count	Physical cores (not logical)
`-ngl 0`	Force CPU-only (no GPU layers)	0 for CPU-only
`--mlock`	Lock model in RAM, prevent swapping	Always use if RAM allows
`-c`	Context window size	2048-4096 for most use
`-b`	Batch size for prompt processing	512 (default is fine)

Or Just Use Ollama

If you don't want to compile anything, Ollama wraps llama.cpp with automatic model downloading and a clean interface. It detects CPU-only systems automatically.

ik_llama.cpp: The Faster Fork

The ik_llama.cpp fork has optimized CPU kernels that nearly doubled inference speed on Broadwell Xeon hardware in benchmarks (2.70 to 5.05 tok/s on 70B models). Worth trying on older Xeon builds specifically.

Quantization Matters Even More on CPU

On GPU, the difference between Q4 and Q8 is modest — maybe 20-30% speed difference. On CPU, it's dramatic. Every extra bit per parameter means more data read from RAM per token, and RAM bandwidth is your bottleneck.

Quantization	7B Model Size	~Speed (DDR5, 2-ch)	Quality vs FP16
Q8_0	~7.5 GB	~7 tok/s	~99%
Q5_K_M	~5.5 GB	~10 tok/s	~95%
Q4_K_M	~4.8 GB	~13 tok/s	~92%

Quantization	7B Model Size	~Speed (DDR5, 2-ch)	Quality vs FP16
Q4_K_S	~4.5 GB	~14 tok/s	~91%
Q3_K_M	~3.8 GB	~14 tok/s	~88%
Q2_K	~3.0 GB	~13 tok/s	~80%

Notice that Q3 and Q2 aren't faster than Q4 despite being smaller. The dequantization compute overhead eats the bandwidth savings. **Q4_K_M is the sweet spot on CPU** — below that, you lose quality without gaining speed.

The rule: Always use Q4_K_M or Q4_K_S for CPU inference. Only go to Q5 if you have bandwidth to spare (server builds). Never go below Q3 — it's slower and worse.

RAM Requirements

The model must fit entirely in RAM. Unlike GPU inference where partial offloading is an option, CPU inference loads everything into system memory. If the model is larger than your available RAM, the OS will swap to disk, and inference drops to near-zero speed.

Model Size	Q4_K_M Weights	+ 4K Context	Total RAM Needed
3B	~1.5 GB	+0.2 GB	~4 GB (with OS)
7B	~4.8 GB	+0.5 GB	~8 GB
13B	~6.5 GB	+0.8 GB	~12 GB
30B	~15 GB	+1.5 GB	~20 GB
70B	~35 GB	+3 GB	~48 GB

Leave at least 4GB free for the OS and other processes. On a 16GB laptop, your practical limit is a 7B model at Q4. On a 32GB desktop, you can stretch to 13B. For 30B+, you need 64GB minimum.

→ Check what fits your hardware with our Planning Tool.

CPU-Only vs. Saving for a GPU

Here's the honest comparison:

Factor	CPU-Only	GPU (RTX 3060 12GB)
7B Q4 speed	~10-15 tok/s	~45 tok/s
13B Q4 speed	~5-8 tok/s	~20 tok/s
70B Q4	Works (3-5 tok/s, 128GB RAM)	Doesn't fit (12GB VRAM)
Cost to start	$0 (existing hardware)	~$200 (used 3060 12GB)
Max model RAM	64-128GB (cheap)	12-24GB (expensive)
Stability	Rock solid	Driver-dependent
Prompt processing	Slow (5-15 tok/s)	Fast (100-500 tok/s)
Power draw (inference)	65-150W (CPU only)	170W (GPU) + CPU

Choose CPU-Only If:

- You already have the hardware and want to try local AI today — $0 investment
- You need to run 30B-70B models and can't afford a $700+ GPU with 24GB VRAM
- You're building a quiet, stable home server for background inference
- You value simplicity — no GPU drivers, no CUDA, no VRAM management
- You're considering a dual Xeon server build as a fun project

Save for a GPU If:

- Interactive speed matters — you want 30+ tok/s for real-time chat
- You mainly need 7B-14B models (fit comfortably in 8-12GB VRAM)
- You also want image generation (Stable Diffusion needs a GPU)
- Budget allows a used RTX 3060 12GB (~$200) or RTX 3090 (~$700-850)

The $200 used RTX 3060 12GB is the inflection point. If you can afford it, GPU inference on 7B-13B models is 3-6x faster than any CPU. But if you need 70B models or have $0 to spend, CPU-only is a legitimate path.

The Bottom Line

CPU inference is slower than GPU. That's the tradeoff. What you get in return is simplicity, stability, cheaper memory, and the ability to run models that don't fit on any consumer GPU.

For laptop/desktop users:

1. Install Ollama and run `llama3.2:3b`. You'll be chatting in two minutes.
2. If you have 16GB+ RAM and DDR5, step up to `qwen2.5:7b` at Q4.
3. Stick to Q4_K_M quantization — it's the fastest format that doesn't sacrifice quality.

For server builders:

1. Source a dual Xeon E5-2699 v4 (or 2696 v4) and a Supermicro X10DAi from eBay.
2. Fill all 8 DIMM slots with DDR4-2400 ECC RDIMMs — bandwidth is everything.
3. Run llama.cpp with `numactl --interleave=all` and threads set to physical core count.
4. Start with Qwen 2.5 14B at Q4 for the best speed/quality balance. Scale to 70B when you need it.

Your CPU is more capable than the benchmarks suggest. Use it.

Related Guides

- How Much VRAM Do You Need for Local LLMs?
- What Can You Run on 8GB VRAM?
- GPU Buying Guide for Local AI
- Build a Local AI PC for Under $500

Sources: OpenBenchmarking llama.cpp, DEV Community DDR5 LLM Benchmark, Hardware Corner Memory Bandwidth Guide, justine.lol CPU Matrix Multiplication, Dual Xeon Gold 5317 LLM Benchmarks, ik_llama.cpp Broadwell Benchmarks, XDA Used Xeon Home Lab, KernelCrash Old Xeon LLMs

---