ROGER'S
RABBIT HOLE



A ROGER'S RABBIT HOLE WHITE PAPER

LOCAL AI MODEL RACKS

Why measuring and routing right-sized models beats one model doing everything



BY ROGGIE D

NOVEMBER 2026

rogersrabbithole.com

LOCAL AI MODEL RACKS

Why measuring and routing right-sized models beats one model doing everything

By Roggie D | November 2026

WHO THIS IS FOR

Local AI hobbyists, homelab builders, IT people and small shops who already have a model or two running on their own hardware and keep wondering whether they picked the right one for each job.

HOW I LABEL THE NUMBERS

CyberTron numbers in this paper are historical results from my own project records, mostly from Oct. 4 and Oct. 16, 2026. I didn't run new benchmarks for this paper. Numbers from research papers are those authors' results on their benchmarks, not mine. When something is a reasonable guess rather than a measurement, I say so.

Related reading on Roger's Rabbit Hole: Bigger Isn't Always Smarter, Building AI on Aging Hardware and GPU VRAM: The Real Currency of Local AI, all at rogersrabbithole.com.

THE SHORT VERSION

My first three white papers made three points. A bigger model isn't automatically a smarter one. A $150 legacy PC can run real local AI if you respect its limits. And on a consumer graphics card, VRAM is the budget everything else spends. This paper puts those pieces together into the architecture I think most local AI setups should use: a model rack.

A model rack isn't a folder full of downloaded models. It's four things working together: slots that each hold a model in a specific mode, a test kit built from your real work, a written rule that decides which slot gets each job, and controls for what stays loaded in memory. Take away the testing and the rule, and you just have a pile of models.

The research on LLM routing backs the idea. Stanford's FrugalGPT, Berkeley's RouteLLM and Microsoft's Hybrid LLM work all found that sending easy requests to cheaper models and saving the expensive one for hard requests can hold quality while cutting cost. On

my own box, the same 9B model with its thinking mode switched off scored the same 7 out of 7 as thinking on and answered about 57 times faster. My router's first full report sent five of six job types to that one model.

A 35B mixture-of-experts model, by contrast, ran about 12 times slower than that 9B in a side-by-side of my recorded numbers, though the two measurements weren't taken under the same conditions. That doesn't make the big model useless. It means it has to earn its latency on the jobs that need it. And it did earn a slot: since Oct. 21, a 35B mixture-of-experts build has handled my everyday requests, while the 9B stays installed as a smaller, faster option. My approach has shifted from picking one favorite model to matching models to workloads. The rack isn't a bet on small models or big ones. It's the habit of measuring, routing and measuring again when the models change.

> **A rack of right-sized models, tested on your real work and picked by a written rule, beats one model doing everything, even when the rack's answer turns out to be mostly one model, and even when that answer changes.**

1. THE QUESTION ONE MODEL CAN'T ANSWER

1.1 WHERE THE FIRST THREE PAPERS LEFT OFF

In Bigger Isn't Always Smarter, I argued that parameter count is not knowledge, that a big model still hallucinates and still can't know your network unless you feed it the facts, and that specialized roles are normal engineering. That paper sketched the roles a rack might hold and introduced a simple yardstick: cost per accepted result, not tokens per second [18].

Building AI on Aging Hardware covered the box itself: an AMD FX-8300, 32 GB of DDR3, an RTX 3060 12 GB on a PCIe 2.0 board [19]. GPU VRAM: The Real Currency of Local AI showed why that 12 GB is the real budget, and why a mixture-of-experts model with only 3 billion active parameters still needs room for all of its experts [6].

What none of them did was lay out how a rack actually works, or show what happens when you run one for a while. That's this paper.

1.2 WHAT A MODEL RACK IS

Here's my working definition:

> **A model rack is a small set of model slots, a test kit drawn from your real work, a written decision rule that sends each job to the cheapest slot that passes, and residency controls that decide what stays loaded.**

Every word in that definition is doing work. "Small" because a 12 GB card can't hold many models at once. "Real work" because a public leaderboard doesn't know what you ask all day. "Written" because a rule in your head drifts. "Cheapest slot that passes" because the goal is the least time and memory for an acceptable answer, not the most impressive model. "Residency" because a model on disk isn't a model in memory.

If you've spent time in IT, the shape will look familiar. Nobody runs DNS, the database and the file server on one process because it's simpler. You split the jobs, you monitor each one, and you swap a part when a better one comes along. A model rack applies the same discipline to language models.

2. ANATOMY OF A RACK

2.1 SLOTS: A MODEL PLUS A MODE

A slot is not just a model name. It's a model, a mode and the settings that matter for the job. That distinction showed up hard on my box, as Exhibit A shows below.

Many current models ship with a "thinking" mode that reasons out loud before answering. The Qwen3 technical report describes building thinking and non-thinking modes into one model, plus a "thinking budget" that lets users trade latency for performance by task [3]. Ollama exposes thinking as a control you can turn on or off per request for models that support it [9]. So one downloaded model can fill two slots: a fast one for everyday work and a slow, careful one for hard problems. That's the cheapest rack you'll ever build, because it costs no extra VRAM.

2.2 THE TEST KIT

The test kit is a set of questions from your own work with answers you already know, grouped by job type. It's the part most people skip, and it's the part that makes everything else honest. Without it, routing is just opinion with extra steps.

My router's October test kit grouped work into six job types: general questions, reasoning, coding, structured output, avoiding made-up answers and vision. It measured

accuracy, total time to answer, time to the first visible word, tokens generated and peak VRAM for each model and mode.

2.3 THE DECISION RULE

The rule turns test results into a choice. Mine is short. Reject any option that can't do the job at all. Prefer higher accuracy. If accuracy ties, prefer lower total time to answer and call it a tie when two options land within 5% of each other. Then prefer lower VRAM, then fewer generated tokens. Quality first, then speed, then memory.

A written rule matters for two reasons. It keeps you from picking the model you're excited about, and it makes the choice repeatable when you add a new model next month.

2.4 RESIDENCY: WHAT STAYS LOADED

On one 12 GB card, the rack is mostly on disk. Residency controls decide what's loaded, for how long and what gets evicted to make room.

The common tools all have these controls. Ollama keeps a model in memory for 5 minutes by default, lets you change that per request or server-wide with keep_alive, and caps concurrent models with OLLAMA_MAX_LOADED_MODELS. Its FAQ is blunt about GPUs: a new model has to fit completely in VRAM to load alongside another one; otherwise requests queue while idle models are unloaded [8]. On the llama.cpp side, llama-swap hot-swaps models on demand and can unload them after a timeout [11], and llama.cpp's own server has a router mode that loads and unloads models by name, with a cap on how many stay loaded and an optional memory margin [10].

2.5 FALLBACK AND ESCALATION

A rack needs a plan for when the chosen slot fails: a timeout, a malformed answer, a "this is beyond me." The simplest version is a cascade. Try the cheap slot, check the result and escalate to the bigger or slower slot only when the check fails. That's the core of FrugalGPT [1]. Tools like LiteLLM's router build fallbacks, retries, timeouts and cooldowns into the plumbing, and it can mix local and paid models [12].

2.6 THE LOG

The last layer is the boring one: write down which slot got each job, how long it took and whether the answer was accepted. The log is how you notice that the router keeps escalating one kind of question, or that a new model quietly got slower. It's also how you rebuild the test kit, because your real questions are sitting right there.

LAYER	ITS JOB	WHAT HAPPENS IF YOU SKIP IT
Slots	Hold a model in a specific mode with job-relevant settings	Every job pays for the same mode, often the slowest one
Test kit	Measure each slot on your real work	You route on reputation and leaderboard scores
Decision rule	Turn results into a repeatable choice	The choice drifts with whatever you read last
Residency	Decide what's loaded and what gets evicted	Constant swaps, or out-of-memory failures
Fallback	Catch failures and escalate	Fast, confident, wrong answers get through
Log	Record what was routed where and how it went	You can't tell when the rack stops working

Table 1. The six layers of a model rack

2.7 THREE DIFFERENT THINGS CALLED A ROUTER

The word "router" gets used for three different things, and mixing them up causes real confusion.

The **decision router** is the one this paper is about. It decides which slot should handle a request, based on the job type, the test results and the rule. Mine was a benchmark and router agent I built for exactly that.

The **serving switchboard** forwards a request to whichever model the request names, loading it if needed. llama-swap, llama.cpp's router mode and LiteLLM's router are this kind [10] [11] [12]. They're excellent plumbing, but by themselves they don't know which model is good at what. You tell them.

The **mixture-of-experts gate** lives inside a single model. It was learned during training and picks a few expert sub-networks for each token [4] [5] [7]. It makes a big model cheaper to run per token, but as GPU VRAM: The Real Currency of Local AI explained, all the experts still need a place to live [6]. A rack router works per request, outside the models, and you can read and change its rule. An MoE gate works per token, inside the model, and you can't.

They stack. The model that handles my everyday requests now, a community build based on Qwen3.6-35B-A3B, is itself a mixture of experts [16], so an MoE gate sits inside one slot of my rack.

3. WHAT THE RESEARCH SAYS ABOUT ROUTING

Routing isn't my invention. It's an active research area, and the results line up with what a homelab finds the hard way.

FrugalGPT (Stanford, 2023) studied cascades of paid language model APIs whose prices differed by up to two orders of magnitude. The authors reported that a learned cascade could match the best single model, GPT-4 in their tests, with up to 98% lower cost, or beat its accuracy by 4% at the same cost [1].

RouteLLM (UC Berkeley, Anyscale and Canva, 2024) trained routers that pick between a stronger and a weaker model for each query. On widely used benchmarks, the authors report cutting costs by more than half in some cases without hurting response quality, and their routers kept working when the two models were swapped for others at test time [2].

Hybrid LLM (Microsoft and the University of British Columbia, 2024) routed between a small model and a large one based on predicted difficulty, with a quality dial you can adjust at run time. The authors report up to 40% fewer calls to the large model with no drop in response quality [13].

RouterBench (2024) built a benchmark specifically for routing systems, with more than 405,000 inference outcomes, starting from the premise that no single model is best at every task once cost is part of the score [14].

On the commercial side, **OpenRouter's Auto Router** picks a model and a reasoning effort for each request. Its documentation makes a point every rack builder should tape to the monitor: switching models costs extra while the new model rebuilds the prompt cache, so the router is deliberately restrained about switching [15].

Two cautions. These are the authors' results on their benchmarks, mostly with paid cloud models, where the cost is dollars per token. On a local box, the cost is seconds, VRAM and electricity. My inference, not theirs, is that the same logic carries over: if a cheaper slot gets an acceptable answer, every job it handles is time and memory the expensive slot didn't spend. Second, none of these papers tested a 12 GB card in an old FX-8300 desktop with DDR3. That's what the exhibits are for.

4. EVIDENCE FROM CYBERTRON

In early October I built a benchmark and router agent. Its whole job was to answer one question: which of my models is actually useful for which job? Everything below comes

from its output and my own terminal pastes, Oct. 4 and Oct. 16, 2026, except the current 35B figure in Exhibit F, which comes from my system's November logs.

EXHIBIT A: THE MODE MATTERED MORE THAN THE MODEL

One of the models I relied on at the time was a 9B Qwen model, with thinking on by default. The router ran it through a seven-question reasoning test twice, once with thinking on and once with it off.

	THINKING ON	THINKING OFF
Score	7 of 7	7 of 7
Average time to answer	20.465 s	0.361 s
Average time to first visible word	20.386 s	0.276 s
Average tokens generated	877.7	4.3
Peak VRAM	8,791 MiB	8,791 MiB

Table 2. Same model, same seven questions, two modes (Oct. 4, 2026)

Same score, about 57 times faster with thinking off. The thinking run generated about 878 tokens per question to deliver answers that averaged about four. Peak memory was identical, so this wasn't a VRAM story. It was pure time.

Seven questions is one small test, and harder problems may well need the thinking. But it proved a slot is a model plus a mode, and the mode can be the bigger decision.

EXHIBIT B: A SWAP COSTS ABOUT 20 FAST ANSWERS

I ran the same prompt on the same 9B model cold and warm, with thinking on.

- Cold, the request took 35.87 seconds, and 7.49 seconds of that was just loading the model.
- Warm, it took 28.50 seconds, with load time effectively zero.
- Generation speed was the same both times, about 42 tokens per second, and it produced 1,178 tokens for a one-sentence answer.

Put that next to Exhibit A. One cold load, 7.49 seconds, is about as long as 20 of the fast thinking-off answers back to back (7.49 / 0.361 = 20.7). That's the hidden tax of a rack on one card. With the 9B loaded, the card peaked at 8,791 MiB of the 11,909 MiB the driver reported, so there was no room for a second model that size alongside it. Every time a job needs a different slot, somebody pays the load.

The lesson I take from it: route by job type and keep it sticky. Don't bounce between slots inside one conversation unless the gain is worth about 20 answers.

EXHIBIT C: MY RACK MOSTLY PICKED ONE MODEL

The router's first full report, on Oct. 4, sorted the work into six job types:

- General, coding, avoiding made-up answers, reasoning and structured output all went to the 9B with thinking off, at 100% on each suite.
- Vision went to a second copy of that 9B that I'd made in August for image work, also with thinking off.

So five of six job types went to one model in one mode, and the sixth went to that August copy. I wrote about that result in my first white paper. What I didn't spell out there is how thin some of that evidence was. The report covered 59 runs in all: 21 on reasoning, 15 on avoiding made-up answers, nine each on coding and vision, three on structured output and two on general questions. A two-run category isn't a verdict. It's a first look.

That's not a knock on the rack. It's the reason the rack has a test kit at all. You start small, you see where the evidence is thin, and you add questions where it matters.

EXHIBIT D: THE BIG MODEL'S NIGHT

In mid-October a post was making the rounds claiming a 35B mixture-of-experts model ran at about 50 tokens per second on a 12 GB RTX 3060. That's my card. So on the night of Oct. 15 into Oct. 16, I built llama.cpp [17] and tried a community build based on Qwen3.6-35B-A3B [16].

- My first standardized benchmark came in at 5.56 tokens per second.
- After tuning, keeping more layers on the GPU while leaving the expert weights of many layers on the CPU, the best I got was 11.75.
- During testing, some inference requests experienced delays or interruptions. The available evidence does not establish whether these were caused by model-loading behavior, resource contention, timeouts or session-management failures.

I couldn't reproduce the claim on my hardware, and the post didn't list the CPU or memory speed behind it. Building AI on Aging Hardware and GPU VRAM: The Real Currency of Local AI cover why an old FX processor and DDR3 hurt when part of a model lives in system memory [19] [6]. The AI assistant I was troubleshooting with pointed at CPU-side work on the FX-8300 and DDR3 as the bottleneck. That's its diagnosis and my best guess, not a measured proof.

On those numbers, the 35B was a candidate for hard questions that can justify the wait, not for quick everyday work. Exhibit F puts the gap in one number.

EXHIBIT E: SMALL TESTS LIE

On one reasoning run, the August copy of the 9B beat the original, 0.229 seconds to 0.361. In the router's later full report, the order flipped and the copy came out slower on reasoning. I made that copy in August as a straight clone, but I haven't confirmed how it was configured by Oct. 4, so this isn't a controlled comparison. Either way, the flip is a caution: a seven-question test can move on run order, a warm cache or GPU clocks stepping up. That explanation is an inference, not something I measured.

EXHIBIT F: A 12-FOLD SPEED GAP, WITH THE CAVEATS ATTACHED

A system handoff document dated Oct. 21, 2026, put two of my recorded measurements side by side. These are historical, non-normalized measurements, not a controlled head-to-head benchmark.

	9B QWEN MODEL	35B COMMUNITY BUILD BASED ON QWEN3.6-35B-A3B
Generation speed	About 66.6 tokens per second	About 5.56 tokens per second
Date measured	Oct. 4, 2026	Oct. 16, 2026
Runtime	Ollama	llama.cpp
GPU placement	Fully on the GPU	Partly on the GPU (16 layers), the rest on the CPU
Context setting	32,768 tokens	8,192 tokens
Test	Router reasoning suite, thinking off, seven questions	Standardized generation benchmark, 128 tokens, three runs

Table 3. Two recorded speeds, taken under different conditions

These two speeds were recorded on the same 12 GB card, but they are not a controlled head-to-head. The 9B rate, about 66.6 tokens per second, comes from an Oct. 4 reasoning test with thinking off, on answers that averaged about four tokens. A caution noted at the time was that token rates on answers that short are noisy. The 35B rate, about 5.56 tokens per second, comes from an Oct. 16 standardized generation benchmark, with only part of the model on the GPU and a shorter context. On those numbers the 35B ran about 12 times slower (66.6 / 5.56 = 12.0).

After tuning that night I saw 11.75 tokens per second, which would narrow the gap, but that setup wasn't adopted as the everyday setting. And neither measurement shows which model gives better answers. The 35B wasn't scored on that reasoning suite at all.

Separate, current figure (a different build and different settings, not a redo of Table 3). On the 35B configuration I use now, a different build based on the same Qwen3.6-35B-A3B model, completed answers have been around 19 tokens per second, with short stretches near 21, and a long 18,000-token prompt took about 83 seconds to finish.

What it shows is the practical point behind a rack. Speed depends on the build, the placement and the settings as much as on the model, and a big model's answers have to be worth the wait for the job at hand.

WHAT THE EXHIBITS PROVE, AND WHAT THEY DON'T

Fact: on one 12 GB card, a model's thinking mode cost about 57 times the time for the same score on one seven-question test; a cold load cost 7.49 seconds; and the router's October report sent five of six job types to one model in one mode.

Inference: for everyday work on hardware like mine, mode and residency decisions save more time than model shopping does. Small test suites can flip on noise.

Not shown: that a small model beats a big one in general, or which of my models gives better answers. Exhibit F compares speed only, under different conditions. These were also my models on my card; your results will differ.

5. WHY THE RACK WINS

This section is argument, and some of it is opinion. I've labeled which is which.

Fit. A rack lets each job use a model and mode that fits the card. On a 12 GB budget, that's the difference between an answer and an out-of-memory error. (Fact, from the VRAM budget in my last paper [6].)

Speed per job. Most of a day's requests are easy. Exhibit A shows how much time an always-on careful mode can burn on them. (Fact on my test; inference for your work.)

Failure isolation. When one slot misbehaves, you can pull it without losing the others. A single model is a single point of failure, which FrugalGPT's authors also note about depending on one provider [1]. (Inference.)

Replaceable parts. A rack treats models like parts in a chassis. When a better one ships, you test it against the same kit and swap one slot. I think this is the biggest win. In mid-November I swapped the build in my everyday slot for a different build of the same 35B

base model, because the first one had trouble following formatting instructions. Each build should be judged on the same work. (Opinion, with the swap as fact.)

Auditability. A written rule and a log let you answer "why did it pick that?" An MoE gate can't tell you, and neither can your memory of what you meant to do. (Fact about the rule; opinion on its value.)

Honesty. The test kit keeps you from believing a post because it's popular, or a model because it's big. Exhibit D is what happens when you check. (Opinion.)

None of that means a rack is automatically faster. Several models fighting over one card can be slower than one sensible model, and a single strong model on enough hardware can dominate the hard stuff. The rack wins when you measure, because measuring is what tells you which of those situations you're in.

6. THE COSTS AND FAILURE MODES

A rack isn't free. Here's what it costs and how to guard against each problem.

FAILURE	WHAT IT LOOKS LIKE	GUARD
Swap latency	A 7.5-second pause before the first word when a cold slot gets called	Route by job type, keep the default resident, make escalation sticky
VRAM contention	A second model won't load, or something else on the card gets evicted	Leave headroom; one large model per 12 GB card is a realistic limit
Router misjudges difficulty	A hard question goes to the fast slot and comes back fast, confident and wrong	Check outputs where you can; escalate on a failed check
Lost context on handoff	Switching models mid-conversation means the new model reprocesses everything	Switch between conversations, not inside them [15]
Small, noisy tests	Results flip between runs (Exhibit E)	Run more than once; grow thin categories first
Stale test kit	A new model wins on old questions that no longer match your work	Refresh the kit from your log every time you change a slot
More plumbing	More services, more settings, more things to break and secure	Add a slot only when you have the work for it
Benchmark drift	A tuned setting looks great in one run but hasn't been checked for stability (Exhibit D)	Don't promote a setting to the default slot until it survives a soak

Table 4. Rack failure modes

Not every failure is the same failure, either. Besides the delays in Exhibit D, I have also seen a request fail because the prompt was larger than the model's context window, and on one evening the model process was stopped under memory pressure, which made in-flight requests error out. Those are different failures from a normal model unload, and I treat them separately.

The costs that are easiest to underestimate are the stale test kit and the extra plumbing. A rack is a living thing. The test kit and the rule need maintenance, just like the models.

7. WHAT CHANGED SINCE OCTOBER

Here's the part I won't hide.

In October my router said the 9B with thinking off should handle almost everything it was tested on, and the 35B looked like an on-call model at best. That's not where things landed.

Real work kept exposing the 9B. In mid-October I was frustrated that it made things up and returned stale or wrong web results, and the 35B still felt too slow. I gave a different 9B model a brief trial. Then, on Oct. 21, I moved my everyday requests to the 35B community build. In mid-November I swapped that for a different build based on the same Qwen3.6-35B-A3B model, because the first build had trouble following formatting instructions.

Since Oct. 21, a 35B mixture-of-experts model has handled my everyday requests, running on llama.cpp. The 9B Qwen model from early October is still installed as a smaller, faster slot, not as the default.

So CyberTron's model strategy evolved from selecting a single preferred general-purpose model toward evaluating multiple models according to workload. Smaller, GPU-resident models remain important for quick work, while larger models earn their place on tasks that justify the extra latency. Model selection depends on task requirements, available compute resources and measured performance.

What we know. The October router compared models and modes on a small test kit, with a rule that put speed right after accuracy. On that kit, one 9B in the right mode covered almost everything. My real work then turned up quality problems the kit didn't measure. The 35B was much slower in its first runs, got faster with tuning (Exhibit D), and today runs at about 19 tokens per second in a different build and configuration (Exhibit F).

What we don't know, and I won't pretend to. There's no formal accuracy benchmark of the 35B against the 9B in my records. The move to the 35B was a judgment call from real-world use, not a router verdict.

What it says about racks (inference). This is the October lesson, not a reversal of it. My test kit was too small and too easy to catch the problems my daily work hit, which is exactly the "stale test kit" failure in Table 4. The fix wasn't to abandon measuring. It was to change the part in the everyday slot, keep the smaller model in the rack, and the next step is to grow the kit to cover the problems daily use exposed. A larger model also makes residency more important, because on a 12 GB card it leaves little room for anything else.

The mistake would be reading the October report as "small models always win" or today's setup as "big models win." Neither is the lesson. The slot is the stable thing, and the model in it is a part. A rack wins through workload specialization, routing and efficient use of the hardware, not by running the largest model you can fit. The measuring is what tells you when to change a part, and it has to be redone when you do.

8. A STARTER RACK FOR READERS

This is generic advice for a single consumer GPU, not my settings.

1. **Start with one default slot.** Pick an instruction model that fits entirely in your VRAM with room to spare for context. GPU VRAM: The Real Currency of Local AI has the budget math [6]. Run it with thinking off for everyday work.
2. **Add a mode before you add a model.** If your model has a thinking mode, make "thinking on" your second slot and call it on purpose for hard problems. It costs no extra memory.
3. **Build a test kit.** Write 10 to 20 questions from your real work, with answers you already know, grouped by job type. Include a few that should get "I don't know" so you catch made-up answers. Measure accuracy and total time to answer, not just tokens per second.
4. **Write the rule down.** Something like: must pass the job's checks; then higher accuracy; then lower total time, with a 5% tie band; then lower VRAM.
5. **Add specialists only when you have the work.** A coder if you write code every day. A vision model if you actually feed it pictures. An embedding model if you're building a searchable document library. Don't fill slots for bragging rights.
6. **Set residency on purpose.** In Ollama, learn keep_alive and OLLAMA_MAX_LOADED_MODELS first [8]. On llama.cpp, look at llama-swap or the server's router mode [10] [11]. If you mix local and paid models, LiteLLM's router handles fallbacks [12].
7. **Keep a log and rerun the kit** every time you change a model, a quantization, a context size or a runtime version. Grow the categories where your evidence is thin.

SLOT	JOB TYPES	MODEL AND MODE	RESIDENT OR ON CALL	HOW IT'S CALLED	PASS CHECK	LAST TESTED
Default	Everyday questions, writing, summaries		Resident	Router default		
Careful	Multi-step reasoning	Same model, thinking on	Resident (same weights)	Router or on request		
Specialist	Code, vision or retrieval		On call	Job type		
Escalation	Anything that fails a check		On call	Fallback		

Table 5. A blank rack card (fill in from your own tests)

If two slots on your card can't both be resident, write that down too. It tells you which swaps you'll be paying for.

9. CONCLUSION

Everybody wants the one model that does everything. On a 12 GB card, that model either doesn't fit, or it makes you wait while it thinks hard about a four-word answer. A model rack is the alternative: right-sized slots, a test kit from your real work, a written rule and honest residency controls.

My own rack taught me two things that look like they disagree. In October, measuring showed one small model in the right mode could handle almost every job I tested. By late October, real work had moved my everyday requests to a 35B mixture-of-experts model, with the 9B kept as a smaller option. Both came from the same habit. Measure the work, route it on purpose, write down what happened and measure again when the parts change.

The rack isn't the models. It's the measuring and the routing. Build that part well and you can swap everything else.

SOURCES

All links checked Nov. 22, 2026.

[1] Lingjiao Chen, Matei Zaharia and James Zou, Stanford University, "FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance," arXiv:2305.05176, 2023. https://arxiv.org/abs/2305.05176

[2] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous and Ion Stoica, "RouteLLM: Learning to Route LLMs with Preference Data," arXiv:2406.18665, 2024. https://arxiv.org/abs/2406.18665

[3] Qwen Team, "Qwen3 Technical Report," arXiv:2505.09388, 2025. https://arxiv.org/abs/2505.09388

[4] Noam Shazeer and others, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," arXiv:1701.06538, 2017. https://arxiv.org/abs/1701.06538

[5] Hugging Face, "Mixture of Experts Explained." https://huggingface.co/blog/moe

[6] Roggie D, "GPU VRAM: The Real Currency of Local AI," Roger's Rabbit Hole, November 2026. https://www.rogersrabbithole.com/Rabbit-Hole/ArtMID/391/ArticleID/102/GPU-VRAM-The-Real-Currency-of-Local-AI

[7] Albert Q. Jiang and others, "Mixtral of Experts," arXiv:2401.04088, 2024. https://arxiv.org/abs/2401.04088

[8] Ollama, FAQ (keeping models loaded, concurrent models). https://docs.ollama.com/faq

[9] Ollama, "Thinking." https://docs.ollama.com/capabilities/thinking

[10] llama.cpp, server README (router mode, MoE CPU placement options). https://github.com/ggml-org/llama.cpp/blob/master/tools/server/README.md

[11] llama-swap project. https://github.com/mostlygeek/llama-swap

[12] LiteLLM, Router documentation. https://docs.litellm.ai/docs/routing

[13] Dujian Ding and others, Microsoft and the University of British Columbia, "Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing," arXiv:2404.14618, 2024. https://arxiv.org/abs/2404.14618

[14] Qitian Jason Hu and others, "RouterBench: A Benchmark for Multi-LLM Routing System," arXiv:2403.12031, 2024. https://arxiv.org/abs/2403.12031

[15] OpenRouter, "Auto Router" documentation. https://openrouter.ai/docs/guides/routing/routers/auto-router

[16] Qwen, Qwen3.6-35B-A3B model page, Hugging Face. https://huggingface.co/Qwen/Qwen3.6-35B-A3B

[17] llama.cpp project. https://github.com/ggml-org/llama.cpp

[18] Roggie D, "Bigger Isn't Always Smarter," Roger's Rabbit Hole, November 2026. https://www.rogersrabbithole.com/Rabbit-Hole/ArtMID/391/ArticleID/97/Bigger-Isnt-Always-Smarter

[19] Roggie D, "Building AI on Aging Hardware," Roger's Rabbit Hole, November 2026. https://www.rogersrabbithole.com/Rabbit-Hole/ArtMID/391/ArticleID/99/Building-AI-on-Aging-Hardware

Test note: The exhibit numbers come from my own runs on Oct. 4 and Oct. 16, 2026, on an RTX 3060 12 GB, an AMD FX-8300 and 32 GB of DDR3. The thinking test used seven reasoning questions. The October 35B figures are llama.cpp benchmark averages of three runs each. The current 35B figure in Exhibit F is from completed requests in November on a different build and settings.

ABOUT THE AUTHOR

Roggie D has been messing with computers since the early 1980s and working in IT since the mid-1990s, from network engineering to the CEO's chair. These days he's

retired, which mostly means he gets to pick which problems are worth solving. He writes Roger's Rabbit Hole, where CyberTron, his budget local AI build, keeps sending him down new ones.

More from the Rabbit Hole at rogersrabbithole.com.