Inference · Technical report

Running a 35B mixture-of-experts model at 111 tokens per second on a 12 GB card

Trevor Rosato

A mixture-of-experts model only uses a fraction of its weights for each token, which makes it a natural fit for small GPUs, if you control where the experts live and how many of them each token consults. Those turn out to be two very different levers.

November 10, 2026 Work conducted August 2026 7 minute read

A mixture-of-experts model replaces some of a dense model's layers with many smaller "expert" networks and a router that picks a few of them for each token. The model we studied, a Qwen-derived 35-billion-parameter model distributed as I-Mini,[1] follows Qwen's 35B-A3B design: 256 routed experts per layer, eight of them selected for each token, plus a shared expert.[2] Qwen's configuration lists 40 backbone layers plus one multi-token-prediction layer; the file we ran stores 41 blocks, and the block numbers below index those stored blocks.

That makes the arithmetic per token small. It does not make the model small. All 256 experts in every layer still have to live somewhere, and on an RTX 4070 with 12 GB of memory they do not all fit beside the rest of the runtime. Running this

model well on a consumer card is a question of placement and routing, and this campaign, part of our consumer-GPU research, separated the two.

Placement: put the right experts on the host

The first lever is where the expert weights live, a question of residency that offloading research has studied for mixture-of-experts models.[3] A modified llama.cpp runtime[4] let us choose a range of blocks whose expert matrices would sit in host memory, while everything else, including attention and the non-expert layers of those same blocks, stayed on the GPU.

The decision is not one switch for the whole network. Different blocks cost different amounts to move. The useful region in this campaign was the last eight blocks, 33 through 40. Moving only those experts to the host, with normal eight-expert routing untouched, ran at about **94 to 96 tokens per second**. The weights and routing are unchanged: it is the same model it would be on a larger card, keeping part of itself somewhere cheaper.

Routing: consult fewer experts where it matters least

The second lever changes the computation itself. The runtime allowed overriding how many experts the router selects, layer by layer, the same kind of per-layer choice that recent work on layer-adaptive expert counts explores.[5] Fewer experts means less work per token, and less weight data read from memory.

This lever is not free, and it is not lossless. The router is part of the model, and asking fewer experts to contribute can change its answers. So for every schedule we tried, we checked the result on an exact-answer probe as well as measuring speed.

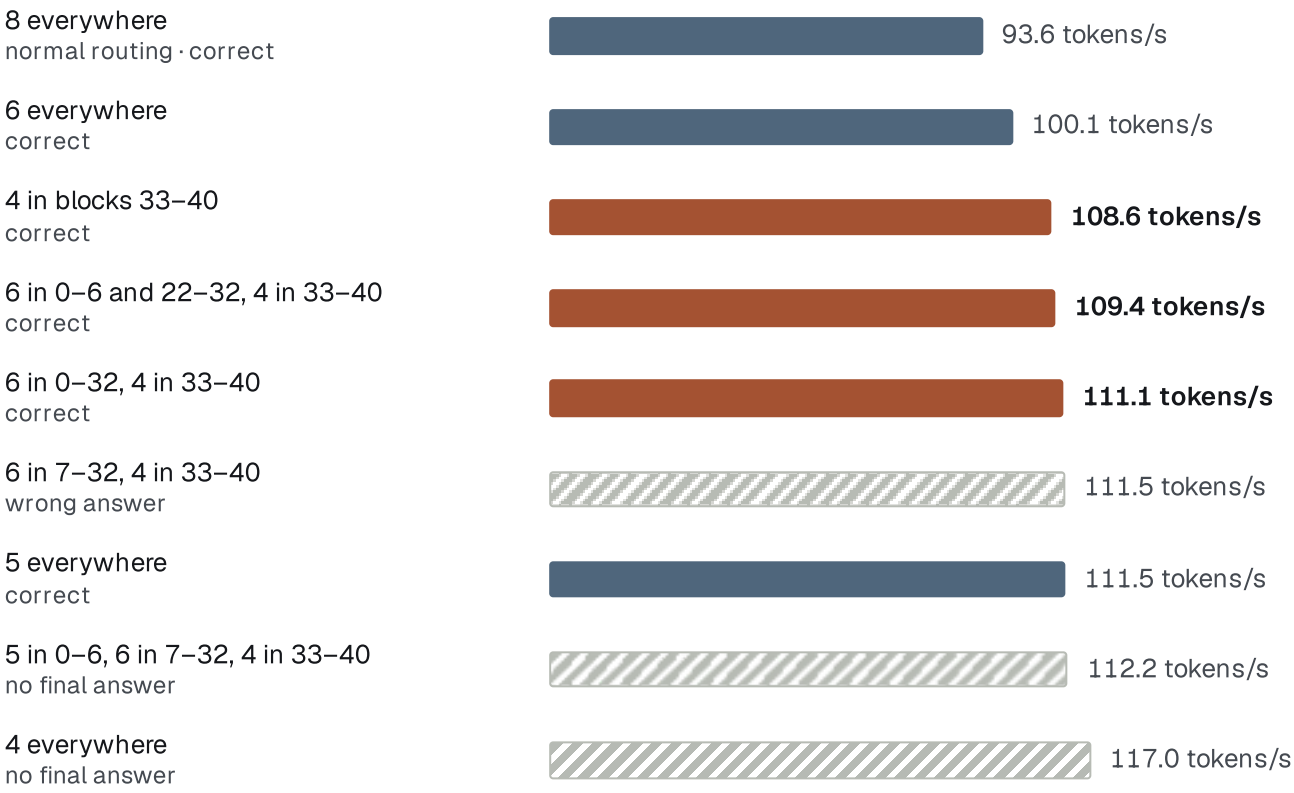


Figure 1

Decoding speed for nine routing schedules, labeled by how many experts each range of blocks consults; blocks not listed keep all eight. Each was checked on the same exact-answer probe. Reducing experts only in the late blocks gained 16% while keeping the answer. Cutting everywhere was fastest and stopped producing an answer at all. Where the reduction happens matters as much as how much is removed.

Reducing only the late blocks to four experts gained **16%**, to 108.6 tokens per second, with the correct answer. Adding a modest reduction to six experts in the earlier blocks reached **111.1** and still answered correctly. Pushing further bought a few more tokens per second and broke things: cutting to four experts everywhere produced 117 tokens per second and no final answer.

The pattern that matters is in the middle of the chart. Two schedules differ only in the first seven blocks: one reduces them to six experts, the other leaves them at the full eight. They run at almost the same speed, and the one that removes *less* computation gets the answer wrong. It was enough to stop us thinking of expert count as a dial that trades quality for speed smoothly. The count of experts

removed predicts neither speed nor damage. Six experts everywhere removes more computation than four in the late blocks and runs slower, 100.1 against 108.6 tokens per second. Which layers you touch decides whether the answer survives.

Fast is not the same as finished

A token rate tells you the cost of generation. It does not tell you whether the request was completed correctly. We tested a small serving policy that treats those separately: start every request on a fast, strict configuration, escalate the ones it cannot handle to an alternate route, and hand exact arithmetic to ordinary code instead of asking the model to compute it, the idea behind program-aided language models.[6]

On a set of five hard math problems, that system produced all five exact answers. Three finished on the fast path, one needed the alternate route, and one needed the calculator. That is a result about the system, not the model, which is precisely the point: a local model does not need to do everything itself to be useful. It needs to do the common case quickly and know when to hand off.

What this campaign changed

This campaign moved much faster than our dense-model work, and the reason is instructive. A mixture-of-experts model exposes its costs in separable pieces. Placement moves memory without changing the weights or routing. Routing changes the computation and needs its own quality checks. Escalation and tools change how a request gets completed and need their own accounting. Keeping those three apart is what made each one easy to tune, and it is a useful habit for anyone fitting large sparse models onto small hardware.

Methods and scope

Measurements come from one RTX 4070 (12 GB) with a modified llama.cpp runtime. Decoding rates exclude prompt processing. The routing sweep generated 128 tokens per configuration with a 4,096-token context limit, and each configuration was checked against a single exact-answer probe; that probe is a diagnostic for obvious breakage, not a measure of model quality. The serving-policy result used five problems and included deterministic code for arithmetic. The exact runtime build was not preserved, so this report describes the experiments rather than providing a reproducible recipe.

The model comes from the Qwen family through the I-Mini distribution; our work concerned runtime configuration and measurement, not training.

References

1. Mudler, Qwen3.6-35B-A3B APEX-MTP GGUF https://huggingface.co/mudler/Qwen3.6-35B-A3B-Claude-4.7-Opus-Reasoning-Distilled-APEX-MTP-GGUF, I-Mini quantization.
2. Qwen Team, Qwen3.6-35B-A3B model card https://huggingface.co/Qwen/Qwen3.6-35B-A3B and config.json https://huggingface.co/Qwen/Qwen3.6-35B-A3B/raw/main/config.json.
3. Artyom Eliseev and Denis Mazur, “Fast Inference of Mixture-of-Experts Language Models with Offloading” https://arxiv.org/abs/2312.17238, 2023.
4. llama.cpp https://github.com/ggml-org/llama.cpp.
5. Krishna Teja Chitty-Venkata, Sandeep Madireddy, Murali Emani and Venkatram Vishwanath, “LExI: Layer-Adaptive Active Experts for Efficient MoE Model Inference” https://arxiv.org/abs/2509.02753, 2025 preprint.
6. Luyu Gao et al., “PAL: Program-aided Language Models” https://proceedings.mlr.press/v202/gao23f.html, ICML 2023.

Cite this

Trevor Rosato. "Running a 35B mixture-of-experts model at 111 tokens per second on a 12 GB card." research.honorlyai.com, November 10, 2026. https://doi.org/10.5281/zenodo.24137189