How a Finnish-RAG experiment caught and corrected its own mistake

A write-up about the process, not the findings. Companion to the results report.

What this is about

I built a retrieval system for my portfolio that answers questions about my projects. At some point I had turned off Finnish support, because I assumed the small local models I run could not handle Finnish well enough. This experiment was meant to check that assumption and decide two things: whether the system could answer in Finnish at all, and if so, whether it needed a model built specifically for Finnish (Poro 2) instead of a general one.

I ran it, got an answer, and wrote it up as a finished PDF. The answer was partly wrong. The way I found that out is the reason for this second write-up. The interesting part is not which model won. It is how the measurement ended up correcting itself.

The setup was careful, and it still produced a wrong result

I was disciplined from the start. Each comparison changed one variable at a time, checked at runtime so the run would stop if a second variable slipped in. The evaluation set was fixed and written before any model ran. The Finnish questions were literal translations of the English ones pointing at the same sources, so a language effect could not be mistaken for a different question. Every phase stopped for review.

None of that stopped a wrong finding from being published. I want to be clear about that, because the discipline was never there to guarantee a correct answer. It was there to make a later mistake findable. That is the part worth paying attention to.

The first answer, and where it broke

The published result had two halves. The synthesis half held up: the small models do produce usable Finnish, and the Finnish-specific model was no better than a general one at it. The containment half was wrong. Containment means refusing off-topic requests, for example a request to write a poem when the system is only supposed to answer questions about my projects. The PDF said the Finnish model was the worst at this and a general model was the best, by a factor of three.

I only caught it because I tried to turn the manual process into a reusable tool. To trust the tool, it had to reproduce the original numbers. It did not.

Four things the measurement could not see

Each problem had the same shape. Something affected the result but was not recorded as part of the measurement, so two runs that were actually different looked the same. The fix was always to make that thing visible.

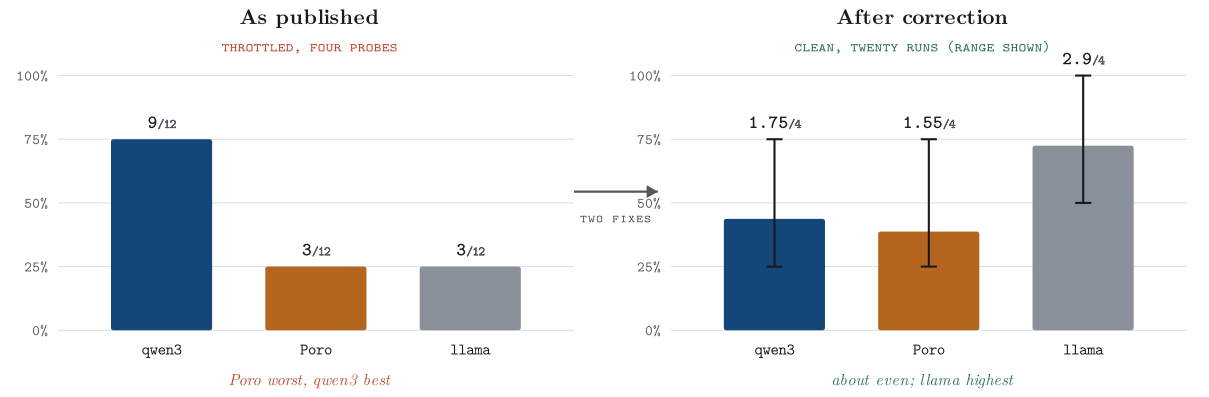
The first was the model's thinking mode. One of the models can run with or without an internal reasoning step, and that changed its answers, but the setting was not recorded anywhere in the configuration. Two runs that differed only in this would have looked identical. I made it a per-run setting and recorded it as part of the run's identity.

The second was the number of runs. The published figure was an average over three runs, but the new tool ran once and produced a different number. Not a bug, just an unrecorded difference in scale. I made the run count part of the run's identity too, so a one-run and a three-run measurement can never be lined up by accident.

The third was the variance itself. With only four refusal probes per run, a containment count swung as much between two of my own runs as it did from the published number. A single figure was hiding a wide spread. I changed the tool to report a range for anything stochastic, and a single value only for things that are actually deterministic.

The fourth was the one that mattered most. The production system has a rate limiter that blocks more than thirty requests a minute from one source. It was rejecting the rapid measurement calls. A timing difference gave it away: some calls took 3.6 seconds, others took 0.2 seconds, which is far too fast to be a real generation. The fast ones were rejections being counted as failures. This had quietly damaged the later runs, and by the same reasoning it had probably skewed the original published numbers as well. I exempted the local measurement path from the limiter, recorded the exemption in the architecture decision log, and added a check that aborts a run the moment a single request is rejected, so the tool can never again treat throttled calls as real data.

The correction



Once the two contaminations were removed, the containment result did not just narrow. It reversed. The model the report had called the worst was actually the best at refusing. The two the report had separated by a factor of three were, within the measured range, about the same. And every containment number was a wide range rather than the firm figure the original showed. The synthesis result survived clean, so I corrected only the containment section, and I wrote into the document why the correction was made.

What I take from it

What actually happened is less flattering than a clean result, and more useful. A careful experiment got a wrong answer, I published it, and then my own tooling forced the correction before I acted on it anywhere. Getting a result right the first time can be luck. Having the process surface its own error is worth more, because it works again the next time.

The tool that found all of this is now reusable. The next time I ask whether to swap one component for another in this system, the same checks run by default: one variable at a time, the run's full identity recorded, ranges for anything noisy, and a hard stop if the measurement is being throttled.