# Why Qwen 3.8 27B Feels Slow: Reasoning Tokens Measured

August 17, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Your answers come back correct, your GPU is not throttling, and a one-line function still takes six minutes. Qwen 3.8 is not slow in the way you are used to diagnosing — it generates at full speed, it just generates vastly more than you asked for, and almost none of it is the answer. I measured all 164 problems on one RTX 3090 at the model's own defaults to find out where it goes.

📚 **More on this topic:** Qwen 3.8 vs 3.6 on RTX 3090 · Why Is My Local LLM So Slow? · Context Length Explained · KV Cache Optimization · Best Qwen Models Ranked

HumanEval problem 108 asks you to count how many numbers in a list have a digit sum greater than zero, with the wrinkle that a negative number's first digit counts as negative. It is a middling problem. Not trivial, not hard.

Qwen 3.8-27B spent **32,000 tokens** on it and never answered.

Not "answered badly" — the response field came back empty. It burned 14.2 minutes of an RTX 3090, hit the token ceiling, and returned no code at all.

What it did with those tokens is not what I assumed before reading the trace. It reasoned coherently for 6,400 characters, worked through the negative-digit rule correctly, and got as far as considering what a hidden test case might contain. Then it emitted a single unbroken run of **30,037 zero characters** and nothing else. That run is 82.4% of its output and it consumed every remaining token.

That is a decode collapse, not a reasoning loop. The model did not think itself into a corner; its generation degenerated and the ceiling merely ended it. It is also **the only problem in all 164 with a degenerate run of even 100 characters**, so 108 is the extreme of the distribution and a different phenomenon from the rest of it. One problem, 7.2% of everything the model generated across the whole benchmark.

Four days earlier I had run problem 108 through the same model file, on the same card, with the same seed, as a plain completion with the chat template switched off. That run answered it in **24 tokens**:

```
return sum(1 for x in arr if sum(map(int, str(abs(x)))) > 0)
```

That answer is wrong — it fails on negatives, the one thing the problem asks you to get right, and the official scorer marks it failed. So the model gets this problem wrong either way. What changed was the price: 24 tokens against 32,000 for the same wrong result, a factor of roughly 1,300, and in the expensive case you do not even get the wrong answer to look at.

The second-worst case is stranger. Problem 2 asks for the decimal part of a float: given 3.5, return 0.5. The model spent 14,953 tokens, got there, and wrote one line.

```
return number - int(number)
```

126 tokens of that were the answer. Two different failures — one reaching a trivial answer expensively, the other never reaching one — and neither is predictable from looking at the task.

I want to be careful about what this proves. It is not evidence the model is bad: it is a strong model, and further down I score it. It is not evidence the model is slow either, because it generated at 37.2 tok/s throughout, which is roughly what this file does on this card. What it shows is where the time goes.

Bar chart of total tokens generated for all 164 HumanEval problems ranked cheapest to most expensive, showing a flat curve that turns sharply upward in the last ten problems

## What I ran

Qwen3.8-27B at Unsloth's UD-Q4_K_XL quant, one RTX 3090, `llama-server` from llama.cpp b10088. The important part is that this run went through `/v1/chat/completions` with `--jinja`, which means the model's own chat template was applied and the reasoning path was live.

That matters because it is the opposite of how I benchmarked this model the first time.

```
reasoning_effort   model default (xhigh, resolved from the GGUF template)
temperature        1.0
top_p              0.95
top_k              20
context            -c 32768
max_tokens         32000
```

```
stop sequences     none
seed               42
```

Sampling follows the model card's own recommendation for thinking mode. I did not pick these numbers; this is what you get if you follow the instructions. The model file was checksummed against upstream immediately before load and again after the run, and both digests matched. That is footing rather than story, after this rig turned up a single-bit page-cache fault last week.

All 164 problems, 3 hours 19 minutes, 444,556 tokens generated. **One seed**, which is the real limit on this page and I come back to it at the end.

**A note on the one truncated problem.** HumanEval/108 hit the ceiling with an empty response, so it is reported as a truncation and excluded from the accuracy question — scoring an empty string as a wrong answer would turn a budget artifact into a capability claim, which is the exact confusion this page exists to undo. It stays in the token distribution, where its true cost is unknown: all we know is that it is at least 32,000.

## Where the tokens go

| | min | median | mean | p95 | p99 | max |
|---|---|---|---|---|---|---|
| **Total generated** | 160 | 993 | 2,711 | 9,213 | 30,304 | 32,000 |
| **Thinking** | 100 | 790 | 2,515 | 8,988 | 30,073 | 32,000 |
| **Answer** | 0 | 173 | 196 | 389 | 515 | 602 |

Across all 164 problems: 444,556 tokens generated, 412,441 of them thinking. **Thinking is 92.8% of output. The answer is 7.2%.** Percentiles are nearest-rank over all 164, including the truncated problem at its censored value of 32,000.

The number to look at is that **the mean is 2.7 times the median** — the signature of a heavy tail, where a handful of problems set the average while typical behaviour sits far below it. Half the set finishes under 993 tokens. The ten most expensive problems produced 44.9% of all output between them; the cheapest hundred produced 14.4%.

Now compare against the raw completions path, from the benchmark I published on the 14th. Same model file, same card, same seed, no chat template, same 164 problems:

| Path | median tokens | mean tokens | total, 164 problems |
|---|---|---|---|
| Raw `/v1/completions` | 41 | 56.6 | 9,282 |

| Path | median tokens | mean tokens | total, 164 problems |
|---|---|---|---|
| Chat `/v1/chat/completions` | 993 | 2,711 | 444,556 |

**24x at the median. 48x at the mean.** Same weights, same hardware, same seed. The only thing that changed is whether the chat template was applied.

Problem 108 from the opening is the extreme of that distribution, not an exception to it. Problem 2 is the same shape at smaller scale: the raw path emitted nine tokens, `return number - int(number)`, and the chat path took 14,953 to reach the identical line.

## The part that actually matters

Look at the answer row again. 0 to 602 tokens, median 173: bounded and boring, which is what a benchmark of small functions should look like. Now the thinking row. 100 to 32,000, **a 320x spread**. Almost all the variance in what this model costs you lives there, and that would be manageable if it tracked how hard the problem was.

Answer length is the obvious thing to check and it is only a loose guide. The two correlate at Spearman r = 0.53, about a quarter of the variance. Among the 39 problems whose answers land between 150 and 200 tokens, thinking still ranges from 131 to 17,636.

Scatter plot of thinking tokens against answer tokens for all 164 HumanEval problems, showing a weak upward trend with very wide vertical spread

## Does the thinking track difficulty?

So I checked four better proxies. None of them is difficulty itself, and I would not lean on any single one, but they disagree usefully.

| Proxy for difficulty | Spearman r | Variance explained |
|---|---|---|
| The model's own raw-path token count | +0.44 | 19% |
| Reference solution, characters | +0.40 | 16% |
| Reference solution, lines | +0.32 | 10% |
| Prompt length | +0.29 | 9% |
| Number of asserts in the test | +0.21 | 4% |

Every one is positive and every one is weak. The best predictor is not a property of the problem at all. It is how many tokens the same model spent on the same problem with the template off, which is closer to how much this model wanted to say than to how hard the task was. Assert count is the outlier at the bottom, and HumanEval/32 shows why: a **single** assert, 26,850 thinking tokens.

Quartiles by reference-solution length separate what these proxies do and do not capture:

| Quartile | Median thinking | Most expensive problem |
|---|---|---|
| Q1, shortest solutions | 369 | 14,827 |
| Q2 | 805 | 17,459 |
| Q3 | 860 | 32,000 |
| Q4, longest solutions | 1,857 | 30,073 |

The medians rise about fivefold, monotonically. **Difficulty sets the floor**, and the cheap end confirms it: the ten cheapest problems have one-line, 32-character reference solutions, three asserts each, a median of 12 tokens on the raw path, and the raw path got all ten right.

What difficulty does not set is the ceiling. The most expensive problem in the run sits in Q3, and the worst blow-up in the easiest quartile ran to 14,827. Every quartile has a tail. Problem 2 belongs to that easy group by every measure I have — one line, 24 characters, three asserts, answered correctly in nine tokens with the template off — and on the chat path it cost 14,827.

Log-log scatter of raw-path tokens against chat-path thinking tokens for all 164 problems, coloured by whether the raw path answered correctly

It is also why the usual local-LLM speed playbook does nothing here. If you came from the slow-model diagnostic route you have already checked offload, quant size, and `num_ctx` overflow, and all three came back clean, because this is not a tokens-per-second problem. Your card is fine. You are being billed for tokens you did not know you ordered.

## This is what reviewers have been running into

The behaviour has been visible since launch, mostly as frustration without a mechanism attached.

Simon Willison covered it on August 16 in a post about the model's default reasoning behaviour: one SVG drawing, around 21 minutes and roughly 22,000 reasoning tokens for about 3,200 tokens of output. His advice is to skip the default and start at low or no reasoning.

Two YouTube reviewers hit it independently in agent workflows. One kept 3.6's settings, ran into session compaction three times and failed all three tasks he set. Another reports burning 87% of a 128K context on a single Kanban app. Those are their observations, not mine, and I repeat them because the numbers above explain what they were seeing: none of them were hitting a broken model, they were hitting a context budget sized for a model that no longer exists.

## Why my own benchmark showed none of this

Here is the awkward part, and I would rather raise it than have someone else raise it for me.

The head-to-head I published on August 14 found 3.8 and 3.6 statistically indistinguishable on HumanEval, 80.49% against 82.32%, McNemar p=0.70. That run used `/v1/completions` with `--no-jinja` and raw prompts. In llama.cpp's source the two endpoints have entirely separate parsers, and the one behind `/v1/completions` contains no reference to templates, reasoning, or thinking at all. Everything on this page lives in the Jinja template, which only `/v1/chat/completions` reaches. I checked all 164 completions from that run: not one contains a `<think>` tag, and the median was 41 tokens. The reasoning path never fired.

**That null still stands and this page does not contradict it.** It compared two models as raw code completers, treated identically, which is how HumanEval is normally run. What it measured was not the reasoning path. This page measures that path's cost.

Whether the cost buys accuracy is a third question, and since I had the completions I scored them. The chat path solved **155 of the 163 problems it answered, 95.09%**, against the raw path's 80.49%, fixing 25 the raw path got wrong and breaking 2 it got right. HumanEval/108 is out of that denominator, because scoring an empty string as a wrong answer is the exact mistake this page exists to warn about.

Hold it loosely: one seed, one model, one quant, no 3.6 comparison yet, and a heavy tail sampled at temperature 1.0 is what a second seed moves. Seed 43 and both 3.6 arms are running and the four-run version gets its own page. But the direction is settled — **the reasoning is buying something substantial**, and nothing below should be read as saying otherwise. On what a HumanEval number is worth either way, we've written the long version.

## What to actually do about it

Qwen 3.8 ships a `reasoning_effort` parameter with three levels. **Qwen 3.6 has no such parameter.** This is new, and it is the single biggest reason 3.8 does not behave like a drop-in swap even though the model card never says so. It defaults to `xhigh`.

Set it per request:

```
{
  "model": "qwen3.8-27b",
  "messages": [{"role": "user", "content": "..."}],
  "chat_template_kwargs": {"reasoning_effort": "low"}
}
```

Server-wide, Unsloth documents the same keys as `--chat-template-kwargs '{"reasoning_effort":"low"}'`.

Two traps in that knob, both found by dumping the template out of the GGUF and neither in any documentation I can find. **`high` is silently rewritten to `xhigh`**, so if you assumed the ladder ran xhigh, high, medium, low and picked "high" to dial back, you set the default and nothing warns you. And **`none` is not valid on this file** despite Unsloth listing it — the template raises on anything outside `xhigh`, `medium`, `low`. `medium` injects no instruction at all, making it the absence of steering rather than moderate steering.

Because of that I would reach for llama.cpp's own flags, which sit outside the template and behave predictably. `--reasoning` takes `on`, `off`, or `auto` and defaults to auto, which does whatever the template says. `--reasoning-budget` defaults to `-1` for no limit; `0` ends thinking immediately, any positive number caps it there. That is a real ceiling, not a suggestion the model can talk itself out of, and the next section is about where to set it. A companion `--reasoning-budget-message` injects text before the closing tag when the budget runs out.

For multi-turn agent sessions, `--no-reasoning-preserve` stops the template carrying every earlier thinking block forward, so your prompt does not grow while your conversation stands still. I have not tested whether that is what the reviewers hit — it is a hypothesis about their sessions, not a measurement of them.

None of this applies to raw completions. Drive `/v1/completions` with `--no-jinja`, as a code-completion backend does, and the template never runs and the thinking never fires.

## Cap the thinking first, then size the context

On the first twenty problems the obvious conclusion was that you should give the model 32K of context and get on with it. The full run says that would have been wrong advice, and it is worth being precise about why, because the obvious correction is also wrong.

The obvious correction is "32K was too small, use more." It is not. HumanEval/108 truncates at 16,384 and at 32,000, and nothing about a generation that collapsed into repeating one character suggests 64,000 would have rescued it. Seven problems went past 16,384 here and four past 20,000. The tail does not taper politely.

So the control is the cap rather than the context. And the reason is not that the tail is expensive; it is that the tail is where the evidence says the thinking stops paying.

**The 25 problems the reasoning genuinely fixed have a median thinking length of 1,697 tokens.** The set median is 790; the eight problems over 10,000 have a median of 20,420. The work that changes outcomes happens in the middle of the distribution, an order of magnitude below the tail.

The tail bears that out. Of the seven expensive problems that answered at all, the chat path got 4 right against the raw path's 2. Better, and nowhere near the 95% it manages across the set. Two of those four were ones the raw path had already solved for nine and 116 tokens. Genuine gain from the roughly 148,000 thinking tokens spent above 10,000: **two problems**.

**Set `--reasoning-budget` first and treat context as whatever you need once the cap is in place.** From this distribution I would start at **8192**:

```
llama-server -m Qwen3.8-27B-UD-Q4_K_XL.gguf --jinja \
  -c 16384 --reasoning-budget 8192
```

8,192 sits above the productive region and below the tail. **154 of 164 problems finish their reasoning without ever reaching it**, and so do 23 of the 25 the reasoning fixed.

The two it costs you are the honest price and I would rather name them than wave at them: HumanEval/132 and /137 were genuine gains at 17,636 and 30,073 thinking tokens, and the cap gives them up. In exchange you bound every worst case in the run. Two problems out of 164 against that is a trade I would take, and it is a trade rather than a free win.

Tighten it further if your work is routine: 4096 touches 23 problems, 2048 touches 42, which is defensible for bulk work where a retry is cheap and a fourteen-minute stall is not.

Context then follows from the cap instead of trying to contain the model. With reasoning bounded at 8,192, the longest prompt here at 457 tokens and answers topping out at 602, 16K is comfortable. Uncapped, no context size is, which is the whole point. On what the cache costs in VRAM at each size, we've measured it. For multi-turn agent work the reviewers say 128K uncapped is still not enough; cap the reasoning there too.

## What this does not tell you

164 problems of self-contained Python function synthesis, one model, one quant, one card, and **one seed**. Specifically not established:

- **That a second seed reproduces any of this.** Sampling at temperature 1.0 is stochastic, and every number on this page comes from seed 42. A heavy tail set by ten problems is exactly the kind of statistic a different seed could move, and the 320x figure in particular rests on single observations at both ends. Seed 43 is running now, as are both Qwen 3.6 arms. I will report what they say whether or not it flatters this page.
- **That the 95.09% holds up.** It is a single arm on a single seed with no 3.6 comparison, which is why it sits in one paragraph above rather than in the headline. The four-run version is a separate article.
- **Anything about other models.** Every current reasoning model has some version of this. I measured one. Do not read these ratios onto GPT-OSS, DeepSeek, or anything else.
- **Multi-turn or agentic behaviour**, which is where reviewers hit the wall and where I have no measurements at all.

One loose end: generation averaged 39.2 tok/s over the first twenty problems and 37.2 tok/s over all 164. Decode slowing as the KV cache fills would produce that, and so would thermal drift over three hours. I did not isolate them.

## The bottom line

Qwen 3.8-27B is not slow. It runs at 37.2 tok/s on a 3090, which is where 3.6 sits too. It generates 24 to 48 times more tokens than the same file does with the template switched off, 92.8% of them are thinking, and the amount varies by 320x in a way that answer length only weakly predicts.

The fix is a cap, not a bigger room. Set `--reasoning-budget 8192` and give it 16K of context; that leaves 154 of 164 problems untouched and bounds the ten that would otherwise cost you minutes each. Drop to `--reasoning off` when you already know the task is easy, and check you are not accidentally setting `high` and getting `xhigh`. Then judge the model on what it produces rather than on how long you waited, because on the evidence so far those are separate questions and only one of them is settled.

Source: https://insiderllm.com/guides/qwen-3-8-27b-reasoning-token-cost/

Free guides for running AI locally