# Qwen's Coding Model Looked Like the Worst. It Isn't, but It Can Break Your Tools.

November 17, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** On HumanEval, scored exactly as I pre-registered it, Qwen3-Coder-30B came last of five: 61% against 84% for the general Qwen3.6-27B. Two-thirds of its failures were correct code with an explanation pasted after it, which a raw-completion harness can't run. On MBPP it tied the 27B, and in the same runs it generated about five times as fast. Here's every number on a 3090 and a 3060, what held up, and the one stop sequence your editor or agent tool needs.

📚 **More on this topic:** Qwen 3.8-27B vs 3.6-27B on a 3090 · The 35B MoE on a 3090 · What Can You Run on 12GB VRAM · What Can You Run on 24GB VRAM · InsiderLLM Benchmarks

You want a local model for code, and the obvious candidate is the one with “Coder” in the name. Qwen has two that matter on consumer cards. Qwen3-Coder-30B-A3B-Instruct is a mixture-of-experts model: 30.5B parameters, about 3.3B active per token. Qwen3-Coder-Next is an 80B MoE with about 3.9B active, counted the same way from the GGUF headers (Qwen’s card says 3B, which leaves out the embedding and output layers). There is no official Qwen Coder after Coder-Next: no 3.5, 3.6 or 3.8 Coder, whatever the fine-tune names on Hugging Face suggest.

I wrote down every prediction before I downloaded anything, ran both Coders on my RTX 3090 and my RTX 3060, and scored them on HumanEval against the two general 27B models I’d already measured. The result I’d predicted didn’t happen. What did happen turned out to be mostly about how a tool reads the model’s output, and that’s worth knowing before you wire one into your editor.

## The question, and the rules I set myself

The question was simple: how fast do the Coders run on a 24 GB and a 12 GB card, and are they better at code than Qwen3.6-27B and Qwen3.8-27B? The two 27Bs are general models, and both scored around 80% or better on HumanEval when I tested them in August.

Everything was pre-registered before any download: the files and their hashes, the build, the server flags, the void rules for a bad measurement, and a band for every result. The lock is public: the main pre-registration (locked November 8) and the MBPP follow-up I’ll get to below. Where I changed or added something after seeing data, it’s appended and dated, and I say so here.

How far you can trust those dates is worth spelling out. The main pre-registration was locked by a commit to my private repository on November 8 (file sha256 `caa23d8e5dc88956fe02ef4d6a26c4e027b719f383a20583efe4810fff829db7`), so its date rests on my own git history and isn't independently verifiable. The public copy also carries two dated additions below the lock: amendment A1, which tightened one rule for voiding a speed measurement, and deviation D1, the post-hoc failure audit described below. The MBPP lock was committed and pushed to the same private repository before its run started. Its public copy went up at 14:24 PST on November 17 (here), after three of the five arms had already been scored. It hashes to the value in the lock commit, so the text didn't change, but the public timestamp doesn't prove it came first either.

The setup, in one paragraph:

- llama.cpp v0.4.0 (`5266f24`) on both rigs.
- The 3090 box is tamanna: Ryzen 7 5700X, DDR4-3000, PCIe 4.0. The 3060 box is rushuna: i7-7700, 32 GB of DDR4-2133, PCIe 3.0. The RAM speeds differ, so any offloaded comparison between the two rigs is directional.
- Coder-30B is unsloth's UD-Q4_K_XL (17.7 GB), the same quant family as the 27B anchors.
- Coder-Next is Qwen's official Q4_K_M (48.4 GB) on the 3090, and unsloth's UD-IQ3_XXS (28.5 GB) on the 3060, because the Q4 doesn't fit in 12 GB of VRAM plus 32 GB of RAM.

## Speed: Coder-30B flies, Coder-Next beat my prediction

Each cell is a streamed chat request with a 4K, 16K or 32K-token prompt of llama.cpp source code, 256 tokens out, three measured repeats after a discarded priming request. L1 is llama-server's default batch sizes; L2 adds `-b 4096 -ub 4096`, which is where prefill speed comes from on an offloaded model. `-ncmoe N` keeps the experts of N layers in system RAM; N was found by a pre-registered search, the lowest that serves a 32K prompt.

**RTX 3090 (tamanna)** · record

| Model, N | Arm | Decode 4K / 16K / 32K, tok/s | Prefill 4K / 16K / 32K, tok/s |
|---|---|---|---|
| Coder-30B UD-Q4_K_XL, N = 0 (all on the card) | L1 | 190.1 / 149.9 / 117.7 | 4,009 / 3,205 / 2,473 |
| | L2 | 190.1 / 149.9 / 117.7 | 6,923 / 5,125 / 3,723 |
| Coder-Next Q4_K_M, N = 28 | L1 | 48.0 / 47.0 / 46.0 | 314 / 313 / 308 |
| | L2 | 48.0 / 47.1 / 46.0 | 1,253 / 1,318 / 1,308 |

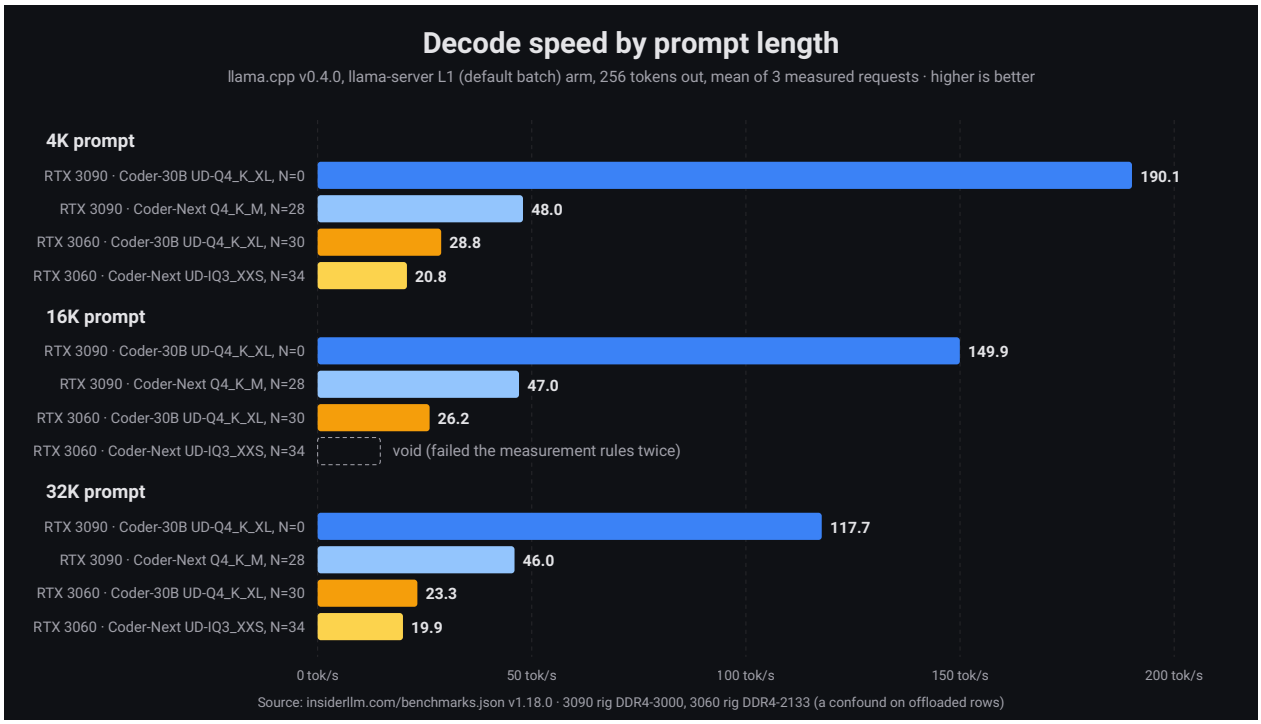
**RTX 3060 12 GB (rushuna)** · record

| Model, N | Arm | Decode 4K / 16K / 32K, tok/s | Prefill 4K / 16K / 32K, tok/s |
|---|---|---|---|
| Coder-30B UD-Q4_K_XL, N = 30 | L1 | 28.8 / 26.2 / 23.3 | 397 / 384 / 354 |
| | L2 | 28.7 / 26.1 / 23.3 | 1,394 / 1,259 / 1,025 |
| Coder-Next UD-IQ3_XXS, N = 34 | L1 | 20.8 / void / 19.9 | 228 / void / 228 |
| | L2 | void / 20.4 / 19.9 | void / 766 / 761 |

Two cells on the 3060 are **void**, and I'm reporting them that way rather than filling them in. My rules void a cell if the box swapped during a measured request, or if the prompt lengths in a cell spread by more than 3 tokens. Each void cell gets exactly one re-run. Both of these failed twice. Coder-Next L1-16K swapped the first time and missed the prompt rule on the re-run. L2-4K missed the prompt rule, then swapped on the re-run, which I'd scheduled straight after Coder-30B with the page cache full. The re-run speeds were within 1% of the void ones, so nothing is hiding there, but a rule is a rule. The prompt spread comes from a random nonce at the top of each request, which tokenizes to slightly different lengths; I've noted it, and kept the rule.

What I got right and wrong, against the bands I wrote down on November 8:

- **Coder-Next decoded faster than I predicted on both cards.** I gave it 22–40 tok/s on the 3090 and got 46–48. On the 3060 I gave it 7–15 and got about 20. Most of its layers are linear attention, and I'd predicted decode would stay nearly flat with depth (32K at least 85% of 4K). It came in at 96% on both rigs. Flat was right; the level was wrong.
- **Coder-30B on the 3090 beat its 4K band too:** 190 tok/s against 120–175, and 6,923 tok/s L2 prefill against 2,500–4,500. It's full attention in every layer, so I predicted decode would fall to 80% or less of 4K by 32K. It fell to 62%.
- **Coder-30B on the 3060 landed inside its bands** (decode 28.8 against 28–45, right at the bottom edge).




The practical number on the 3060 is time to first token. A 32K prompt into Coder-30B at default batch sizes took **94 seconds** before the first token. With `-b 4096 -ub 4096` it took **34**. If you paste whole files into an offloaded model, set those two flags.

## HumanEval, scored as locked: the Coders lost

HumanEval is OpenAI's 164 Python problems; you get the function signature and docstring, the model writes the body, and the official harness runs the tests. I scored it exactly the way I'd scored the two 27Bs in August: raw text completion, no chat template, greedy, lm-eval's five stop sequences, the official scorer in a sandbox. All five arms ran on the 3090, including the 3060's IQ3 file (at N = 8) so the two Coder-Next quants are scored on the same box. Every file hashed the same before and after its run, and a repeat of the first arm at the end came out byte-identical. HumanEval record.

| Arm | pass@1 | My predicted band | |
|---|---|---|---|
| Qwen3.6-27B UD-Q4_K_XL | **137/164 (83.5%)** | 132–138 | hit |
| Qwen3.8-27B UD-Q4_K_XL | 130/164 (79.3%) | 129–135 | hit |
| Coder-Next Q4_K_M | 120/164 (73.2%) | 80–93% | **miss, below** |
| Coder-Next UD-IQ3_XXS | 113/164 (68.9%) | 75–91% | **miss, below** |
| Coder-30B UD-Q4_K_XL | **100/164 (61.0%)** | 75–88% | **miss, below** |

The paired tests (exact McNemar, which only counts the problems where the two models disagree) make it plain:

| Pair | Only the first passes | Only the second passes | p | |
|---|---|---|---|---|
| Coder-30B vs Qwen3.6-27B | 12 | 49 | 0.000002 | the 27B measurably wins |
| Coder-Next vs Qwen3.6-27B | 7 | 24 | 0.0033 | the 27B measurably wins |
| Coder-Next vs Coder-30B | 42 | 22 | 0.017 | Coder-Next measurably wins |
| Coder-Next IQ3 vs Q4 | 13 | 20 | 0.30 | tie |
| Qwen3.8-27B vs 3.6-27B | 11 | 18 | 0.27 | tie |

As locked, that's the result, and it stays the primary one: **both Coders scored significantly worse than the general Qwen3.6-27B.** I'd predicted Coder-Next would sit at or above it and Coder-30B would tie it. Every Coder prediction missed, low.

## What the failures actually were (post-hoc)

A 23-problem miss on a coding model is strange enough that I read every failure before writing anything. To be clear about status: this audit was decided after I'd seen the scores, so it's exploratory. It's recorded in the pre-registration as deviation D1, and it changes no score above. Audit files.

The harness treats every model identically. There's no code extraction at all; whatever the model returns is appended to the prompt and run. The difference is what Coder-30B returns. A typical failure is a correct-looking function body, then a closing ``` it never opened, then "This function works by: 1. Iterating through…". Python can't parse that, so the problem fails.

| Failure type | Qwen3.6-27B | Coder-30B | Coder-Next |
|---|---|---|---|
| Wrong logic | 23 | 12 | 26 |
| Closing fence then prose, passes once cut there | 0 | 41 | 3 |
| Fence or prose, fails even when cut | 0 | 7 | 1 |
| Other junk tokens after the body | 0 | 0 | 4 |

| Failure type | Qwen3.6-27B | Coder-30B | Coder-Next |
|---|---|---|---|
| Helper function cut off by a stop sequence | 0 | 1 | 2 |
| Truncated at 512 tokens | 0 | 3 | 3 |
| Missing import, or timeout | 4 | 0 | 5 |
| **Format failures** | **0 of 27** | **49 of 64** | **10 of 44** |

By my classification, Coder-30B got only 12 problems wrong on logic, fewer than the 27B's 23. Its trouble is that it explains itself like a chat model, in a harness that doesn't expect chat.

Cutting every completion at its first ```, the same rule for all five models, and re-scoring in the same sandbox:

| Arm | As locked | Cut at first fence (post-hoc) |
|---|---|---|
| Qwen3.6-27B | 83.5% | 83.5% |
| Qwen3.8-27B | 79.3% | 79.3% |
| Coder-30B | 61.0% | **86.6%** |
| Coder-Next Q4_K_M | 73.2% | 75.0% |
| Coder-Next IQ3_XXS | 68.9% | 72.0% |

On that view, Coder-30B goes from last to first, and its gap to Qwen3.6-27B becomes a tie ($p = 0.46$). But I picked the cut rule after looking at its failures, which means it's fitted to the data it explains. That makes it a hypothesis. So I tested it.

## The re-test, locked in advance: MBPP

MBPP's sanitized split is 257 short Python tasks, a different benchmark the cut rule had never seen. I wrote a new pre-registration before generating anything. It covers the same five arms, servers and settings, and fixes the scoring in advance: primary score cut at the first fence, secondary raw. It states six predictions, with a rule for what counts as confirmed. The hypothesis: with the cut, Coder-30B ranks at or near the top; scored raw, it drops again.

The lock's timing is covered above: committed before the run (sha256 `62c5e120…`), public partway through it. MBPP record.

| Arm | Cut (primary) | Raw |
|---|---|---|
| Qwen3.6-27B | **217/257 (84.4%)** | 217 |
| Coder-30B | **205 (79.8%)** | 205 |
| Qwen3.8-27B | 204 (79.4%) | 204 |
| Coder-Next Q4_K_M | 198 (77.0%) | 198 |
| Coder-Next IQ3_XXS | 190 (73.9%) | 190 |

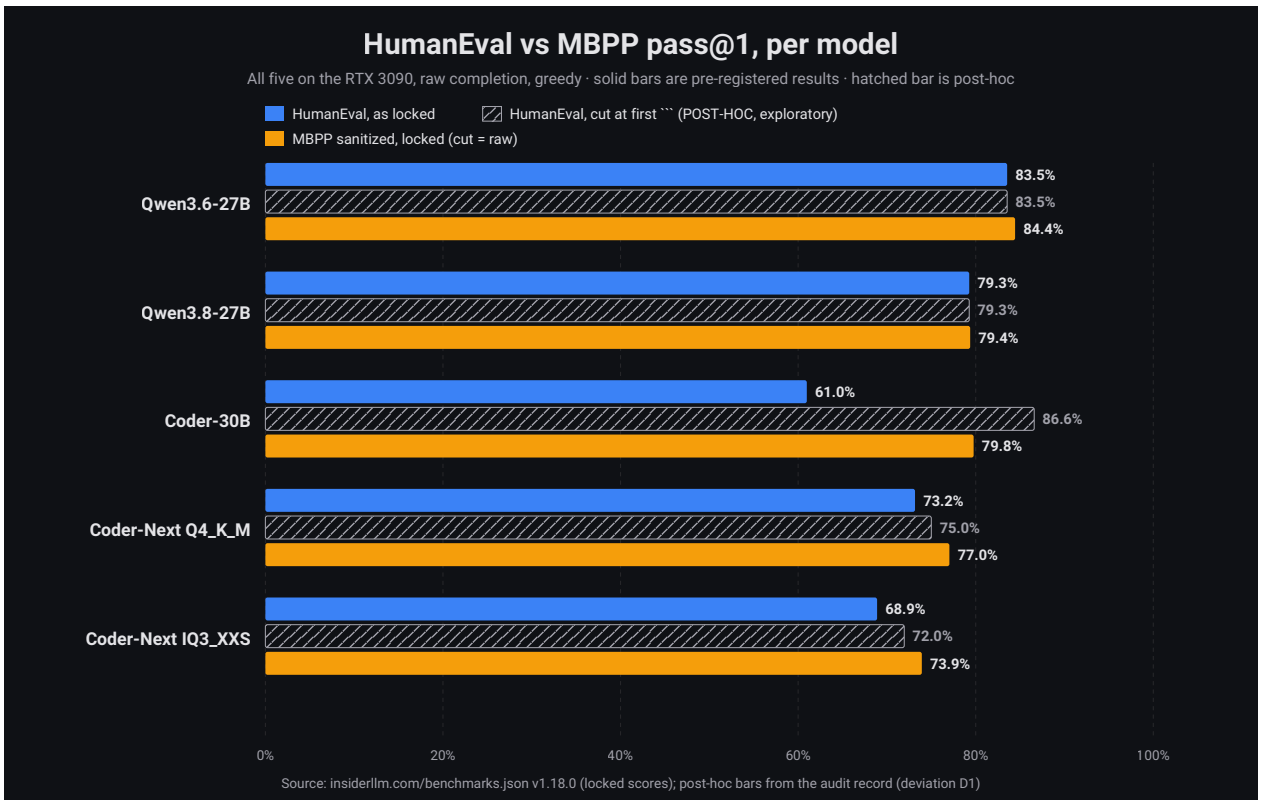
Coder-30B came second, and against Qwen3.6-27B it's a statistical tie: 12 problems only it solved, 24 only the 27B solved, p = 0.065. That's close to the line, and the 27B's point estimate is higher, but under the rule I wrote in advance it isn't a measurable win. Predictions H1 (rank 1st or 2nd) and H2 (no measurable 27B win) hit.

The part that didn't happen: **no model wrote a code fence in any of its 257 answers.** So cut and raw scored identically, and H3, the prediction that raw scoring would knock Coder-30B down again, missed: there was nothing for the cut to remove. The verdict under my own rule is **PARTIAL**: not confirmed, not disconfirmed.

My best explanation, flagged as an observation rather than a result: 136 of Coder-30B's MBPP answers end on a comment like `# Test the function`, against 11 for the 27B. It was about to do the same thing it did on HumanEval. MBPP's stop sequences include `\nassert` and `\nprint`, which cut it off before it got to the prose. HumanEval's stops don't include `\nassert`, and nothing catches a fence.

So, on a second benchmark with the same raw-completion setup, Coder-30B ties the general 27B instead of losing to it badly. That fits the fence explanation. It isn't proof of it, because the mechanism never fired.

## What held on both benchmarks




Two findings came out the same way on HumanEval and MBPP, both pre-registered pairs:

- **Coder-Next is measurably below Qwen3.6-27B.** On HumanEval it was 7 against 24 discordant problems (p = 0.0033). On MBPP it was 12 against 31 (p = 0.0054). Coder-Next's format failures were few (10 of 44), so the fence story doesn't rescue it: even with the post-hoc cut, the 27B still wins (p = 0.020). A 48 GB, 80B-total coder lost to a 17.6 GB dense general model, twice.
- **The 3060's IQ3_XXS file ties the 3090's Q4_K_M.** On HumanEval it was p = 0.30, and on MBPP p = 0.33. A 3060 owner running Coder-Next at IQ3 gives up nothing either test can detect against the bigger quant. One caveat: I scored the IQ3 file on the 3090 at N = 8, and a different CPU/GPU split can flip a problem or two.

## One I didn't predict: Qwen3.6-27B beat Qwen3.8-27B

On MBPP, Qwen3.6-27B beat Qwen3.8-27B measurably: 217 against 204, 21 against 8 discordant, p = 0.024. On HumanEval the same pair was a tie (p = 0.27), the same verdict as my August comparison. I made no prediction for this pair on MBPP, and it's one benchmark, so I'm treating it as a lead. But it's the second time the newer 27B hasn't shown up ahead on code in my tests.

## What to use

**RTX 3090 or another 24 GB card:**

- **For speed, run Coder-30B fully on the card.** It decodes at 190 tok/s at 4K, against roughly 31–32 tok/s for the 27Bs in the same HumanEval runs (that's the harness's whole-run rate, not a speed measurement). It prefills a 4K prompt in under a second with L2 flags, uses about 21 GB of VRAM, and ties Qwen3.6-27B on MBPP.
- **For the best score per problem, Qwen3.6-27B** was top on both benchmarks.
- **Skip Coder-Next on this card** unless you need something I didn't test, like its long context. It's a 48 GB file, runs at 48 tok/s offloaded, and lost to the 27B twice.

**RTX 3060 12 GB:**

- **Coder-30B at `-ncmoe 30` is the pick.** It decodes at 28.8 tok/s at 4K and 23.3 at 32K, nearly 40% faster than Coder-Next IQ3 here. It scored 205 to Coder-Next IQ3's 190 on MBPP, though that pair wasn't one of my pre-registered tests. On HumanEval the order flips, which is the fence habit again.
- **Coder-Next IQ3 at `-ncmoe 34`** decodes at about 20 tok/s and barely slows with depth, but scores lower.
- **Either way, use `-b 4096 -ub 4096`**. It's 94 seconds against 34 to first token on a 32K prompt.

**If your tool uses raw completion, add a stop sequence.** Editor autocomplete, fill-in-the-middle and some agent tools call `/v1/completions` and paste whatever comes back straight into your file. That's exactly what HumanEval does, and it's what broke Coder-30B there. Chat interfaces are fine: a closing fence and an explanation are normal markdown, and the UI renders them. In a raw-completion tool, add ``` ``` ``` to the stop list, or cut the output at the first fence before inserting it. It's one line of config, and it's the difference between 61% and 87% on the benchmark that made this model look like the worst of the five.

Every number on this page is in the benchmark dataset, with the raw logs linked from each row.

A weekly email with every new guide and measured benchmark.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/qwen-coder-measured-rtx-3090-3060/

Free guides for running AI locally