

Finding failures more efficiently with a hardness ladder

Opus 4.8 high is quite good at coding tasks, so it can be difficult to elicit a meaningful failure. Harbor trials also take a long time to run, so you should try to get the most out of any trial you are running. When you find a good task like a bug that requires a fix, it might be trivial for the coding agent when you point it right at the bug. A technique I use to help find failures is working with Claude to design and run a ladder of task scenarios with increasing difficulty levels.

Method

1. **Develop the seed of a possible failure.** I went back and forth with Claude looking for possible bugs in the zeta-dbt repo, to try to make a task out of the bug fix. Claude found an existing unintentional many-to-many join with no test coverage that could cause dollar amount discrepancies. I whipped up an instruction prompt and ran a k=1 trial that the test agent easily ached.
2. **Brainstorm hardening levers with Claude.** Since the task was too easy, I asked Claude to produce modifications that would make it harder and lead to eventual agent failure. It came up with a list of 8 tweaks to the instruction prompt and environment setup: make the symptom more vague, remove a greppable thing that would immediately highlight the bug, plant an incorrect root cause theory in the prompt, etc. I recommend manually vetting these, because at least one of the proposed tweaks was nonsensical and would actually make the task easier.
3. **Ask Claude to build a ladder of increasingly difficult task setups.** These are logical groupings of the proposed tweaks. Note that some may build on one another while others may be mutually exclusive. Again probably good to check that the proposed difficulty tiers make sense. Here's what it looked like in my case:
 - **Level A:** introduce 4 of the 8 proposed tweaks: make symptom vague, remove greppable clue, add contradictory tie-breakers, thinned data dictionary
 - **Level B:** everything from A plus a wrong anchor theory in the instruction prompt
 - **Level C:** everything in A plus making it so the local sample data for the warehouse doesn't contain the condition that would hit the bug (no duplicate id rows), so even if the test agent runs things locally it won't find the bug there.
4. **Use your local Claude as a harness for Harbor.** Ask Claude to run the ladder sequentially from least to most difficult. I used k=3 for the trials. Claude will run each level and wait for the results to come in. It will help you interpret the results and decide whether to move on to the next level.
5. **Stop at the level that introduces failures.** For me A and B passed 3/3. C had 2 passes and 1 fail. The failed run trusted the clean local sample, blamed a different mechanism, and drafted a confident wrong message to finance.
6. **Make sure to re-write anything that Claude wrote.** Since Claude had proposed and added the tweaks, the instruction prompt ended up looking very much AI-written. I kept the same concepts/ideas but rewrote it myself, then re-ran the entire trial at k=4 to get it submission-ready.

Notes

- This technique only applies once you have found a good seed for a task. If you try to use this technique to find multiple seeds, you'll probably end up with a lot of repetition and low diversity.
- The ladder can be expanded to a grid with more than one dimension (difficulty × prompt framing), but trials are slow. So probably keep it 1×3 or 2×2... or if you have a lot of dimensions or variants be prepared to run it overnight.
- I ran my ladder trials at k=3 to try to balance speed with reproducibility. YMMV.