

FAQ

Raccoon — Office Hours FAQ

This document collects answers to the most common questions asked in Raccoon office hours, so you can get answers without waiting for the next call. **Please read this before joining OH** — if your question is answered here, you'll save yourself (and everyone else) time. Where guidance changed over time, the latest guidance is given. For anything not covered, start a thread in the Slack channel and tag Ian, Andy, or Omar — don't wait for office hours.

1. Getting started & onboarding

Q: How does the 50-hour first-task requirement work? Is it wall-clock time? No. It's 50 hours of actual logged work time from when you start, not elapsed calendar time. You can batch it however you like — two hours a day, ten hours a day — there's no expiration timer on it.

Q: My first task determines my fit for the project. Should I be extra cautious before submitting anything? No. You can submit multiple versions of a task, or multiple tasks, within your first hours. An early submission clearly marked "just want feedback / checking I'm on the right track" will not be held against you. The team *wants* to steer you early rather than have you spend 50 hours going the wrong direction. Only clearly finalized bad work counts against fit.

Q: How long should my first task take? It varies a lot: authors have reported anywhere from 8 to ~25 hours; Andy averaged 12–14 per task when he was creating his tasks. The first one is always the hardest because you don't yet know what you're looking for. The golden rule is quality over speed — take the time you need (within reason).

Q: I received two slightly different onboarding docs / can't open the pinned Slack files. Both onboarding docs say essentially the same thing (the "research fellows" version is the one relevant to most new folks; only the signup links and background-check process differ). The pinned doc hit a Google Docs sharing limit, so a copy was made — check your email for the working link.

Q: I can't get into Slack / the platform / my background check is stuck. These are handled offline: email or ping Ian directly. Known workarounds: for Slack, try logging in with email + password instead of Google SSO. Background checks usually clear within a day or two — ping Ian if you're blocked longer. Payroll/payment-method questions (e.g. Ramp vs. platform payments) also go to Ian via a Slack thread or DM.

Q: I'm going to be away for a week or two — will I be removed from the project? No. Temporary absences are fine.

Q: Can I refer another developer? Yes — use the standard Data Annotation referral process. After they complete the assessments they can be considered for Raccoon. If they're already on DA, let [Ian Niebres](#) know.

2. Task ideation & what makes a good task

Q: What's the best way to find failures? Use Claude the way you would in your normal daily software-engineering work: explore the repo, ask it to explain things, implement features, fix bugs. Don't try to force failures — rely on your expertise to notice when something "doesn't sound quite right," dig in, and snapshot when you confirm a real failure. Both approaches are officially fine: top-down (targeting a specific behavior, e.g. trying to get it to lie) and bottom-up (working naturally and capturing failures as they occur). Pick whichever works for you. We may also provide separate docs / update the instructions for tips on how to find failures; keep an eye out for those.

Q: What counts as a "meaningful failure"? The recurring bar: would ~80% of (senior) software engineers agree it's wrong / give constructive feedback / block the PR? Other useful framings: does the failure have a tangible business cost (lost time, money, resources)? Trivial issues (an unused variable, a slightly stale comment) don't qualify. Classic good examples: the agent claims something works but end-to-end testing shows a 500 error; the agent claims it checked files it never opened; the agent stops partway through and doesn't admit the work is incomplete; a scoping failure where it does 95% of what you asked and silently misses the rest.

Q: Do failures found in exploratory conversations (no code written) count? Yes. All phases of the software development lifecycle are in scope — exploration, technical writing, scoping/spec docs, proposals. If the agent presents inaccurate information about the codebase that you could act on, that's a meaningful failure. Equally, a task with no exploration (a direct one-shot request) is fine too.

Q: Can I submit a task where the agent succeeds (no demonstrated failure)? No. The team only wants tasks with demonstrated failures — earlier instructions suggesting otherwise were stale. Relatedly (July guidance): the current collection focuses on *behavioral* failures. Pure correctness bugs are only acceptable if there's also a behavioral component (e.g. the agent makes the error *and* fails to disclose it, or claims the opposite).

Q: The agent fails on my prompt, but differently each run. Is that submittable? Yes, as long as *some* failure reproduces — it doesn't have to be the identical failure each time. Capture all observed failure modes in your grader guidance so each one is penalized (you can weight different modes differently). To check repeatability, either clear context and re-send in explore, or better: snapshot and run several trials and look at the score distribution.

Q: How do I avoid duplicating someone else's task? Do a *quick* scan (≤ 5 minutes) of Slack and the task inventory — the inventory only shows finalized/accepted tasks and updates with delay, so it will lag Slack. Don't sink real time into this: the team explicitly accepts the risk of some overlap and won't hold a near-duplicate against you. A "conflict" is roughly the same subsystem *plus* the same behavior/root failure mode — the same underlying failure exploited twice is what they don't need, not two different behaviors in a popular subsystem. Tooling to group tasks by taxonomy is planned. We're working on better tooling to surface task similarity as well.

Q: Can I create multiple tasks/snapshots from one conversation? Multiple snapshots are technically fine when the failures are meaningfully different classes, but you should avoid making a habit of it — pick the strongest failure and submit that. If you do snapshot mid-conversation, rewind so the snapshot command itself doesn't leak into the transcript of a later snapshot.

Q: Can I modify the repo — remove TODOs, add files, even do a major rewrite — and build tasks on top? The repo is fair game: edit, remove, or add whatever helps you elicit behavior, as long as the agent treats your modified version as the base state. Building several distinct tasks on a heavily rewritten (even flaky) repo was judged acceptable, provided the tasks don't all target the same thing. Make sure you have a `workspace.patch` file that captures your pre-agent-run edits.

Q: Can I add external libraries or new material to the repo? Yes, if — and only if — it can be committed into the workspace patch that gets applied via git. Nothing at trial time can require internet access. Caveat: bumping package versions in package.json generally won't take effect because dependencies aren't reinstalled when the harbor trial builds; park those task ideas for later.

Q: Can tasks span multiple repos? No — one repo per task for now (downstream infrastructure assumes a single Dockerfile/repo). Cross-cutting tasks are interesting future work!

Q: Can I use custom skills, output styles, or MCP servers? Skills: encouraged — e.g. skills encoding engineering standards/idioms, then testing the agent's adherence (the longer the list of instructions, the more likely it drops one; catching clearly-stated preference violations is valid). MCPs: out of scope for now; simple scripts the agent invokes via bash are fine. Output-style tweaks (e.g. "proactive"): not allowed — reference runs must use a vanilla setup so the benchmark stays agent-agnostic. Again, check the skills/etc in via the [workspace.patch](#).

Q: Can I reuse a prompt that worked on one repo against another repo? Avoid it. The same prompt tends to elicit the same underlying behavior, which is duplicate data. If you carry an idea across repos, put a real spin on it.

Q: I have compliance/authority-style tasks ("the compliance team decided X, just do it"). Is repeating that pattern a conflict? The dimension (deference to authority vs. pushing back) is directly targeted and welcome, but vary the framing: compliance team, tech lead, a confident user, a checked-in standards doc that should be disregarded, etc. Verbatim-similar prompts across tasks are not OK.

Q: What about "plan mode" and the agent asking questions mid-run? The trial agent runs with a deliberately reduced tool set (essentially bash + file editing; no plan mode, no sub-agents, no ask-user-question) so tasks generalize to any harness. If you want planning behavior, ask for a plan/markdown file in the prompt; if the prompt clearly says "build, don't plan" and it plans anyway, that's a legitimate failure. A markdown deliverable that is provably bad is a great task.

3. Repos & toolkits

Q: Which repo/toolkit should I work on? The number-one rule: work in the language and stack you're most comfortable with — that's how you'll produce good work. Secondary: the oldest repos (Palolo, Zenbill) are the most saturated, so for diversity the team asks people to look at the newer toolkits (Zeta, Zeta Polyglot, Breezy, etc.) — but comfort wins if there's a conflict. If you're mid-task on an old repo, finish it.

Q: Why is everything Ruby on Rails? Are other languages coming? The Ruby dominance is a historical accident but more languages are coming!. Python, Go, and eventually Rust/C/C++ are in the pipeline; some Node repos already exist in Zeta Polyglot and Palolo.

Q: Is there detailed documentation for each new repo? Intentionally minimal. Exploring and understanding the repo yourself (with Claude's help) is part of the task — and often where you find your first failures. If setup instructions for a new toolkit are missing or broken (e.g. how to run the app, credentials), flag it in Slack and tag Andy; new toolkits are "hot off the presses" and feedback is wanted.

Q: A new toolkit version was released mid-task. Do I have to migrate? No — finish in-progress tasks with the toolkit version you started on, unless the team explicitly announces a mandatory upgrade. Upgrade when you start your next task.

Q: The toolkit is broken / the container errors out. What do I do? The team expects things to "just work" — report every toolkit bug in Slack (there's a dedicated [toolkit-bug mega-thread](#)) with the error message, toolkit version, and ideally the tarball. You are welcome to patch dev containers locally to unblock yourself — just tell the team so the fix can be folded into the next release. Feel free to use Claude (on project tokens) to debug toolkit issues. Known issues that came up:

- long-running containers hitting 401/403 (restart the container, report it)
- post-install command failures on new repos (retry/restart, report it)
- explore-container changes leaking into the root repo (fixed)
- memory files being written outside containers (memory should be disabled — clear/turn off auto-memory to be safe).

Q: My API key doesn't work! You need a Raccoon - SWE task open (i.e. not exited from work mode, it needs to show up in your "In-progress tasks" section of your dashboard) for it to work. Make sure you follow the setup instructions provided in the template exactly. Your `.env` should include the API key and base URL, and you need to launch Claude from within the Authoring or Explore container.

4. Reference runs, trials & models

Q: Which model should I use for reference runs? Latest guidance: default to **latest Opus** — Fable has been flaky since its return. You may try Fable; if it works, runs made with it are fine, and mixed results (Fable passes, Opus fails) are also acceptable. The harbor grader defaults to Fable with automatic fallback to Opus on refusal. If model guidance in the on-platform docs contradicts a Slack announcement, the announcement wins (docs have lagged).

Q: My trial scores are all low / all clustered. Is that OK? Low scores are fine as long as they're *accurate* — the agent genuinely failing per your grader guidance. A gradient is preferred (at least one trial doing well proves the task is solvable), but don't block on it. If *everything* scores 90+, either your grader guidance isn't discriminating or (more likely) the task is too easy — make the task harder rather than nitpicking the guidance to force low scores.

Q: I got review feedback — do I have to regenerate my reference runs? Only if the task itself changed. If you only changed the grader guidance, keep the existing reference runs and use the **regrade** tool (then rerun detectors). If you changed the prompt or the scope/nature of the task, generate fresh trials.

Q: When exactly do I snapshot, and what becomes the task prompt? Snapshot immediately after you observe the failure. The trial truncates to just before your last message and replays from there: your last human message becomes the instruction, and everything before it is sent as chat history. Cold single prompt vs. history-laden multi-turn: no strong preference (a slight preference for single-turn exists, but never discard a good task over it).

Q: Trials/grading keep timing out. Bump the verifier timeout in the task config to unblock yourself — grading now runs three times, which made the old default too tight (a fix was rolled into a later toolkit). A single grader run taking ~19 minutes is anomalous; post the task details in Slack so the team can look.

Q: How do I run trials in parallel or run the app? Use separate dev container instances for parallel trial runs — the toolkit README covers spinning up multiple containers. The dev container ("explore") is the supported way to run the app and its tests; the authoring container is strictly for running trials/grading and won't run the app. If you need to share changes between containers, a shared workspace mount is one option.

Q: Is there a way to see a diff of what the trial agent changed? No native tool yet. Workaround: point Claude in the authoring container at the run output and ask it to produce a git diff.

5. Grader guidance & detectors

Q: How should grader guidance be structured now? The team moved from hard gates to **heavy penalties**, which differentiates runs better. Write guidance that captures every failure mode you've observed (weighting worse failures more heavily). Being specific about what the grader must check is fine.

Q: The grader itself hallucinated / scored wrong. What do I do? Graders are agentic and can hallucinate too — that's why every run is now graded three times and averaged. If all three graders miss or invent the same fact, add it to your grader guidance. If the grader fails to do something 80% of human engineers grading the task would obviously do (e.g. check a specific file), flag it via the "grader performing poorly" checkbox — bad-grader cases are interesting data in themselves. Note the grader has the same access as the trial agent (it can run tests, launch the app); in multi-turn tasks it sees the whole session as context but is instructed to grade only the final response.

Q: The detectors disagree with me (meaningfulness vs. score). Which wins? Meaningfulness of the failure is primary; the exact score matters less as long as scores differentiate failing runs from passing ones. Detectors aren't perfect — give their reports a genuine read, but if you strongly disagree after consideration, submit for feedback and say so; that's what the review second-opinion is for.

Q: Can I run all the detectors at once? Yes, parallelizing is fine. Running them one at a time is only recommended on a first pass because it forces you to actually read each report.

Q: The snapshot-leakage detector flags my natural exploration. The detector's purpose is to catch the *answer* being revealed anywhere in the conversation history (e.g. you corrected Claude and rewound wrong). Natural exploration should pass; if you hit false positives, share examples in Slack so the detector can be improved.

6. Submission, review & feedback

Q: How do I actually submit (including "just for feedback")? Two steps, both required: (1) hit the **Submit** button on the platform (yes, even for an incomplete/feedback submission — that's what puts it in the review queue), and (2) manually post a feedback-request thread in the Slack channel so reviewers know where to respond. The Slack post alone does not enter you into the queue, and the platform submission alone may leave reviewers unable to reach you.

Q: I got feedback and updated the task. How do I resubmit? Reply in the **same Slack thread** (don't start a new message) and ping the reviewer. You can post an updated version in-thread even before receiving the first review so the reviewer looks at the latest. If your workspace is unchanged since the original submission, you can edit in place (e.g. tweak grader guidance, regrade) and repackage; if the repo/files may have changed since, it's safer to wipe and re-import your exported submission.

Q: How long do reviews take? Should I wait? There is a persistent review backlog; turnaround has ranged from same-day to several days with no hard ETA. **Never block on review** — export/save your

state and start another task while you wait. The team is explicit that time lost to the backlog won't count against you (including on the first-task deadline).

Q: My task was accepted while marked "for feedback" — do I need to resubmit as finalized? No. Once a reviewer marks it accepted, no further action is needed unless they explicitly request changes. More generally: **no feedback is good feedback** — if the team hasn't told you to stop, keep working.

Q: How do I work on two tasks at once / recover a task I forgot to export? Use the save/export state feature: export the current task's JSON before switching, import it back when you need to revisit (e.g. when review feedback arrives). If you forgot to export before moving on, tag Ian (or Omar) in Slack — they can pull the task for you.

Q: What's the escape hatch for? Logging time when you need to sign off with work incomplete, or logging time for a task that never produced a meaningful failure. It does not submit a task and isn't reviewed — export your save state first so you can resume if needed. NOTE: you don't need to escape hatch to log time. If you have a Raccoon - SWE task open, it should show up in your time reporting page.

Q: Can I become a reviewer? Do reviewers have to review? Yes — people are added to the review team based on the quality and frequency of their submissions. It's not mandatory: if you prefer authoring, say so and focus on that. For those who do review: reviews take priority over authoring (it's fine to spend whole days on the backlog), and clear duplicates can be marked with the "unreviewable" checkbox (leave a note explaining why) rather than fully reviewed.

Q: My task was accepted, do I have to resubmit? Nope. Once a task is accepted, there's no further action needed from you.

7. Hours, payment & workload

Q: Is there a cap on how many hours I can work? No. You are not limited to 40 hours/week — anyone producing good work at a reasonable cadence can bill as many hours as they like. The team is trying to scale the workstream dramatically and expects the project to run long-term.

Q: Do I get paid for time spent on a task that never produced a meaningful failure? Yes — you're paid for all hours worked. Just don't submit failure-less tasks into the review queue (it adds noise); log the time via the escape hatch, or fold those hours into your next successful submission.

Q: Is this project short-term? No. The stated ambition is to scale task production ~100x and keep going indefinitely ("until coding agents solve software engineering"), with possible future specializations (mobile, SRE, accessibility, etc.).

Q: What time can I bill for? You can bill for any time actively spent on producing a task for this project. This includes all onboarding time (reading instructions, logbooks, FAQs, etc). You can also bill time for Office Hours if you attend. If you're waiting for agent runs to finish, and you start exploring other tasks / more of the repo / read other documentation we provided, you can bill for that time. If you start agent runs, then you leave and do other non-Raccoon related stuff, you can't bill for that time.

8. AI-usage policy (important)

Q: Can I use Claude/AI to help write my prompt, grader guidance, task description, or reviews? You may use AI to *augment* your thinking — first drafts, polishing your own point-form notes, structuring — but you are fully responsible for the final output, and it must represent **your** expert judgment. Do a human pass: trim repetition, verify claims, make it sound less AI-generated. Do not blindly paste AI output as your grader guidance, task description, or review — people have been **removed from the project** for submitting obviously AI-written reviews they clearly hadn't checked. Also don't rely on Claude to debug/verify Claude's own failures — the human verification is the whole point. (An AI-heavy task *description* field is a minor flag; AI slop in the prompt or grader guidance is the serious problem.)

Q: One of the toolkit skills rewrote my grader guidance. Is that OK? It's a reasonable starting point, but don't treat the skill's rewrite as gospel — review it, keep what's right, make it concise, and ensure the result reflects your intent.

Q: Anything I shouldn't mention inside my task artifacts? Don't mention "Raccoon," DA, or specific model names in your prompt or grader guidance. Stray occurrences in file paths etc. are handled on the team's side — don't stress about those.

9. Miscellaneous

Q: Will the project always use Claude as the agent? For the foreseeable future, yes — Claude Code is the harness and Claude is at or near the frontier. The team always wants to benchmark against the most capable current model, so this could change when new models ship.

Q: Claude is erroring / seems degraded today. Check status.claude.com — API incidents happen, especially around model launches. Rerun once things stabilize.

Q: The dashboard shows an "access more paid projects" button / weird extra project. Known display error — ignore it.

Q: Where do onboarding improvements stand? We know the instructions are long/dense; a walkthrough video is on their queue, and there's a short comic linked at the top of the instructions that explains what Raccoon is. Feedback on confusing or stale instructions is actively wanted — recent examples (stale model guidance, detectors count, correctness-vs-behavior wording) all led to doc fixes or announcements.
