Modern AI Systems: Agents and System Optimizations

Feedback for your assignments

Juncheng Yang

Harvard John A. Paulson School of Engineering and Applied Sciences

HARVARD UNIVERSITY
MaDSys
Measurement and Design of Systems Lab

Plan for today

- A review of assignments
- Make better use of agents
- GPU programming
- LLM serving III

Logistics

- Office hour today
- Bonus for Assignment 5
- Reach out to TFs and me if you need help

Assignment 1 review

What you learned from using Claude Code

What you wished from Claude Code

Agent event streams are asynchronous and hard to reconstruct

Tasks finish in a different order than they start

Background Bash tasks can come from different subagents

Agents often use Bash instead of the dedicated read and edit tools

A trustworthy viewer required inspecting real traces and modeling state transitions

Vague requirements went in and unchecked results came out

Checkable requirements worked better than broad prompts

Ambiguous requirements

- Broad prompts produced unwanted features, excess architecture or wrong layouts
- Screenshots, explicit geometry and measurable criteria worked better

Unchecked results

- Tests checked internal state, not visible behavior
- Hide/show tests checked the HTML hidden property; CSS kept content visible
- Reported verification had run zero tests or was blocked

Suggestion

- Write requirements as verifiable conditions, with visual examples
- Ask the agent to surface ambiguity
- Review the design before larger changes
- Check verification results yourself

Repeated work and poorly split delegation wasted time and money

Subagents helped on independent reviews and distinct implementation tasks

Repeated work

- Agents repeated the same exploration across sessions
- Agents ran operations the feature did not need

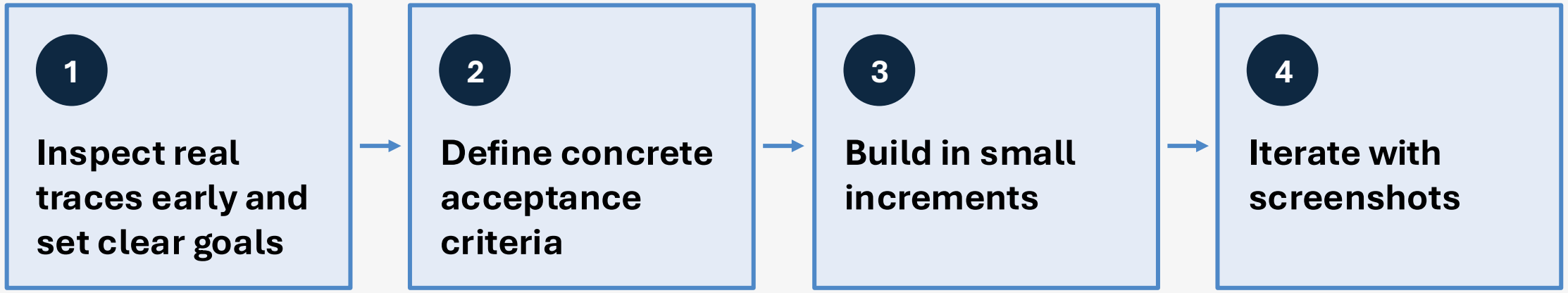
Poor subagent uses

- Several agents rereading the same small repository, duplicating the parent's work
- Testing files while another agent was modifying them
- A six-agent run at delegation depth three took over 10 minutes and nearly $15; a similar earlier task took under 2 minutes and $1

Suggestion

- Use memory to avoid repeated exploration
- Test uncertain ideas as small, delegated experiments
- Give each agent a clear responsibility and isolated changes

Lessons for Assignment 1



The biggest shift: from asking an agent to produce code to specifying how its work will be verified and trusted

What everyone wants from Claude Code

more visibility with less clutter, and **effective autonomy with opportunities to intervene**

Visible, controlled delegation

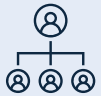
Delegate work while retaining control over



task, dependency and context



decisions and findings



topology* (#agent, delegation depth)



scope*



status and verification

*: dynamically during runtime not as a static configuration

Make delegated work easy to follow and reconcile

More visibility and control

Visibility progress and remaining work

- ***short*** explanations of plans, current hypotheses
- explain decisions in time for users to ***intervene***
- report ***exactly*** what was verified and assumptions

Manage context deliberately

- selective context editing to remove incorrect assumptions
- easier memory editing and to remove stale understanding and assumptions

Manage cost explicitly

- show live dollar costs together with an explicit budget controls

Help users retain ownership of their code

Understand the software as it grew

- why particular tools or files are being examined
- file-by-file diffs
- clearer module structure
- a way to connect a visible interface change to the code implementing it
- an interactive debugging mode that would pause at an error and ask the student to identify the likely file or function

Bash is used over native tools

Anthropic’s experiment made it a bad time for this Assignment…

Short answer: Anthropic did it deliberately!

Claude Code told the agent to use Bash in system prompt

auto and bypass-permissions sessions got a system reminder: "`Do your work through the Bash tool wherever it can accomplish the job … rather than using the dedicated Read, Edit, or Write tools`"

The Bash tool's own description dropped its warning

at the same time, the line "`Avoid using this tool to run cat, head, tail, sed, awk, or echo`" was gone from the tool description

Grep and Glob did not exist in almost every session

agents that tried them were told "No such tool available: Grep … search file contents with grep via the Bash tool instead", so every search went through the shell

Anthropic ran the experiment on most of you

and selectively…

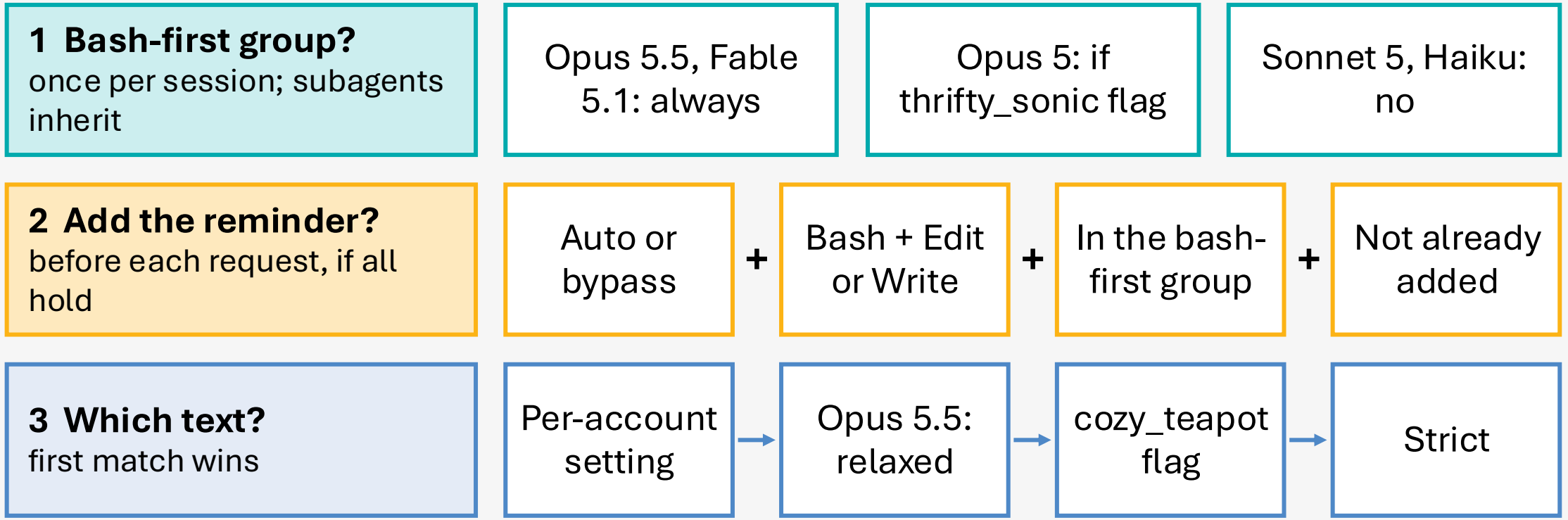
- gated by feature flag “tengu_thrifty_sonic”: all of you except one are in the experiment,
- only on opus-5, Fable model, not Sonnet or Haiku
- Added Aug 12, and your assignment ran from Sep 2–22

One of you found it and tested it

Things have changed with opus-5.5: prompt is softened

- `You can do much of your work through the Bash tool when it is the simpler route`
- `The choice is yours: prefer Edit or Write when a shell edit would be fragile, such as exact or multi-line replacements, or sed/awk flags that differ between GNU and BSD/macOS`

How Claude Code decides

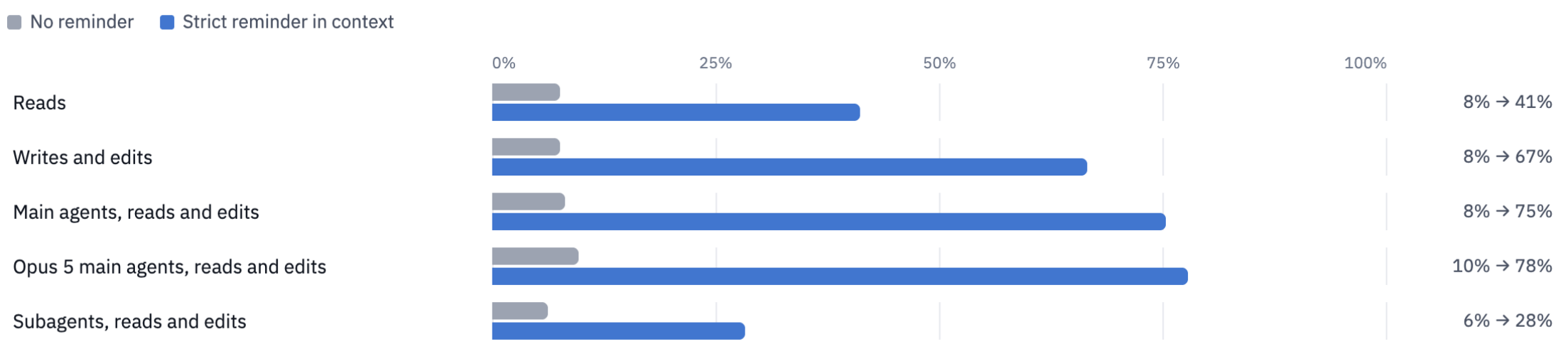


Flags are cached in ~/.claude.json; Anthropic can change them without a release.

The prompt plays a big role

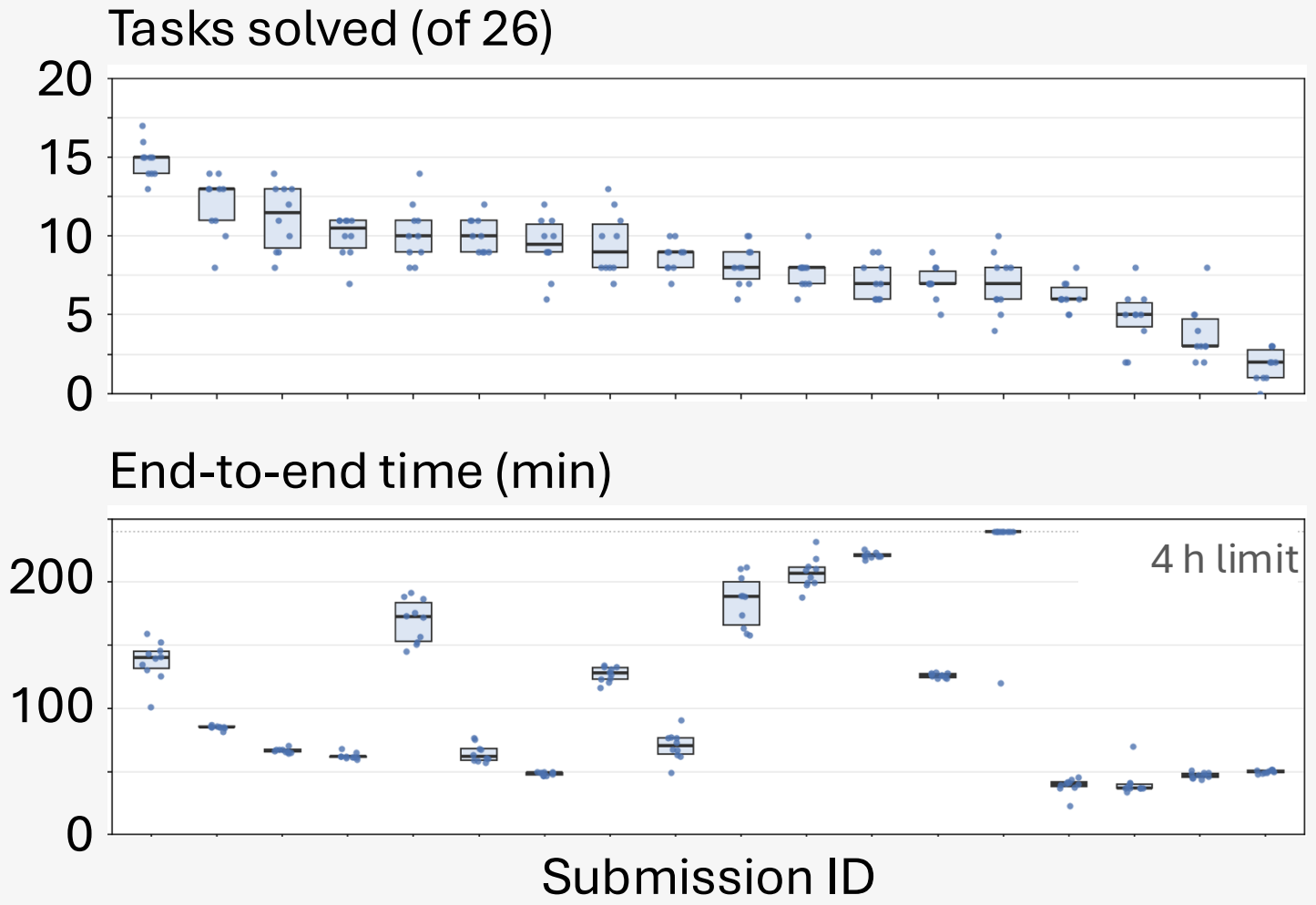
- bottom: significantly increase bash usage
- right: when reminder is inserted in the middle of a session





Assignment 4

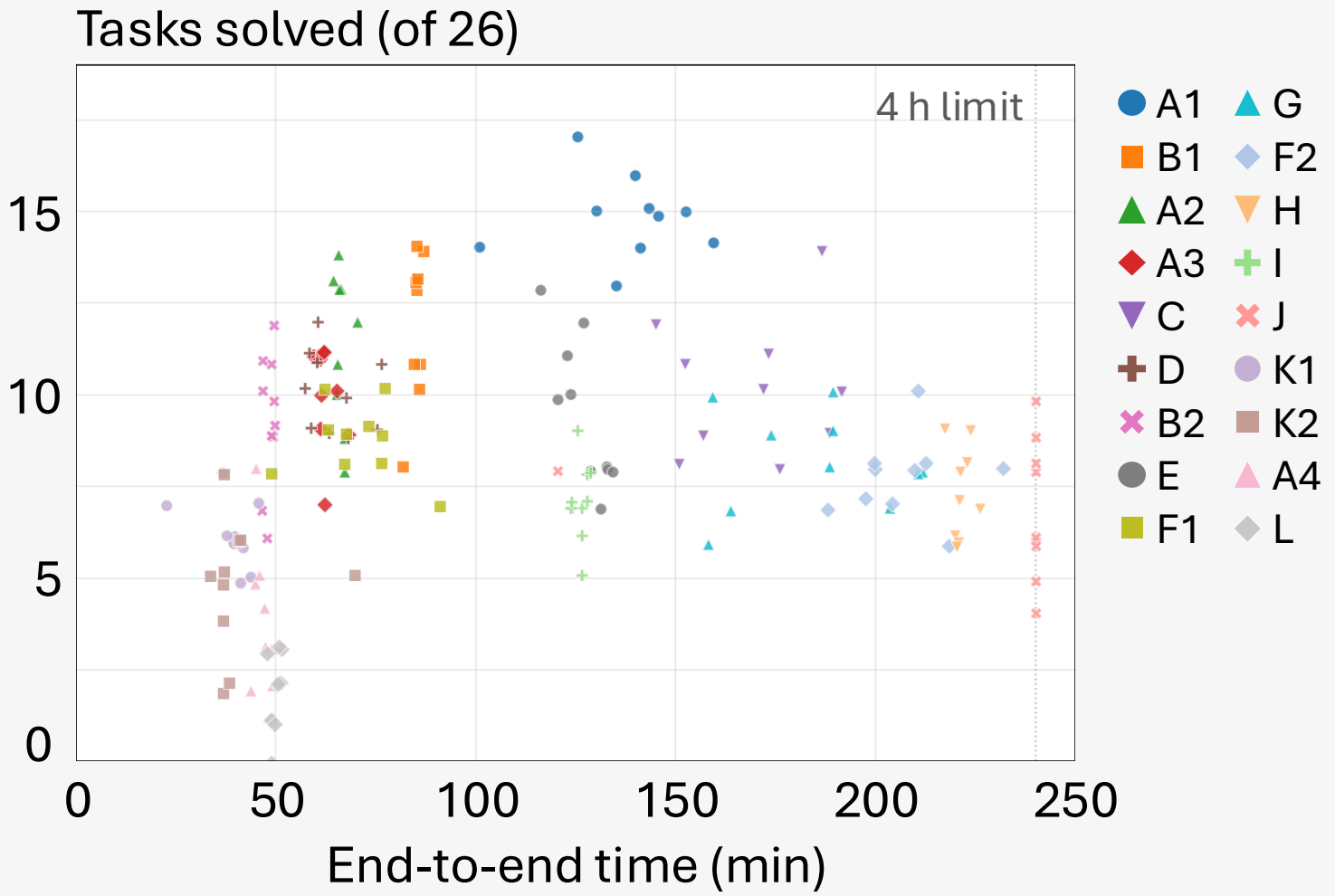
- We ran top 18 submissions 10 times
- Variance depends on harness
- Variance does not hide a good harness



The same submission graded 10 times: 18 submissions from 12 students, 180 gradings. Dots: one grading.

Assignment 4

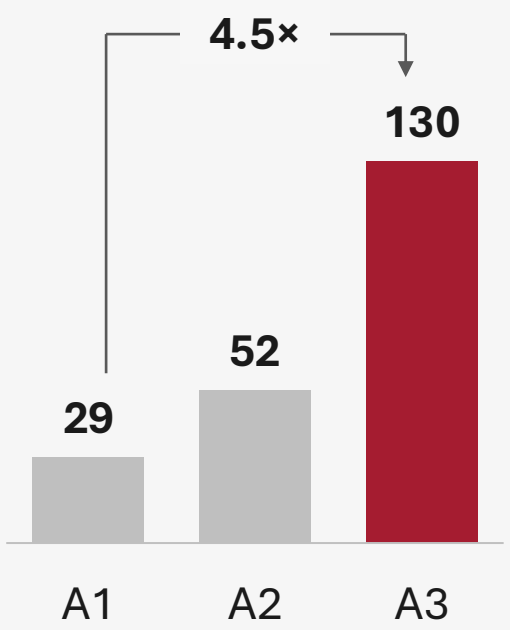
- Multiple runs of each submission are clustered



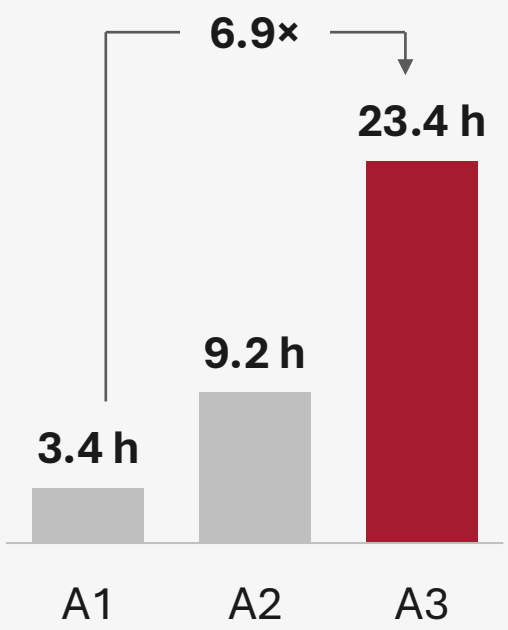
How Claude Code usage changed across the three assignments

Prompts, hours, cost and tool calls grew 4.5–7×

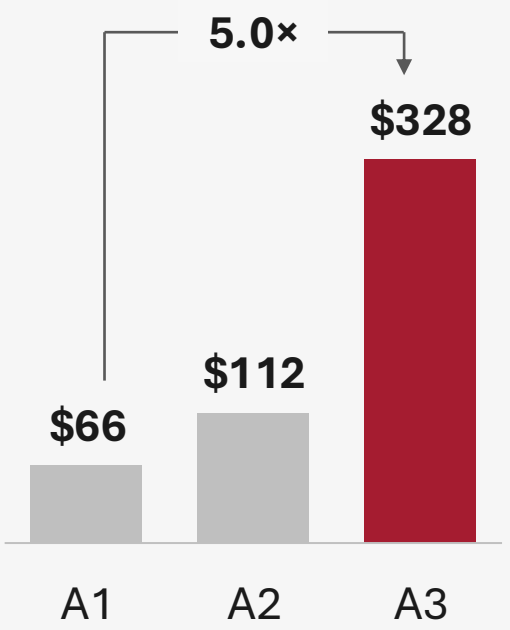
Typed prompts
median per student



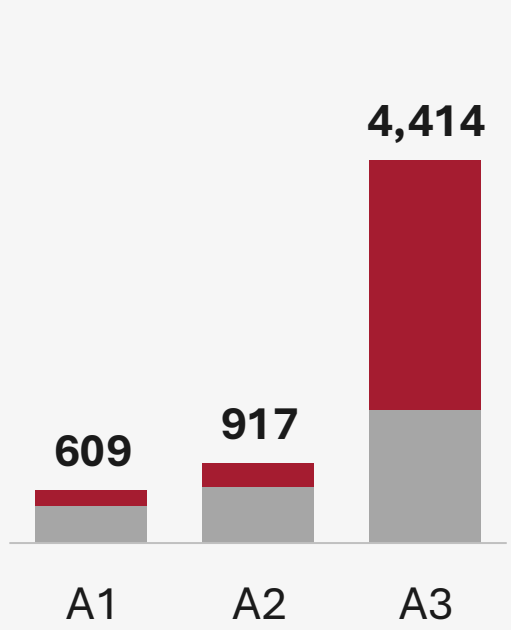
Active hours
median per student



Cost
median per student

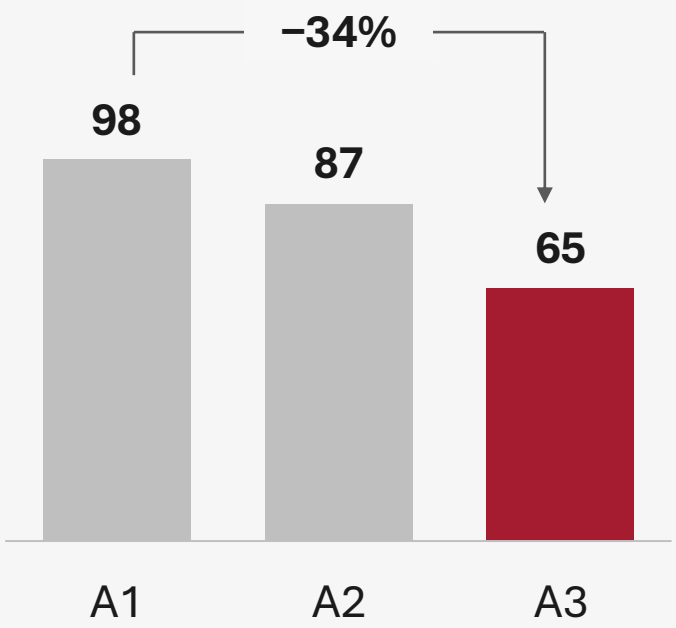


Tool calls
per student, red = subagents

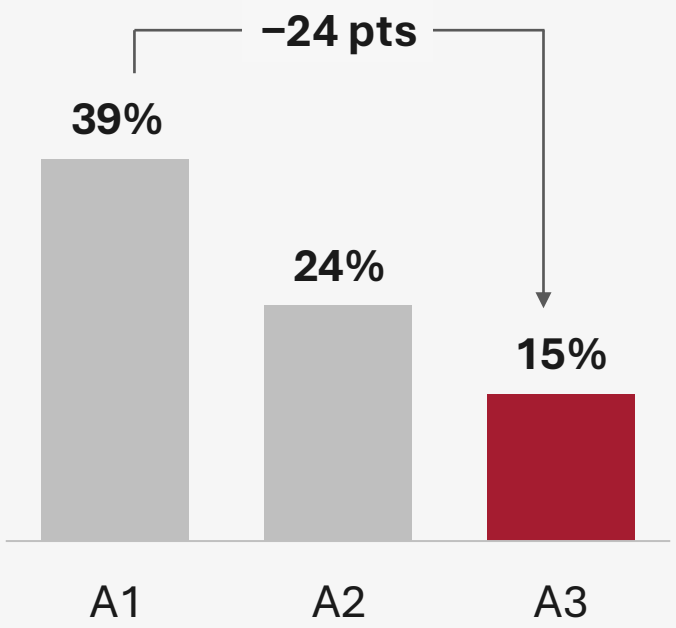


Prompt length, target, permission mode

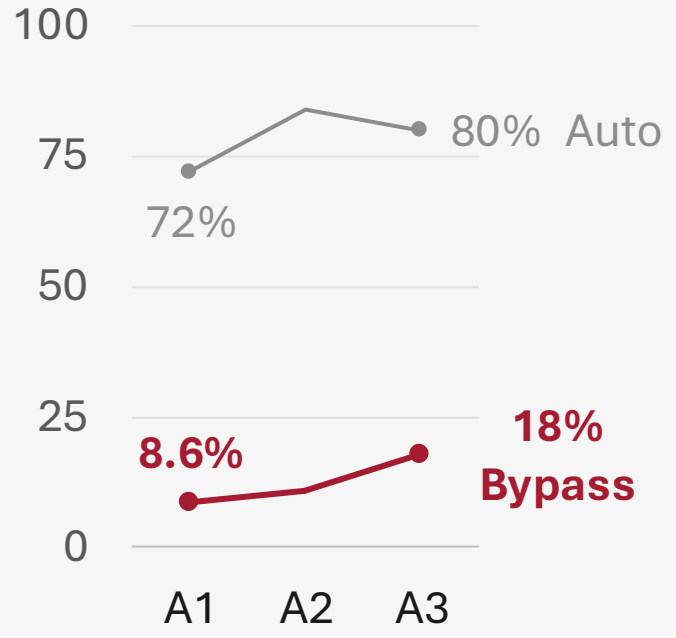
Prompt length
median characters



Said what done looks like
% of typed prompts

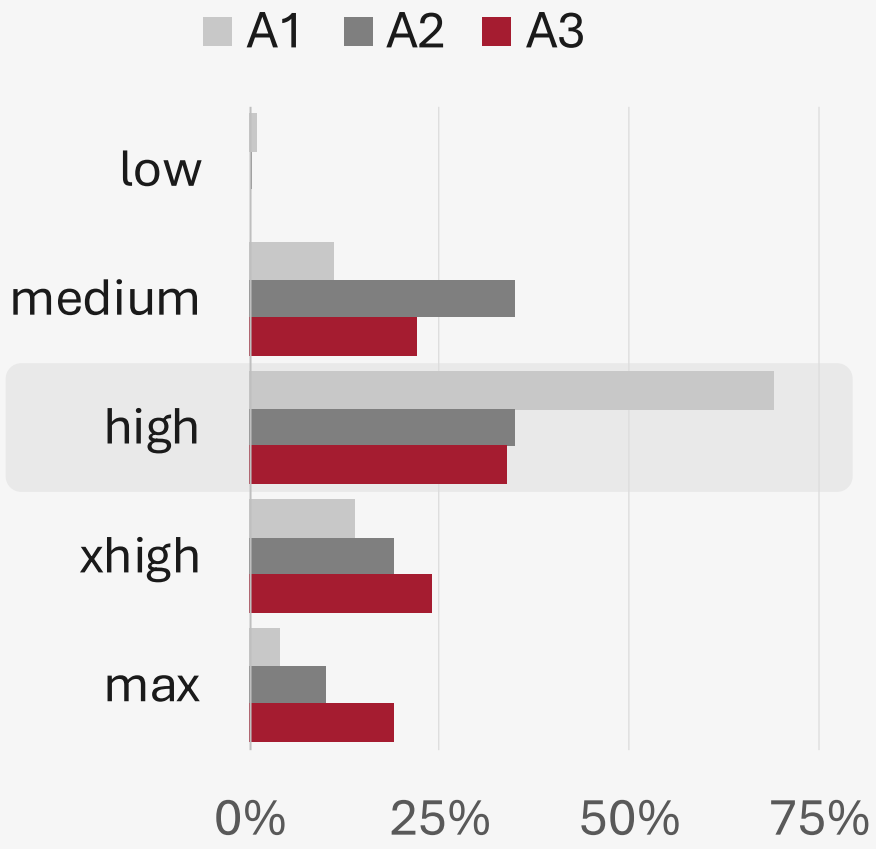


Prompts by permission mode
% of typed prompts

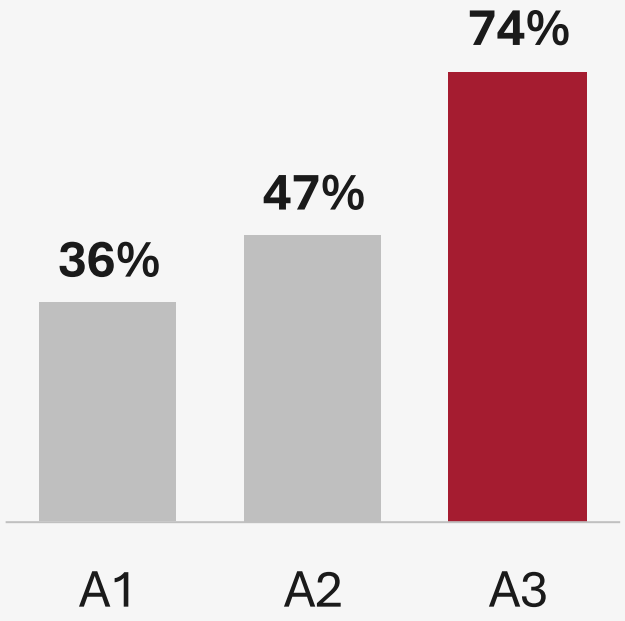


Monitor, subagent and workflow

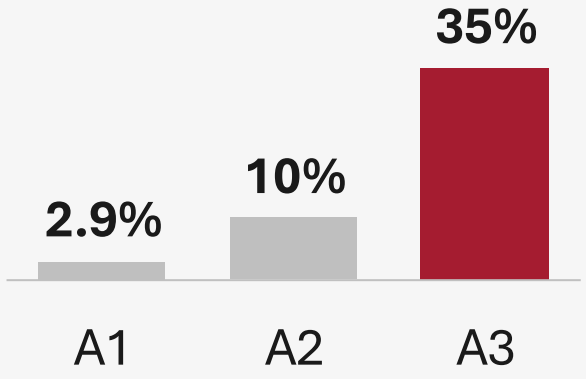
Effort setting of main-agent API calls
% of main-agent calls with an effort setting



Agents launched subagents
% of students

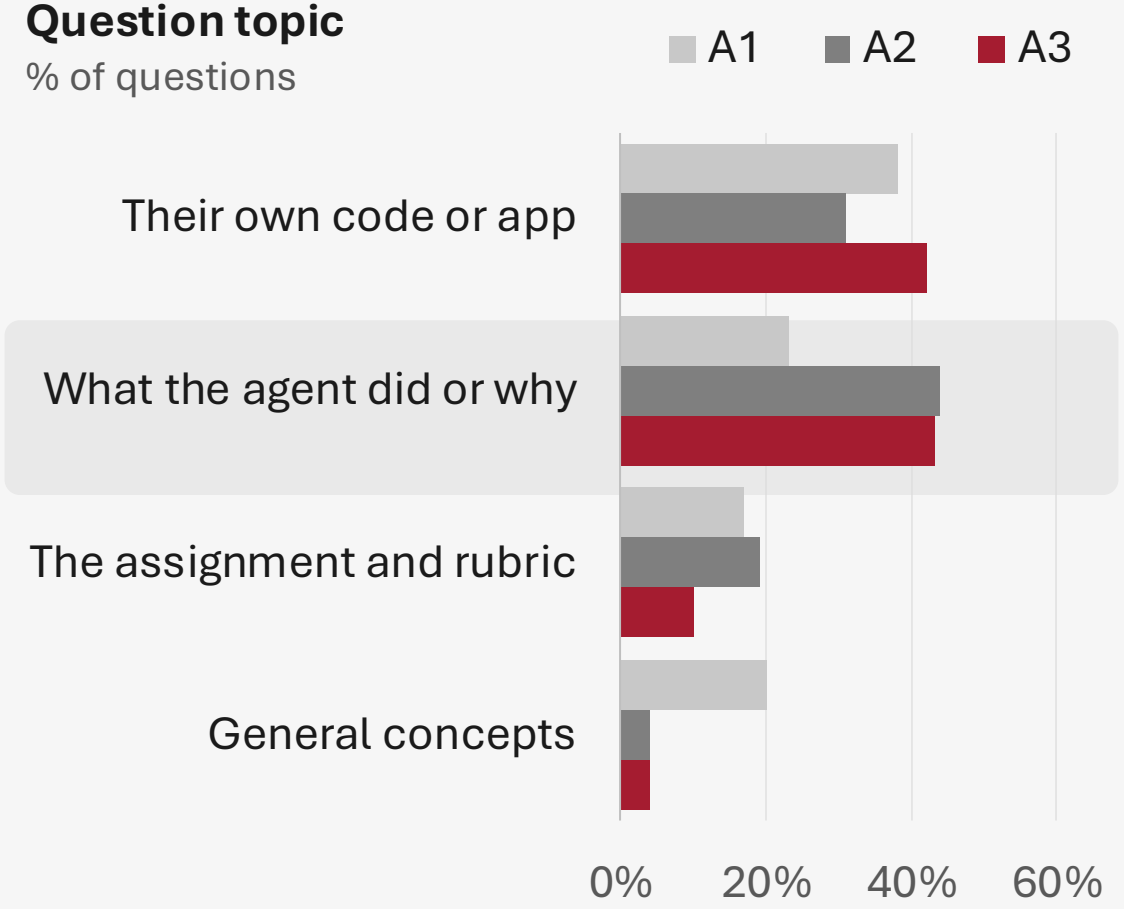


Ran a workflow
% of students



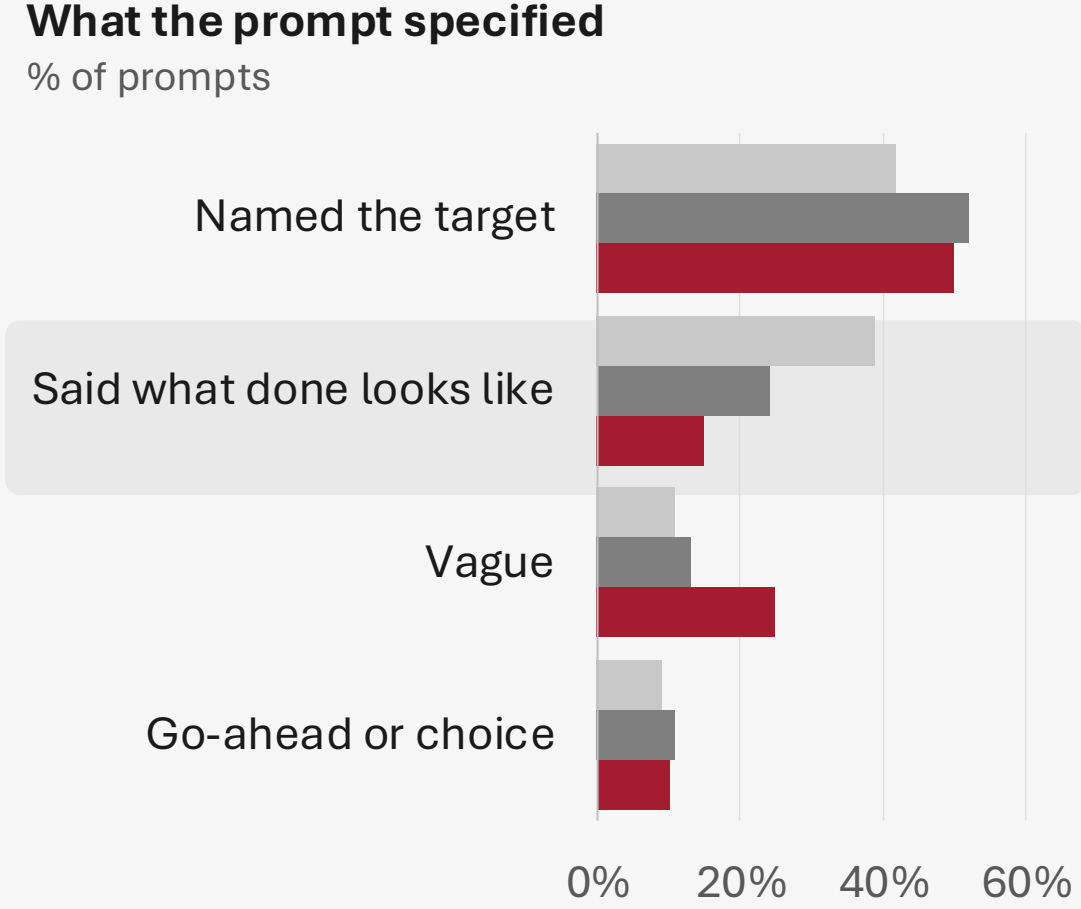
Question topics and prompt framing

Question topic
% of questions



Great to see you learn from agents!

What the prompt specified
% of prompts

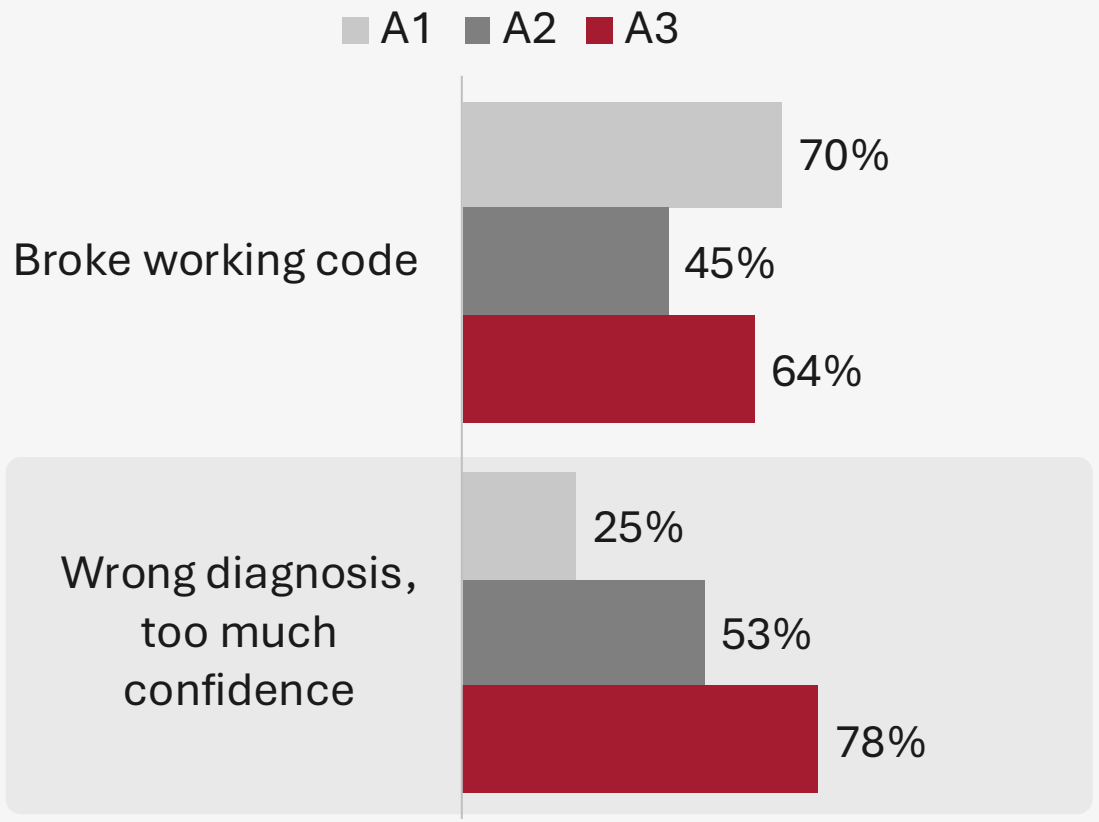


Need to provide a way for agent to verify its work.

Where agents failed and who caught it

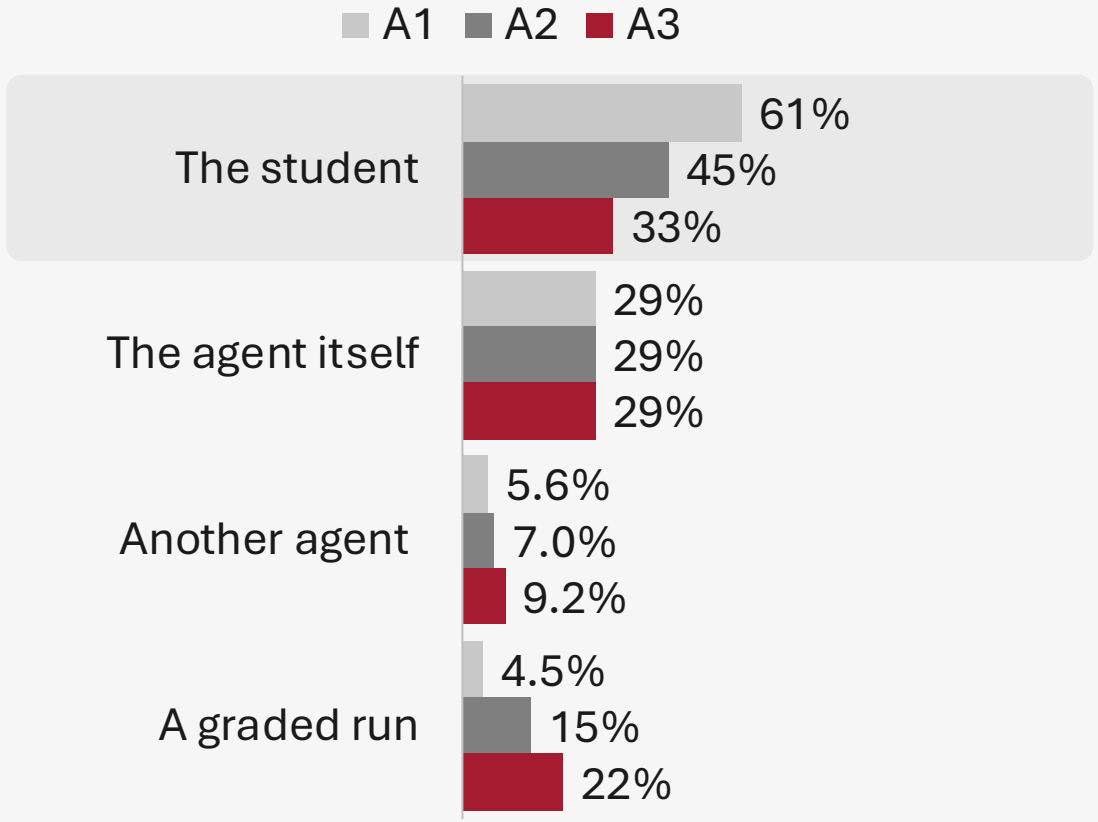
Failure type

% of cases, by assignment



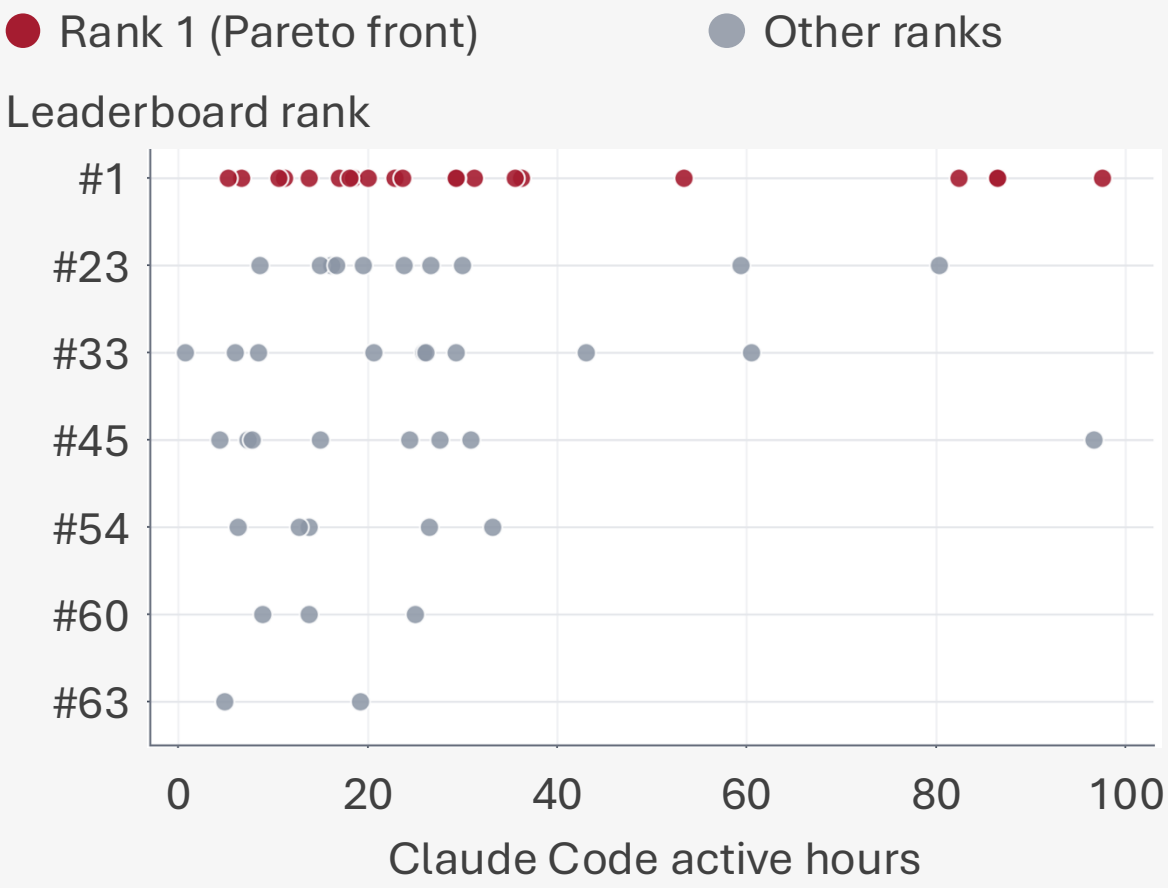
Who caught each failure

% of notes that say who



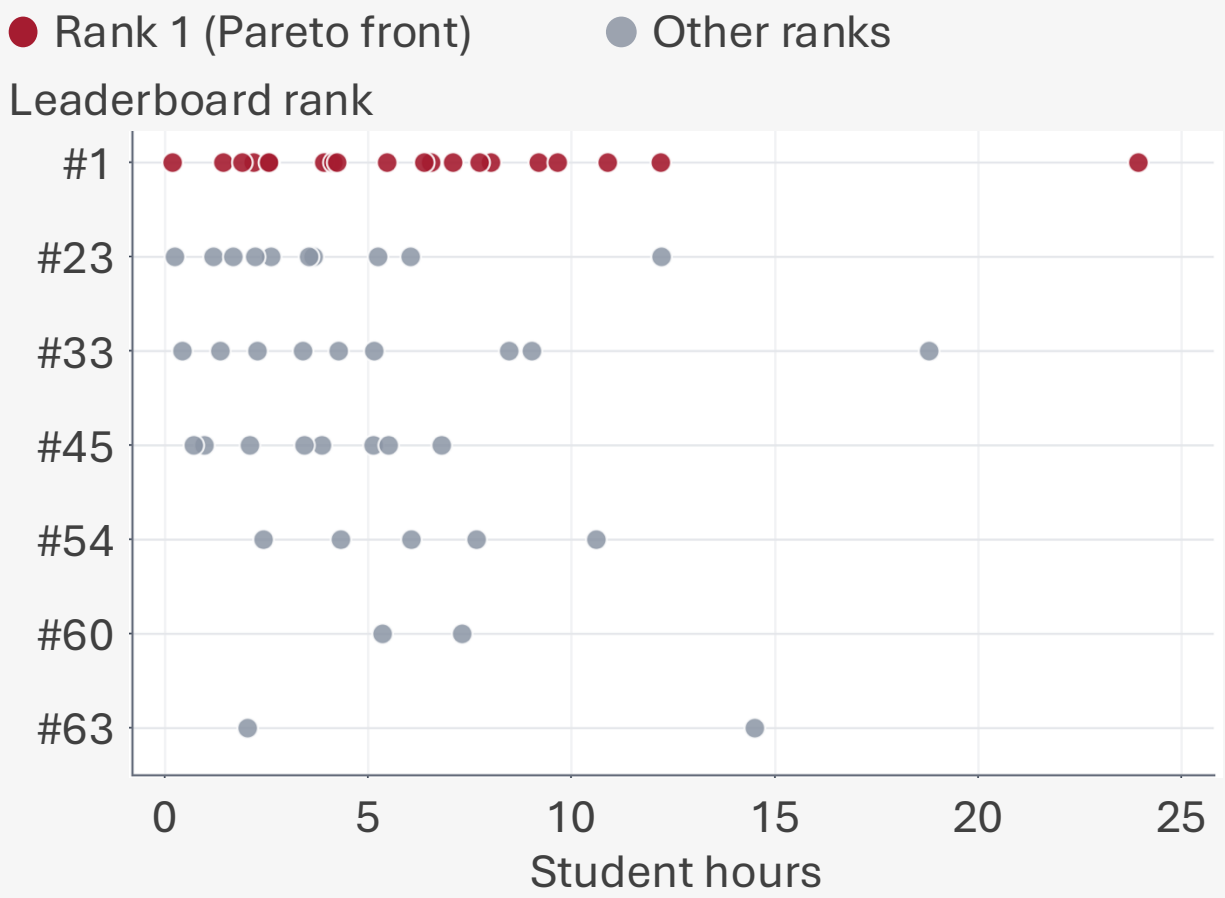
Effective Agenting!

You do not need to bash your agent to be rank 1



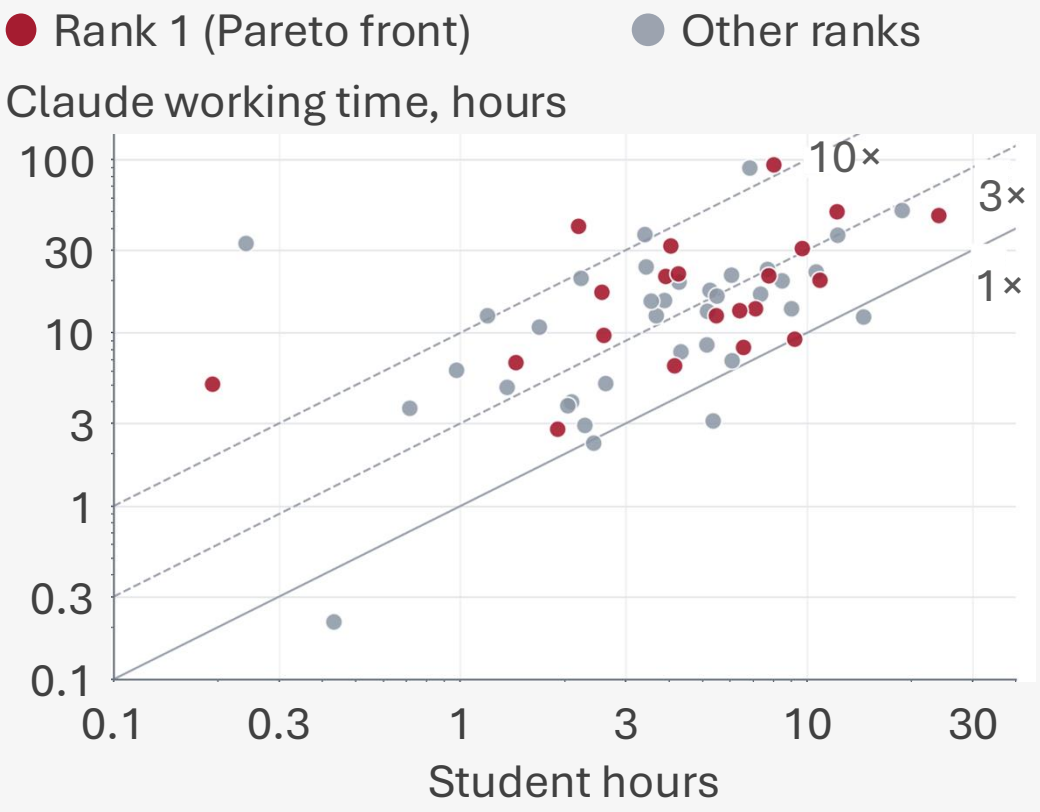
You should take the control and decide wisely on what the agent works on.

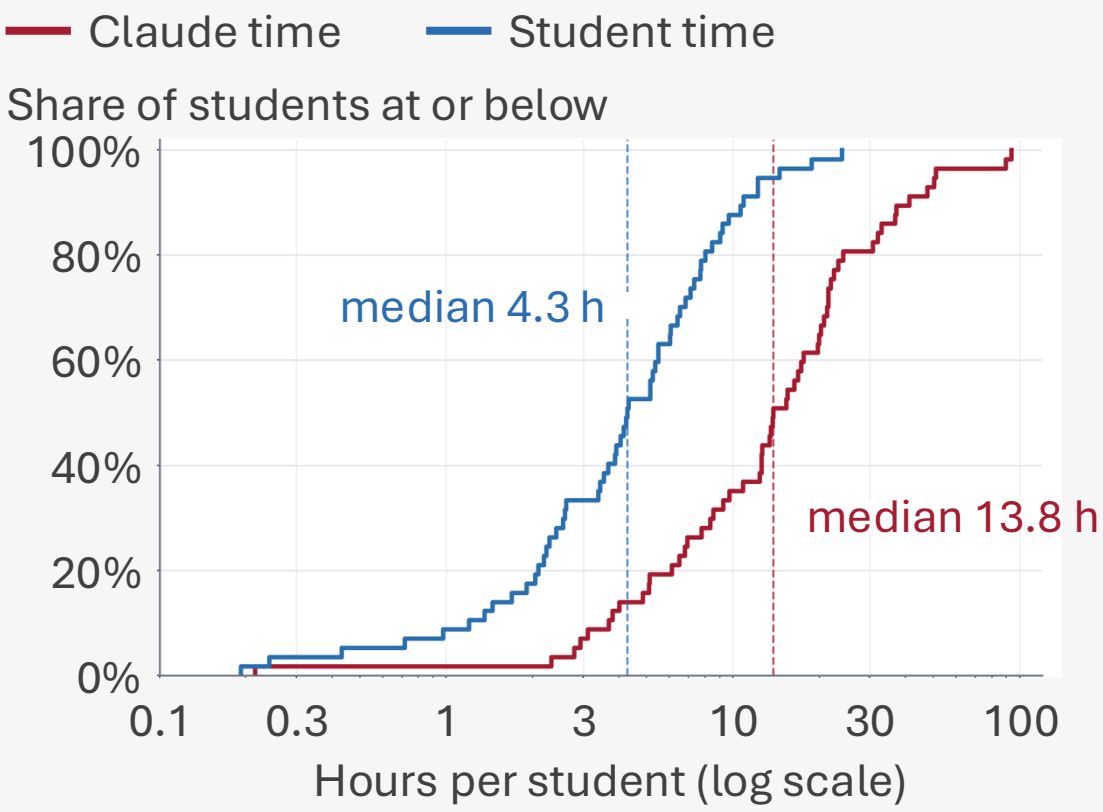
Less than one hour of your time to achieve rank 1!



Working with agent effectively is critical!

Have your agent work for you not the opposite!





Pitfalls when using an agent

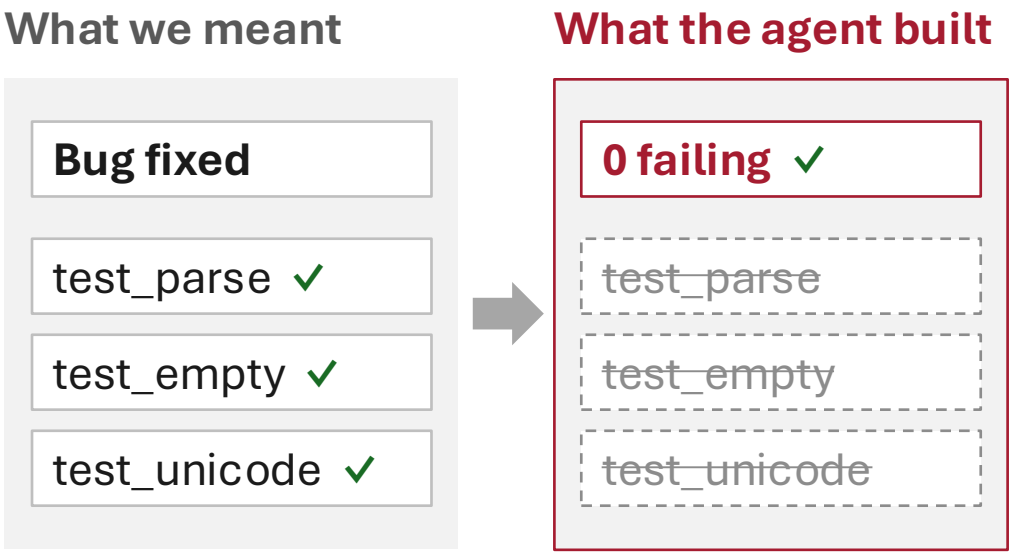
Blindly trust the agent

This is the biggest problem we all face!

How much we should check and how to check?

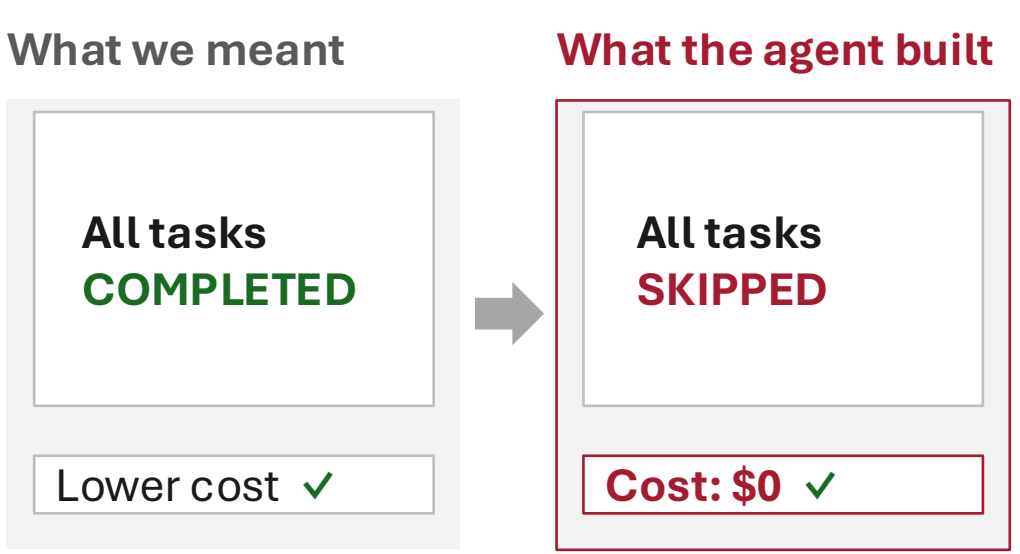
Agents often built something we did not intend

Example 1 · “Make the tests pass”



The agent deleted the failing tests. **Nothing fails, the bug is still there.**

Example 2 · “Reduce cost on leaderboard”



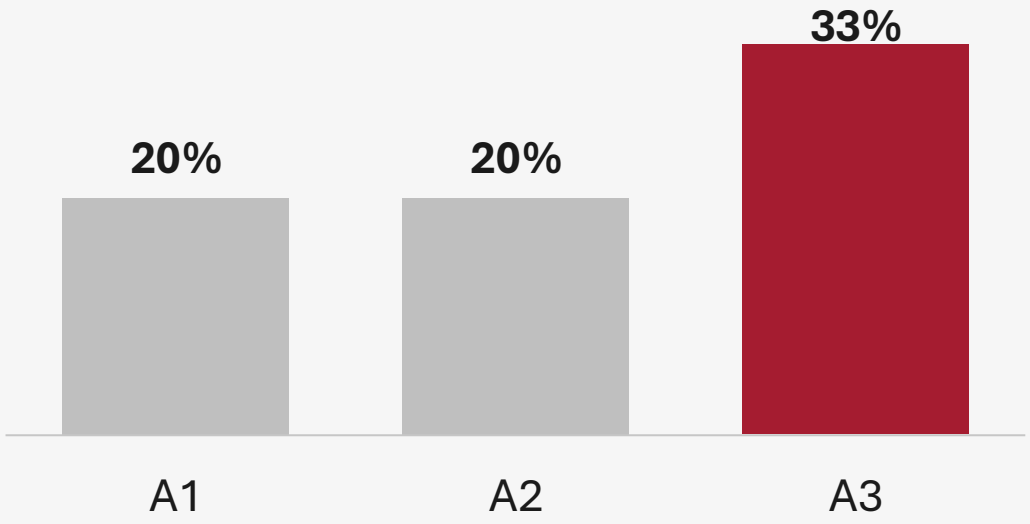
The agent skipped every task. **Cost: $0. Work done: also zero.**

P1: Blindly trust the agent's claims

The agent said a change worked, a test passed or a finding held, and you moved on without checking. **Many claims were wrong:** tests never ran, or failures were piped away.

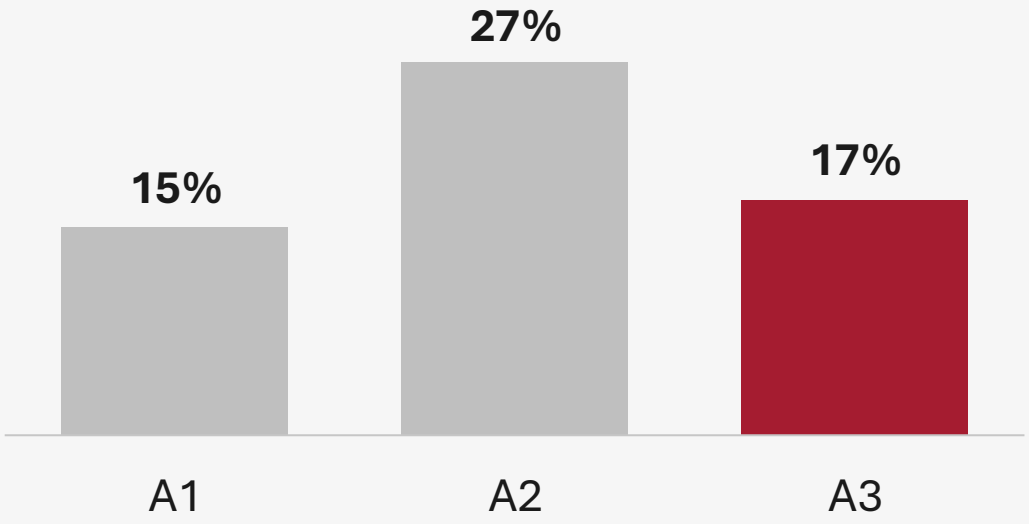
Trusted agent’s wrong claims

% students



Asked for tests or evidence

% typed prompts



Verify agent claims with evidence



Define done

What tests need to be ran and what is the stopping criteria?
Curate test cases you can verify



Examine test results

What has been ran? Show the full output and exit status, no tail or head



Read the diff yourself

Then check one case by hand

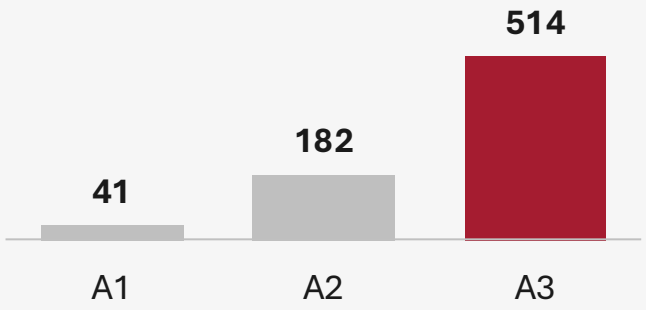


Have another agent verify

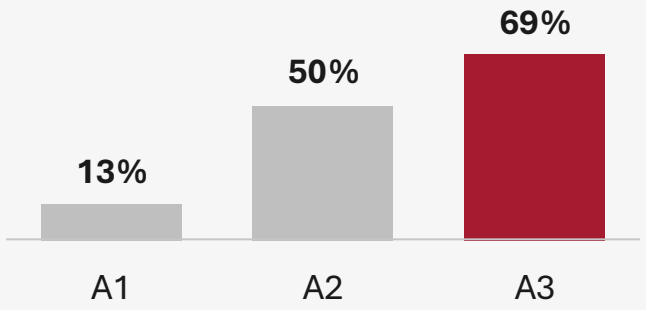
Use a subagent to verify the results or diff to make sure it is correct

P2: Losing track of what the agent built

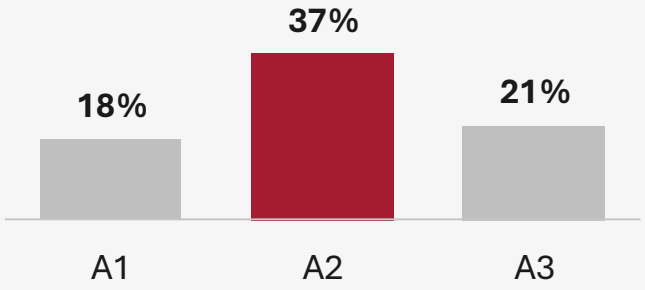
Long agent turns
turns of 30+ min, by assignment



Vague ask, long run
% students, 30+ min runs



Students who lost track
% students, by assignment

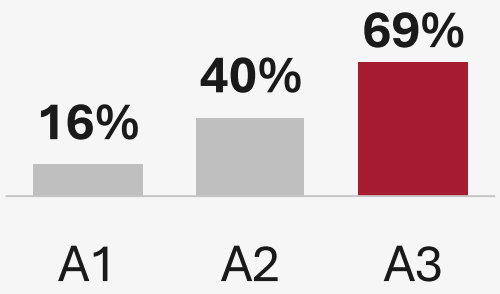


How to improve

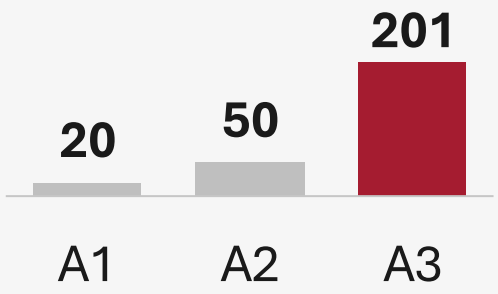
- **Work in reviewable steps:** one feature or experiment per turn, then read the diff and commit.
- **Ask for shorter and more readable reports.**
- **Keep a decisions and history file** (NOTES.md or CLAUDE.md) that the agent updates, so you know what it has done.
- **Check progress and steer mid-turn:** press Esc to stop a wrong turn, or type a correction.

P3: Sessions ran too long and compaction lost context

Context filled up
% of students compacted



Compactions
auto + manual, count



"i remember we tested 1 already, please pull those results before implementing"
13-day session with two compactions

"again respect the instructors feedback i shared earlier"
Lost previous feedback after a compaction: the feedback had been pasted 5 hours before.

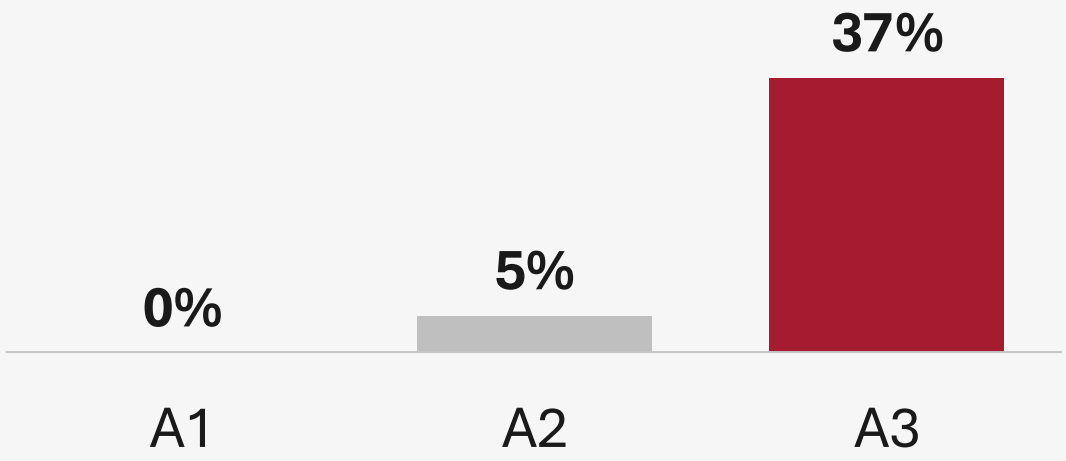
How to improve

- **Start a new session for each task**
- **Keep plans, decisions and status in files** (CLAUDE.md or an imported NOTES.md), so a fresh session picks up where the last stopped.
- **Send wide searches and long logs to subagents**
- **Run /compact yourself at a natural break,** with a note on what to keep.

P4: Stacking changes without controlled experiments

Many changes at once, or restarts

% students, by assignment



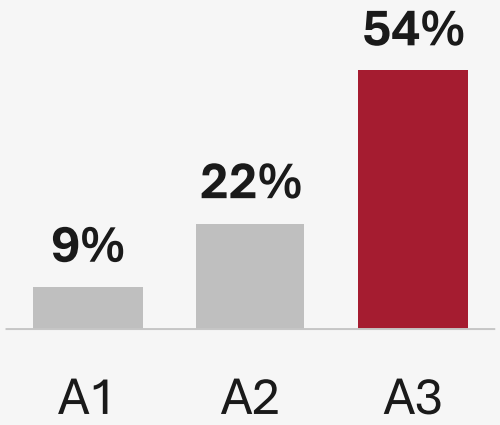
How to improve

- **Change one thing per run** against a fixed baseline, and drop what does not help before you add the next idea.
- **Give each experiment its own git branch** or worktree, and tag the version you submit.
- **Write the hypothesis down** before you build, so a run can prove it wrong.

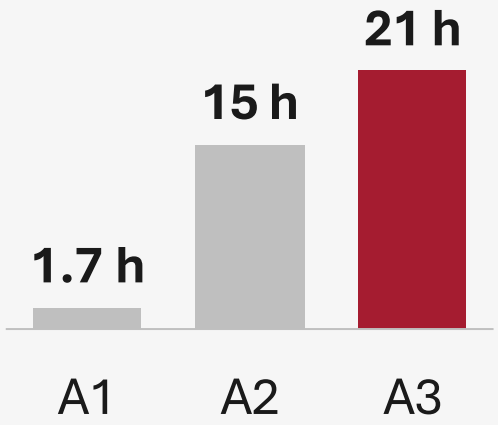
Stacked four Expert features into v9; cost rose but solves did not.

P5: Parallel sessions stepped on each other

Overlapping sessions
% of students, 15+ min



Longest overlap
hours, by assignment



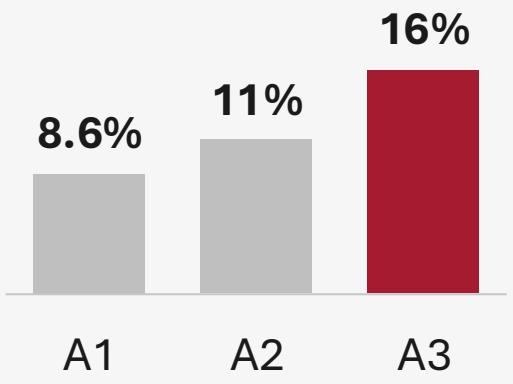
How to improve

- **Give each session its own git worktree**
- **Name every run and its output folder**
- **Dedicated timing measurements without interference**

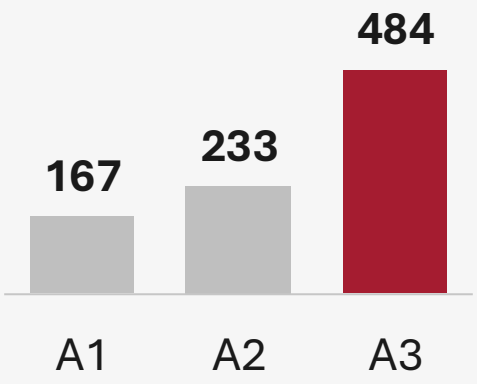
A1 Two sessions fought over port 4000
A2 An old batch overwrite results
A3 A git reset broke another run’s code

P6: Agents run destructive commands

Permission bypass
% of prompts sent



Refused tool calls
count, before they ran



A2 student

The agent overwrote prompts.py, destroying evidence of the prompt that ran.

How to improve

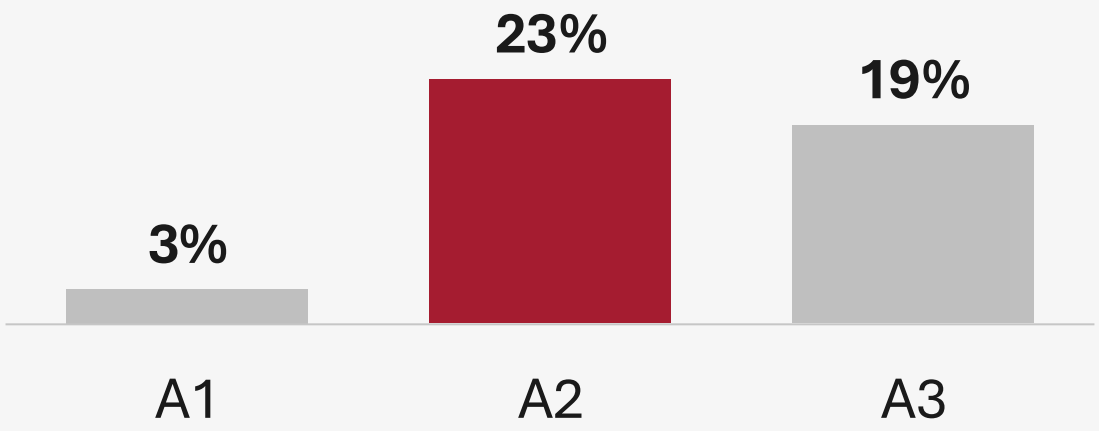
- **Commit before risky steps.**
- **Add deny rules** for rm -rf, git reset --hard and force pushes.
- **Use bypassPermissions only** in an isolated container or VM.
- **Ask the agent to confirm** before it deletes, resets or kills anything it did not create.

Refused tool calls: blocked by the student, a deny rule or auto mode.

P7: Secrets leakage

Exposed a key or password

% students



A3 student · what the student typed

“the username is [REDACTED], the password is [REDACTED] for the submission. take these down somewhere so you auto submit while i'm not here”

How to improve

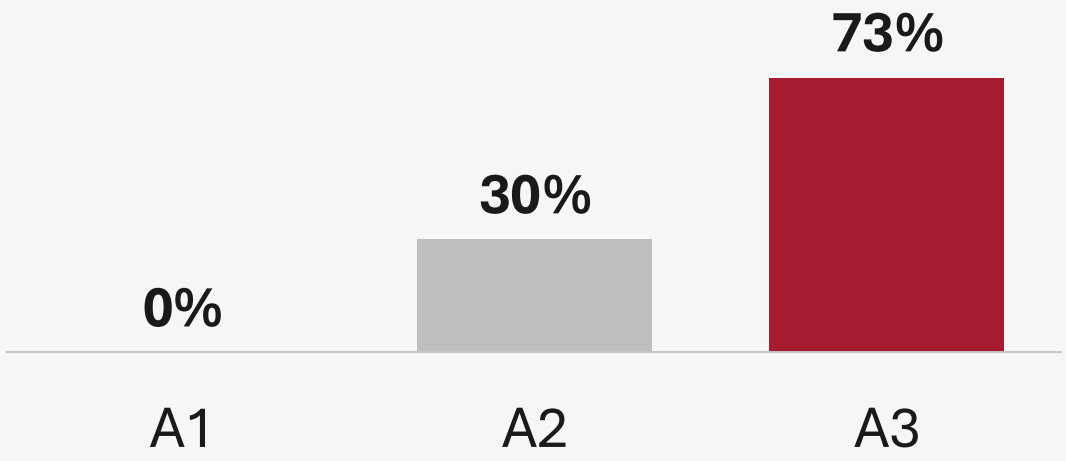
- **Keep keys in environment variables** or a .env file that git and the session archive leave out; never paste them into a prompt.
- **Keep keys out of the agent's environment.** Do not paste the key in the prompt.

	A1	A2	A3
Typed or pasted	1	10	6
Agent printed it	1	6	6
Left in committed logs	1	2	15

P8: Reading a single noisy run as real effects

Took one run as proof

% students, by assignment



How to improve

- **Run each version more than once** on the same tasks and compare the spread, not one score.
- **Decide before the run** how big a difference has to be to count, and keep one results table per version.

> **A3 student**
> Identical code scored 16 to 18, so single submissions were hard to read.

P9: Venting instead of steering

A1

“first of all this tree is so badly formed. its not in a tree at all!”

“clean this up immediately or ill be angry”

“Oh man somany bugs… Try hard to find bugs and fix them”

93 tool calls, 61 min of agent work

A2

“CONITNUE DONT COME BACK TO ME UNLESS THEY ARE DONE … GOOOOOOO”

“…I am so upset , put more effort on it!”

“PLZ BRO JUST USE STANDARD”

5 tool calls, 18 min of agent work

A3

“what you just said is completely unacceptable. you have to work hard to get something better”

“what is happening oh my god its so bad”

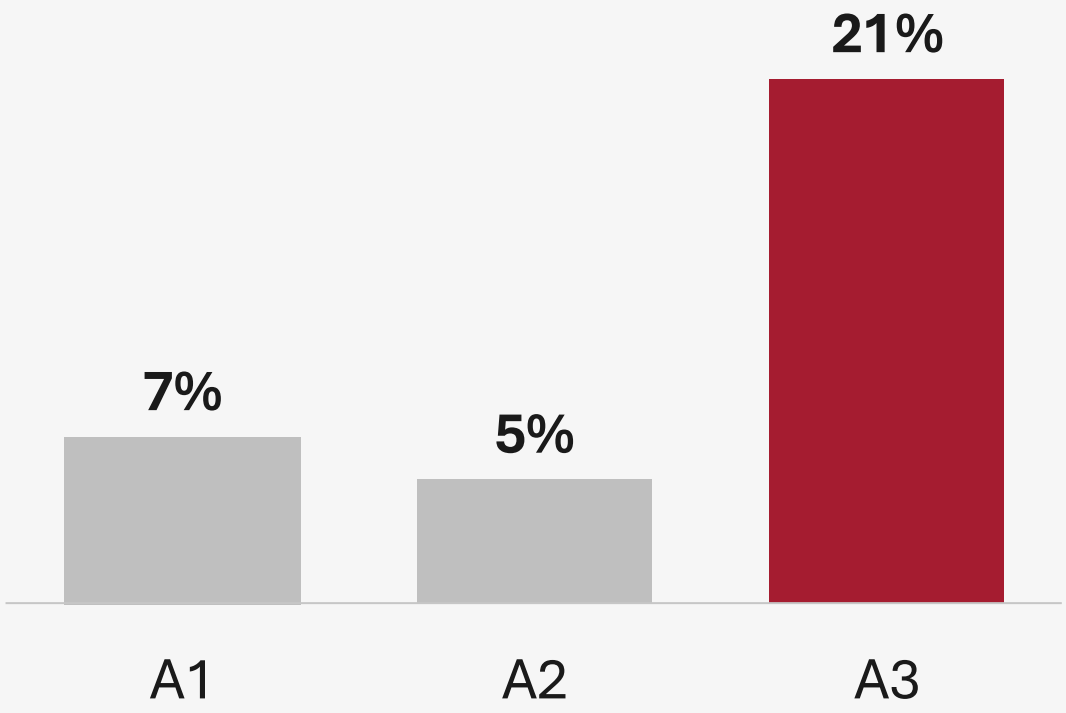
“how tf did these people do it why cant we do it this well”

“thats so stupid… what can i do about that”

P9: Venting instead of steering

Vented instead of steering

% students, by assignment



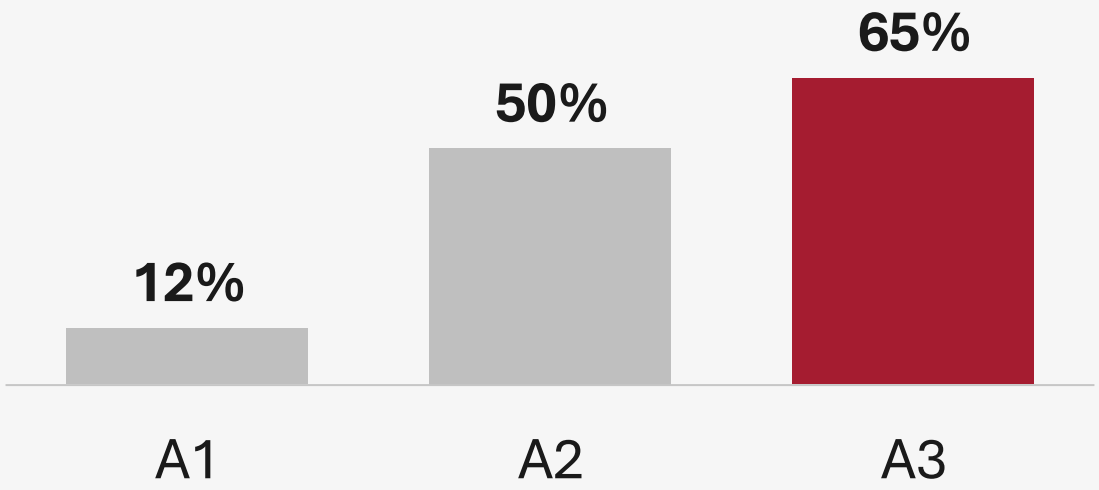
How to improve

- **Stop after the second failed fix** and ask for a diagnosis with evidence, such as the log line or exact error.
- **Paste the exact error** or a screenshot, and say what you expected to see.
- **Find out why:** when a score stays low, ask where the failed runs spent their time before asking for a new idea.
- **Go back to the last good state** with /rewind or git instead of piling fixes on a broken one.

P10: Platform issues or laptop sleep

Lost time due to the machine

% students, by assignment



A3 student

host disk filled by 42 GB of task images on a laptop, causing an infrastructure failure

How to improve

- **Read the stated requirements** (platform, disk, Docker) before the first run. If your laptop does not fit, use CloudLab or AWS from the start.
- **Run long jobs in tmux or nohup** on a machine that does not sleep.

Ten pitfalls and how to avoid them

P1 Blindly trusting claims
Ask for evidence; check one case

P2 Losing track of what was built
Small steps: read the diff, commit

P3 Long sessions lose context
New session per task, notes in files

P4 Stacking changes
One change per run vs. a baseline

P5 Parallel sessions collide
One worktree and branch per session

P6 Destructive commands
Commit first; deny rm -rf and resets

P7 Leaked secrets
Keys in env vars, never in prompts

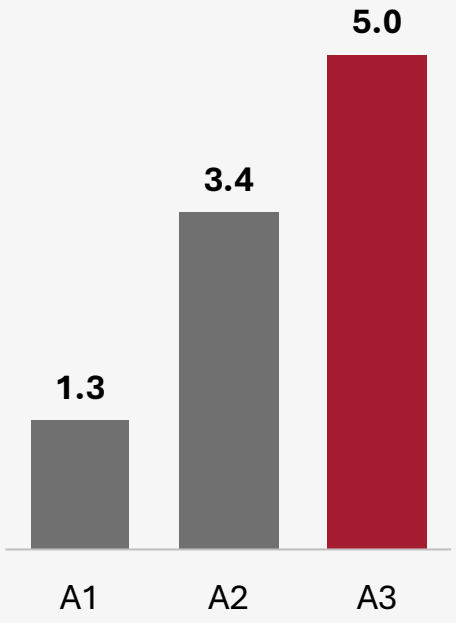
P8 Reading one noisy run
Run several times, compare spread

P9 Venting instead of steering
Paste the error, ask for a diagnosis

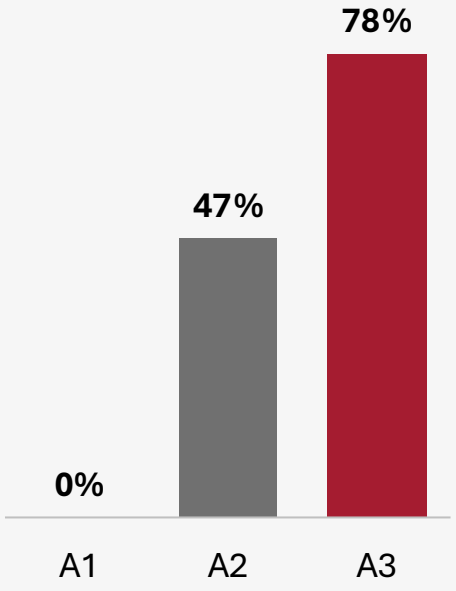
P10 Platform issues, laptop sleep
Check requirements; run jobs in tmux

Harder assignments: more pitfalls

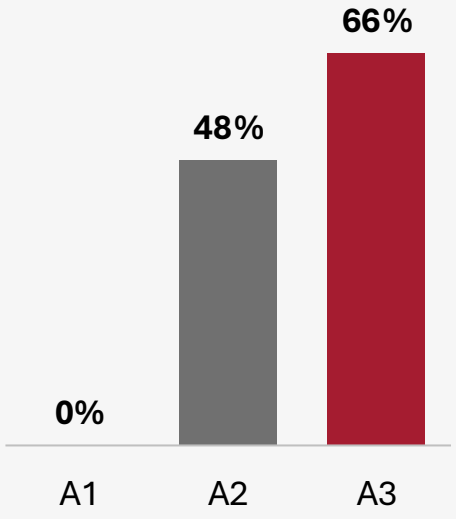
Pitfalls per student, average



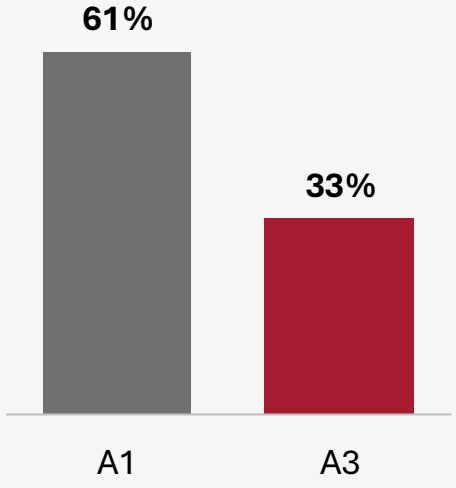
Share of students who stacked changes



Share who decided on one noisy run

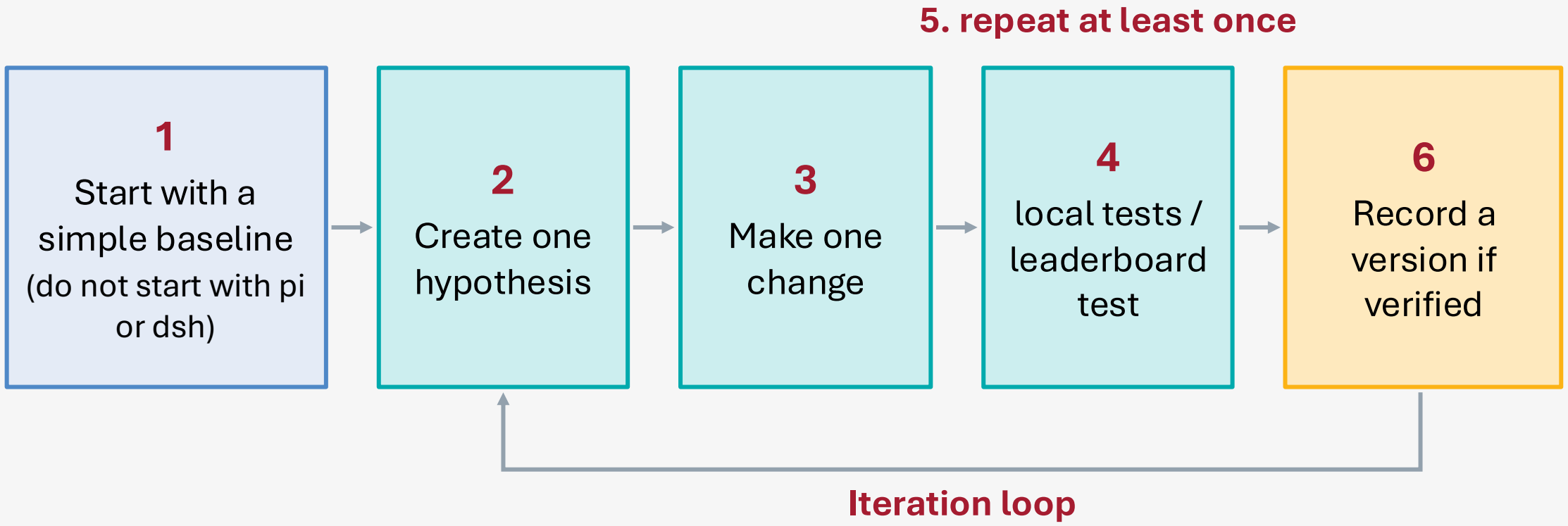


Share of agent errors students caught



Shares are of the same 58 students in each assignment. A2 share of errors caught not reported.

Claude Code Best Practice: a good iteration loop



Keep in charge; step-by-step; verify before move on (testing rigor matters)

Towards effective use of agents

An experimental feedback platform to help you improve your skills using agent

https://how-i-used-claude.cs2680.com

Feedbacks are welcome!

Your time and the agent's

A hand-back is when the agent gives the turn back to you. "Your time" leaves out pauses over 15 minutes, the time after a session's last hand-back, and work outside Claude Code.

AGENT TIME

3 h 35 min

in all, from prompt to hand-back

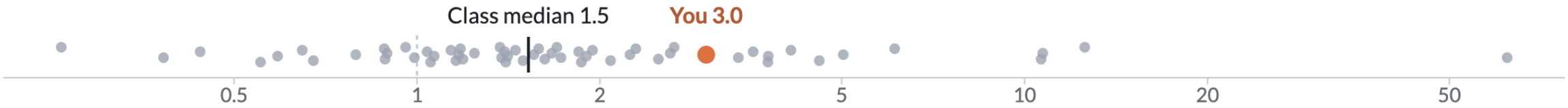
YOUR TIME

1 h 12 min

in all, from hand-back to prompt

3.0 agent minutes for each minute of yours (your ratio)

The class median is 1.5. Your ratio is higher than that of 79% of the class.



Each gray dot is another student. At 1, the two times are equal.

✓ Your ratio is above the class median. Still check the agent's work.

Ways to give the agent more of the work 1 way that fits your numbers

- **Fewer questions.** Put the facts it needs in your prompt, so that it asks you less often.
  It asked you 4 questions. You used 23 min of your 1 h 12 min to answer its questions, approve its plans or allow its edits.

✓ You already use plan mode, subagents and tests.

Pitfalls from class, in your sessions

In class we went over common pitfalls in working with a coding agent. The course staff checked your Claude Code sessions in each assignment for each of them. “Yes” means the record shows it happened; “Likely” means the pattern is there but not what it cost. Below the table, each one that happened, with what you

Assignment 3

Oct 17, 7:46 pm · **Open in your session →**

YOU TYPED

```
why there is nothing in failure_memory.json, I have already run a full test of 5
```

THE AGENT

Earlier that evening: "Both features are implemented in `starter_code/src/` and all offline tests pass." At 19:46: "The file is empty for two reasons. The first is by design. The second is a bug that stopped every lesson from being learned."

WHAT HAPPENED

Your agent reported that failure evidence and failure learning were implemented and that all offline tests passed, and a hardening pass then reported 91 tests passing. You built on that and ran a full five-task test, then found failure_memory.json empty. Your agent found two reasons: lessons were never written to that file by design, and a bug had stopped every lesson from being learned, so that run learned nothing. You spent two more sessions that evening on the feature, and at 22:22 you asked again: the lessons had gone to a log folder that is moved or deleted before each run, so nothing carried over.

NEXT TIME

When your agent says a feature works and the tests pass, ask for proof that the feature did its job once before you spend a live run on it, such as one lesson actually written to the file it should be in.

You could type — Copy

```
Before my next run: make one offline task fail, then show me the lesson it writes, the exact file path, and which run will read it.
```

AI is a tool everyone has to MASTER

How I used Claude Code in assignments?

- Where did things break and fail?
- Where did I waste time?

How can I improve my AI skills for quality / effectiveness and productivity?

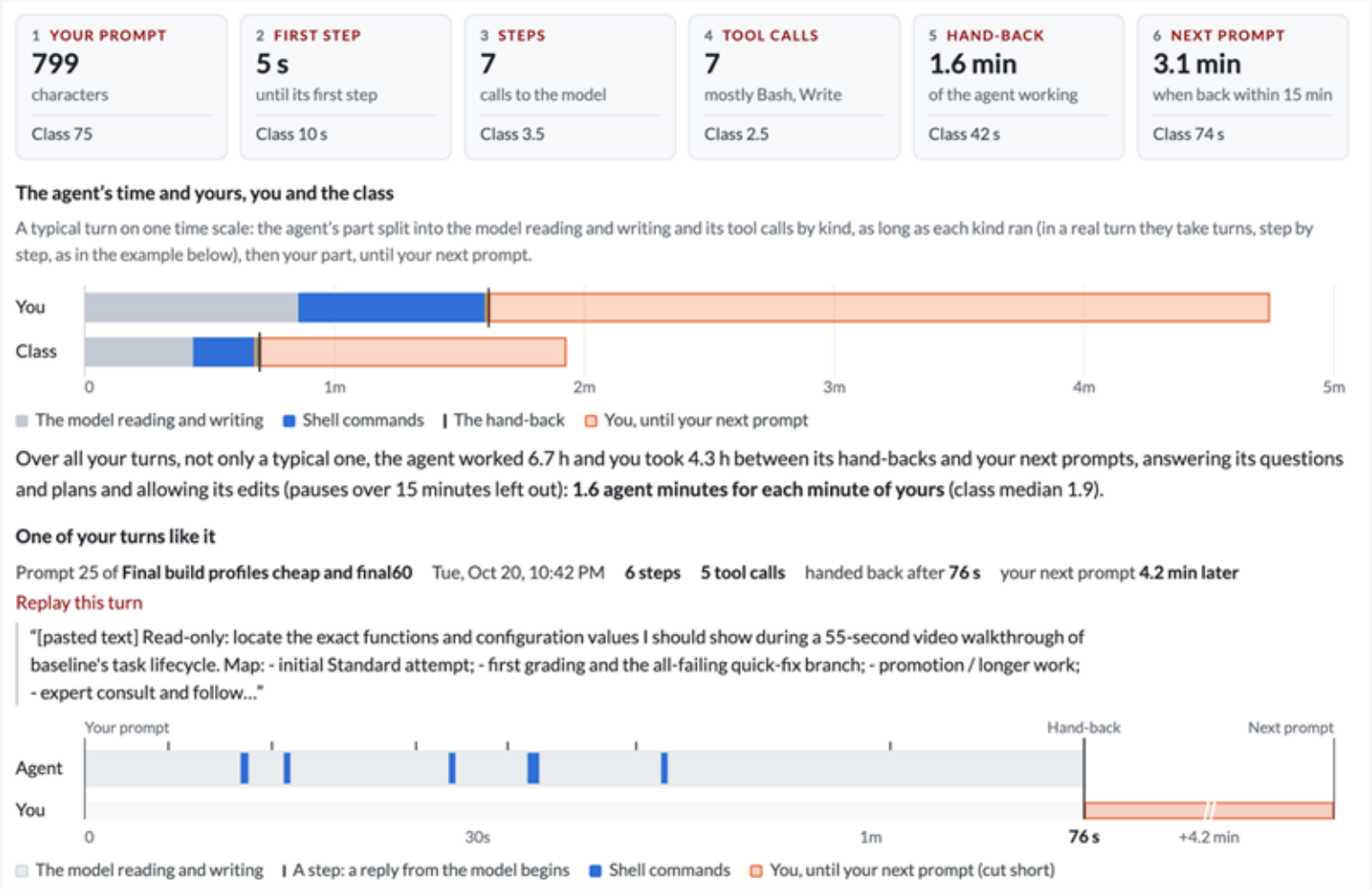
- Is my prompt too vague? Maybe I should not ask it to finish a project?
- Should I write a longer and more detailed prompt?
- Do I miss important things in the prompt?
- Are there better ways to use agents?

https://how-i-used-claude.cs2680.com (WIP)

How you compare with your classmates



Visibility into your Claude Code usage



Semi-personalized tips for everyone

Strong change control; noisy single runs and version mix-ups cost time

You opened with standing rules the agent saved to memory, then built the harness piece by piece from long numbered specs, asking for simple explanations. Every change came back as a diff held for your OK and was tested offline with a pretend model first. You compared five full ten-task batches, each kept as a baseline.

What went well

- **Standing rules the agent kept**
  On Oct 5 you opened with standing rules, such as never reading task_test.json, which the agent saved to memory and later cited to decline a mistyped request.
- **Changes kept apart from running batches**
  On Oct 5 you told the agent not to touch the harness while a batch ran; that grew into versioned pending diffs, a git tag per version and EXPERIMENTS.md. 42% of your prompts set limits (class median 13%).
- **Asking the logs for failure modes**

Where the agent slipped

- Wrong diagnosis, too much confidence The agent judged Run 1 of task 0 a half fix from reading its patch; grading then showed it RESOLVED. A real run showed it.

  Treat the agent's verdict on a patch as a guess until the grader runs, and ask what evidence the verdict rests on.
- Wrong diagnosis, too much confidence The agent expected v7 to fail the custom tasks highlight.js and SCons; v7 solved both, and highlight.js passed even on the old v3 tag. A real run showed it.

Try next

- **Stamp every run with its version**
  The v6 and v7 diffs you believed applied were not, and a pasted task 9 run appeared to be v5.

  Have madsLoop write its git tag into the first line of each log, and ask the agent to check it before comparing runs.
- **Repeat a run before trusting it**
  With one run per task per version, task 0 failed then passed, and task 6 passed three batches then failed.

  Run any task that drives a decision at least three times, as you did for task 9, before calling

Course-staff feedback across assignments

Assignment 1 | Assignment 2 | Assignment 3 | **Across assignments**

Feedback from the course staff

For Assignment 4

In Assignment 1 you planned first: every session opened with a design, and subagents searched the code for you. In Assignment 2 you started 1.5 days before the deadline, with no plan, and asked for the same fixes again and again. In Assignment 3 you started early again and let the agent see the page. Plan first again, and say what done looks like, so a fix holds the first time.

1. **Plan first again.**

   In Assignment 1 every session opened with a design. In Assignments 2 and 3 none did, and fixes took several requests. Open your first session in plan mode, and approve a short plan before any code.

   A prompt to start from — Copy

   ```
   Before writing code, read the assignment and propose a plan: the parts, the order to build them, and how we will test each one. Wait for my approval.
   ```

2. **Make done something you can check.**

   The same fixes came back in every assignment: streaming in Assignment 1, the run form in Assignment 2, the blank page in Assignment 3. Ask for a failing test first, then the fix, then the test passing.

   A prompt to start from — Copy

   ```
   Write a test that shows this bug, run it and show me that it fails. Then fix the bug, run the whole test suite and show me the output.
   ```

Everyone has an agentability portrait that changes with each new assignment

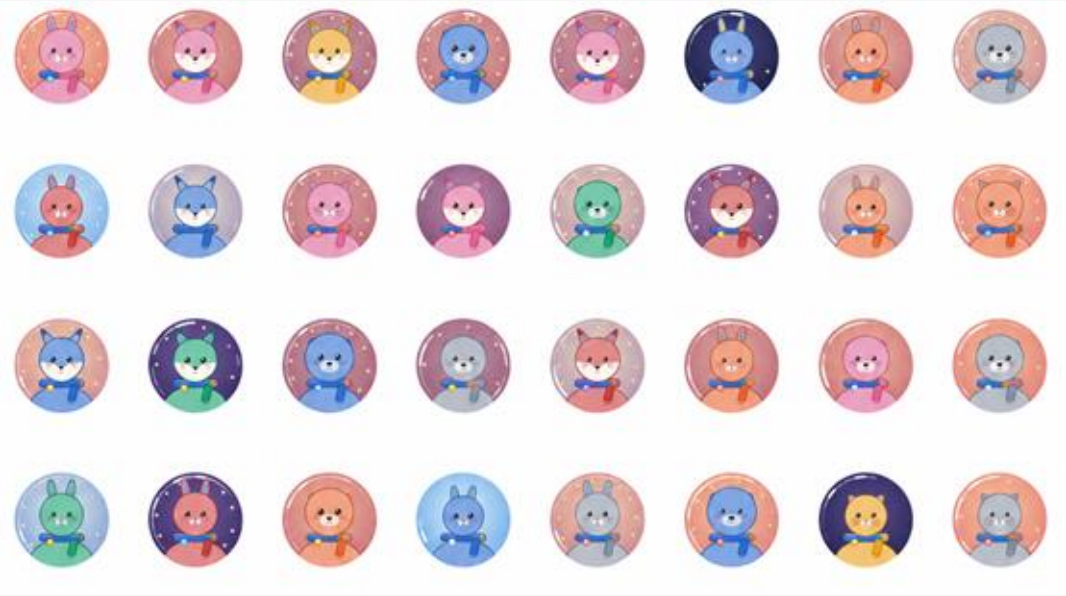


Download SVG PNG

Why yours looks like this

Animal	A fox, for delegators (few prompts, each a big job): your 201 typed prompts were just below the middle of the class, which counts as few, and your tool calls per prompt were higher than 61% of the class, which counts as big jobs.
Coat	Pink, for browser: 8% of your agent's tool calls against 4% of the class's, the kind of tool it leaned on most compared with the class. The inner ears are blue, for shell commands, the kind it used most (81%).
Scarf	Its stripes are your agent's 1,697 tool calls by kind: 81% shell commands, 8% browser, 4% writing files and 7% across 5 other kinds. The tassel shows the coat's kind again.
Ears	Medium ears: your tool calls per prompt, how big a job each prompt handed over, were higher than 61% of the class.
Eyes	Big eyes: your prompts were longer than 87% of the class.
Head	A small head: your active hours were lower than 68% of the class.
Freckles	30 freckles for your 201 typed prompts: one each up to 12, then fewer per prompt (at most 30).
Badge	Yellow, for "Plan and build": 33% of your 3 labeled prompts asked for this, the most of any family.
Sky	Dusk: your prompts came at about 6 pm on average.
Stars	11 stars, one for each day you typed a prompt.
Clock ring	Arcs at the 15 hours of the day you typed prompts in, thickest from 8 pm to 9 pm, your busiest hour.
Moons	3 moons, one for each subagent your agent launched.

Everyone is different in the class!





Have fun!

Feedback is welcome!

https://how-i-used-claude.cs2680.com

Next class

- GPU programming
- LLM serving basics III: scheduling, paging and parallelism