

Introduction

ar018 · 9 July 2026

Here is an observation. Software engineers use coding agents constantly; scientists, far less so. That gap is not about talent or curiosity, it is about tooling: the agent arrived in a form built for shipping software, not for running experiments and writing them up. demolab is an attempt to close that gap. It is a lab notebook for computational science in which the science lives as code, an agent operates it in plain language, and one build turns the whole repository into a website and a set of PDFs. This page makes the case for it, in the same two acts as the talk it is drawn from: first the agents, then the workflow built on them.

Coding agents got good

A coding agent is not autocomplete. It reads a repository, runs programs, edits files, and checks its own work; it can do anything a terminal can do. The line runs autocomplete (2021), to chat (2023), to agents that do real work (2025), and only the last step made it useful for anything as unforgiving as science.

The clearest evidence is SWE-bench Verified [1, 2], a benchmark of real, human-validated GitHub issues that an agent must resolve end to end. The trajectory, shown below, is the point: at the benchmark's 2024 launch the best systems closed about a third of the issues, and by late 2025 that figure had climbed past three quarters. A tool that was a novelty two years ago now does a large fraction of a working engineer's tickets unaided. Many such agents exist, commercial and open, so there is no vendor lock-in.

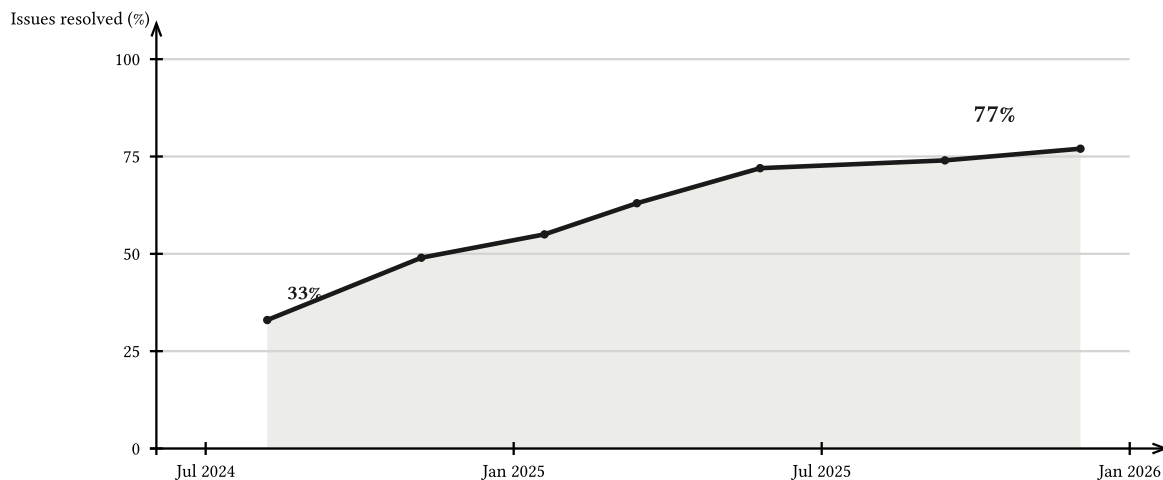


Figure 1: Share of real, human-validated GitHub issues a coding agent resolves end to end, on SWE-bench Verified (swebench.com), from the benchmark's 2024 launch to late 2025. The vertical axis is percent of issues resolved; the horizontal axis is date. The resolved fraction climbs from 33% to 77% in about eighteen months. These are software bugs, not science, so the level does not transfer to research; the slope is what matters.

It would be dishonest to sell only the strengths. Agents do the same work faster, clear the backlog of things you never had time for, document as they go, and read a paper straight into runnable code, and they work far better over a whole repository than over a snippet. They are also confidently wrong: they invent APIs, numbers, and citations. They leave you with comprehension debt, code you shipped but do not understand. Their judgement is weak; an agent will not kill a bad experiment on its own. So the honest summary is short. A coding agent

is a powerful tool that needs supervision, and a framework worth having is one that supplies the rails.

What demolab is

demolab is that framework. It writes code, runs programs, and documents the results, and it holds the whole thing to one loop: run, produce data, write it up, publish. One build compiles the repository into a website (with real, selectable math) and a PDF per entry plus a bound book. The discipline underneath is a single rule with teeth: nothing on the page is typed by hand. Numbers and figures come from the run, so a result cannot quietly drift from the code that produced it, and every result carries the git provenance of the run that made it. It is reproducible by default, and it stays reproducible a year later because one command rebuilds all of it.

Those properties answer real failure modes. Results drift when numbers are retyped and figures go stale; here they come from the run. Reproducibility rots until, a year on, nothing builds; here one command rebuilds everything. Code and paper drift apart into separate repositories; here the experiment and its write-up live together. And agents need rails; here they operate in plain language while the science stays yours.

A few principles hold the shape together:

1. **Tools compute, experiments analyse.** A clean split between the reusable science and the runner that exercises it.
2. **One experiment is a runner plus a write-up.**
3. **Components talk through files, never imports.** A tool emits data; a runner reads it.
4. **Raw data is disposable; the record is committed.**
5. **Bring any stack.**
6. **Provenance is built in.**
7. **Documentation is everything.**

The shape of a repository

A lab is four content directories, each with one job:

- **tools/:** the science, as reusable models and solvers. A tool emits data (a `numbers.json` plus the data behind each figure) and stays language-agnostic.
- **experiments/:** the runners. One `expNNN.py` per experiment calls a tool, renders the figures, and stages a `numbers.json` of the headline metrics.
- **writings/:** the write-ups. One `.typ` per entry reads the run and embeds its figures and numbers. Prose articles like this one, and slide decks, live here too.
- **artifacts/:** the committed record, holding the figure data and a PDF per entry.

The engine itself ships inside the `demolab-cli` package, a black box you never edit and update with a dependency bump. Publishing is built on Typst, a modern take on LaTeX: instant incremental compiles instead of slow multi-pass ones, readable errors with line numbers instead of macro soup, and one source that renders to both web and PDF instead of PDF alone. Because Typst imports files, the numbers on the page are the numbers from the run.

Driving it with an agent

demolab is meant to be run by a coding agent, though you can operate every command by hand. You type a name in capitals and it acts. **HELP** lists everything it can do. A guide name (`RULES`, `HOUSESTYLE`, `SLIDES`, `STRUCTURE`, `GLOSSARY`, `SUPPORT`) walks you through an always-

on convention. A runbook name (`GETTING-STARTED`, `LINT`, `DOCTOR`, `FROM-PAPER`, `MIGRATE-STACK`, and the rest) runs an on-demand procedure step by step: read a paper into code, convert a notebook, migrate the language, update the engine. Underneath it all is an ordinary CLI, so `uv run python experiments/exp000.py` runs an experiment, `demolab dev` serves the site with live reload, and `demolab build` compiles the lot. The default stack is Python, but tools talk through files rather than imports, so you can switch to MATLAB, R, Julia, or Octave by asking, and the contract and the typesetting stay put.

Where it's going

Everything above is `demolab` today: implemented and ready to use. Two directions define where it goes next.

The first is autoresearch. Give the agent a goal and some compute, and it runs the loop itself: goal, code, compute, read the data, document, iterate. Because the rails already exist, every attempt lands as a committed experiment and write-up with provenance and a published trail, for free. It is built, and still being tested.

The second is scale. Version one is one person's lab notebook. The next step is a lab on a GitHub organisation: many researchers, one shared flow, with questions and answers, code review, and shared code compounding across the group. The notebook becomes the lab.

References

1. Jimenez CE, Yang J, Wettig A, Yao S, Pei K, Press O, Narasimhan K (2024). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *ICLR 2024*. doi:10.48550/arXiv.2310.06770
2. OpenAI (2024). Introducing SWE-bench Verified. Live leaderboard at swebench.com.