

pi profiles, part 2: agents, plural

Adrian Tame Jacobo, writing for Plicara Labs · 2026-10-13 ·

A while ago I noticed that every Pi session I started was already 15 to 20% full before I had typed anything. One of my extensions, a to-do tracker, was loading a huge amount into context on every start. It was useful, but it was absolutely massacring my ability to work on anything else. Debugging that, and working out which other extension could do the same job, is how I ended up building Pi profiles, which is part 1 of this series.

This piece is about why I work that way, and where I think it is all heading. The practical side, how the launcher works, is in the companion article. This one is the why.

pi, and the people it is for

Pi is a pretty beautiful system. It is useful, it is extensible, it is open by design and it lets you play. And we are in an era where you can use AI tools to build the system with itself, which is a bit crazy when you think about it. A tool that builds itself is a very interesting idea, and it is really how Pi is meant to be used.

The other harnesses are powerful too. Claude Code, Codex, Cursor, Goose, and a long tail of smaller ones. There is even Oh My Pi, which is built on Pi and goes in the opposite direction: a ton of functionality, a ton of skills, everything switched on. I like it and I use it quite a bit. But for me it is overkill, and it cuts against what I think Pi is about.

There is a lot of excitement around Pi among the people I talk to who build systems. It is the same kind of person who got excited when TypeSafe released Jev, or who goes deep into LazyVim. Engineers who like to work with their tools, take them apart and modify them. There is always going to be a place for those people and for the work they do.

That is a very different kind of user from someone using a mainstream application where the decisions are made for you. It is the Linux crowd versus the Mac crowd. I'm not arguing about the merits of either, whether you're on a Mac or on a 14-year-old ThinkPad that is modded to high heaven. There is a need for both, and a beauty in both. Some people just want their tools to work for them. Some people want modular control. Pi is for the second group, and its whole philosophy is minimalism plus extensibility by the people who use it.

the one thing i was missing

Pi is minimal, obviously, so I understand why some things are not in it. Durability is a good example. Pi Durable is an incredible direction and very interesting, but it is also very opinionated, so it lives separately from the minimal core. That is the right call, and I tested it (write-up coming soon) because long unattended runs are exactly where it matters.

The one thing I felt was actually missing from Pi was the ability to keep more than one agent: to create profiles and move between them. So I built my own. I am not saying it is for everyone. I am saying it is for me, and that is the point of Pi. Several other people have built their own versions, which tells me I'm not the only one who wanted it.

The to-do extension was the first push. Testing was the second. Ponytail is a popular extension, and a good one, but it changes the agent's behaviour quite drastically, and with everything else loaded alongside it I couldn't tell what it was doing to my setup. Then NVIDIA released SoL-Pi and I wanted to try it on long sessions. I tried it inside my normal setup, and another extension silently switched off its main feature for the whole trial. I even opened an issue on their GitHub before I realised it was my setup, not their code (the full story is in the other article). I had changed more than one thing at once, and the trial told me nothing.

a lab, not a single stream

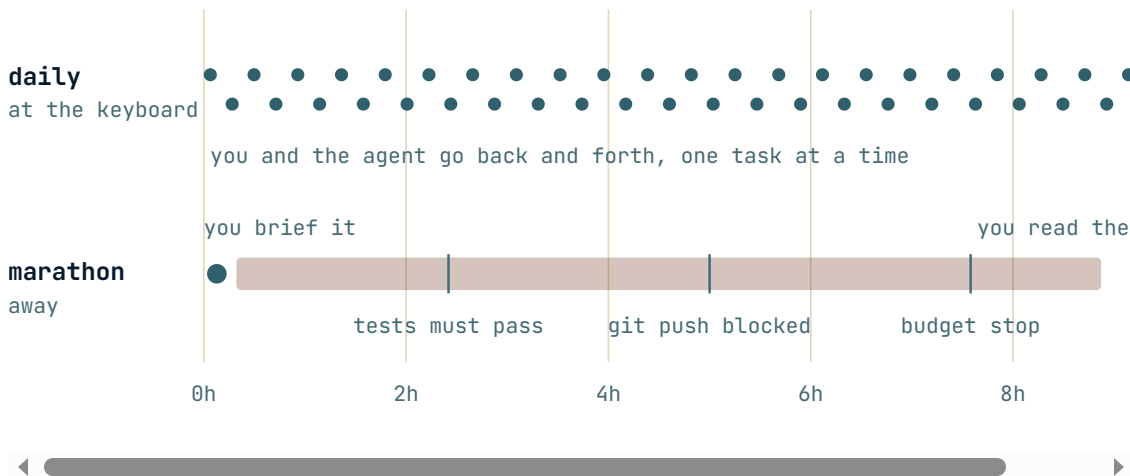
Native Pi in its minimalism is absolutely beautiful, but it is also restrictive. Some people will like that. What I want is a lab. The way I think is more like a laboratory than a single stream of work. I want to isolate environments, test one thing at a time and separate tasks by profile.

Pi is a generalist tool, which is part of the point. But sometimes you need a specialised one. A screwdriver will take ten times less effort than trying to do the same job with a hammer, if a hammer is all you have. People talk about Pi as a Swiss army knife. I don't think it is one. It is a very simple stick that you build into something useful to you. I want to build many things that are useful to me.

The difference between my profiles is not really the tools. It is the way of working. **daily** assumes I am in the loop the whole time: back and forth, one task, everything visible. **marathon** assumes I am gone for ten hours, so it needs things I don't want anywhere near my daily work, like Pi sessions that can spawn other Pi sessions, an approval mode that won't stop at 3am, and hard rules about budgets, worktrees and when to give up. Those are

different permissions and different failure modes. Putting them in one configuration makes both worse.

FIGURE 3. TWO WAYS OF WORKING



An illustration of how the two profiles are meant to be used, not a measurement. With **daily** I am in the loop the whole time. With **marathon** I leave it, so the rules I rely on are checks that run in code.

This is the one place where I disagree with the philosophy of the team behind Pi, and I'm probably the only one. It is not agent, singular. It is agents, plural. If you really want to get into Pi, profiles are a necessary part of that learning journey, because you need to be able to test things out.

isn't this the opposite of minimalism?

The obvious objection is that more profiles means more configuration to maintain, which is exactly what Pi is trying to avoid. I think it is the other way around. The alternative to several small profiles is not one small profile. It is one large profile that slowly accumulates everything you have ever found useful, until it looks like Oh My Pi.

Profiles let each setup stay minimal. My SoL-Pi profile has one package. **daily** has seven. **marathon** has eleven, because unattended work needs more machinery. None of them carries the others' tools. The complexity moves from what is loaded to what you choose, and choosing is the part I want to be deliberate about anyway.

There is a second opinion baked in that is worth naming: my profiles share session history. I care more about the continuity of my work than about keeping agents apart. I can start something in `daily` , realise it is a marathon job and pick up the same thread there. Other people want each profile to be its own agent with its own memory, and that is a fair choice too. It is just not mine.

the models got good

The other part of this is that the models keep getting better. Something people forget, and I forget too, is that this is the worst these models will ever be.

At some point this year the models crossed a line for me. They understand context well enough to do what you mean without careful wordsmithing. I like to talk things through, and most of what I write, including this article, starts as a voice conversation with an agent: bouncing ideas around, arguing, recording. That only works because the models now follow a messy spoken explanation.

So here is a strong claim. I think prompt engineering is basically dead, and the skills way of working is going with it. If you came into these tools today with no skills and none of the old habits, I suspect you would get better results than someone carrying over everything from six months or a year ago. Boris, who created Claude Code, recently shared how he prompts it, and it is about as basic as a prompt gets: build me a game. And it works. What matters now is harness and context engineering.

That is my experience. It is not yet evidence, and I would not accept it from anyone else on those terms. But some of the evidence I do have points the same way.

FIGURE 4. DO MODELS READ SKILLS?

Before the edit: 7 of 21 sessions opened the skill file



Right after I put the rule there: 0 of 3



☒ opened SKILL.md ☐ did not



One square per session in my Pi Audit Loop experiments. Small numbers from one setup, so this is an observation, not a proof.

- **Models often don't read skills.** In my earlier [Pi Audit Loop](#) experiments, 7 of 21 sessions opened the bundled skill files at all, and 0 of 3 did right after I put an important rule there. Moving it into a tool description and a tool message, at the point where the model had to act on it, was what got it in front of the model at all, and even then it did not reliably change what the model did.
- **The guarantees I rely on are code.** In `marathon`, the review gate refuses a clean verdict unless the tests ran since the last change, and the loop blocks `git push`. Those are extensions. The written rules in that profile matter, but they are not what I trust when nobody is watching.
- **The context cost is real.** The to-do extension at the start of this piece is the clearest case I have. Every skill, tool definition and extension takes up context and can trigger when you don't want it to.

None of that proves skills are fading in general. It is one person's setup and a few small experiments. So the honest version of my claim is a hypothesis: skills will stop encoding ways of working and mostly encode taste, how you like things done, while the important parts move into hooks, extensions, permissions and slash commands you invoke at a particular moment.

That is testable. When I wrote that [the harness is the signal](#), the idea was to watch what we build around models to understand what they can and cannot do. The same approach works here. The [GitSkills](#) dataset has about 3.8 million `SKILL.md` files from 282,200 public repositories, and it is what my earlier articles on [the languages skills are written in](#) were built on. If I'm right, I'd expect skill files to shift over time from procedures towards preferences, and hooks and extensions to grow alongside them, especially among people who actively customise their harnesses. If I'm wrong, the data should show that too. That is the next piece.

become a harness engineer

There is a call to action here regardless of how that test comes out. The way Pi was created, the philosophy of the team and the purpose of the tool make it a very useful base for everything. But it also forces you to think about context engineering. In Pi you see the cost

of every skill and extension, because you put each one there yourself. I'd love to have every skill and extension loaded for free, but that isn't how it works.

If you want to become a harness engineer and actually think about these problems, the tool you need is something like Pi. Instead of building software by importing libraries, you are designing the harness and the agents that help you through the work.

So design the environment, not just the prompt. And keep more than one.