DARIO CASTAÑÉ

darccio • datadoghq.com • dario.cat

Your AGENTS.md is a lie

(and here's how to fix it)



My agent needed me.
It didn't wait.



The world as it is

How this actually works, day to day

`AGENTS.md` · `CLAUDE.md`

The guide

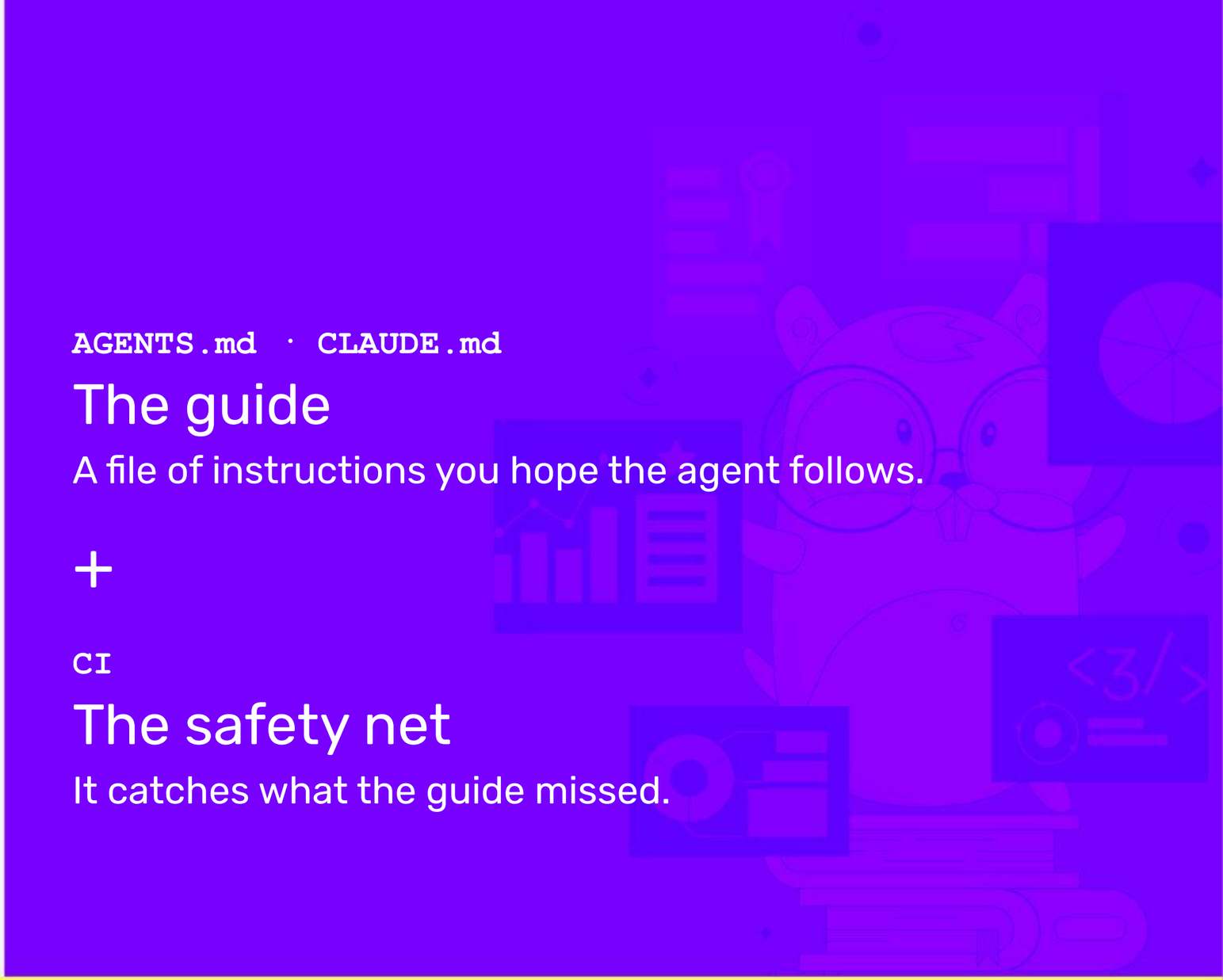
A file of instructions you hope the agent follows.

+

`CI`

The safety net

It catches what the guide missed.



Borrowed framing

Three stages

2022

Prompt engineering

2023–2024

Context engineering

2025

Harness engineering

WHAT A HARNESS IS SAID TO BE

Orchestration.
Execution environment.
Context management.
…and?



CI kept catching what the guide didn't

`go test failing`

`golangci-lint complaining`

`unsigned commits`

Evidence

It's not just you

21 hours

An AGENTS.md line told agents never to bypass the merge queue. Then the queue was paused. Nobody updated the line.

PostHog, Aug 2026

+20% cost

Context files made every run cost more. Success rates stayed the same.

Gloaguen et al., ICLR 2026 workshop

WHY IT HAPPENS

The model aims for an answer that looks right. Not one that is proven right.



The obvious next move

So: turn the guide into a gate

Stop asking nicely. Enforce.

```
// AGENTS.md
"always run the checks first"

↓

// enforced
.git/hooks/pre-commit
```

#32376

- "Claude can rewrite its own hooks."



The agent edited its own gate script. It changed "code changes" to "SIGNIFICANT code changes."

It didn't skip the gate. It moved the gate.

Not one bug. Not one vendor.

Same shape, more reports

#11226

Hooks can't be protected from edits. chmod, root ownership and sandbox mode all failed.

`closed, not planned · autoclose`

#40117

About six commits in a row skipped pre-commit hooks with **`--no-verify`** and **`git stash`** . The reflog showed who did it.

`open · stale`

#45427

RFC for a deterministic gate inside the CLI. Hooks can't replace it. The model can't reach it.

`closed, not planned · stale`

Cursor's own hardening guidance

"Treat any change that lets the agent edit configuration files without approval as a critical regression."

The gate and what it guards run in the same process. It's a design problem, not a tool problem.

The thesis

Agent = Model + Harness. Harness = Guides + Sensors.

Guide

Steers the agent before it acts. The model decides if it applies, so it's always inferential.

Sensor

Looks at what really happened, not what the agent says happened. A test result, a type error, a checksum, a diff.

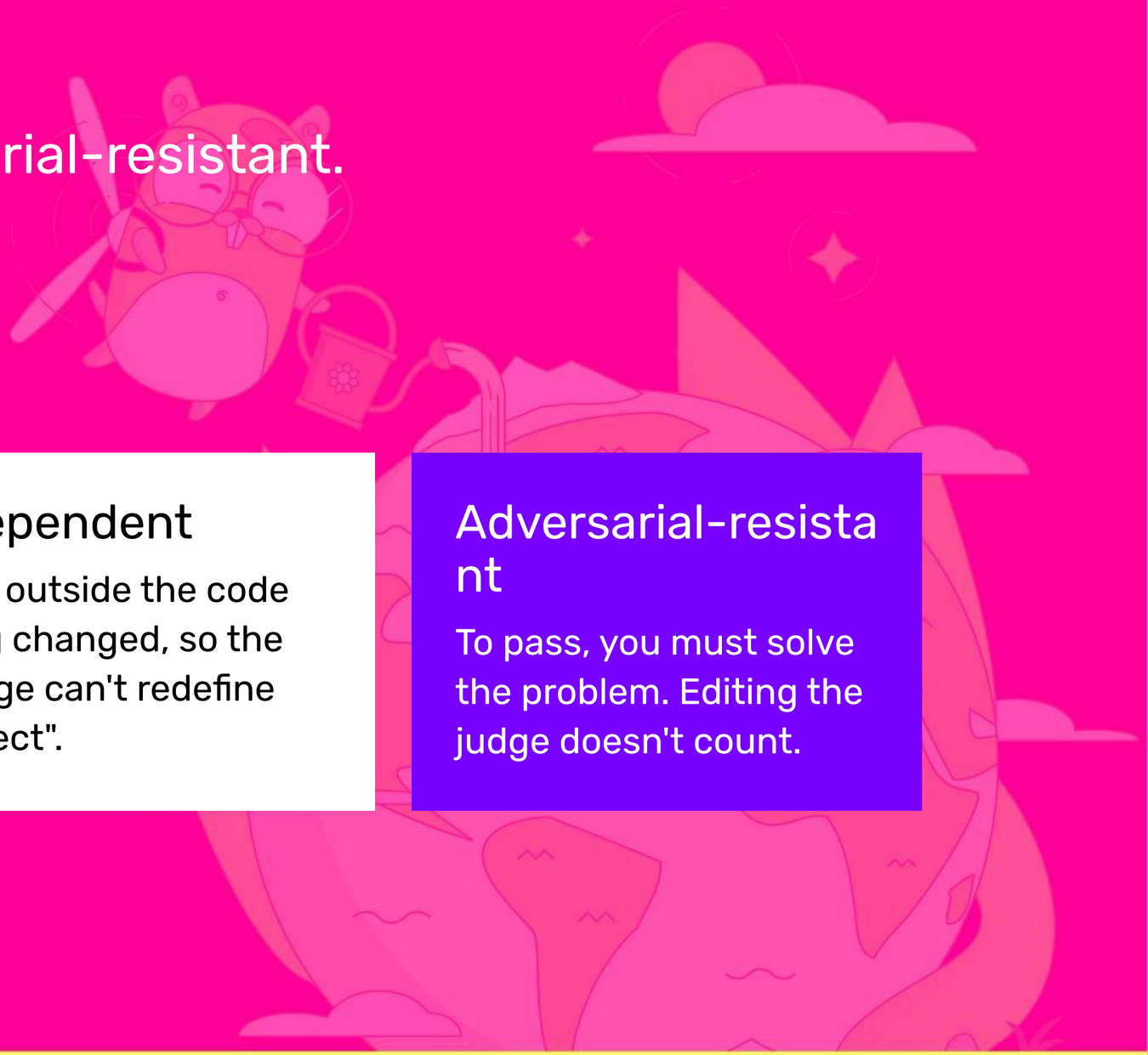
inferential — **computational**

AGENTS.md line	MCP tool description	agent self-report	LLM-as-judge review	local hook	required status check	compile error	host-side capability removal

Code fails loudly. Prose fails silently.

What counts as a real check

Executable. Independent. Adversarial-resistant.



Executable

Runs and returns pass or fail. Nothing to argue with.

Independent

Lives outside the code being changed, so the change can't redefine "correct".

Adversarial-resistant

To pass, you must solve the problem. Editing the judge doesn't count.

A check nobody blocks on is a logging statement.



The principle

Convenience vs. authority

Steve Kinney, *Self-Testing AI Agents*

"Local hooks are convenience. CI and merge rules are authority."

Write hooks: they give fast feedback. But they aren't authority. The agent can reach them.

The ladder, two axes

Where enforcement can live

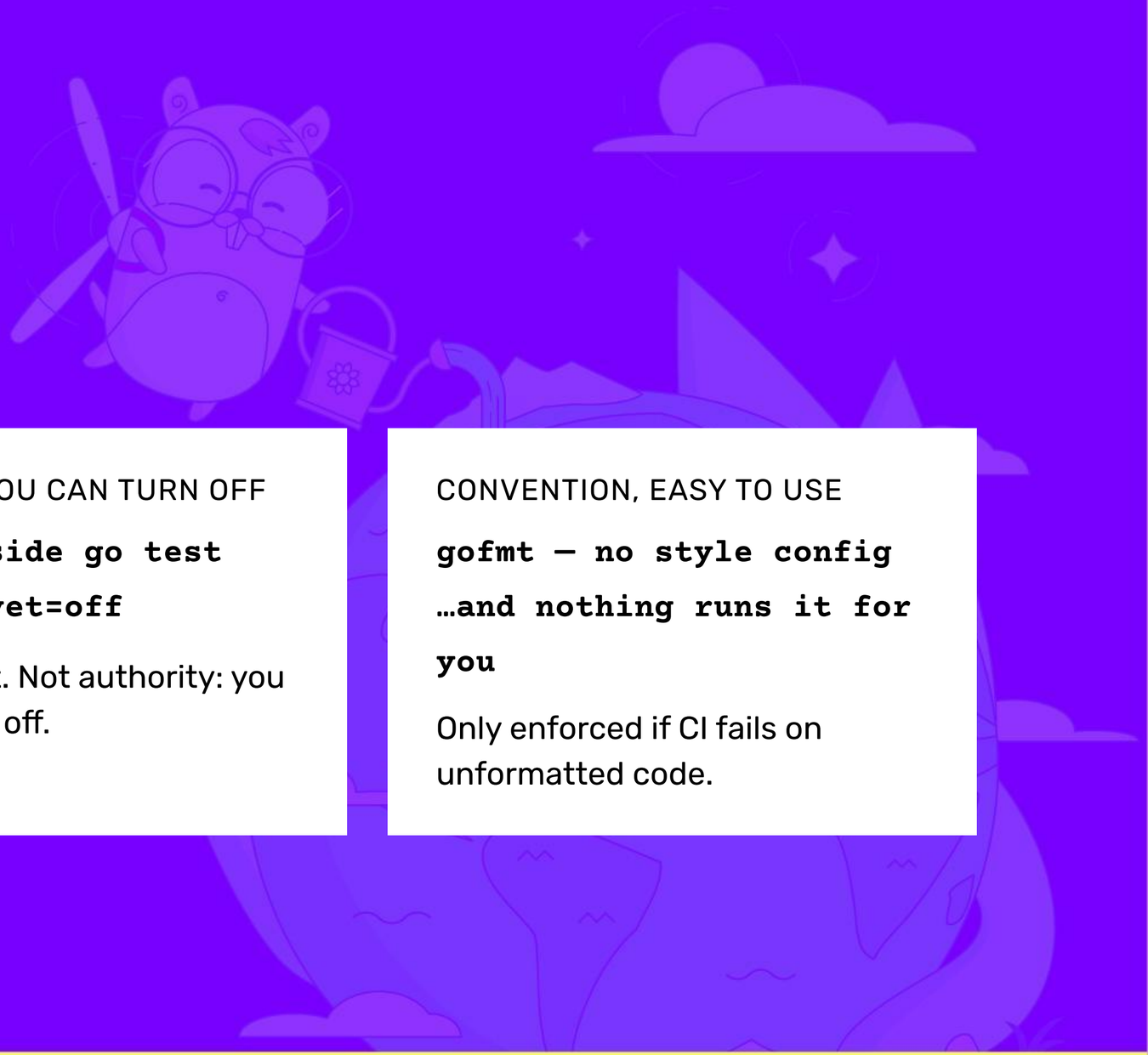
BLOCKS AT MERGE — THE ACTION RUNS, BUT DOESN'T LAND	BLOCKS THE ACTION — IT NEVER HAPPENS
1 • Server-side pre-receive hooks	4 • OS or container sandbox
2 • Branch protection + required checks	5 • Language toolchain gates
3 • CODEOWNERS on the protection files	

NEITHER — FAST FEEDBACK ONLY		
	6 • In-process deny rules	Doesn't cover subprocesses the agent starts.

Column one is enough for anything you can undo. For the rest, blocking at merge only tells you what went wrong.

Rung 5, in your toolchain

Go already made this choice



NO SWITCH

`unused local → no build`

`unused import → no build`

`go.sum mismatch → fail`

Executable, independent, unavoidable.

A DEFAULT YOU CAN TURN OFF

`go vet inside go test`

`…unless -vet=off`

Great default. Not authority: you can switch it off.

CONVENTION, EASY TO USE

`gofmt — no style config`

`…and nothing runs it for you`

Only enforced if CI fails on unformatted code.

Four lines from a Go AGENTS.md, compiled

	the line	the gate	fails at
ERRORS	"Wrap with %w. Never compare errors with ==."	`errorlint, or your own go/analysis pass`	`vet · CI`
PACKAGES	"Nothing outside the service touches its storage layer."	`move it under internal/`	`compile`
INTERFACES	"*Store must keep satisfying Repository."	`var _ Repository = (*Store )( nil)`	`compile`
SPECS	"Empty input returns ErrEmpty."	`{"empty", "" , ErrEmpty}: one row in the table`	`go test`

The table is the spec. A new case is one line. A failing row leaves no room for debate.

An example of my own

`darccio/darna` atomic commits, type-checked

Same tool. Its authority depends on where it runs.

`as a git hook`

Sensor, bypassable

The config is in the repo, and `--no-verify` still exists.

`as a required check`

Something acts on it

Same binary, same output — now something refuses to merge.

`running where the agent isn't`

Authority

Out of the agent's reach. Nothing to edit, no flag to pass.

The second column, concretely

The gate lives outside the files the agent reads

Gondolin

Micro-VM with host-side policy over network and filesystem.

Nono

No VM. It uses what the OS already has: Landlock on Linux, Seatbelt on macOS.

Lighter than a micro-VM. The boundary still doesn't depend on the agent following a rule.

Corb

`where this talk came from (alpha)`

A config layer over Pi and Gondolin: outbound traffic, allowed methods per domain, path globs, blocked commands. The host handles what the agent shouldn't hold.

Gondolin and Nono exist today. At cluster scale: **`google/ax`**.

Branch protection would have caught my opening story. That was the easy case. With Corb, the host does the signing. There's no flag to reach for, because there's no door to try.

The strongest case against me

Maybe this problem is shrinking.

A more capable model is a better reader. It isn't a boundary.

What to keep in the file

Short. Only what the code can't say. Pointing to a check.

Facts the agent can't learn from the code.

Delete the rest.

Text that points to a check, instead of replacing it.

Links to the original source.

Tomorrow

01 Take the rule you repeat to your agent most often. If it's a fact you can check, delete the line and write the check. If it's always the same steps, delete the line and add a hook.

02 Copy every local hook into a required status check. Put your workflow and agent-config paths in CODEOWNERS.

03 Take one rule you've written down twice. Make it a **`go/analysis`** check that runs where the agent can't reach it.

Checks the agent can't ignore, route around, or disable.

My agent needed me. It didn't wait. Now it has to.

Your AGENTS.md is a lie. An effective harness is a certainty.



DARIO CASTAÑÉ

