DOGBENCH.AI

What AI agents get wrong when writing docs

Based on DoGBench (**Do**cumentation **G**eneration **Bench**mark), the first AI benchmark for user-facing documentation.

Promptless

Built with the community

A big thank you to the Write the Docs community.

CO-AUTHORS

Paige Calvert
Helm

Ayu Adiati
Mautic

Sarah Sanders
PostHog

Manny Silva
Promptless, Doc Detective

REVIEWERS

Dachary Carey

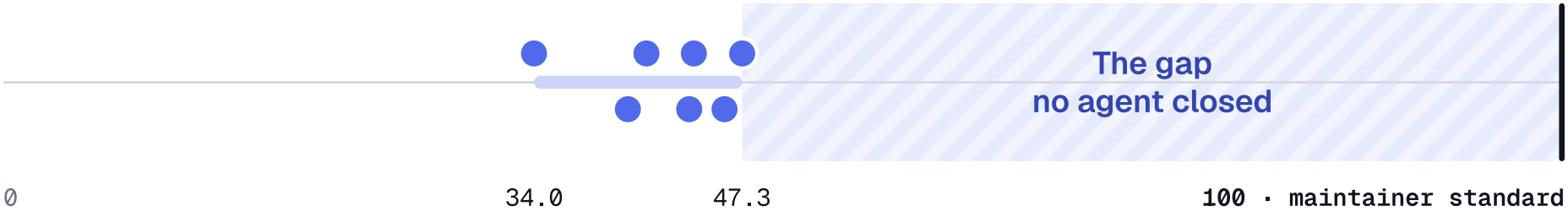
Fabrizio Ferri-Benedetti

CT Smith

We validated the benchmark with rubrics from expert technical writers who are also official maintainers of their open source projects.

02 WHAT THE RESEARCH SHOWS

Every agent fell well short of maintainer standards



7
agents, each a frontier model with a frontier harness

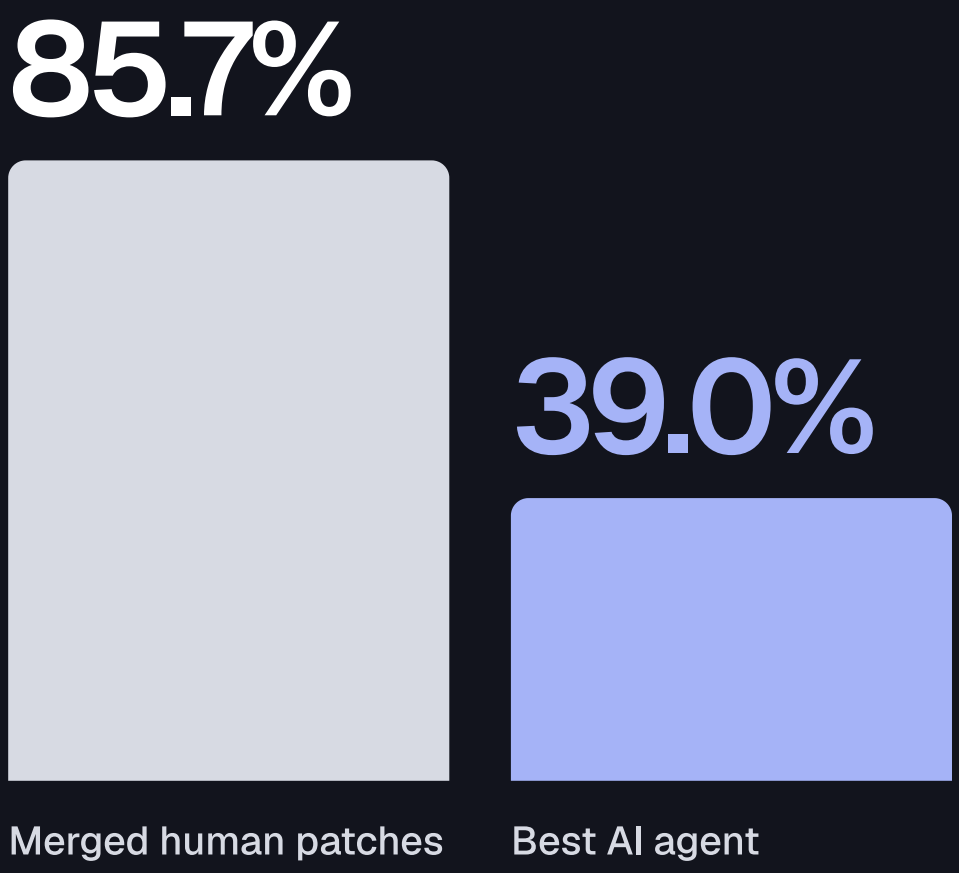
117
held-out docs tasks from open source projects

47.3
The best agent's score

03 WHAT THE RESEARCH SHOWS

How often was a patch free of critical failures?

A critical failure is an error or omission that would mislead readers or block the task.



04 HOW DOCS AGENTS FALL SHORT

They are very eager to help, but sometimes that creates more work

37.1%

Average rate at which agents wrote a patch when no update was needed

DogBench held-out split, 7 agents, 35 no-update items

WHAT AGENTS DOCUMENT THAT READERS DON'T NEED

- Internal refactors and renames
- Performance, test, CI, and dependency changes
- Generated files

05 HOW DOCS AGENTS FALL SHORT

They don't really question the status quo

Legacy style violations

If an old section breaks the style guide, the agent treats it as precedent and keeps breaking it.

Accidental gaps

If a feature was never documented by mistake, the agent assumes the omission was deliberate. When related code changes arrive, it skips the update instead of filling the gap.

From the agent's perspective, deliberate exclusion and an unfilled gap produce the same search result.

06 HOW DOCS AGENTS FALL SHORT

They can be "lazy" on longer tasks if you let them be

They jump to conclusions

They find related material, stop short of the key code change or other decisive evidence, and fill in with a plausible guess.

33.1%

of runs skipped decisive evidence and guessed instead

They focus too narrowly

They edit the first reasonable page and miss the other pages that may be impacted by the same change.

30.1%

of runs stopped at the first plausible page

DogBench trajectory analysis, 1,267 runs

LESSON 1 WORKING WITH AGENTS

Lesson 1 - Help the agent understand your company, your docs, and your workflows

A Conventions

Terminology

The words you use and the ones you avoid

Style guide

How your docs should read

Authoritative sources

Where the truth lives

B Audience

Personas

Who reads your docs

Goals

What they came to do

Common user journeys

The paths they take through the product

C Information architecture

Progressive disclosure

What goes in an overview and what goes deeper

Entry points

Search, navigation, related guides, examples

Content graph

Pages that link, mirror, or must change together

LESSON 2 WORKING WITH AGENTS

Lesson 2 - Ask the agent to do research, and ask for a report with pre-defined fields

GIVE THE AGENT

Relevant sources

Code changes, issues, specs, and support tickets

MCP connections

Live access to trackers, chat, and knowledge bases

Skills

Reusable instructions for your repo, tools, and review steps

ASK FOR

Scope and impact of the change

The old behavior, the new behavior, and who the change affects

Verified boundaries

Conditions, defaults, exceptions, and compatibility limits

Documentation impact map

What's documented today, what isn't, and which other pages the change touches

Open questions and conflicts

What's still uncertain, where sources disagree, and what to check before drafting

LESSON 3 WORKING WITH AGENTS

Lesson 3 - Check that the doc update include the why and when, not just the what and how

Agents describe what a feature does, not the decision it supports.

EXAMPLE FROM DOGBENCH

A Strawberry GraphQL patch explained how to select an older Apollo Federation version. It never said why a reader would want to do that: to upgrade Strawberry while staying compatible with an older Apollo Router or Gateway.

ASK OF EVERY DRAFT

- Who is this for?
- When would they choose it, and why?
- What must they get right?

19.9%

of agent patches lacked audience and purpose framing

26.5%

left out rationale, boundaries, or examples

DogBench audit of 1,267 submissions

LESSON 4 WORKING WITH AGENTS

Lesson 4 - Walk through the reader's whole task

Agents don't understand that a change is part of a larger workflow. Readers need to complete the whole task, start to finish.

WALK THE READER'S PATH

Land	Can they find the right page when they hit a problem?
Set up	Permissions, credentials, versions, enablement
Decide	Which option fits their situation
Do	Every step, in order, with branching if needed
Verify	How they know it worked
Recover	What to do when it didn't

45.5%

of agent patches had a task-completion gap, the most common defect

21.0%

omitted prerequisites

DogBench audit of 1,267 submissions

LESSON 5 WORKING WITH AGENTS

Lesson 5 - Fact-check claims, especially their scopes or applicabilty

Frontier agents don't invent features as much. More often, they take something true in one case and present it as true in every case.

EXAMPLE FROM DOGBENCH

An agent wrote that Helm 4 uses Server-Side Apply by default for installs and upgrades. That holds only for new installations. Releases created with Helm 3 keep client-side apply unless the user switches them.

CHECK KEY CLAIM'S

- Scope
- Defaults
- Versions
- Compatibility
- Exceptions

36.6%

of agent patches had a technical inaccuracy, often a claim stated more broadly than it holds

6.1%

contained fabricated content

DogBench audit of 1,267 submissions

Case study - pantsbuild/pants #23219

AN AGENT WALKS INTO A REPO

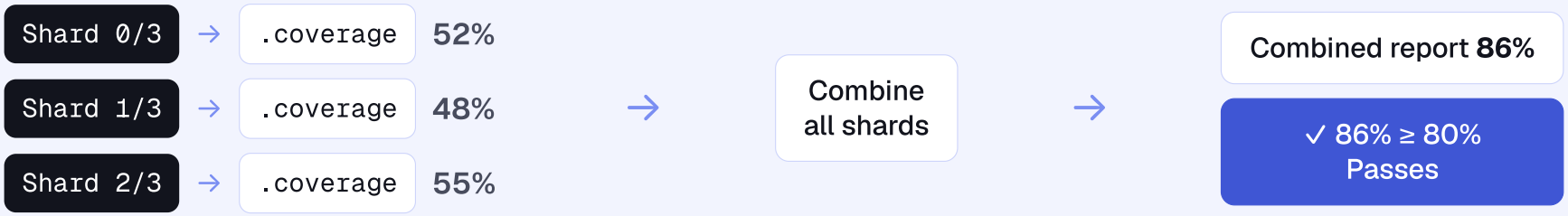
Pants is a build tool that can split Python tests across CI jobs. Each job runs only some tests, so its coverage report describes **that shard**. The team wants to know whether **all tests together** meet its coverage target.

DOCUMENTATION REQUEST

Explain how to collect the raw `.coverage` file from **every** shard, combine the files after the shard jobs finish, and apply the threshold to the **combined** report.

Coverage target: 80%

Illustrative numbers



✗ **Don't check each shard.** Every shard falls short of 80% on its own, even though the project meets its target.

✓ **Check once, after combining.** Only the combined report reflects all tests.

A complete patch changes two pages

docs/docs/python/goals/test.mdx
Full procedure

```
   ## Coverage
   … fail_under and coverage configuration …
+  ### Coverage with test sharding
+  Request the raw report in every shard
+  Save each shard's .coverage file
+  Combine all files after the shard jobs finish
+  Report and check the threshold on the
   combined total
   ## JUnit XML results
```

docs/docs/using-pants/advanced-target-selection.mdx
Caveat + link

```
   ## Sharding the input targets
   … --test-shard=k/N explanation …
+  Each shard reports only part of your
   Python coverage.
+  See Coverage with test sharding to combine them.
```

docs/docs/python/goals/test.mdx
Short note, abridged

```
  ## Coverage
+ :::note Coverage with --shard
+ Each shard only reports coverage for its
  own tests.
+ You will also likely want report to
  include "raw".                                  (1)
+ You must ensure .coveragerc
  sets relative_files.                            (2)
+ :::
```

Missing: save each .coverage → combine → check the total (3)

```
  ## JUnit XML results
```

docs/docs/using-pants/advanced-target-selection.mdx
No change

```
  ## Sharding the input targets

  … --test-shard=k/N explanation …
```

No caveat, no link to the coverage procedure (4)

1. **A required step sounds optional**
   raw produces the .coverage files the combine step needs.
2. **An invented manual step**
   Pants already sets relative_files. The request said otherwise, and the agent didn't check.
3. **No working procedure**
   The threshold still fails in each shard.
4. **Readers on the sharding page never find out**

TURNS 1-3

Doesn't know your docs

FAILURE

Searches `.md` files and initially misses the relevant `.mdx` docs. Eventually finds the Python test page and sharding guide, but never makes a list of pages that need changes.

TURN 4

Jumping to conclusions

FAILURE

The grep result shows Pants setting `relative_files = true`. The agent still tells readers they must set it themselves. Mentions `raw`, but never verifies what it produces or when the threshold runs.

TURNS 5-8

Lacking understanding of user's workflow

FAILURE

Rereads the coverage and sharding pages. Its reasoning shows that it never maps how a CI user gets from partial shard results to one checked coverage total.

TURNS 9-10

Stops after the first edit

FAILURE

Adds one note to the coverage page. The patch has no file collection or combine steps, no final threshold check, and no link from the sharding guide.

An effective agent trajectory can look like -

STEPS	STEP	WHAT IT DOES	LESSON
1-2	**Map the docs**	Reads the Python test coverage page and the sharding guide.	1 · Give the agent your docs architecture
3-4	**Settle the facts**	Reads `coverage_py.py` and `test.py`. Separates combining within one invocation from combining across CI jobs.	2 · Require evidence before drafting 5 · Check the scope of claims
5	**Define the reader's goal**	An accurate project-wide threshold from separately run shards.	3 · State who the page is for and why
6-7	**Write both surfaces**	Adds `### Combining coverage from sharded test runs` to the test page, plus a caveat and link on the sharding page.	1 · Respect entry points 4 · Walk the whole task
8	**Inspect the result**	Rereads the edited section and checks the link and wording.	5 · Verify the final claims

17 END

Thank you!

frances@promptless.ai. Or find me on WTD Slack