

ptbr-writer: controlled technical Portuguese for people and agents

Agentic writing in Brazilian Portuguese is a cheap abstraction

Matheus Hoffmann [ORCID](#)

[@h0ffmann](#)

Rob Kler Soares da Silva Junior [ORCID](#)

[@robkler](#)

October 2026, not peer reviewed · v0.3.0, preprint · rev. 6e9da8d+130



OUROBORIC.DEV

Abstract. ptbr-writer rewrites technical text in Brazilian Portuguese for a single reading: explicit subject, one proposition per sentence and consistent terms. Its seed is Português Técnico Controlado (PTC), by kayquer: a Claude Code skill with controlled-language rules in Portuguese for a reader that may be a model. ptbr-writer adapts the 8 rules of PTC and adds 5, for 13 rules in total, PTW-1 to PTW-13. Each request goes through one of 5 levels of rigor, from *estrito* to *smart-note*. For text with sources, the *cientifico* skill adds 12 rules, CIE-1 to CIE-12. The same rules reach the writer by three paths: a Claude Code plugin, an installer for OpenCode and a browser extension for Chrome and Firefox. The extension treats a review as a decision. Besides models that generate text (Claude, DeepSeek or a local model through Ollama), it accepts a local decision model, served by Ollaya, which says which rule each sentence breaks, with a probability, and generates no text. The runtime tools are static Rust binaries pinned by sha256, and each claim that was not checked carries the mark Δ . This paper describes the rules, the levels, the architecture and the evidence policy. It reports no measure of accuracy: the regression cases in *evals/* are written and were not run, and the latency and accuracy of the local models on pt-BR were not measured. The code is under MPL-2.0 since v0.3.0.

Keywords: technical writing; Brazilian Portuguese; controlled language; coding agents; decision models; reproducibility.

1 Introduction

A runbook is read by the person who runs the procedure and, more and more, by an agent that runs it. Neither can stop to ask “did you mean X or Y?”. PTC, by kayquer, wrote this need down as a Claude Code skill, with 8 rules for Brazilian Portuguese [1]. ptbr-writer starts from that seed.

The thesis of this paper is that agentic writing in Brazilian Portuguese is a cheap abstraction. It is cheap because the whole abstraction is text: one file of rules in Markdown, which any agent that reads skills loads, and a small binary that decides what to ask before it writes. There is no trained model, no fine-tuning and no server. The paper supports the thesis with three pillars: tailored Brazilian Portuguese (sections 2 and 7), portability (sections 3 to 5) and a harness that improves itself (section 6). Section 8 shows what was measured of cost, and section 9, what was not.

The abstraction is also cheap because it stands on an open-source foundation. Ollama (MIT) serves, on the user’s machine, the model that writes the replacement in the extension [4].

Ollaya (Apache-2.0) serves the decision models [3]. OpenCode (MIT) is the open coding agent where the skills run outside Claude Code [5]. OpenCodeReview (Apache-2.0), by Alibaba, is the code review tool the project will use to review its own pull requests with a local model [6]; that integration is not done yet $\triangle$.

2 Tailored Brazilian Portuguese

PTC notes that English requires the subject, so ASD-STE100, the controlled language of the aerospace industry, never needed a rule for it [1]. Portuguese lets the writer drop the subject. The first rule of ptbr-writer, PTW-1, asks for an explicit subject and avoids the passive **-se**, **o mesmo** (“the same”) and ambiguous third-person clitics (**SKILL.md**). A sentence such as “deve-se verificar os logs” (“one must check the logs”) becomes “verifique os logs” (“check the logs”).

Rules PTW-1 to PTW-8 adapt the 8 of PTC: one proposition per sentence, explicit modality and quantity, full verbs, flat syntax, frozen terms, numbers with units and the spelling of the 1990 Orthographic Agreement, with care for technical prefixes such as **microserviço**. Rules PTW-9 to PTW-13 add an evidence policy: every claim has a mark, no reference is invented, a number has a unit and a source, there is no priority claim, and fact stays apart from analysis.

The rigor changes with the reader. The levels are **estrito** (runbooks and output for agents), **descritivo** (system documentation), **leve** (announcements), **científico** (text with sources) and **smart-note** (notes for a second brain). The **científico** level loads the **cientifico** skill, which applies the 13 PTW rules and adds 12 CIE rules: nothing invented, a source read before it is cited, what was searched for and not cited, a DOI as a link, code as **file:line** at a commit, and the final decision with the human. The **smart-note** level follows the Zettelkasten: atomic notes, each with an identifier and links to the others, and it combines with another level (**smart-note** + **estrito**).

The same request can come out in pt-BR and in ASD-STE100, the simplified technical English, one text per language. Every text the plugin writes carries its reading time, computed by the binary, at 238 words per minute [2]. That rate is the one for English; pt-BR uses the same, with no scaling $\triangle$.

There are also planned author levels, outside **SKILL.md**: **machado-writer** and **saramago-writer**. They imitate the style and never copy text. **REUSE.md** sets the rules: the author must be in the public domain, and **machado-writer** always cites Machado de Assis. For Saramago, the repository already has a prose linter in Vale (**lint/saramago**), which checks the author’s punctuation and runs in **just quality**.

3 Plugin architecture

The Claude Code plugin has three parts. The first is **SKILL.md**, the rules. The second is a hook on the **UserPromptSubmit** event: on each message, it runs the binary **ptbr ask-levels**, with a 5 s timeout. A request that starts with **ptbr** gets one instruction; any other request gets nothing. The third is the configuration, in a **ptbr-writer.toml**.

The first time, with no such file, Claude asks once, in multiple choice: the levels, the formats (**txt**, **md**, **pdf**, **notes**), the folder and the languages. The answers go into the file, and the plugin does not ask again. The configuration has layers, each over the one before: the plugin’s defaults, **~/.config/ptbr-writer/**, the repository’s **./ptbr-writer.toml** and,



Figure 1: **Claude Code plugin flow.** Takeaway: you write `ptbr: ...`, and the skill writes one text per level and per language. How to read: left to right; the top lane is the path of every request, the bottom one is the configuration and the files, and the orange border marks where you come in and what you get back. Not shown: the rules, the `smart-note` level and the `ptbr reconfigurar` request. Evidence: `docs/archify/en/em-um-minuto.json`.

last, the request (`ptbr leve: ... uses leve`). An unknown value or a folder outside the root is ignored, with a warning. The files come out with one name, `ptbr-writer-<UTC date>-<goal>-<level>.<ext>`, and the same stamp holds for every file of one request.

A request with no level goes through routing. The `ptbr-routing` skill reads `routes.tsv`, a table of skill, level and words: “runbook”, “procedimento” and “incidente” lead to `estrito`; “arquitetura” and “API” lead to `descritivo`. The binary `ptbr-route` picks the skill and the level before a skill loads, and only the chosen one enters the context. A new level in `SKILL.md` needs a row in the table, or just `quality` fails.

4 Portability

The rules live in one file, `SKILL.md`, with 807 words (`wc -w`), and the `cientifico` skill has 1,087. An agent that reads skills loads these files with no change. `ptbr-writer` serves two audiences, by different paths.

Developers use `ptbr-writer` in the terminal and in the repository, through a coding agent:

- **Claude Code:** a plugin, installed with one command and pinned by tag. A repository pins the version in its own `.claude/settings.json`, and everyone who opens it uses the same one.
- **OpenCode:** a script copies the skills as files and installs a plugin that does what the Claude Code hook does. It was tested on OpenCode 1.19.0: the skills load and the instruction reaches the model. That the model answers with the `question` tool was not measured $\triangle$. The repository also starts OpenCode with a local Qwen3.6-27B, through `llama.cpp` (just `opencode-qwen`).

Non-developers use the Chrome extension (and the Firefox 128+ one), in the field where they already write: WhatsApp Web, Gmail and GitHub are on by default. The extension underlines what the level asks to change, and nothing changes without a click on **Aceitar** (accept). Today it is still installed from the source: the roadmap lists it as in progress, not yet on the Chrome Web Store.

What runs on the machine is a static Rust binary, `ptbr`, of 570,112 bytes on `linux-x86_64`, with a twin for `linux-aarch64`. `tools.lock` pins the picasso commit that builds it and the

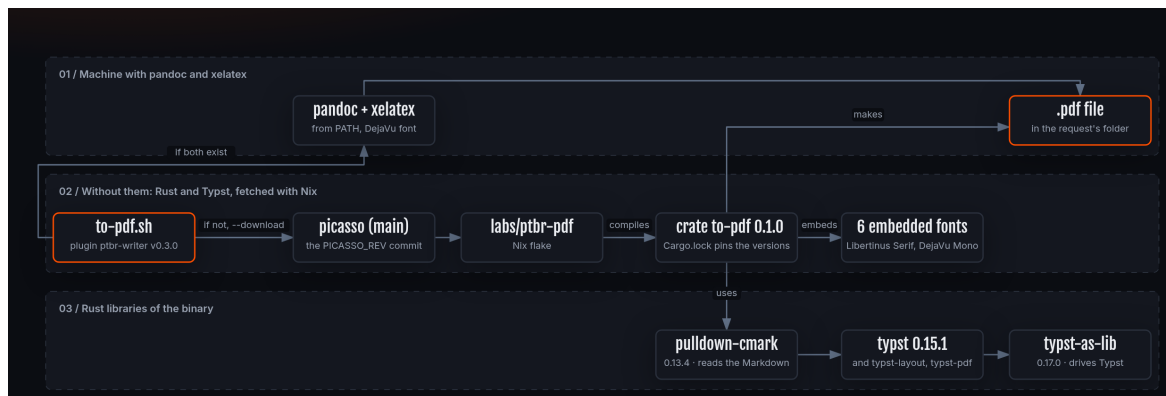


Figure 2: **Where the .pdf comes from.** Takeaway: the .pdf comes from the machine’s pandoc, when it is there with XeLaTeX; otherwise, from a Rust binary that Nix builds from a pinned picasso commit, with each library at the version of `Cargo.lock`. How to read: left to right; the middle lane is the path without pandoc, the bottom one holds the binary’s Rust libraries, and the top one, the local path and the finished file. Not shown: the dependencies of the dependencies and Nix itself. Evidence: `plugins/ptbr-writer/scripts/to-pdf.sh` and `docs/archify/en/pdf-dependencias.json`.

sha256 of each binary. There is no Python and no Node at run time. The .pdf comes from the machine’s pandoc, when pandoc and XeLaTeX are there. Without them, and only with the user’s permission, Nix builds `to-pdf` 0.1.0, a Rust binary of picasso’s `labs/ptbr-pdf`, at a pinned commit. Its `Cargo.lock` pins `pulldown-cmark` 0.13.4, which reads the Markdown, `Typst` 0.15.1, which sets the page, and `typst-as-lib` 0.17.0, which joins the two; six fonts are embedded (Libertinus Serif and DejaVu Mono). The first build, on a fresh GitHub runner with 4 cores, took 892 s; from the second time on, Nix reuses what it built.

5 The review as a decision

The extension waits for 1 s without typing and only checks fields with 3 words or more. Password fields are left out, Gmail’s quoted history is not sent and, above 4,000 characters, only the cursor’s paragraph goes. The suggestions come back as underlines drawn in a shadow DOM, outside the editor: the page’s DOM is not touched. Each underline opens a card with **Aceitar** (accept) and **Ignorar** (ignore).

The extension talks to Claude and DeepSeek, with the user’s key, and to two local servers, with no key. Ollama serves a model that writes the replacement. Ollaya serves decision models, which answer closed questions with probabilities and never generate text [3]. For the extension, pointing out a broken rule is a decision. For each sentence of the field, up to 32, it sends `/api/decide` one choice question, “which rule does this sentence break?”, with `nenhuma` (none) and the level’s rules as the answers. The sentence is the state, and each option is only the rule’s code and title. A rule at 50% or more becomes an underline; the card names the rule and the probability, and only has **Ignorar**, because the decision model does not write the corrected sentence.

The two servers also work together, as independent switches. With Ollama chosen, the switch **Ollaya escolhe o estilo** (Ollaya picks the style) makes Ollaya read the field and pick the tone of the conversation: concise, Socratic or friendly. Ollama writes the replacements in that tone, and the card shows the chosen tone. When Ollaya is down, Ollama writes alone.

The default decision model comes from a survey in the repository itself (`docs/SURVEY_DECISION_MODEL.md`),

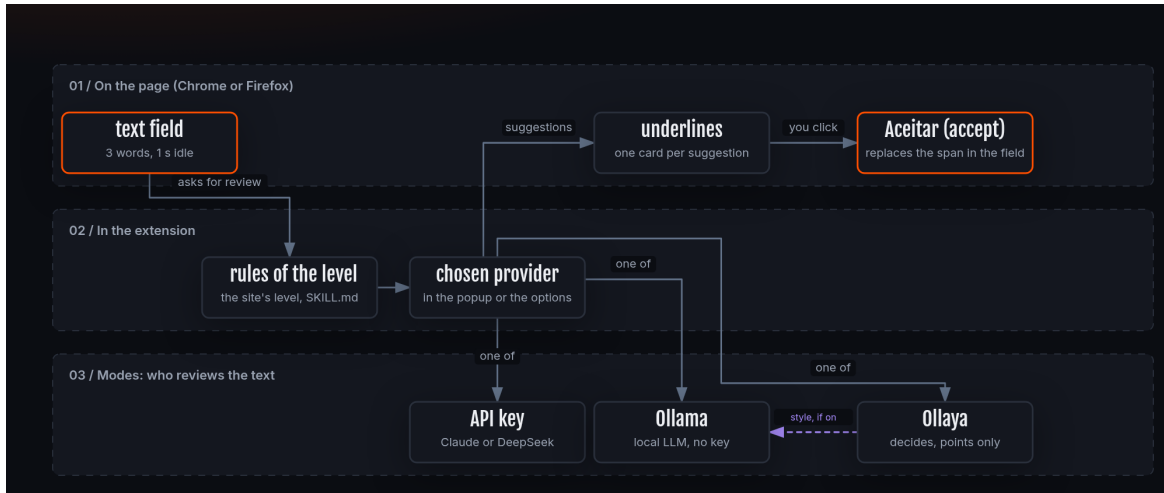


Figure 3: **Browser extension flow.** Takeaway: the extension sends the field’s text, with the rules of the site’s level, to a provider you pick, and only your click on **Aceitar** changes the text. How to read: top to bottom; the top lane is the page, the middle one is the extension, the bottom one holds the modes, and the dashed arrow is the switch where Ollaya picks the style and Ollama writes. Not shown: the **Ignorar** button, the 4,000-character limit and the error badge in the field. Evidence: `docs/archify/en/extensao-fluxo.json`.

with the numbers of Ollaya Bench, on tasks in English and German:

Model	Use	CPU, 5 questions	Calibration error	Reads Portuguese
<code>decima</code>	default, CPU	1,212.5 ms	0.052	the card lists pt
<code>laya:multilingual</code>	CPU, faster	515.5 ms	0.155	the card lists por
<code>pplx-decider</code>	GPU, 10.6 GB	5,533.6 ms	0.016	not stated $\triangle$

`decima` is the default because it is the CPU candidate whose card lists Portuguese and that has the lowest calibration error. None of these numbers was measured on pt-BR or on the PTW rules $\triangle$. The extension has 169 unit tests (Vitest, on 22 October 2026) and end-to-end tests in Chromium, with a fake server and no real API.

6 A harness that improves itself

In this paper, RSI has one meaning only, the operational one the repository uses: the label `rsi` marks “a mistake the harness let through, whose fix changes the loop so that it does not come back” (ISSUE_LABELS.md). The loop is **just quality**, which CI runs on each pull request. Four cases from October 2026 show the pattern, all with the agent itself as the author of the mistake:

1. An edit put two roadmap rows at the top of the README, outside the table. **just quality** passed, because it only checked that each piece’s name appeared somewhere. The fix added a structure check: no table row before the first heading.
2. The README reached ~31 minutes of reading. It was cut to ~8, and **just quality** now fails above 10, by the same binary that measures the user’s text.

3. The English README showed the figure with Portuguese labels. Each figure got an English twin, and the check fails when a README shows the figure of the other language.
4. The release recipe opened an editor in a shell with no terminal and stopped halfway. Without a terminal, it now does not open the editor and says what to do.

In each case, the fix to the file came with a new check, and the check is what stays. The project does not measure whether this loop reduces mistakes over time $\triangle$.

The loop stands on two rules of **AGENTS.md**. The first is about evidence: the mark $\triangle$ flags a claim that was not checked, nobody removes it without doing the check, and a number without a command or a source does not enter a table. The second is determinism: the versions are pinned by the plugin's tag, the picasso commit, **flake.lock** and **Cargo.lock**, and the same input gives the same output. **just pins** checks that the submodule and the flake point at the same commit. This PDF follows the same rules: the logos are fetched at a pinned commit, with their sha256 checked, and **just whitepaper** fails above 8 pages.

7 Bilingual and citable

Portuguese comes first, and English is the mirror. **README.md** and **README.en.md** have the same sections and the same facts, and a change to one changes the other in the same pull request. Each figure has an English twin, with the same boxes and arrows and only the words translated; **just figure-check** compares the two drawings and fails when a README shows the figure of the other language. Pull requests follow the **bipr** skill: the body comes from the commits, pt-BR first with the English mirror, each half with the rules of its language (PTW at the **estrito** level for pt-BR; short sentences and active voice, in the spirit of ASD-STE100, for English). This paper also comes in both languages, with the same sections.

The project is citable. One version ties together the tag **vX.Y.Z**, the **version** of **plugin.json** and the heading of **CHANGELOG.md**; **just release-check** compares the three before a release, and CI runs it when the tag arrives. From v0.3.0 on, the release goes to Zenodo, which gives it a DOI. The record is a preprint, with a description in pt-BR and in English, and it cites PTC with the relation **isDerivedFrom**; **CITATION.cff** carries the authors with their ORCID. **just release** checks Zenodo's webhook before the tag and waits for the DOI after it $\triangle$: that path has not run once yet.

The project is also transparent. Every fact on the site and in the README comes from a file in the repository or a pinned commit. The site is written in Gleam, with Lustre, and prerendered. The rules, the levels, the recipes, the roadmap, each rule's origin and the related projects come from **SKILL.md**, the **justfile** and the tables of **site/data/**, and none of them is typed into the site's code; a new rule without its origin row breaks the build. The site also publishes the code graph of this repository and the extension's, built on each build by picasso's **code-graph**, with tree-sitter and no model, and each figure as an interactive Archify page, pinned at a commit.

8 What was measured of cost

Three real requests to the plugin, with the model **claude-opus-5-5**, at plugin commit **ca0e859**, on 21 October 2026, were measured by **claude -p --output-format json (docs/eli5/RESULTS.tsv)**:

Request	Level	Tokens in total	Read from cache	Output	Cost (US\$)
deploy runbook	estrito	350,529	301,876	8,184	0.5477
announcement	leve	184,688	155,591	2,401	0.2927
explain to a child	descritivo	232,366	200,269	3,434	0.3380

Between 84% and 86% of each request’s tokens were read from the cache. The `bipr` skill, which writes the bilingual body of a pull request, was measured in the transcript of the session that opened PR #49: its two steps added up to 4 input tokens, 235,576 read from the cache and 1,525 of output (`docs/bipr/exemplo-pr-49.md`). That document does not convert tokens into dollars $\triangle$.

The abstraction itself has no training or server cost. The cost per request is that of the model the user picks. With a local decision model, there is no charge per request, and the cost is the CPU time of section 5.

9 Limitations

The regression cases in `evals/` are written and were not run by a harness, so the accuracy of the rules was not measured. The extension was tested by hand on WhatsApp Web, with a real key; Gmail and GitHub were not tested with the real editor. The latency and accuracy of Ollama and Ollaya on pt-BR were not measured, and Ollaya’s 50% threshold was not tuned. The reading rate for pt-BR is the one for English. The four cases of section 6 are from one project and one month. The three requests of section 8 are a sample of three, with one model. The tag `v0.2.0` was never created; `v0.3.0` is the first tag since `v0.1.0`.

10 References

1. kayquer. *Português Técnico Controlado (PTC)*. Claude Code skill, MIT license. <https://github.com/kayquer/portugues-tecnico-controlado/tree/cfdffad0ed98cdebfb7f591481889090a4f4e65b>, commit `cfdffad`, read on 22 October 2026.
2. Brysbaert, M. (2019). How many words do we read per minute? A review and meta-analysis of reading rate. *Journal of Memory and Language*, 109, 104047. <https://doi.org/10.1016/j.jml.2019.104047>
3. ollaya-dev. *Ollaya*. Apache-2.0 license. <https://github.com/ollaya-dev/ollaya/tree/39619accce0f5836dbeb60d2b14f5a056c2459ce>, commit `39619ac`, read on 21 October 2026.
4. Ollama. *Ollama*. MIT license. <https://github.com/ollama/ollama/tree/efabf6bc450454dc856c1f799aac7ea865e3a4cf>, commit `efabf6b`, read on 22 October 2026.
5. Anomaly. *OpenCode: the open source coding agent*. MIT license. <https://github.com/anomalyco/opencode/tree/c93aaae37157d785f76a8e9a9243a6560e286468>, commit `c93aaae`, read on 22 October 2026.
6. Alibaba. *Open Code Review*. Apache-2.0 license. <https://github.com/alibaba/open-code-review/tree/375f64cd6f67a48a135da83cfcf59c52669a09cb>, commit `375f64c`, read on 22 October 2026.

The code, the rules and this text are at <https://github.com/h0ffmann/ptbr-writer>, under MPL-2.0. The template of this PDF comes from the Paper Fabric of h0ffmann/closed-source, commit 563c339 (MIT).