

ptbr-writer: português técnico controlado para pessoas e agentes

Escrita agêntica em português brasileiro é uma abstração barata

Matheus Hoffmann [ORCID](#)

[@h0ffmann](#)

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

[@robkler](#)

Outubro de 2026, sem revisão por pares · v0.3.0, preprint · rev. e8146fd+137



OUROBORIC.DEV

Resumo. O ptbr-writer reescreve texto técnico em português brasileiro para uma leitura única: sujeito explícito, uma proposição por frase e termos consistentes. A semente é o Português Técnico Controlado (PTC), de kayquer: uma skill do Claude Code com regras de linguagem controlada em português para um leitor que pode ser um modelo. O ptbr-writer adapta as 8 regras do PTC e acrescenta 5, num total de 13 regras, PTW-1 a PTW-13. Cada pedido passa por um de 5 níveis de rigor, de **estrito** a **smart-note**. Para texto com fontes, a skill **científico** acrescenta 12 regras, CIE-1 a CIE-12. As mesmas regras chegam por três caminhos: um plugin do Claude Code, um instalador para o OpenCode e uma extensão de navegador para Chrome e Firefox. A extensão trata a revisão como uma decisão. Além de modelos que geram texto (Claude, DeepSeek ou um modelo local pelo Ollama), ela aceita um modelo de decisão local, servido pelo Ollaya, que diz qual regra cada frase quebra, com uma probabilidade, sem gerar texto. As ferramentas de execução são binários Rust estáticos, fixados por sha256, e cada afirmação não conferida leva a marca Δ . Este artigo descreve as regras, os níveis, a arquitetura e a política de evidência. Ele não relata medida de acerto: os casos de regressão em **evals/** estão escritos e não foram rodados, e a latência e o acerto dos modelos locais em pt-BR não foram medidos. O código está sob a MPL-2.0 desde a v0.3.0.

Palavras-chave: escrita técnica; português brasileiro; linguagem controlada; agentes de programação; modelos de decisão; reprodutibilidade.

1 Introdução

Um runbook é lido por quem executa o procedimento e, cada vez mais, por um agente que o executa. Nenhum dos dois pode parar para perguntar “você quis dizer X ou Y?”. O PTC, de kayquer, escreveu essa exigência como uma skill do Claude Code, com 8 regras para o português do Brasil [1]. O ptbr-writer parte dessa semente.

A tese deste artigo é que escrita agêntica em português brasileiro é uma abstração barata. Barata porque a abstração inteira é texto: um arquivo de regras em Markdown, que qualquer agente que lê skills carrega, e um binário pequeno que decide o que perguntar antes de escrever. Não há modelo treinado, ajuste fino nem servidor. O artigo sustenta a tese com três pilares: o português brasileiro sob medida (seções 2 e 7), a portabilidade (seções 3 a 5) e um harness que melhora a si mesmo (seção 6). A seção 8 mostra o que se mediu de custo, e a seção 9, o que não se mediu.

A abstração é barata também porque se apoia numa fundação de código aberto. O Ollama (MIT) serve, na máquina do usuário, o modelo que escreve a troca na extensão [4]. O Ollaya

(Apache-2.0) serve os modelos de decisão [3]. O OpenCode (MIT) é o agente de programação aberto em que as skills rodam fora do Claude Code [5]. O OpenCodeReview (Apache-2.0), da Alibaba, é a ferramenta de revisão de código que o projeto vai usar para revisar os próprios pull requests com um modelo local [6]; essa integração ainda não foi feita $\triangle$.

2 Português brasileiro sob medida

O PTC observa que o inglês obriga o sujeito, e por isso o ASD-STE100, a linguagem controlada da indústria aeroespacial, nunca precisou de uma regra para ele [1]. O português permite omitir o sujeito. A primeira regra do ptbr-writer, PTW-1, pede o sujeito explícito e evita o `-se` apassivador, o `mesmo` e os clíticos ambíguos de terceira pessoa (SKILL.md). Uma frase como “deve-se verificar os logs” vira “verifique os logs”.

As regras PTW-1 a PTW-8 adaptam as 8 do PTC: uma proposição por frase, modalidade e quantidade explícitas, verbo pleno, sintaxe plana, termos congelados, números com unidade e a ortografia do Acordo de 1990, atenta a prefixos técnicos como `microserviço`. As regras PTW-9 a PTW-13 acrescentam uma política de evidência: toda afirmação tem marca, nenhuma referência é inventada, número tem unidade e origem, não há alegação de pioneirismo e fato fica separado de análise.

O rigor muda com o leitor. Os níveis são `estrito` (runbooks e saídas para agentes), `descritivo` (documentação de sistema), `leve` (comunicados), `científico` (texto com fontes) e `smart-note` (notas para um segundo cérebro). O nível `científico` carrega a skill `cientifico`, que aplica as 13 regras PTW e acrescenta 12 regras CIE: nada inventado, fonte lida antes de citada, o que foi procurado e não citado, DOI como link, código como `arquivo:linha` num commit, e a decisão final com o humano. O nível `smart-note` segue o Zettelkasten: notas atômicas, com um identificador e ligações entre si, e combina com outro nível (`smart-note + estrito`).

O mesmo pedido pode sair em pt-BR e em ASD-STE100, o inglês técnico simplificado, um texto por idioma. Todo texto que o plugin gera traz o tempo de leitura, calculado pelo binário, com 238 palavras por minuto [2]. Essa taxa é a do inglês; o pt-BR usa a mesma, sem escala $\triangle$.

Há também níveis de autor planejados, fora da SKILL.md: `machado-writer` e `saramago-writer`. Eles imitam o estilo e nunca copiam texto. O REUSE.md fixa as regras: o autor tem de estar em domínio público, e o `machado-writer` sempre cita Machado de Assis. Para Saramago, o repositório já tem um linter de prosa no Vale (`lint/saramago`), que confere a pontuação do autor e roda no `just quality`.

3 Arquitetura do plugin

O plugin do Claude Code tem três partes. A primeira é a SKILL.md, as regras. A segunda é um hook no evento `UserPromptSubmit`: a cada mensagem, ele roda o binário `ptbr ask-levels`, com prazo de 5 s. Um pedido que começa com `ptbr` recebe uma instrução; qualquer outro pedido não recebe nada. A terceira é a configuração, num `ptbr-writer.toml`.

Na primeira vez, sem esse arquivo, o Claude pergunta uma vez só, em múltipla escolha: os níveis, os formatos (`txt`, `md`, `pdf`, `notes`), a pasta e os idiomas. As respostas ficam no arquivo, e o plugin não pergunta mais. A configuração tem camadas, cada uma sobre a anterior: os padrões do plugin, `~/config/ptbr-writer/`, o `./ptbr-writer.toml` do repositório e, por último, o pedido (`ptbr leve: ... usa leve`). Um valor desconhecido ou uma pasta fora



Figura 1: **Fluxo do plugin do Claude Code.** Em uma frase: você escreve `ptbr: ...`, e a skill escreve um texto por nível e por idioma. Como ler: da esquerda para a direita; a faixa de cima é o caminho de todo pedido, a de baixo é a configuração e os arquivos, e a borda laranja marca onde você entra e o que recebe. Fora da figura: as regras, o nível `smart-note` e o pedido `ptbr reconfigurar`. Evidência: `docs/archify/em-um-minuto.json`.

da raiz é ignorado, com aviso. Os arquivos saem com um nome só, `ptbr-writer-<data UTC>-<objetivo>-<nível>.<ext>`, e o mesmo carimbo vale para todos os arquivos de um pedido.

Um pedido sem nível passa pelo roteamento. A skill `ptbr-routing` lê `routes.tsv`, uma tabela de skill, nível e palavras: “runbook”, “procedimento” e “incidente” levam ao **estrito**; “arquitetura” e “API” levam ao **descritivo**. O binário `ptbr-route` escolhe a skill e o nível antes de carregar uma skill, e só a escolhida entra no contexto. Um nível novo na `SKILL.md` precisa de uma linha na tabela, ou o `just quality` falha.

4 Portabilidade

As regras ficam num arquivo, `SKILL.md`, com 807 palavras (`wc -w`), e a skill `cientifico` tem 1 087. Um agente que lê skills carrega esses arquivos sem adaptação. O `ptbr-writer` atende dois públicos, por caminhos diferentes.

Quem programa usa o `ptbr-writer` no terminal e no repositório, por um agente de programação:

- **Claude Code:** um plugin, instalado com um comando e fixado por tag. Um repositório fixa a versão no próprio `.claude/settings.json`, e todos que o abrem usam a mesma.
- **OpenCode:** um script copia as skills como arquivos e instala um plugin que faz o que o hook do Claude Code faz. Ele foi testado no OpenCode 1.19.0: as skills carregam e a instrução chega ao modelo. Que o modelo responda com a ferramenta `question` não foi medido Δ . O repositório também sobe o OpenCode com o Qwen3.6-27B local, pelo `llama.cpp` (`just opencode-qwen`).

Quem não programa usa a extensão do Chrome (e do Firefox 128+), no campo onde já escreve: WhatsApp Web, Gmail e GitHub vêm ligados de início. A extensão sublinha o que o nível pede para mudar, e nada muda sem o clique em **Aceitar**. Hoje ela ainda é instalada a partir do código: o roadmap a registra como em andamento, ainda fora da Chrome Web Store.

O que roda na máquina é um binário Rust estático, `ptbr`, com 570 112 bytes em `linux-x86_64`, e um par para `linux-aarch64`. O `tools.lock` fixa o commit do picasso

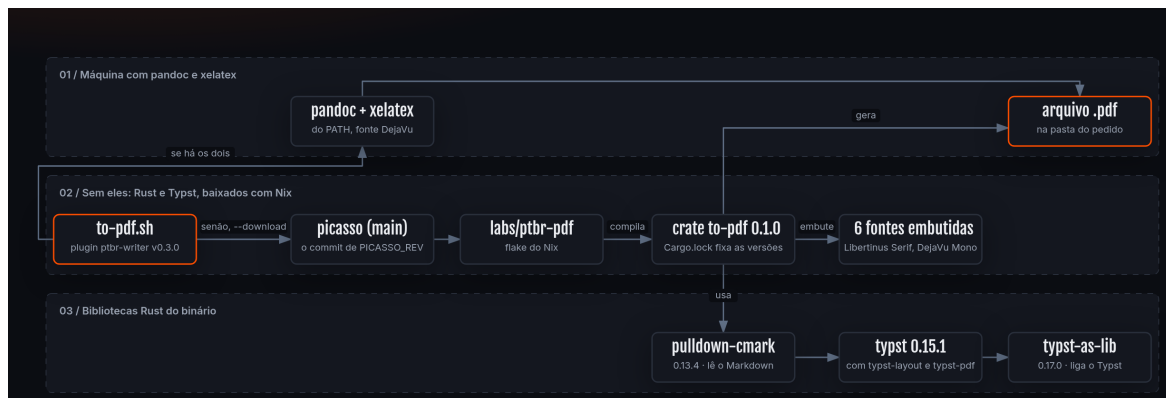


Figura 2: **De onde vem o .pdf.** Em uma frase: o .pdf sai do pandoc da máquina, se ele existir com o XeLaTeX; senão, de um binário Rust que o Nix compila a partir de um commit fixo do picasso, com cada biblioteca na versão do Cargo.lock. Como ler: da esquerda para a direita; a faixa do meio é o caminho sem pandoc, a de baixo são as bibliotecas Rust do binário, e a de cima, o caminho local e o arquivo pronto. Fora da figura: as dependências das dependências e o Nix em si. Evidência: `plugins/ptbr-writer/scripts/to-pdf.sh` e `docs/archify/pdf-dependencias.json`.

que o compila e o sha256 de cada binário. Não há Python nem Node no tempo de execução. O .pdf sai do pandoc da máquina, se houver pandoc e XeLaTeX. Sem eles, e só com a permissão do usuário, o Nix compila o `to-pdf` 0.1.0, um binário Rust do `labs/ptbr-pdf` do `picasso`, num commit fixo. O `Cargo.lock` dele fixa o `pulldown-cmark` 0.13.4, que lê o Markdown, o `Typst` 0.15.1, que compõe a página, e o `typst-as-lib` 0.17.0, que liga os dois; seis fontes vão embutidas (Libertinus Serif e DejaVu Mono). A primeira compilação, num runner novo do GitHub com 4 núcleos, levou 892 s; da segunda vez em diante, o Nix reusa o que já compilou.

5 A revisão como decisão

A extensão espera 1 s sem digitação e só confere campos com 3 palavras ou mais. Campos de senha ficam de fora, o histórico citado do Gmail não é enviado e, acima de 4 000 caracteres, só vai o parágrafo do cursor. As sugestões voltam como sublinhados desenhados num shadow DOM, fora do editor: o DOM da página não é tocado. Cada sublinhado abre um cartão com **Aceitar** e **Ignorar**.

A extensão fala com Claude e DeepSeek, com a chave do usuário, e com dois servidores locais, sem chave. O Ollama serve um modelo que escreve a troca. O Ollaya serve modelos de decisão, que respondem perguntas fechadas com probabilidades e nunca geram texto [3]. Para a extensão, apontar uma regra quebrada é uma decisão. Para cada frase do campo, até 32, ela manda ao `/api/decide` uma pergunta de escolha, “qual regra esta frase quebra?”, com **nenhuma** e as regras do nível como respostas. A frase é o estado, e cada opção é só o código e o título da regra. Uma regra com 50% ou mais vira um sublinhado; o cartão diz a regra e a probabilidade, e só tem **Ignorar**, porque o modelo de decisão não escreve a frase corrigida.

Os dois servidores também andam juntos, como chaves independentes. Com o Ollama escolhido, a chave **Ollaya escolhe o estilo** faz o Ollaya ler o campo e escolher o tom da conversa: conciso, socrático ou amigável. O Ollama escreve as trocas nesse tom, e o cartão mostra o tom escolhido. Com o Ollaya fora do ar, o Ollama escreve sozinho.

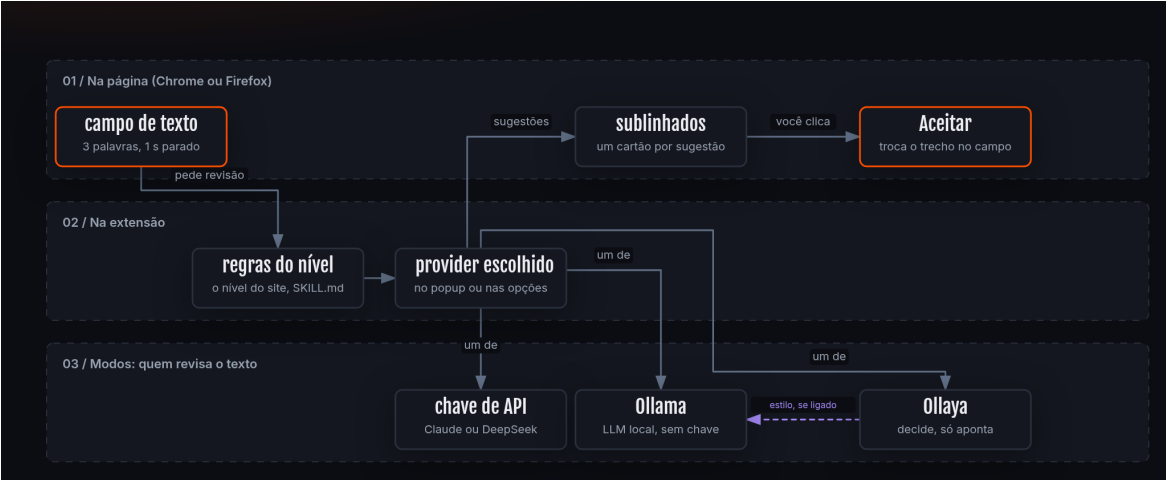


Figura 3: **Fluxo da extensão de navegador.** Em uma frase: a extensão manda o texto do campo, com as regras do nível do site, a um provider que você escolhe, e só o seu clique em **Aceitar** muda o texto. Como ler: de cima para baixo; a faixa de cima é a página, a do meio é a extensão, a de baixo são os modos, e a seta tracejada é a chave em que o Ollaya escolhe o estilo e o Ollama escreve. Fora da figura: o botão **Ignorar**, o limite de 4 000 caracteres e o selo de erro no campo. Evidência: docs/archify/extensao-fluxo.json.

O modelo de decisão padrão vem de uma pesquisa no próprio repositório (docs/SURVEY_DECISION_MODEL.md), com os números do Ollaya Bench, em tarefas em inglês e alemão:

Modelo	Uso	CPU, 5 perguntas	Erro de calibração	Lê português
decima	padrão, CPU	1 212,5 ms	0,052	a ficha lista pt
laya:multilingual	CPU, mais rápido	515,5 ms	0,155	a ficha lista por
pplx-decider	GPU, 10,6 GB	5 533,6 ms	0,016	não declarado △

O decima é o padrão porque é o candidato de CPU com a ficha em português e o menor erro de calibração. Nenhum desses números foi medido em pt-BR nem nas regras PTW △. A extensão, Rust compilado para WebAssembly, tem 113 testes unitários (cargo test, em 22 de outubro de 2026) e 32 testes de ponta a ponta no Chromium, com um servidor falso e nenhuma API real.

6 Um harness que melhora a si mesmo

Neste artigo, RSI tem um sentido só, o operacional que o repositório usa: o rótulo rsi marca “um erro que o harness deixou passar; a correção muda o loop para que ele não volte” (ISSUE_LABELS.md). O loop é o just quality, que o CI roda em cada pull request. Quatro casos de outubro de 2026 mostram o padrão, todos com o próprio agente como autor do erro:

1. Uma edição pôs duas linhas do roadmap no topo do README, fora da tabela. O just quality passou, porque só conferia se o nome de cada peça aparecia em algum lugar. A correção acrescentou uma checagem de estrutura: nenhuma linha de tabela antes do primeiro título.

2. O README chegou a ~31 minutos de leitura. Ele foi cortado para ~8, e o `just quality` passou a falhar acima de 10, pelo mesmo binário que mede o texto do usuário.
3. O README em inglês mostrou a figura com rótulos em português. Cada figura ganhou um gêmeo em inglês, e a checagem falha se um README mostra a figura do outro idioma.
4. A receita de release abriu um editor num shell sem terminal e parou no meio. Sem terminal, ela agora não abre o editor e diz o que fazer.

Em cada caso, a correção do arquivo veio com uma checagem nova, e a checagem é o que fica. O projeto não mede se esse loop reduz erros ao longo do tempo $\triangle$.

O loop se apoia em duas regras do **AGENTS.md**. A primeira é a de evidência: a marca $\triangle$ indica uma afirmação não conferida, ninguém a tira sem fazer a checagem, e um número sem comando ou fonte não entra numa tabela. A segunda é o determinismo: as versões são fixadas pela tag do plugin, pelo commit do picasso, pelo `flake.lock` e pelo `Cargo.lock`, e a mesma entrada dá a mesma saída. O `just pins` confere que o submódulo e o flake apontam para o mesmo commit. Este PDF segue as mesmas regras: os logos são baixados num commit fixo, com o sha256 conferido, e o `just whitepaper` falha acima de 8 páginas.

7 Bilíngue e citável

O português vem primeiro, e o inglês é o espelho. O `README.md` e o `README.en.md` têm as mesmas seções e os mesmos fatos, e uma mudança num muda o outro no mesmo pull request. Cada figura tem um gêmeo em inglês, com as mesmas caixas e setas e só as palavras traduzidas; o `just figure-check` compara os dois desenhos e falha se um README mostra a figura do outro idioma. Os pull requests seguem a skill `bipr`: o corpo sai dos commits, em pt-BR primeiro e com o espelho em inglês, cada metade com as regras do seu idioma (PTW no nível `estrito` para o pt-BR; frases curtas e voz ativa, no espírito do ASD-STE100, para o inglês). Este artigo também sai nos dois idiomas, com as mesmas seções.

O projeto é citável. Uma versão só amarra a tag `vX.Y.Z`, o `version` do `plugin.json` e o cabeçalho do `CHANGELOG.md`; o `just release-check` compara os três antes de uma release, e o CI o roda quando a tag chega. A partir da `v0.3.0`, a release vai ao Zenodo, que dá a ela um DOI. O registro é um preprint, com descrição em pt-BR e em inglês, e cita o PTC com a relação `isDerivedFrom`; o `CITATION.cff` traz os autores com os ORCID. O `just release` confere o webhook do Zenodo antes da tag e espera o DOI depois dela $\triangle$: esse caminho ainda não rodou uma vez.

O projeto também é transparente. Todo fato do site e do README vem de um arquivo do repositório ou de um commit fixo. O site é escrito em Gleam, com Lustre, e pré-renderizado. As regras, os níveis, as receitas, o roadmap, a origem de cada regra e os projetos relacionados saem da `SKILL.md`, do `justfile` e das tabelas de `site/data/`, e nenhum deles é digitado no código do site; uma regra nova sem a linha da sua origem quebra o build. O site também publica o grafo do código deste repositório e o da extensão, gerados a cada build pelo `code-graph` do picasso, com `tree-sitter` e sem modelo, e cada figura como uma página interativa do Archify, fixado num commit.

8 O que se mediu de custo

Três pedidos reais ao plugin, com o modelo `claude-opus-5-5`, no commit `ca0e859` do plugin, em 21 de outubro de 2026, foram medidos pelo `claude -p --output-format json`

(docs/eli5/RESULTS.tsv):

Pedido	Nível	Tokens no total	Lidos do cache	Saída	Custo (US\$)
runbook de deploy	estrito	350 529	301 876	8 184	0,5477
comunicado	leve	184 688	155 591	2 401	0,2927
explicar a uma criança	descritivo	232 366	200 269	3 434	0,3380

Entre 84% e 86% dos tokens de cada pedido foram lidos do cache. A skill `bipr`, que escreve o corpo bilingue de um pull request, foi medida no transcript da sessão que abriu o PR #49: os dois passos dela somaram 4 tokens de entrada, 235 576 lidos do cache e 1 525 de saída (docs/bipr/exemplo-pr-49.md). Esse documento não converte tokens em dólares Δ .

A abstração em si não tem custo de treino nem de servidor. O custo por pedido é o do modelo que o usuário escolhe. Com um modelo de decisão local, não há cobrança por pedido, e o custo é o tempo de CPU da seção 5.

9 Limitações

Os casos de regressão em `evals/` estão escritos e não foram rodados por um harness, então o acerto das regras não foi medido. A extensão foi testada à mão no WhatsApp Web, com uma chave real; Gmail e GitHub não foram testados com o editor de verdade. A latência e o acerto do Ollama e do Ollaya em pt-BR não foram medidos, e o limiar de 50% do Ollaya não foi ajustado. A taxa de leitura do pt-BR é a do inglês. Os quatro casos da seção 6 são de um projeto e de um mês. Os três pedidos da seção 8 são uma amostra de três, com um modelo só. A tag v0.2.0 nunca foi criada; a v0.3.0 é a primeira tag desde a v0.1.0.

10 Referências

1. kayquer. *Português Técnico Controlado (PTC)*. Skill do Claude Code, licença MIT. <https://github.com/kayquer/portugues-tecnico-controlado/tree/cfdffad0ed98cdebfb7f591481889090a4f4e65b>, commit `cfdffad`, lido em 22 de outubro de 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*. Licença Apache-2.0. <https://github.com/ollaya-dev/ollaya/tree/39619accce0f5836dbeb60d2b14f5a056c2459ce>, commit `39619ac`, lido em 21 de outubro de 2026.
4. Ollama. *Ollama*. Licença MIT. <https://github.com/ollama/ollama/tree/efabf6bc450454dc856c1f799aac7ea865e3a4cf>, commit `efabf6b`, lido em 22 de outubro de 2026.
5. Anomaly. *OpenCode: the open source coding agent*. Licença MIT. <https://github.com/anomalyco/opencode/tree/c93aaae37157d785f76a8e9a9243a6560e286468>, commit `c93aaae`, lido em 22 de outubro de 2026.
6. Alibaba. *Open Code Review*. Licença Apache-2.0. <https://github.com/alibaba/open-code-review/tree/375f64cd6f67a48a135da83cfcf59c52669a09cb>, commit `375f64c`, lido em 22 de outubro de 2026.

O código, as regras e este texto estão em <https://github.com/h0ffmann/ptbr-writer>, sob a MPL-2.0. O modelo deste PDF vem do Paper Fabric de h0ffmann/closed-source, commit 563c339 (MIT).