

INSTITUTO FEDERAL DE BRASÍLIA
Campus Estrutural
Especialização em Matemática, Educação e Tecnologias

RICARDO CESAR ZIMMERMANN

PIPELINES AUTOMÁTICOS DE AVALIAÇÃO DE SLMS EM CÓDIGO EM PORTUGUÊS: ESTUDO DE VALIDADE DE CONSTRUTO

Brasília
2026

RICARDO CESAR ZIMMERMANN

PIPELINES AUTOMÁTICOS DE AVALIAÇÃO DE SLMS EM CÓDIGO EM PORTUGUÊS: ESTUDO DE VALIDADE DE CONSTRUTO

Artigo apresentado ao Instituto Federal de Brasília como requisito parcial para obtenção do título de Especialista em Matemática, Educação e Tecnologias.

Orientador: Prof. Dr. Rodrigo Figueiredo Abdo
Co-orientador: Prof. Me. Pedro Carvalho Brom

Brasília
2026

MINISTÉRIO DA EDUCAÇÃO
Instituto Federal de Educação, Ciência e Tecnologia de Brasília

RICARDO CESAR ZIMMERMANN

PIPELINES AUTOMÁTICOS DE AVALIAÇÃO DE SLMs EM CÓDIGO EM PORTUGUÊS: ESTUDO DE VALIDADE DE CONSTRUTO

Trabalho de conclusão de curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Brasília Campus Estrutural como parte dos requisitos necessários à obtenção do título de Especialista em Matemática, Educação e Tecnologias.

Documento assinado digitalmente
gov.br RICARDO CESAR ZIMMERMANN
Data: 11/06/2026 13:22:51-0300
Verifique em https://validar.iti.gov.br

Aprovado em 11 de Junho de 2026.

BANCA EXAMINADORA

Assinaturas digitais

_____

Dr. Rodrigo Figueiredo Abdo – Orientador

_____

Me. Pedro Carvalho Brom – Coorientador

_____

Me. Bruno Marx de Aquino Braga – membro

_____

Dr. Rodrigo Lima Peraira - membro

Brasília

2026

Documento assinado eletronicamente por:

- **Rodrigo Figueiredo Abdo**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 11/06/2026 12:12:42.
- **Bruno Marx de Aquino Braga**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 11/06/2026 12:13:34.
- **Rodrigo Lima Pereira**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 11/06/2026 12:14:34.
- **Pedro Carvalho Brom**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 11/06/2026 12:39:14.

Este documento foi emitido pelo SUAP em 11/06/2026. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifb.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 703143
Código de Autenticação: eba7e73878

INSTITUTO FEDERAL
Brasília

Campus Estrutural
Área Especial n° 01, Quadra 16, None, Cidade do
Automóvel/SCIA, ESTRUTURAL / DF, CEP 71.255-200

Pipelines Automáticos de Avaliação de SLMs em Código em Português: Estudo de Validade de Construto

Ricardo Cesar Zimmermann* **Rodrigo Figueiredo Abdo†** **Pedro Carvalho Brom‡**

Resumo

ZIMMERMANN, Ricardo C.; **Pipelines Automáticos de Avaliação de SLMs em Código em Português: Estudo de Validade de Construto.** 2026. Artigo (Especialização em Matemática, Educação e Tecnologias). Instituto Federal de Brasília. Brasília, 11 de junho de 2026.

Apresentamos um estudo de validade de construto (*construct validity*) de *pipelines* automáticos de avaliação para Small Language Models (SLMs) em tarefas de programação em português. Utilizando o SLM Benchmark Kit como instrumento experimental, integramos execução local via Ollama, avaliação automática com múltiplos juízes, avaliação humana cega e análise estatística com Welch, tamanho de efeito, intervalo de confiança *bootstrap* e regressão OLS com erros robustos. Em um experimento com 1296 execuções, comparamos os modelos Qwen2.5-Coder:1.5B e DeepSeek-Coder:1.3B e observamos diferença estatisticamente significativa entre médias (Cohen's $d = -0,440$, tamanho de efeito pequeno a médio). Observamos baixa concordância entre juízes automáticos (Spearman $\leq 0,302$) e descolamento expressivo entre avaliação automática e consenso humano (Spearman = 0,286; MAE = 2,35 pontos), evidenciando superestimação sistemática dos juízes automáticos. Em contraste, a concordância entre avaliadores humanos foi excelente (κ_w entre 0,966 e 1,000). Os resultados indicam que desempenho médio e significância estatística, isoladamente, são insuficientes para sustentar inferência robusta quando o *pipeline* apresenta degradação de juiz, baixa calibração automático versus humano e escores automáticos sistematicamente superestimados.

Palavras-chave: Small Language Models; Validade de Construto; LLM-as-a-Judge; Avaliação Humana; Engenharia de Software; Estatística.

*Instituto Federal de Brasília. E-mail: ricardo74469@estudante.ifb.edu.br
†Instituto Federal de Brasília. E-mail: rodrigo.abdo@ifb.edu.br
‡Instituto Federal de Brasília. E-mail: pedro.brom@ifb.edu.br

Abstract

We present a construct validity study of automatic evaluation pipelines for Small Language Models (SLMs) in programming tasks in Portuguese. Using the SLM Benchmark Kit as an experimental instrument, we integrated local execution via Ollama, automatic evaluation with multiple judges, blind human evaluation, and statistical analysis with Welch's test, effect size, bootstrap confidence intervals, and robust OLS regression. In an experiment with 1296 runs, we compared the Qwen2.5-Coder:1.5B and DeepSeek-Coder:1.3B models and observed a statistically significant difference between means (Cohen's $d = -0,440$, small to medium effect size). We found low agreement among automatic judges (Spearman $\leq 0,302$) and a notable gap between automatic evaluation and human consensus (Spearman = 0,286; MAE = 2,35 points), evidencing systematic overestimation by automatic judges. In contrast, human evaluators showed excellent agreement (κ_w between 0,966 and 1,000). Results indicate that average performance and statistical significance alone are insufficient to support robust inference when the pipeline shows judge degradation, poor calibration against humans, and systematically overestimated automatic scores.

Keywords: Small Language Models; Construct Validity; LLM-as-a-Judge; Human Evaluation; Software Engineering; Statistics.

Data da aprovação: 11/06/2026

1 Introdução

Modelos de linguagem de pequeno porte (SLMs) têm sido aplicados em tarefas como detecção de *bugs*, explicação de código, refatoração e geração de testes. Em comparação com soluções fechadas de grande escala, SLMs viabilizam execução local, menor custo operacional e melhor controle de dados sensíveis.

Neste trabalho, o SLM Benchmark Kit é tratado como instrumento de investigação, e não como contribuição final em si. O objetivo central é examinar a validade de construto de *pipelines* automáticos de avaliação para SLMs em código em português, testando se os sinais produzidos por juízes automáticos permanecem consistentes entre si e alinhados ao julgamento humano. Assim, além de comparar médias de *score*, investigamos descolamento entre juízes, confiabilidade operacional do *pipeline* e consistência entre avaliadores humanos [3].

Além das métricas clássicas de *benchmark*, estudos recentes mostram que a avaliação por LLMs-juízes pode escalar bem, mas exige tratamento explícito de vieses e validação cruzada com julgamento humano. Em particular, Zheng et al. discutem que o paradigma *LLM-as-a-judge* é útil, porém sujeito a vieses de posição, extensão da resposta e auto-favoritismo, o que reforça a necessidade de protocolos experimentais transparentes e auditáveis [9]. Sob a ótica de engenharia, essa preocupação é consistente com práticas modernas de *AI Engineering* para

sistemas em produção, nas quais avaliação, confiabilidade e observabilidade devem ser tratadas como requisitos de primeira classe [10].

2 Fundamentação Teórica

2.1 O Que São SLMs

SLMs são modelos de linguagem com menor *footprint* computacional, tipicamente direcionados a cenários com restrição de recursos, como execução local, *edge* e ambientes com orçamento limitado de memória e latência [11, 7]. Em engenharia de *software*, esse perfil é particularmente relevante por viabilizar uso contínuo em ferramentas de desenvolvimento e *pipelines* de validação com maior controle sobre dados sensíveis.

Na literatura recente, a noção de *small* é tratada menos como um limiar fixo de parâmetros e mais como uma propriedade de projeto orientada por restrições operacionais [11]. Sob essa perspectiva, Nguyen et al. estruturam o campo em dois eixos: (i) técnicas de otimização ao longo do ciclo do modelo (arquitetura, treinamento e compressão) e (ii) restrições-alvo, como custo de inferência, tempo de resposta e eficiência energética [11]. Isso desloca a discussão de tamanho absoluto para adequação ao contexto de uso.

Do ponto de vista técnico, esse enquadramento também explica por que estratégias como destilação, poda e quantização aparecem com frequência no ecossistema de SLMs: elas buscam reduzir custo computacional sem perda desproporcional de qualidade [11]. Contudo, ganhos de eficiência não são uniformes entre tarefas, e podem introduzir efeitos colaterais em robustez, calibração e estabilidade de saída.

Para *benchmarking*, a consequência direta é metodológica: comparar SLMs exige avaliação multicritério e leitura conjunta de qualidade de resposta e desempenho operacional. Em outras palavras, *score* médio isolado não captura adequadamente valor prático quando o objetivo é suportar fluxo de engenharia em tempo real e com requisitos de confiabilidade.

2.2 Engenharia de Prompts

Engenharia de *prompts* refere-se ao desenho sistemático das instruções, do contexto e do formato de saída enviados ao modelo para maximizar precisão técnica, consistência e utilidade da resposta [8]. Em aplicações de software, ela não deve ser tratada como ajuste *ad hoc*, mas como componente metodológico do experimento, pois pequenas variações de instrução podem alterar substancialmente o comportamento observado.

Esse processo é complementar ao controle de hiperparâmetros de inferência (por exemplo, temperatura, *top-p* e *top-k*), que regulam variabilidade e aderência ao enunciado. Na prática, *prompt* e decodificação formam um sistema único de decisão: o texto define a estrutura da tarefa, enquanto os hiperparâmetros modulam estabilidade e diversidade de geração.

Com base em *surveys* da literatura de *prompting*, as técnicas podem ser entendidas como um gradiente de controle sobre o raciocínio do modelo [12]. Em geral, *zero-shot* é adequado para instruções diretas; *few-shot* introduz exemplos para induzir padrão; *Chain-of-Thought* explicita passos intermediários em problemas mais complexos; e *self-consistency* melhora robustez ao selecionar respostas recorrentes entre múltiplas amostras [12].

Além disso, estratégias orientadas a fluxo e ação, como encadeamento de *prompts* e ReAct, permitem decompor tarefas em subtarefas e integrar consulta a ferramentas externas [12]. Em *benchmark* de geração de código, tais abordagens reduzem ambiguidade de requisito, aumentam conformidade de formato (por exemplo, JSON estrito) e tornam a avaliação mais reprodutível. Por essa razão, o projeto de *prompt* deve ser versionado e auditável, no mesmo nível de rigor aplicado à configuração de modelo e ao protocolo estatístico.

2.3 SLMs Como Assistentes de Código

No ciclo de desenvolvimento de *software*, SLMs podem atuar como assistentes de código em tarefas como explicação de trechos, diagnóstico de falhas, geração de testes, sugestões de refatoração e apoio à prototipação. A atratividade prática decorre, sobretudo, do menor custo de inferência, da menor latência e da possibilidade de execução local, o que favorece integração contínua em IDEs e *pipelines* com menor dependência de infraestrutura remota [11, 13, 14].

Sob a perspectiva metodológica, a utilidade desses assistentes não deve ser inferida apenas por acurácia média. Em linha com a taxonomia de SLMs orientada por restrições, o desempenho precisa ser interpretado conjuntamente com métricas operacionais (tempo de resposta, uso de memória, estabilidade de execução e custo), especialmente quando o objetivo é suporte em tempo real a tarefas de programação [11]. Esse enquadramento é coerente com relatórios técnicos recentes de modelos abertos de código, nos quais privacidade, controle de implantação e responsividade são requisitos centrais [13, 14].

Ao mesmo tempo, há *trade-offs* relevantes: modelos menores tendem a perder robustez em tarefas de maior complexidade semântica ou raciocínio profundo, podendo demandar estratégias de mitigação como *prompts* estruturados, ajuste fino orientado ao domínio e validação por múltiplos critérios [11, 13, 14]. Portanto, uma adoção tecnicamente consistente em engenharia de *software* combina ganhos de eficiência dos SLMs com protocolos de avaliação que controlem risco de erro em cenários críticos.

2.4 Benchmarks De Código E Cobertura Avaliativa

A literatura de geração de código consolidou *benchmarks* funcionais que servem de base para comparação entre modelos. HumanEval introduz uma coleção de problemas de síntese com verificação por testes unitários e popularizou a leitura por $pass@k$ [15]. MBPP amplia esse cenário com tarefas de programação introdutórias em linguagem natural, úteis para medir robustez em instruções curtas e variadas [16].

Além dos *benchmarks* iniciais, trabalhos recentes expandem a cobertura de validade do construto "qualidade de código gerado". MultiPL-E estende HumanEval e MBPP para múltiplas linguagens, reduzindo viés de avaliação monolíngue [17]. EvalPlus mostra que *suites* de teste fracas inflacionam desempenho e propõe avaliação mais rigorosa via expansão automática de casos de teste [18]. BigCodeBench, por sua vez, aumenta complexidade de instrução e uso de bibliotecas reais, aproximando a medição de cenários de engenharia [19].

Esses avanços são diretamente relevantes para este trabalho: ao focar tarefas em português e validação humano-versus-juízes, buscamos complementar a linha clássica de *benchmarks* com uma leitura de validade de construto em contexto linguístico e operacional específico. À luz desses fundamentos, este estudo utiliza o SLM Benchmark Kit como instrumento central de investigação. Diferente dos *benchmarks* clássicos, que se limitam a coleções de tarefas, o kit é um programa de computador que organiza a execução dos modelos, aplica juízes automáticos e humanos e realiza análises estatísticas de forma integrada. Em outras palavras, funciona como um laboratório digital para avaliar SLMs em condições controladas e reprodutíveis.

3 Metodologia Do Benchmark

Com base no enquadramento teórico anterior, a metodologia foi estruturada para traduzir requisitos de eficiência, confiabilidade e reprodutibilidade em procedimentos explícitos de execução, julgamento e análise. O *pipeline* do SLM Benchmark Kit é estruturado em três camadas: *Runner*, *Judge* e *Analyzer*. O *Runner* executa combinações de modelos e hiperparâmetros em um conjunto de tarefas estratificadas. O *Judge* produz pontuações automáticas por múltiplos avaliadores. O *Analyzer* agrega resultados e executa inferência estatística.

O experimento foi executado com 1296 *trials* válidos no total (648 por modelo). A infraestrutura de inferência utilizou Ollama [5]. As saídas foram analisadas por juízes automáticos e por avaliação humana cega em amostra dedicada, distribuída entre avaliadores com sobreposição parcial de 25% (`overlap_rate = 0,25`) para estimar concordância interavaliador.

Os modelos avaliados pertencem às famílias Qwen2.5-Coder e DeepSeek-Coder, documentadas em relatórios técnicos públicos [13, 14].

Como princípio de desenho experimental, adotamos avaliação multicritério (desempenho, latência, saúde operacional e alinhamento com humanos), em linha com recomendações contemporâneas de engenharia de aplicações com modelos fundacionais [10].

3.1 Caracterização Do Dataset

O *dataset* `slm_tasks_ptbr.jsonl` foi construído especificamente para este experimento, cobrindo cinco classes de tarefas de programação em português que refletem atividades recorrentes no ciclo de engenharia de *software*: explicação de código, detecção de *bugs*, refatoração, geração de testes e análise de segurança/desempenho. Cada tarefa é composta por um *prompt* em

linguagem natural e uma resposta-referência concisa, usada como âncora pelos juízes automáticos e como guia durante a fase de avaliação humana.

3.1.1 Distribuição Por Classe, Dificuldade E Partição

O corpus contém 40 tarefas distribuídas igualmente entre as cinco classes (8 por classe), estratificadas em três níveis de dificuldade e três partições, conforme a Tabela 1.

Tabela 1: Distribuição do dataset por classe, dificuldade e partição. Split indicado como tr/dev/te (train/dev/test).

Classe	Fácil	Médio	Difícil	Total	*tr/de/te*
`code_explanation`	3	3	2	8	5/1/2
`bug_detection`	2	3	3	8	5/1/2
`refactoring`	2	3	3	8	5/2/1
`test_generation`	2	3	3	8	5/2/1
`security_performance`	2	3	3	8	4/2/2
Total	**11**	**15**	**14**	**40**	24/8/8

A Tabela 1 apresenta a partição do corpus em *tr/dev/test* (*train/dev/test*), onde *train* é usado para desenvolvimento e ajuste de parâmetros, *dev* para seleção de configuração e validação interna, e *test* para avaliação final e comparação entre modelos. Em amostras pequenas, como a presente (40 tarefas; *split* 24/8/8), essa divisão exige cuidados: estimativas pontuais tornam-se mais variáveis e a probabilidade de *overfitting* ao conjunto de desenvolvimento aumenta. Para mitigar esses riscos, adotamos estratificação por classe e dificuldade ao criar os *splits*, reportamos tamanhos de cada partição e utilizamos intervalos de confiança por *bootstrap* nas métricas agregadas. Quando apropriado, recomenda-se complementar o esquema de *hold-out* com validação cruzada estratificada ou análises de sensibilidade (por exemplo, repetir partições com *seeds* diferentes) para avaliar a estabilidade dos resultados e a robustez das conclusões.

3.1.2 Exemplos Canônicos Por Classe

A Tabela 2 apresenta um exemplo canônico por classe, selecionado pelo grau fácil como referência de menor ambiguidade semântica.

3.1.3 Processo De Validação Das Respostas-Referência

As respostas-referência foram redigidas pelos autores com base em conhecimento canônico de engenharia de *software* e revisadas em duas rodadas. Na primeira rodada, cada referência foi confrontada com literatura técnica estabelecida (por exemplo, critérios de OWASP para segurança, princípios SOLID para refatoração, práticas de *testing* em Node/Jest). Na segunda rodada, um autor verificou consistência interna entre o *prompt*, a resposta-referência e o nível

Tabela 2: Exemplos canônicos por classe de tarefa

Classe	Prompt	Resposta-referência
`code_explanation`	*Explique este código JS:* `function soma(a,b){ return a+b; }`	A função recebe dois parâmetros e retorna a soma.
`bug_detection`	*Encontre o problema:* `function div(a,b){ return a/b }`	Falta tratamento para divisão por zero e validação de entrada.
`refactoring`	*Refatore:* `function isAdult(age){ if(age>=18){ return true } else { return false } }`	Simplificar para `return age >= 18` e manter legibilidade.
`test_generation`	*Crie casos de teste para:* `function max(a,b){ return a>b?a:b }`	Cobrir maior, menor, iguais, negativos e zero.
`security_performance`	*Analise risco:* `app.get('/u/:id', (req,res)=> db.find(req.params.id))`	Verificar validação de id e controle de acesso por usuário autenticado.

de dificuldade declarado. Referências que não cobriram o ponto central do *prompt* foram reformuladas antes da finalização. O critério de aceitação exigiu que cada referência contivesse ao menos um elemento diagnóstico essencial e fosse formulada de modo suficientemente sucinto para orientar o julgamento sem impor estilo de resposta ao modelo avaliado.

3.1.4 Protocolo De IAA (*Inter-Annotator Agreement*) Para Construção Do *Gold Standard*

A construção do *gold standard* humano seguiu protocolo de anotação cega com múltiplos avaliadores. Os IDs (identificadores únicos) originais foram substituídos por identificadores cegos (`BLIND-XXXX`) antes da distribuição, impedindo rastreabilidade direta ao modelo ou à tarefa. A amostra humana cobriu 120 itens, distribuídos entre três avaliadores com sobreposição de 25% (`overlap_rate = 0,25`), gerando entre 9 e 11 itens de dupla anotação por par.

Os avaliadores são profissionais de tecnologia da informação com formação e experiência semelhantes na área, e as tarefas do *dataset* são específicas desse domínio. Esse perfil homogêneo favorece alta concordância, mas também representa uma limitação de escopo: os resultados de IAA são válidos dentro dessa coorte e não devem ser generalizados para avaliadores com formações distintas ou sem conhecimento especializado na área.

Cada item foi avaliado em seis dimensões, todas na escala 0–10: correção técnica (`score_technical`), cobertura/profundidade (`score_completeness`), utilidade prática (`score_actionability`), clareza e concisão (`score_clarity`), conformidade de formato (`score_format`) e julgamento global (`score_overall`). O campo `score_overall` não é média aritmética das dimensões; reflete o julgamento holístico do avaliador, podendo divergir de uma combinação linear simples.

A concordância foi estimada por ρ de Spearman, τ de Kendall e κ_w ponderado. Os resultados obtidos (κ_w entre 0,966 e 1,000 nos três pares; cf. Seção 5.5) classificam-se como *excelente* pelo critério de Landis e Koch, validando o uso do agregado humano (mediana por `blind_id`) como referencial estável para calibração de juízes automáticos.

4 Técnicas Estatísticas

Derivadas do desenho metodológico descrito na seção anterior, as técnicas estatísticas foram selecionadas para atender objetivos complementares de inferência: descrição dos dados, quantificação de incerteza, teste de hipótese, magnitude de efeito e análise condicional por covariáveis [1, 2, 4]. Medidas descritivas oferecem visão inicial da distribuição dos escores e da latência. O intervalo de confiança por *bootstrap* foi adotado para estimar incerteza da média com menor dependência de pressupostos paramétricos estritos, sendo apropriado para distribuições potencialmente assimétricas e cenários em que a forma da distribuição populacional é desconhecida [4].

Para comparação entre modelos, o teste t de Welch foi escolhido por ser mais robusto à hipótese de variâncias desiguais entre grupos, reduzindo risco de inferência distorcida quando a homocedasticidade não é garantida [4]. O tamanho de efeito de Cohen foi incluído para complementar a significância estatística com interpretação prática da magnitude da diferença observada [2]. Por fim, a regressão OLS (*Ordinary Least Squares*) com erros robustos permite estimar associações entre escore e fatores experimentais (modelo, temperatura, *top-p*, *top-k* e tipo de tarefa), mitigando o impacto de heterocedasticidade sobre os erros-padrão e fortalecendo a validade inferencial do modelo linear [4].

4.1 Intervalo De Confiança Bootstrap

Para cada modelo, o intervalo de confiança de 95% da média foi estimado por *bootstrap percentile*:

$$\hat{\mu}^{*b} = \frac{1}{n}\sum_{i=1}^{n} X_i^{*b}, \quad b = 1, \ldots, B \tag{1}$$

onde X_i^{*b} é a amostra com reposição no *bootstrap* b, com $B = 5000$. O IC é obtido pelos percentis 2,5% e 97,5%.

4.2 Hipóteses E Teste De Welch

A comparação entre os dois modelos foi conduzida com teste t de Welch (variâncias desiguais), com hipóteses:

H_0: as médias populacionais dos escores agregados de Qwen2.5-Coder:1.5B e DeepSeek-Coder:1.3B são iguais.

H_1: as médias populacionais dos escores agregados são diferentes.

A estatística utilizada foi:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} \tag{2}$$

4.3 Tamanho de Efeito

A magnitude da diferença foi quantificada por Cohen's d:

$$d = \frac{\bar{X}_1 - \bar{X}_2}{s_p} \tag{3}$$

com s_p como desvio-padrão combinado.

4.4 Regressão OLS

Para avaliar associação entre escore e covariáveis do experimento, estimamos o modelo base:

$$Y = \beta_0 + \beta_1 \text{Model} + \beta_2 \text{Temp} + \beta_3 \text{TopP} + \beta_4 \text{TopK} + \beta_5 \text{TaskType} + \epsilon \tag{4}$$

com erros robustos HC3. Um modelo estendido inclui o termo de interação TaskType $\times$ Temp para capturar heterogeneidade do efeito de temperatura entre classes de tarefa.

4.5 Diagnósticos De Regressão

A normalidade dos resíduos foi verificada pelo teste de Shapiro-Wilk [4]:

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{5}$$

onde $x_{(i)}$ são os resíduos ordenados e a_i são constantes derivadas da distribuição normal esperada. H_0: resíduos normalmente distribuídos; rejeitada quando $p < 0{,}05$.

A homocedasticidade foi verificada pelo teste de Breusch-Pagan [4]: regride-se o quadrado dos resíduos sobre as covariáveis originais; a estatística LM segue distribuição χ^2 sob H_0 de variâncias homogêneas. Rejeição ($p < 0{,}05$) confirma heterocedasticidade e justifica o uso de HC3.

4.6 Coeficientes De Correlação E Concordância

Três coeficientes de associação são usados para avaliar concordância entre juízes e calibração automático-humano [4]:

- *Spearman* (ρ): correlação de postos, robusta a outliers e distribuições assimétricas.
- *Kendall* (τ): proporção líquida de pares concordantes, mais conservador que Spearman em amostras pequenas.

• *Pearson* (r): correlação linear, sensível à escala e a outliers.

O MAE (*Mean Absolute Error*) quantifica viés absoluto médio entre dois conjuntos de escores: $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$, interpretado aqui como desalinhamento de calibração entre avaliação automática e humana.

O *kappa* ponderado (κ_w) mede concordância entre avaliadores descontando concordância ao acaso, com pesos que penalizam discordâncias proporcionalmente à sua magnitude [2, 4]. Valores $\kappa_w \geq 0{,}75$ classificam-se como concordância excelente pelo critério de Landis e Koch.

4.7 Intervalo De Confiança Para Correlações

Para os coeficientes de Spearman (ρ) e Pearson (r) reportados nas Tabelas de concordância entre juízes e de calibração automático-humano, os intervalos de confiança de 95% foram obtidos pelo método assintótico de Fisher z [4]:

$$z = \frac{1}{2}\ln\left(\frac{1+\hat{\rho}}{1-\hat{\rho}}\right), \quad \text{SE}(z) = \frac{1}{\sqrt{n-3}} \tag{6}$$

O IC para z é $z \pm 1{,}96 \cdot \text{SE}(z)$ e o resultado é transformado de volta à escala original por $\hat{\rho} = \tanh(z)$. O método assume que n é suficientemente grande; para Spearman, a transformação é aplicada como aproximação, procedimento padrão na literatura [4]. Coeficientes de Kendall (τ) e o MAE não possuem expressão analítica fechada análoga e são reportados sem IC.

5 Características dos Modelos e Hardware Utilizado

Os modelos avaliados neste experimento foram:

- *Qwen2.5-Coder:1.5B* — modelo específico para código, derivado da arquitetura Qwen2.5. Possui 1,54 bilhões de parâmetros (1,31B não-*embedding*), 28 camadas, atenção GQA com 12 cabeças para Q e 2 para KV, e contexto de até 32.768 tokens. Foi pré-treinado em 5,5 trilhões de *tokens*, incluindo código-fonte, dados sintéticos e textos técnico-científicos. Apresenta melhorias significativas em geração, raciocínio e correção de código, mantendo competências gerais em matemática e linguagem natural .
- *DeepSeek-Coder:1.3B* — modelo de código treinado do zero em 2 trilhões de tokens, com composição de 87% código e 13% linguagem natural (inglês e chinês). Possui 1,3 bilhões de parâmetros, janela de contexto de 16K *tokens* e tarefa adicional de preenchimento de lacunas para suportar completude em nível de projeto.

Os juízes automáticos empregados para avaliação foram:

- *Gemma2:2B* — modelo da família Gemma, com 2 bilhões de parâmetros, projetado para eficiência em dispositivos locais e tarefas de julgamento. Treinado com *corpus* multimodal e textual, suporta contexto de até 8K *tokens*. É otimizado para avaliação leve e rápida, mantendo boa calibragem em tarefas de linguagem natural e código.

- *CodeLlama:7B* — modelo da família LLaMA, especializado em programação, com 7 bilhões de parâmetros e janela de contexto de até 16K *tokens*. Treinado em grandes *corpora* de código-fonte e instruções técnicas, é amplamente usado como *baseline* em *benchmarks* de geração e avaliação de código. Suporta múltiplas linguagens de programação e tarefas de completude, explicação e correção.

A infraestrutura de execução foi baseada em um MacBook Pro M3 Max com 36 GB de memória unificada, utilizado para todas as execuções locais via Ollama. Essa configuração garantiu consistência nos tempos de resposta e viabilizou a execução integral dos 1296 *trials* sem necessidade de cluster ou GPU externa.

O uso de hardware único e controlado assegura reprodutibilidade dos resultados e elimina variabilidade decorrente de diferenças de ambiente computacional.

6 Resultados

6.1 Resumo Geral Por Modelo

Tabela 3: Resumo absoluto por modelo. IC 95% *bootstrap* ($B = 5000$) sobre a média.

Modelo	n	Média	IC 95% Média	Desvio	Mediana	Lat. p50 (ms)
qwen2.5-coder:1.5b	648	7,74	[7,716; 7,767]	0,33	8,00	7246
deepseek-coder:1.3b	648	7,55	[7,509; 7,589]	0,52	7,50	6849

No teste de Welch entre modelos, obtivemos $t = -7,918$, $p = 5,858 \times 10^{-15}$ e $d = -0,440$. Assim, ao nível de 5%, rejeita-se H_0 e conclui-se diferença estatisticamente significativa entre as médias, com tamanho de efeito pequeno a médio.

6.2 Concordância Entre Juízes Automáticos

Tabela 4: Concordância entre juízes automáticos. IC 95% assintótico para Spearman via Fisher z.

Juiz A	Juiz B	n	Spearman	IC 95% ρ	Kendall	Pearson
Heuristic	Gemma2_judge	1296	0,095	[0,041; 0,149]	0,092	0,062
Heuristic	Codellama_judge	1255	0,006	[−0,050; 0,061]	0,005	-0,015
Gemma2_judge	Codellama_judge	1255	0,302	[0,252; 0,353]	0,293	0,392

A correlação entre os pares de juízes varia de quase nula (Heuristic × Codellama, Spearman = 0,006) a moderada (Gemma2 × Codellama, Spearman = 0,302). O par Heuristic × Gemma2

permanece fraco (Spearman = 0,095). Essa heterogeneidade de sinal entre avaliadores automáticos confirma alta sensibilidade ao critério de avaliação, reforçando a necessidade de triangulação com julgamento humano.

O juiz heurístico (`Heuristic`) é um avaliador automático baseado em regras determinísticas e métricas sintáticas (por exemplo, presença de trechos esperados, conformidade de formato, checagens simples de saída) em vez de inferência estatística por SLMs. Por ser rápido e transparente, serve como linha de base operacional para detectar respostas inválidas, erros de *parsing* e conformidade estrutural, mas tende a ser insensível a nuances semânticas e à correção funcional profunda. Em termos práticos, o juiz heurístico complementa juízes baseados em modelos: ele filtra ruídos óbvios e reduz carga de avaliação humana, mas não substitui calibração contra o *gold standard* humano quando o objetivo é medir qualidade semântica.

6.3 Saúde Operacional Dos Juízes

Tabela 5: Saúde técnica dos juízes automáticos

Juiz	Total	Válidos	Parse errors	Judge errors
Codellama_judge	1296	1255	41	0
Gemma2_judge	1296	1296	0	0
Heuristic	1296	1296	0	0

6.4 Diagnóstico Do Codellama_judge

O juiz Codellama foi configurado como `codellama:7b` no *benchmark*. O *prompt* de avaliação exigia saída em JSON estrito com as chaves `score` e `rationale`. Após execução, o juiz apresentou taxa de validade de 96,8% (1.255 de 1.296 chamadas), com 41 erros de *parse* (3,16%), 54 *parse_fallbacks* (4,17%) e 0 *judge_errors*.

Os erros de *parse* indicam que, em aproximadamente 3% dos casos, a resposta do Codellama não pôde ser convertida para o formato esperado, sendo parcialmente compensados pelos *fallbacks*. Esse padrão aponta degradação pontual de formato de saída. A concordância do Codellama com os demais juízes permanece baixa — Spearman = 0,302 com Gemma2 e quase nulo com o Heuristic (Spearman = 0,006) — indicando que a confiabilidade desse juiz é inferior à dos demais.

6.5 Concordância Humana

Tabela 6: Concordância humano-humano por par em itens sobrepostos

Avaliador A	Avaliador B	n_overlap	Spearman	κ_w
eval01	eval02	10	1,000	0,975
eval01	eval03	11	0,998	0,966
eval02	eval03	9	1,000	1,000

A concordância foi calculada somente nos itens com dupla anotação (`n_overlap` entre 9 e 11 por par), oriundos da sobreposição parcial da amostra humana. A mediana de κ_w entre os pares foi 0,975, com amplitude de 0,966 a 1,000, classificando-se como concordância excelente em todos os pares. A alta consistência interavaliador fortalece o referencial humano e torna o descolamento em relação à avaliação automática ainda mais relevante para fins de calibração.

6.6 Calibração Automático Versus Humano

Tabela 7: Calibração entre score automático e agregado humano ($n = 120$). IC 95% assintótico via Fisher z para Spearman e Pearson; Kendall e MAE sem IC analítico disponível.

Métrica	Estimativa	IC 95%
n itens	120	—
Spearman (ρ)	0,2855	[0,112; 0,440]
Kendall (τ)	0,2261	—
Pearson (r)	0,2856	[0,112; 0,440]
MAE	2,3492	—

Os coeficientes mostram associação fraca entre avaliação automática e consenso humano (Spearman = 0,286). Mais relevante é o MAE de 2,35 pontos: os juízes automáticos superestimam sistematicamente as respostas em comparação ao julgamento humano (médias automáticas em torno de 7,5–7,7 versus médias humanas em torno de 5,2), evidenciando viés de calibração que inviabiliza o uso do *score* automático como *proxy* fiel de qualidade.

6.7 Regressão OLS

O modelo OLS base apresentou $R^2 = 0,226$ (ajustado 0,222), com coeficiente positivo para temperatura e significância estatística nas covariáveis especificadas. O diagnóstico formal dos resíduos e a extensão do modelo com interação `task_type` $\times$ `temperature` são reportados na seção seguinte.

6.8 Diagnóstico OLS E Modelo Com Interação

6.8.1 Shapiro-Wilk (Normalidade Dos Resíduos)

O teste de Shapiro-Wilk sobre os resíduos do modelo base retornou $W = 0,892$, $p = 3,08 \times 10^{-29}$, indicando desvio significativo de normalidade[1]. Com $n = 1296$, esse resultado é esperado pela sensibilidade do teste a grandes amostras, e o impacto prático é mitigado pelo Teorema Central do Limite sobre os coeficientes. O uso de erros robustos HC3 já contempla esse cenário.

6.8.2 Breusch-Pagan (Homocedasticidade)

O teste de Breusch-Pagan rejeitou homocedasticidade (LM $= 70,16$, $p = 4,56 \times 10^{-12}$; $F = 9,21$, $p = 2,15 \times 10^{-12}$), confirmando empiricamente a presença de heterocedasticidade. Esse resultado valida retroativamente a escolha de erros robustos HC3 na especificação principal.

6.8.3 Modelo Com Interação `task_type` × `temperature`

O modelo estendido com o termo de interação apresentou $R^2 = 0,279$ (ajustado 0,272), ganho de 5,3 pontos percentuais sobre o modelo base, sugerindo que o efeito da temperatura é heterogêneo entre classes de tarefa. Os coeficientes de interação significativos estão na Tabela 8.

Tabela 8: Coeficientes de interação `task_type × temperature` (HC3). Classe de referência: `bug_detection`.

Interação	coef	std err	z	p	IC 95%
`code_explanation × temp.`	−0,439	0,095	−4,63	<0,001	[−0,624; −0,253]
`refactoring × temp.`	−0,222	0,096	−2,32	0,020	[−0,409; −0,034]
`security_performance × temp.`	−0,644	0,115	−5,57	<0,001	[−0,870; −0,417]
`test_generation × temp.`	+0,469	0,197	+2,39	0,017	[+0,084; +0,854]

O coeficiente base de temperatura permanece positivo (+0,473, $p < 0,001$), mas as interações revelam que esse efeito se concentra em `bug_detection` e `test_generation`, enquanto classes de análise e refatoração são penalizadas pelo aumento de temperatura. `top_p` ($p = 0,830$) e `top_k` ($p = 0,245$) não alcançaram significância em nenhuma especificação.

7 Discussão

Os resultados apontam vantagem média do Qwen2.5-Coder:1.5B sobre DeepSeek-Coder:1.3B, com significância estatística e efeito pequeno a médio ($d = -0,440$). Do ponto de vista prático,

[1] O *p-value* é a probabilidade, sob a hipótese nula, de observar uma estatística de teste tão extrema quanto a observada.

a diferença existe e é mais substantiva do que a observada na execução anterior, porém ainda insuficiente para afirmar superação de grande magnitude.

Além da comparação de escores, os resultados documentam o perfil operacional dos modelos no cenário de implantação local que motivou este trabalho. Ambos foram executados inteiramente em *hardware* local via Ollama, sem dependência de API remota, eliminando custo variável de inferência e garantindo que os dados processados não trafeguem fora do ambiente controlado — atributos centrais para adoção em contextos com requisitos de privacidade ou conformidade regulatória. As latências medianas observadas (Qwen: 7.246 ms; DeepSeek: 6.849 ms) confirmam viabilidade operacional abaixo de 8 segundos por chamada em *hardware* de consumo, compatível com fluxos de revisão assistida e validação em *pipelines* de CI. Esse perfil — custo próximo de zero por inferência, dados sob controle local e tempo de resposta adequado a tarefas de desenvolvimento — constitui a vantagem competitiva central dos SLMs frente a alternativas de grande escala baseadas em nuvem [11, 13, 14]. Essa vantagem se sustenta independentemente dos achados de validade de construto discutidos a seguir, pois pertence ao eixo operacional, não ao eixo de qualidade de resposta.

A contribuição principal deste estudo está no exame de validade de construto do pipeline automático de avaliação. Evidenciamos três sinais de fragilidade do construto "qualidade de resposta" quando inferido apenas por julgamento automático: degradação de formato em um dos juízes (Codellama, 3,16% de *parse_errors* após correção operacional), descolamento entre juízes automáticos (Spearman $\leq 0,302$) e superestimação sistemática dos juízes em relação ao consenso humano (MAE = 2,35; Spearman = 0,286). Em contraste, a concordância entre avaliadores humanos foi excelente (κ_w entre 0,966 e 1,000), indicando que o referencial humano é estável dentro da coorte avaliada. Cabe notar, contudo, que os três avaliadores possuem formação e experiência semelhantes em TI e que as tarefas são específicas desse domínio; essa homogeneidade de perfil favorece alta concordância e constitui uma limitação de escopo: o referencial pode não ser igualmente estável com avaliadores de áreas distintas ou sem conhecimento especializado [20]. O descolamento observado entre avaliação automática e humana permanece atribuível principalmente a viés nos juízes automáticos, e não a ruído na avaliação humana dentro desta coorte. De forma mais ampla, os resultados deste estudo — tanto os de desempenho comparativo entre modelos quanto os de calibração de juízes — são válidos para o par de modelos, o conjunto de tarefas e os juízes aqui empregados; experimentos com modelos de outras famílias, tarefas de outros domínios ou outros conjuntos de juízes podem produzir padrões distintos, o que reforça a necessidade de replicação e extensão do protocolo em contextos variados.

Nossos achados empíricos são coerentes com a literatura: embora *LLM-as-a-judge* seja promissor para reduzir custo de avaliação, há riscos metodológicos quando sua saída é tomada como *proxy* final de qualidade sem calibração humana. Esse ponto é alinhado com a análise de Zheng et al. [9]. Do ponto de vista de engenharia de sistemas de IA, o comportamento observado — degradação parcial do Codellama, baixa concordância entre juízes (Spearman

$\leq 0,302$) e superestimação sistemática em relação ao julgamento humano (MAE = 2,35) — reforça a recomendação de *pipelines* com validação em múltiplas camadas, monitoramento de saúde e critérios de fallback explícitos [10].

Em concordância com literatura recente [3], a presença de *bootstrap* e análises robustas melhora a qualidade inferencial. O diagnóstico formal dos resíduos confirmou heterocedasticidade significativa (Breusch-Pagan $p = 4,56 \times 10^{-12}$), validando empiricamente o uso de erros robustos HC3. O modelo com interação `task_type` $\times$ `temperature` revelou heterogeneidade no efeito da temperatura entre classes: temperatura mais alta beneficia `test_generation` (+0,469, $p = 0,017$) mas penaliza `security_performance` ($-$0,644, $p < 0,001$), `code_explanation` ($-$0,439, $p < 0,001$) e `refactoring` ($-$0,222, $p = 0,020$). Esse padrão sugere que a escolha de hiperparâmetros não deve ser uniforme entre classes de tarefa em implantações práticas.

8 Conclusão

Este trabalho deve ser interpretado, prioritariamente, como um estudo de validade de construto de pipelines automáticos de avaliação para SLMs aplicados a tarefas de código em português. O protocolo experimental implementado no SLM Benchmark Kit permitiu demonstrar, de forma reprodutível, dois resultados complementares: (i) diferença estatisticamente significativa entre os modelos avaliados, com magnitude prática pequena a média; e (ii) limitações relevantes de inferência quando a avaliação automática é adotada isoladamente, sem calibração com referência humana.

Do ponto de vista metodológico, os achados reforçam que significância estatística, por si só, não é condição suficiente para sustentar conclusões robustas em cenários com potencial degradação de juízes e descalibração automático-humano. A combinação entre métricas operacionais, análise inferencial, diagnósticos de suposições e validação humana cega mostrou-se necessária para reduzir risco de interpretações espúrias e para preservar a validade do construto qualidade de resposta no contexto estudado.

Como agenda de continuidade, propomos três extensões principais. Primeiro, incorporar *bootstrap* estratificado por tipo de tarefa, de modo a estimar incerteza com maior sensibilidade à heterogeneidade entre classes e reduzir o risco de que médias globais ocultem padrões específicos relevantes para decisão prática. Segundo, estender a inferência para cenários com mais de dois modelos por meio de ANOVA seguida de comparações *post-hoc* com correção para múltiplos testes, assegurando controle adequado de erro do Tipo I em comparações simultâneas. Terceiro, adotar modelagem hierárquica (multinível), com efeitos aleatórios por item, para representar explicitamente a dependência intra-item entre repetições e aprimorar a estimativa de efeitos de modelo em desenhos com estrutura aninhada.

Em conjunto, essas extensões ampliam a robustez estatística e a generalização analítica do protocolo, mantendo o compromisso com auditabilidade e reprodutibilidade em avaliações de

SLMs executados localmente.

Referências

[1] BOX, G. E. P.; HUNTER, W. G.; HUNTER, J. S. *Statistics for Experimenters*. 2. ed. Wiley, 2005.

[2] COHEN, J. *Statistical Power Analysis for the Behavioral Sciences*. 2. ed. Lawrence Erlbaum, 1988.

[3] DI OLIVEIRA, V.; BROM, P. C.; LI, W. Two-Step RAG for Metadata Filtering and Statistical LLM Evaluation: Integrating Prompt-Aware Retrieval and Bootstrap Mixed Models. *IEEE Latin America Transactions*, v. 23, n. 12, p. 1201–1210, 2025.

[4] MONTGOMERY, D. C. *Design and Analysis of Experiments*. 9. ed. Wiley, 2017.

[5] OLLAMA. Documentação técnica. Disponível em: https://ollama.com e https://github.com/ollama/ollama. Acesso em: 25 abr. 2026.

[6] RAMOS, M. V. M.; NETO, J. J.; VEGA, I. S. *Teoria de Linguagens Formais*. LTC, 2009.

[7] RUSSELL, S.; NORVIG, P. *Inteligência Artificial: Uma Abordagem Moderna*. 3. ed. São Paulo: Pearson, 2021.

[8] SOUZA, R. *ChatGPT: do zero aos prompts avançados*. Independente, 2023.

[9] ZHENG, L.; CHIANG, W.-L.; SHENG, Y.; et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv preprint arXiv:2306.05685, 2023. Disponível em: https://arxiv.org/abs/2306.05685. Acesso em: 25 abr. 2026.

[10] HUYEN, C. *AI Engineering: Building Applications with Foundation Models*. O'Reilly Media, 2025. ISBN-13: 9781098166298. Disponível em: https://oreil.ly/ai-engineering. Acesso em: 25 abr. 2026.

[11] NGUYEN, C. V.; SHEN, X.; APONTE, R.; et al. A Survey of Small Language Models. arXiv preprint arXiv:2410.20011, 2024. Disponível em: https://arxiv.org/abs/2410.20011. Acesso em: 28 abr. 2026.

[12] LIU, P.; YUAN, W.; FU, J.; et al. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Computing Surveys*, v. 55, n. 9, Article 195, 2023. Disponível em: https://doi.org/10.1145/3560815. Acesso em: 1 maio 2026.

[13] HUI, B.; YANG, J.; CUI, Z.; et al. Qwen2.5-Coder Technical Report. arXiv preprint arXiv:2409.12186, 2024. Disponível em: https://arxiv.org/abs/2409.12186. Acesso em: 1 maio 2026.

[14] GUO, D.; ZHU, Q.; YANG, D.; et al. DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence. arXiv preprint arXiv:2401.14196, 2024. Disponível em: https://arxiv.org/abs/2401.14196. Acesso em: 1 maio 2026.

[15] CHEN, M.; TWOREK, J.; JUN, H.; et al. Evaluating Large Language Models Trained on Code. arXiv preprint arXiv:2107.03374, 2021. Disponível em: https://arxiv.org/abs/2107.03374. Acesso em: 1 maio 2026.

[16] AUSTIN, J.; ODENA, A.; NYE, M.; et al. Program Synthesis with Large Language Models. arXiv preprint arXiv:2108.07732, 2021. Disponível em: https://arxiv.org/abs/2108.07732. Acesso em: 1 maio 2026.

[17] CASSANO, F.; GOUWAR, J.; NGUYEN, D.; et al. MultiPL-E: A Scalable and Extensible Approach to Benchmarking Neural Code Generation. arXiv preprint arXiv:2208.08227, 2022. Disponível em: https://arxiv.org/abs/2208.08227. Acesso em: 1 maio 2026.

[18] LIU, J.; XIA, C. S.; WANG, Y.; ZHANG, L. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. arXiv preprint arXiv:2305.01210, 2023. Disponível em: https://arxiv.org/abs/2305.01210. Acesso em: 1 maio 2026.

[19] ZHUO, T. Y.; VU, M. C.; CHIM, J.; et al. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. arXiv preprint arXiv:2406.15877, 2024. Disponível em: https://arxiv.org/abs/2406.15877. Acesso em: 1 maio 2026.

[20] HOSKING, T.; BLUNSOM, P.; GOLDSTEIN, M. Human Feedback is not Gold Standard. In: *International Conference on Learning Representations (ICLR)*, 2024. Disponível em: https://arxiv.org/abs/2309.16349. Acesso em: 2 maio 2026.

APÊNDICE A

Ética e Conformidade no Uso de IA e Benchmarks

O presente estudo foi conduzido em conformidade com princípios éticos e de boas práticas de pesquisa em inteligência artificial e avaliação experimental. Destacamos os seguintes aspectos:

- *Transparência e Reprodutibilidade* — todos os experimentos foram documentados com detalhamento de modelos, juízes, hardware e protocolos estatísticos, permitindo replicação independente e auditoria metodológica.
- *Uso Responsável de Modelos de IA* — os modelos avaliados (`Qwen2.5-Coder:1.5B` e `DeepSeek-Coder:1.3B`) e os juízes automáticos (`Gemma2:2B` e `CodeLlama:7B`) foram utilizados em conformidade com suas respectivas licenças abertas e documentação técnica oficial, sem modificação indevida ou uso em contextos proibidos.
- *Proteção de Dados e Privacidade* — nenhum dado sensível ou pessoal foi utilizado nos benchmarks. O dataset construído (`slm_tasks_ptbr.jsonl`) contém apenas tarefas sintéticas e exemplos técnicos, assegurando ausência de risco de exposição de informações privadas.
- *Avaliação Humana Ética* — os avaliadores humanos participaram de forma voluntária, com anonimização de identificadores (`BLIND-XXXX`) e distribuição cega das tarefas, garantindo imparcialidade e proteção da identidade dos participantes.
- *Conformidade com Normas Acadêmicas* — o estudo segue recomendações contemporâneas de *AI Ethics* e *AI Engineering*, incluindo mitigação de vieses, validação cruzada com julgamento humano e documentação transparente dos protocolos experimentais.

Assim, asseguramos que o uso de IA e benchmarks neste trabalho respeita princípios de integridade científica, responsabilidade social e conformidade regulatória.

APÊNDICE B

Disponibilidade de Artefatos

Este artigo disponibiliza os artefatos completos de pesquisa para execução, auditoria e replicação do benchmark no repositório do projeto: `https://github.com/ricozimmermann/slm-benchmark-kit`.

Para orientar uso e reprodutibilidade, destacam-se os seguintes componentes:

- documentação de uso geral e execução rápida: README.md, README.en.md, README.es.md e `QUICK_GUIDE.md`;
- protocolo formal de release e rastreabilidade experimental: `docs/release_protocol.md`, `docs/release_protocol.pt-br.md`, `VERSION` e `CHANGELOG.md`;
- especificação metodológica estatística e rubrica de avaliação humana: `docs/metodologia_estatistica.md`, `docs/metodologia_estatistica.pt-br.md`, `docs/human_eval_rubric.md` e `docs/human_eval_rubric.pt-br.md`;
- configurações experimentais versionadas: `configs/benchmark_ollama.yaml`, `configs/benchmark_ollama_docker.yaml`, `configs/benchmark_local_smoke.yaml` e variantes de smoke test;

- scripts de execução e análise: `scripts/run_benchmark.py`, `scripts/analyze_results.py`, `scripts/prepare_human_eval.py` e `scripts/agreement_report.py`;
- implementação do pipeline analítico e de avaliação: `src/slm_benchmark/analysis.py`, `src/slm_benchmark/agreement.py`, `src/slm_benchmark/human_eval.py`, `src/slm_benchmark/runner.py` e módulos auxiliares;
- resultados e saídas reproduzíveis de referência: `results/raw_benchmark.jsonl`, results/report.md, `results/human_agreement.md`, `results/human_assignment.csv` e `results/human_assignment_scored.csv`;
- infraestrutura de execução reprodutível em contêiner: Dockerfile e docker-compose.yml;
- metadados de dependências e licença: pyproject.toml e LICENSE.