

# Análise Comparativa entre LLMs com o uso de Engenharia de Prompt, PMD e CheckStyle na Detecção Automatizada de Code Smells em Projetos Java

**Arthur C. Bretas<sup>1</sup>, Bernardo C Biondini<sup>2</sup>, Gabriel V. de O. Morais<sup>3</sup>, Igor M. Santos<sup>3</sup>,  
Júlia B. A. Silva<sup>4</sup>, Letícia R. B. de Paula<sup>5</sup>**

<sup>1</sup>Instituto de Ciências Exatas e Informática – Pontifícia Universidade de Minas Gerais (PUC Minas)  
Belo Horizonte – MG – Brasil

{arthur.bretas.1416299, bbiondini, igor.santos.1419578}@sga.pucminas.br

{gabriel.morais.1425529, julia.silva.1403728, leticia.blom}@sga.pucminas.br

**Resumo.** Este artigo apresenta uma análise comparativa da precisão das ferramentas de análise estática (PMD e Checkstyle) na detecção de code smells em projetos Java. Através da avaliação empírica de 1.000 repositórios populares do GitHub, avaliamos a eficácia das ferramentas em diferentes categorias de smells (complexidade, duplicação e problemas de design). Os resultados fornecem diretrizes práticas para desenvolvedores e insights de pesquisa sobre as capacidades das ferramentas de análise estática.

**Abstract.** This article presents a comparative analysis of the precision of static analysis tools (PMD, and Checkstyle) in detecting code smells in Java projects. Through empirical evaluation of 1,000 popular GitHub repositories, we assess the tools' effectiveness across different smell categories (complexity, duplication, and design problems). The results provide practical guidelines for developers and research insights into static analysis tool capabilities.

## 1. Introdução

Esta seção serve como introdução ao nosso estudo. A seguir, serão explorados os seguintes tópicos: Visão do Problema Geral, que aborda os desafios impostos pelos Code Smells; Visão do Problema Específico, que detalha a comparação das ferramentas de análise estática; Motivação, explicando a relevância de nossa pesquisa; e Justificativa, que destaca a importância de nossos achados para a qualidade do software.

### 1.1. Visão do Problema Geral

Code smells (CS), também conhecidos como "bad smells", estão associados a sintomas de problemas de manutenibilidade no software. Eles comprometem princípios básicos de design e influenciam negativamente a eficiência futura do sistema, podendo dificultar sua evolução e aumentar a probabilidade de falhas. Sua identificação é considerada um aspecto crítico para prevenir dificuldades de manutenção e promover melhorias contínuas na qualidade do código-fonte

## **1.2. Motivação**

A motivação para este artigo surge da necessidade de entender as discrepâncias na eficácia das ferramentas de análise estática, que podem resultar em falsos positivos ou na negligência de problemas críticos. A escolha inadequada de uma ferramenta pode comprometer a qualidade do código e aumentar os custos de manutenção. Portanto, analisar e comparar a precisão dessas ferramentas é fundamental para apoiar desenvolvedores, equipes de engenharia e pesquisadores na tomada de decisões informadas.

## **1.3. Justificativa**

Resolver esse problema é importante porque a qualidade do software está diretamente ligada à capacidade de identificar e corrigir code smells de forma eficiente. Ao fornecer uma análise comparativa detalhada, este artigo contribui para a seleção de ferramentas mais adequadas a cada contexto, otimizando o processo de desenvolvimento e aprimorando a qualidade do código. Além disso, os resultados podem servir como base para futuras pesquisas acadêmicas na área de qualidade de software.

## **1.4. Objetivos**

O objetivo geral deste artigo é avaliar a precisão das ferramentas SonarQube, PMD e Checkstyle na detecção de code smells em projetos Java. Como objetivos específicos, buscamos:

- Comparar a eficácia das ferramentas em diferentes categorias de code smells
- Identificar discrepâncias na classificação de code smells como críticos
- Analisar a correlação entre a popularidade de projetos Java e a quantidade de code smells detectados

## **1.5. Perguntas de Pesquisa**

Este artigo busca responder às seguintes perguntas de pesquisa:

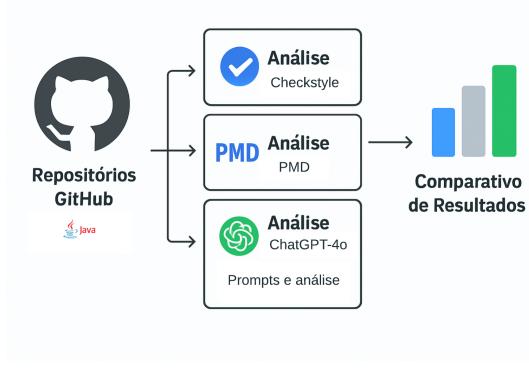
- As ferramentas analisadas priorizam da mesma maneira os code smells classificados como críticos?
- Qual ferramenta detecta a maior quantidade absoluta de code smells em categorias específicas?
- Existe correlação entre a popularidade dos projetos Java e a quantidade de code smells detectados?

## **1.6. Organização do Texto**

Este artigo está organizado da seguinte forma: a Seção 2 apresenta a metodologia utilizada no estudo; a Seção 3 descreve os resultados obtidos; a Seção 4 discute as implicações dos resultados; e a Seção 5 conclui com recomendações e direcionamentos para pesquisas futuras.

## **2. Metodologia**

Este estudo adota uma abordagem quantitativa comparativa para avaliar a detecção de *code smells* em projetos Java por três métodos distintos. Conforme ilustrado na Figura 1, o experimento segue o fluxo geral de (i) seleção e preparação do corpus, (ii) aplicação das ferramentas de análise, e (iii) comparação dos resultados obtidos. O corpus consiste



**Figure 1. Diagrama de metodologia**

em 50 repositórios Java populares (ordenados por número de estrelas no GitHub), todos contendo arquivos de build Maven ou Gradle. Cada repositório foi clonado localmente e submetido a três abordagens de análise: Checkstyle, PMD e um modelo de linguagem grande (LLM) via ChatGPT-4o. Estudos recentes mostram que modelos LLM são capazes de identificar *code smells* com precisão competitiva. Inspirados nesses trabalhos, exploramos três estratégias de *prompt* distintas para o ChatGPT-4o – *zero-shot*, *one-shot* (com exemplo de I/O) e *one-shot* calibrado (com guia técnico de code smells) – de modo a avaliar o impacto do desenho do prompt na detecção automática de smells.

## 2.1. Procedimentos experimentais

Os procedimentos do experimento podem ser divididos em 5 e estão listado a seguir:

1. Seleção e preparação do corpus: Utilizou-se a API do GitHub para recuperar repositórios Java públicos, ordenados por popularidade (estrelas) e filtrados para incluir apenas aqueles com arquivos de build Maven (pom.xml) ou Gradle (build.gradle). Os 50 primeiros repositórios que atenderam a esses critérios foram clonados via git.

2. Análise com Checkstyle: Para cada repositório, executou-se o Checkstyle (versão 7.13.0) com um conjunto predefinido de regras de estilo e manutenção de código. As mensagens de violação geradas pelo Checkstyle (indicando potenciais code smells ou inconsistências de código) foram capturadas em formato bruto (XML) e posteriormente parseadas.

3. Análise com PMD: De forma semelhante, cada projeto foi analisado com o PMD (versão 10.24.0) usando um ruleset que cobre code smells conhecidos (como *Long-Method*, *GodClass*, *DataClass*, etc.). O PMD produz relatórios de violações em XML; esses arquivos foram processados por um parser Python, extraíndo os detalhes de cada ocorrência de smell (tipo, arquivo, linha e mensagem).

4. Análise com ChatGPT-4o: Cada arquivo de código-fonte relevante (classes Java) foi enviado ao ChatGPT-4o via API (OpenAI) para identificação de *code smells*. Três variações de *prompt* foram empregadas: *Zero-shot*: o modelo recebeu somente a instrução de identificar *code smells* no código, sem exemplos prévios. *One-shot*: além do código alvo, o prompt incluía um exemplo demonstrativo de código e saída desejada em JSON, ilustrando como reportar um *code smell*. *One-shot* calibrado: similar ao *one-shot*, mas acrescentando um guia técnico resumido de code smells (baseado na literatura) para

orientar o modelo sobre os tipos de smells esperados.

Essa escolha de estratégias segue evidências de que prompts com exemplos e contexto técnico melhoram a performance dos LLMs. Em cada execução, solicitou-se explicitamente ao ChatGPT que respondesse em formato JSON contendo os atributos das instâncias de smell (veja adiante).

5. Coleta e padronização dos outputs: Os resultados de cada abordagem foram agregados em um formato JSON unificado. Desenvolveu-se um script Python que mapeou as saídas: cada ocorrência de *code smell* foi representada como um objeto JSON com campos padronizados (por exemplo, "tipo" do *smell*, arquivo, linha, mensagem e severidade/regra associada). Esse formato uniforme permitiu comparar diretamente as saídas das três abordagens e cruzar os dados de maneira sistemática.

## 2.2. Ferramentas e Tecnologias Utilizadas

**GitHub API (Python):** Bibliotecas como PyGitHub e requests foram utilizadas para autenticação e requisição de repositórios. Um script Python automatizou a coleta dos metadados (URL de clone, estrelas, presença de pom.xml ou build.gradle) e a clonagem via git.

**Checkstyle:** Ferramenta estática para Java que detecta violações de estilo e práticas de código. Foi usada a versão 7.13.0, com um conjunto de regras voltadas a métricas de código e heurísticas de smells. A saída foi exportada em XML e convertida para JSON.

**PMD:** Outra ferramenta de análise estática (versão 10.24.0) que identifica problemas de código, incluindo diversos code smells. Executou-se o PMD via linha de comando apontando para o ruleset padrão de Java; os relatórios em XML foram parseados para JSON com campos padronizados.

**ChatGPT-4o (OpenAI API):** O modelo foi acessado por meio da API do OpenAI, com modelo gpt-4o. Os prompts foram gerados dinamicamente em Python, seguindo a estratégia (zero/one-shot) definida. O parâmetro temperature foi fixado em 0 para maximizar determinismo. O JSON de resposta do modelo foi extraído e validado.

**Python (versão 3.10.12):** Usado para orquestrar todo o experimento. Bibliotecas para processamento de arquivos (JSON, XML), execução de comandos do SO (subprocess) e análise de similaridade (ex.: scipy, pandas) foram empregadas.

## 2.3. Métricas e Variáveis

Para avaliar comparativamente as abordagens, definiram-se as seguintes métricas:

**Contagem total de *code smells*:** número absoluto de ocorrências identificadas por cada abordagem em cada repositório (e agregada no corpus). Essa métrica fornece a base de comparação inicial do volume de detecções.

**Similaridade entre conjuntos de *smells*:** mediu-se a sobreposição de resultados entre pares de abordagens usando o índice de Jaccard (razão entre interseção e união dos conjuntos de *smells* identificados). Altos valores de Jaccard indicam forte correspondência.

Divergência/exclusividade: quantidade e porcentagem de *smells* detectados por uma ferramenta, mas não pelas outras. Essa métrica evidencia quão distintos são os conjuntos de detecções (por exemplo, smells únicos ao ChatGPT vs. únicos ao PMD).

Distribuição por tipo de smell: análise qualitativa da frequência de cada categoria de smell (e.g., *Long Method*, *God Class*) em cada abordagem, permitindo verificar se certos tipos são mais frequentemente capturados por uma ferramenta.

Como variáveis, considerou-se principalmente a abordagem de análise (Checkstyle, PMD, ChatGPT *zero-shot*, ChatGPT *one-shot*, ChatGPT calibrado) e os valores derivados acima (contagem, índices de similaridade, etc.). As comparações estatísticas focaram em testar diferenças nas médias/medianas de contagem de *smells* e nas medidas de similaridade entre abordagens.

## 2.4. Configuração Experimental

O experimento foi conduzido em ambiente controlado. Todas as ferramentas (Checkstyle, PMD, Java SDK 11) foram instaladas localmente nas versões fixas mencionadas. Para o ChatGPT-4o via API, utilizou-se chave de acesso e parâmetros padronizados (ex. *time-out* de requisição, temperatura=0, máximo de *tokens* suficiente para abranger a resposta JSON).

Foram adotadas as seguintes validações e precauções: (i) os prompts foram testados inicialmente em um *subset* de exemplos para assegurar que o modelo retornasse a estrutura JSON esperada; (ii) eventuais respostas inválidas ou incompletas do ChatGPT-4o foram detectadas e reexecutadas; (iii) os scripts de parsing de XML e JSON incluem verificações de esquema (ex. campos obrigatórios de smell) para evitar dados corrompidos. Todo o processo foi documentado e executado de forma repetível, sem intervenção manual nos resultados.

## 2.5. Questões de pesquisa e hipóteses

Para guiar a análise quantitativa, definiram-se três perguntas de pesquisa implícitas, cujas hipóteses nulas ( $H_0$ ) e alternativas ( $H_A$ ) são:

1. Pergunta 1: Há diferença no número de *code smells* detectados por cada abordagem?

$H_{0,1}$ : Não existe diferença significativa no número médio de code smells detectados entre Checkstyle, PMD e ChatGPT-4o.

$H_{A,1}$ : Há diferença significativa no número médio de *code smells* detectados por pelo menos uma das abordagens.

A análise para resposta da Pergunta 1 considera as seguintes métricas: Métrica 1.1: Número total de *code smells* detectados por abordagem. Métrica 1.2: Diferença média por repositório. Métrica 1.3: Similaridade com PMD (índice de Jaccard). Métrica 1.4: Divergência com PMD. Métrica 1.5: Similaridade com Checkstyle (índice de Jaccard) Métrica 1.6: divergência com Checkstyle.

Essas métricas serão calculadas comparando os conjuntos de smells extraídos de cada ferramenta para os mesmos repositórios, padronizados em JSON. O índice de Jaccard será usado para medir a sobreposição entre os conjuntos, e a divergência será obtida com base nos smells exclusivos da LLM.

2. Pergunta 2: Os conjuntos de smells identificados são semelhantes entre as abordagens?

$H_{0,2}$ : As abordagens produzem conjuntos de smells com grau de sobreposição (similaridade) estatisticamente equivalente; ou seja, não há diferença significativa na composição das detecções.

$H_{A,2}$ : Existe diferença significativa na similaridade/dissimilaridade dos conjuntos de smells detectados por diferentes abordagens.

A metodologia segue os mesmos princípios da Pergunta 1, utilizando agora os resultados da variação *one-shot* do LLM. As métricas associadas são: Métrica 2.1: Total de *code smells* por abordagem. Métrica 2.2: Diferença média de detecção por repositório. Métrica 2.3 e 2.4: Similaridade e divergência com PMD. Métrica 2.5 e 2.6: Similaridade e divergência com Checkstyle. A análise busca verificar se a inclusão de um exemplo no prompt (estratégia *one-shot*) melhora a performance do LLM em relação às ferramentas tradicionais.

3. Pergunta 3: O tipo de prompt afeta o desempenho do ChatGPT-4o?

$H_{0,3}$ : As variações de prompt (*zero-shot*, *one-shot*, *one-shot* calibrado) não têm efeito significativo na quantidade ou qualidade dos code smells detectados pelo ChatGPT-4o.

$H_{A,3}$ : Pelo menos uma das variações de prompt (ex.: *one-shot* calibrado) leva a diferenças significativas nos resultados de detecção de smells.

Será feita uma comparação focada exclusivamente entre PMD e a variação calibrada da LLM. As métricas envolvidas são: Métrica 3.1: Número total de *code smells* detectados. Métrica 3.2: Diferença média por repositório. Métrica 3.3: Similaridade com PMD (índice de Jaccard). Métrica 3.4: Divergência com PMD. Essa etapa visa identificar se a adição de contexto técnico ao prompt (regras derivadas do PMD) resulta em maior alinhamento ou desempenho superior por parte do modelo LLM.

Essas hipóteses serão testadas estatisticamente com base nas métricas definidas. A seguir, os dados coletados serão submetidos a análise comparativa para confirmar ou rejeitar  $H_0$  em cada caso.

Referências do framework de prompts: conforme evidenciam [Mesbah et al. 2025], a inclusão de exemplos nos prompts (*few-shot*) pode melhorar significativamente a precisão de modelos LLM em tarefas de detecção de code smells. Essa motivação, aliada a trabalhos que recomendam guias de domínio nos prompts, fundamentou a escolha das estratégias de prompt usadas aqui.

### 3. Trabalhos Relacionados

Os trabalhos relacionados discutidos nesta seção incluem a aplicação de *Large Language Models (LLMs)* na detecção automatizada de *code smells*, comparativos entre ferramentas tradicionais de análise estática, e estudos sobre o uso de técnicas de engenharia de *prompt* para análise de código.

## Aplicação de *LLMs* na Detecção de *Code Smells*

[Mesbah et al. 2025] investigam a aplicação de *LLMs* baseados em *prompts* para detecção de *code smells*, utilizando modelos *GPT-4* e *LLaMA* no *dataset Machine Learning Code Quality (MLCQ)*. Os autores conduzem uma análise extensa do desempenho desses modelos quando solicitados a identificar e classificar *code smells*, avaliando sistematicamente precisão, *recall* e capacidade de generalização. Os resultados evidenciam que *LLMs* podem ser eficazes na detecção de *code smells* quando adequadamente *prompt-engineered*. Contudo, esse estudo se concentra exclusivamente em *LLMs* e não estabelece comparações diretas com ferramentas tradicionais de análise estática, limitando-se a um único *dataset* e não explorando diferentes estratégias de *prompting* como *zero-shot*, *one-shot* e calibração específica.

[Silva et al. 2024] apresentaram *insights* iniciais sobre a eficácia do *ChatGPT* (versão 3.5-Turbo) na detecção de *code smells* em projetos *Java*. Eles utilizaram um grande conjunto de dados compreendendo quatro *code smells* (*Blob*, *Data Class*, *Feature Envy* e *Long Method*) classificados em três níveis de severidade. Para avaliar a proficiência do *ChatGPT*, empregaram dois *prompts* diferentes: um genérico e outro que especificava os *smells* selecionados. Os resultados revelaram que a probabilidade de o *ChatGPT* fornecer um resultado correto com um *prompt* específico é 2,54 vezes maior em comparação com um *prompt* genérico. Além disso, o *ChatGPT* foi mais eficaz na detecção de *smells* com severidade crítica (*F-measure* = 0.52) do que aqueles com severidade menor (*F-measure* = 0.43). Este artigo se relaciona diretamente com este trabalho, pois valida a motivação para explorar diferentes estratégias de *prompt* (*zero-shot*, *one-shot*, *one-shot* calibrado) com *LLMs* para a detecção de *code smells*. Os resultados de [Silva et al. 2024] sobre a eficácia de *prompts* específicos em melhorar o desempenho do *ChatGPT* na identificação de *smells* apoiam a hipótese de que o tipo de *prompt* afeta o desempenho do *ChatGPT-4o* na detecção de *code smells*. A análise de precisão, *recall* e *F-measure* também se alinha com as métricas usadas no trabalho.

## Comparação entre ferramentas de análise estática na Detecção de *Code Smells*

[Lenarduzzi et al. 2021] realizaram uma comparação em larga escala de seis ferramentas populares de análise estática (ASATs) para projetos *Java*: *Better Code Hub*, *CheckStyle*, *Coverity Scan*, *Findbugs*, *PMD* e *SonarQube*. O estudo buscou entender quais problemas de qualidade de código podem ser detectados por essas ferramentas, qual o grau de concordância entre elas e qual a precisão de suas recomendações. Os principais resultados apontam para uma baixa concordância entre as ferramentas e um baixo grau de precisão, com a precisão variando entre 18% e 86%. Eles também observaram que, entre as ferramentas, o *SonarQube* foi capaz de detectar a maioria dos problemas de qualidade que podem ser identificados pelas outras ASATs, mas diferentes ferramentas identificam diferentes formas de problemas de qualidade. Este estudo é relevante para o trabalho, pois fornece uma base sólida para a análise comparativa das ferramentas *SonarQube*, *PMD* e *CheckStyle*. Os achados sobre a baixa concordância e precisão dessas ferramentas tradicionais reforçam a necessidade da pesquisa e podem ser usados para contextualizar os resultados, especialmente ao comparar a eficácia dos *LLMs* com essas abordagens estabelecidas.

## **Engenharia de *Prompt* e Análise de Código**

[Cordeiro et al. 2024] realizaram um estudo empírico abrangente sobre a capacidade de refatoração de código de *LLMs*, utilizando o *StarCoder2*. O estudo avaliou se o código refatorado pela *LLM* ou por desenvolvedores é mais eficaz na melhoria da qualidade do código, analisou as diferenças entre os tipos de refatoração aplicados pela *LLM* e desenvolvedores, e avaliou se a qualidade do código refatorado pela *LLM* poderia ser melhorada por meio de *one-shot prompting* e *chain-of-thought prompting*. Os autores descobriram que o *StarCoder2* supera os desenvolvedores na redução de *code smells* em 20,1% em refatorações geradas automaticamente. O *StarCoder2* se destaca na redução de mais tipos de *code smells*, como *Long Statement*, *Magic Number*, *Empty Catch Clause* e *Long Identifier*. No entanto, os desenvolvedores se saem melhor na correção de problemas complexos, como *Broken Modularization*, *Deficient Encapsulation* e *Multifaceted Abstraction*. Além disso, o *StarCoder2* supera os desenvolvedores em tipos de refatoração mais sistemáticos e repetitivos, enquanto os desenvolvedores se destacam em refatorações que exigem uma compreensão mais profunda do contexto e da arquitetura do código. Suas descobertas mostraram que o *one-shot prompting* melhora a taxa de aprovação de testes de unidade em relação ao *zero-shot prompt* em 6,15% e reduz *code smells* em uma taxa 3,52% maior. Este artigo se relaciona diretamente com o nosso trabalho, pois ambos investigam a eficácia de *LLMs* na detecção e refatoração de *code smells*, embora nosso estudo foque na comparação da precisão de ferramentas de análise estática tradicionais (*SonarQube*, *PMD*, *Checkstyle*) com o *ChatGPT-4o* e explore o impacto de diferentes estratégias de *prompt* especificamente na detecção de *code smells*.

[ZHA 2024] fornecem uma avaliação sistemática de *LLMs* de código em várias tarefas de engenharia de software, incluindo análise de código. Os autores exploram capacidades e limitações dos *LLMs* em compreensão, geração e análise de código, destacando tanto oportunidades quanto desafios significativos. O estudo conclui que, embora *LLMs* demonstrem potencial considerável, pesquisas adicionais são necessárias para otimizar eficiência, precisão e segurança. Este trabalho oferece fundamentação teórica para o presente estudo, mas mantém escopo mais amplo e não foca especificamente na detecção de *code smells* nem em comparações com ferramentas tradicionais como *PMD* e *CheckStyle*.

## **Limitações e Oportunidades**

A análise dos trabalhos relacionados revela lacunas importantes na literatura atual. Primeiramente, poucos estudos realizam comparações sistemáticas e diretas entre *LLMs* e ferramentas tradicionais estabelecidas como *PMD* e *CheckStyle*. Segundo, a maioria dos trabalhos não explora diferentes estratégias de engenharia de *prompt* (*zero-shot*, *one-shot*, calibração específica) e seu impacto na eficácia da detecção. Terceiro, há ausência de metodologias padronizadas para avaliação quantitativa e comparativa dessas abordagens em cenários reais de desenvolvimento.

O presente trabalho se diferencia da literatura existente ao: (i) realizar uma comparação sistemática e quantitativa entre *LLMs* (através de diferentes estratégias de *prompting*) e ferramentas tradicionais consolidadas (*PMD* e *CheckStyle*); (ii) explorar sistematicamente três abordagens de engenharia de *prompt* (*zero-shot*, *one-shot* e calibração baseada em regras do *PMD*); (iii) estabelecer métricas padronizadas de similaridade e divergência para quantificar diferenças entre abordagens; e (iv) avaliar eficácia em projetos

reais de software, fornecendo *insights* práticos para escolha de ferramentas em contextos de desenvolvimento industrial.

## Resultados

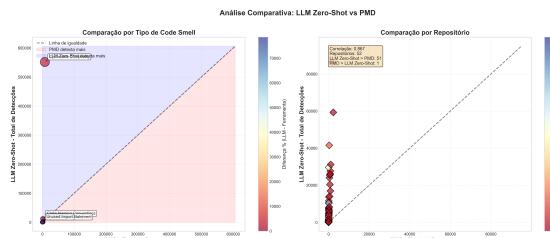
Esta seção apresenta os resultados comparativos da detecção de *code smells* utilizando LLMs com diferentes estratégias de *prompt* (zero-shot, one-shot e calibrado) em relação às ferramentas tradicionais PMD e CheckStyle.

### Questão 1: Qual das abordagens detecta mais *code smells*, LLM com *prompt* zero shot, PMD ou CheckStyle?

#### Análise Comparativa: LLM Zero-Shot vs PMD

A análise comparativa entre LLM Zero-Shot e PMD e para a detecção de *code smells* revelou o seguinte:

- **Comparação por Tipo de *Code Smell*:** A figura 2, apresenta o gráfico de comparação por tipo de *code smell*. Nela, é possível observar que o PMD detectou uma quantidade significativamente maior de *code smells* em comparação com o LLM Zero-Shot para a maioria das categorias. A linha de igualdade no gráfico indica os casos em que as detecções foram semelhantes.
- **Comparação por Repositório:** A figura 2 também mostra a comparação por repositório. A correlação entre as detecções do LLM Zero-Shot e PMD foi de 0.867 para 52 repositórios analisados. Em 51 repositórios, o LLM Zero-Shot detectou mais *code smells* do que o PMD, enquanto em 1 repositório o PMD detectou mais.

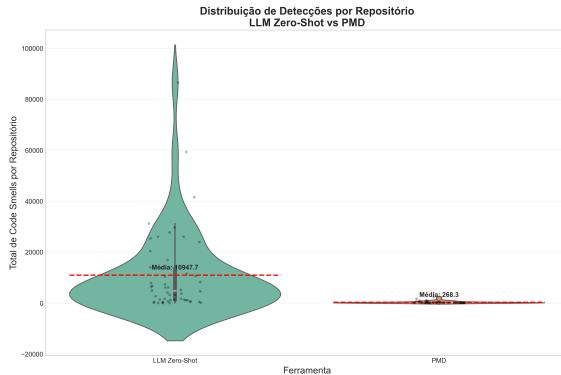


**Figure 2. Analise Comparativa: LLM Zero Shot vs PMD**

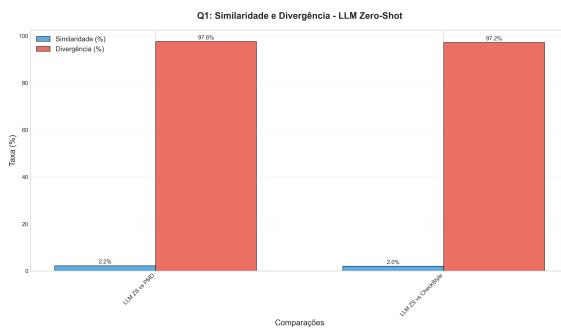
- **Distribuição de Detecções por Repositório:** A figura a seguir, ilustra a distribuição total de *code smells* por repositório. Nela, percebe-se que o LLM Zero-Shot teve uma média de 10947.7 detecções, enquanto o PMD teve uma média de 268.3 detecções.
- **Similaridade e Divergência:** A figura abaixo detalha as taxas de similaridade e divergência. A taxa de similaridade entre os *code smells* detectados pelo LLM Zero-Shot e PMD foi de 2.2%, e a taxa de divergência foi de 97.6%.

#### Análise Comparativa: LLM Zero-Shot vs CheckStyle

A análise comparativa entre LLM Zero-Shot e CheckStyle para a detecção de *code smells* revelou o seguinte:

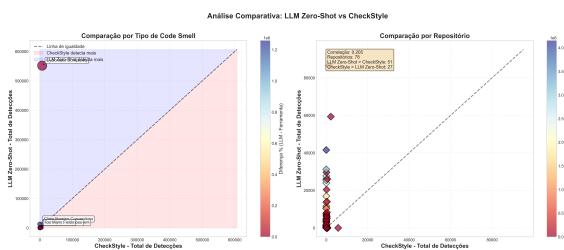


**Figure 3. Distribuição de Detecções por Repositório LLM Zero Shot e PMD**



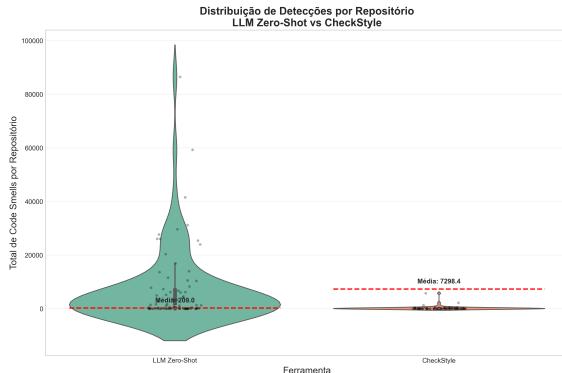
**Figure 4. Similaridade e Divergência - LLM Zero-Shot**

- **Comparação por Tipo de *Code Smell*:** A figura a seguir apresenta o gráfico de comparação por tipo de *code smell* para LLM Zero-Shot e CheckStyle. Pode-se observar que o CheckStyle detectou uma quantidade maior de *code smells* em comparação com o LLM Zero-Shot para a maioria das categorias.
- **Comparação por Repositório:** A figura abaixo mostra a comparação por repositório para LLM Zero-Shot e CheckStyle. A correlação entre as detecções do LLM Zero-Shot e CheckStyle foi de 0.295 para 78 repositórios analisados. O LLM Zero-Shot detectou mais *code smells* em 51 repositórios, enquanto o CheckStyle detectou mais em 27 repositórios.



**Figure 5. Analise Comparativa: LLM Zero Shot vs PMD**

- **Distribuição de Detecções por Repositório:** A figura a seguir ilustra a distribuição total de *code smells* por repositório para LLM Zero-Shot e CheckStyle. Nela, percebe-se que o LLM Zero-Shot teve uma média de 209.0 detecções, enquanto o CheckStyle teve uma média de 7298.4 detecções.



**Figure 6. Distribuição de Detecções por Repositório**

- **Similaridade e Divergência:** Conforme a figura 4, a taxa de similaridade entre os *code smells* detectados pelo LLM Zero-Shot e CheckStyle foi de 2.0%, e a taxa de divergência foi de 97.2%.

**Questão 2: Qual das abordagens detecta mais *code smells*, LLM com *prompt one shot*, PMD ou CheckStyle?**

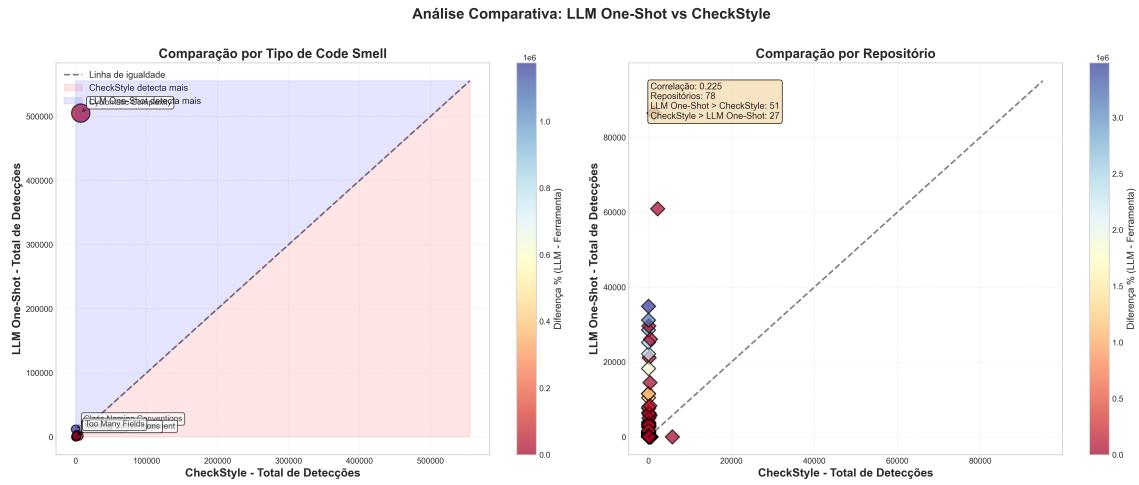
#### Análise Comparativa: LLM One-Shot vs CheckStyle

A análise comparativa entre LLM One-Shot e CheckStyle para a detecção de *code smells* apresentou os seguintes resultados:

- **Comparação por Tipo de *Code Smell*:** A figura a seguir, apresenta o gráfico de comparação por tipo de *code smell*. Similar à LLM Zero-Shot, o CheckStyle detectou um número maior de *code smells* na maioria das categorias em comparação com a LLM One-Shot.

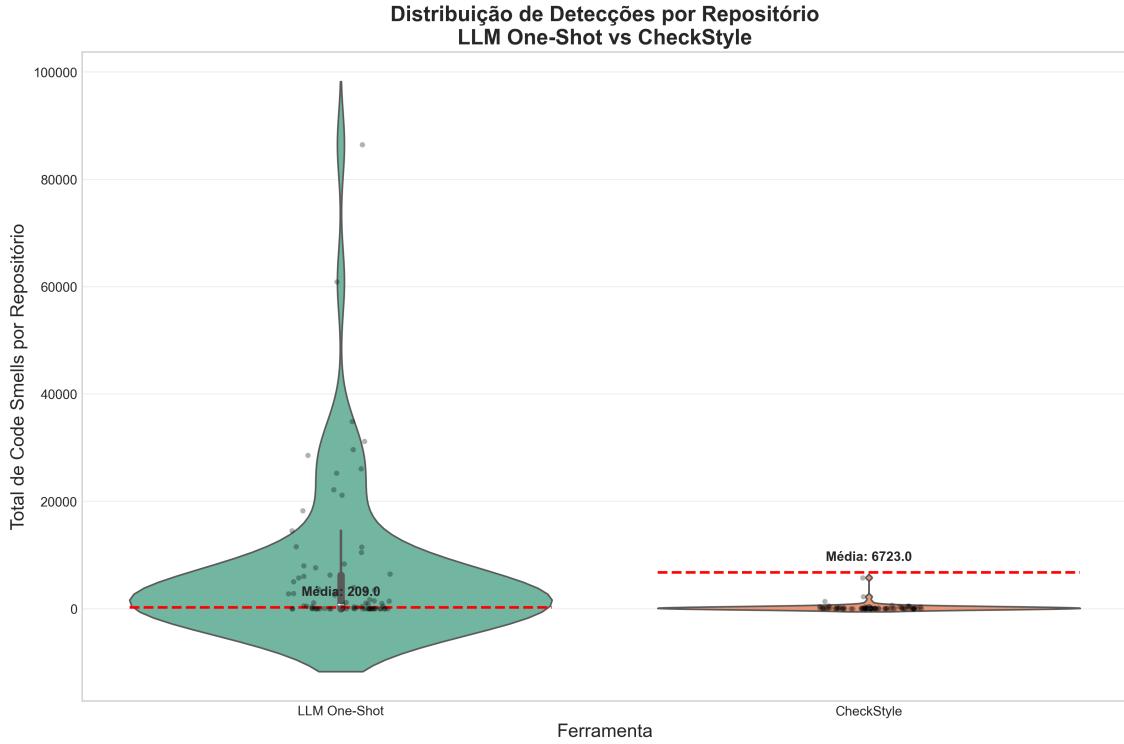
#### Análise Comparativa: LLM One-Shot vs CheckStyle

- **Comparação por Repositório:** A figura abaixo, mostra a comparação por repositório. A correlação entre as detecções do LLM One-Shot e CheckStyle foi de 0.225 para 78 repositórios analisados. O LLM One-Shot detectou mais *code smells* em 51 repositórios, enquanto o CheckStyle detectou mais em 27 repositórios.



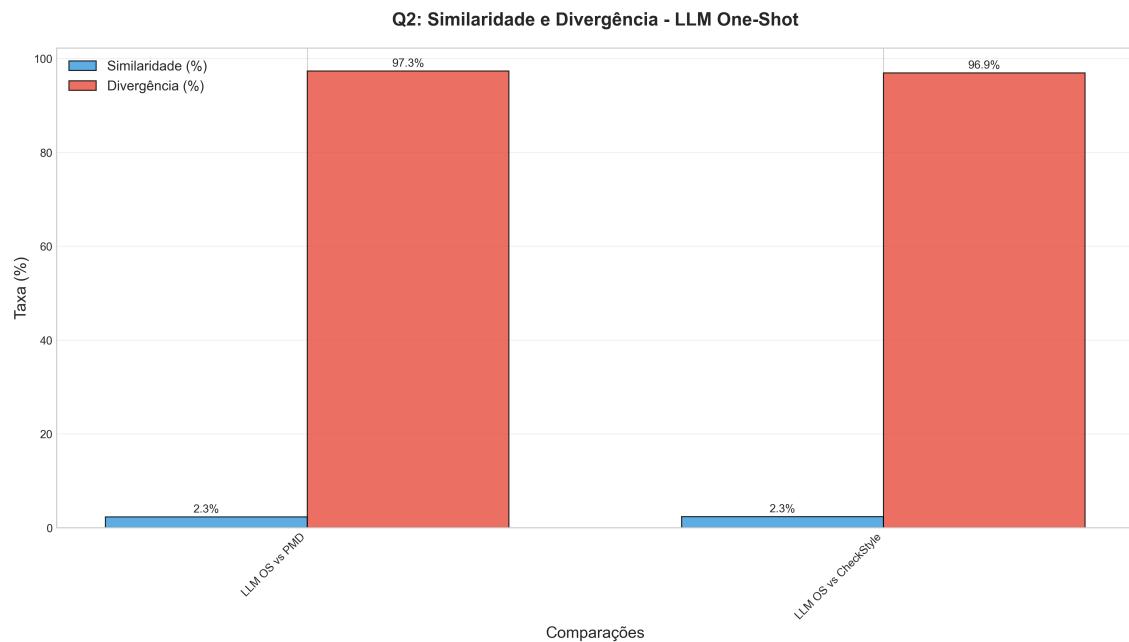
**Figure 7. Comparação por Tipo de Code Smell entre LLM (one-shot) e CheckStyle**

- **Distribuição de Detecções por Repositório:** A figura a seguir ilustra a distribuição de detecções. A média de detecções para LLM One-Shot foi de 209.0, enquanto para CheckStyle foi de 6723.0. [cite: 56]



**Figure 8. Distribuição de Detecções por Repositório LLM (one-shot) e CheckStyle**

- **Similaridade e Divergência:** A figura a seguir detalha as taxas de similaridade e divergência. A taxa de similaridade entre os *code smells* detectados pelo LLM One-Shot e PMD foi de 2.3%, com uma divergência de 97.3%. Para CheckStyle, a similaridade foi de 2.3% e a divergência foi de 96.9%.



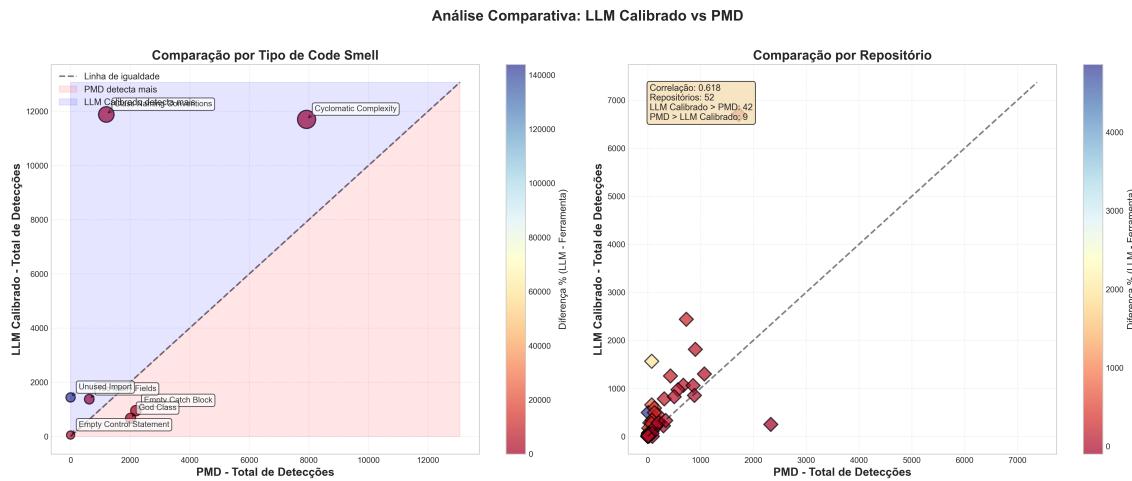
**Figure 9. Similaridade e Divergência LLM (one-shot) e CheckStyle**

**Questão 3: Qual das abordagens detecta mais *code smells*, LLM com um *prompt* calibrado baseado nas regras do PMD, ou a ferramenta PMD?**

### Análise Comparativa: LLM Calibrado vs PMD

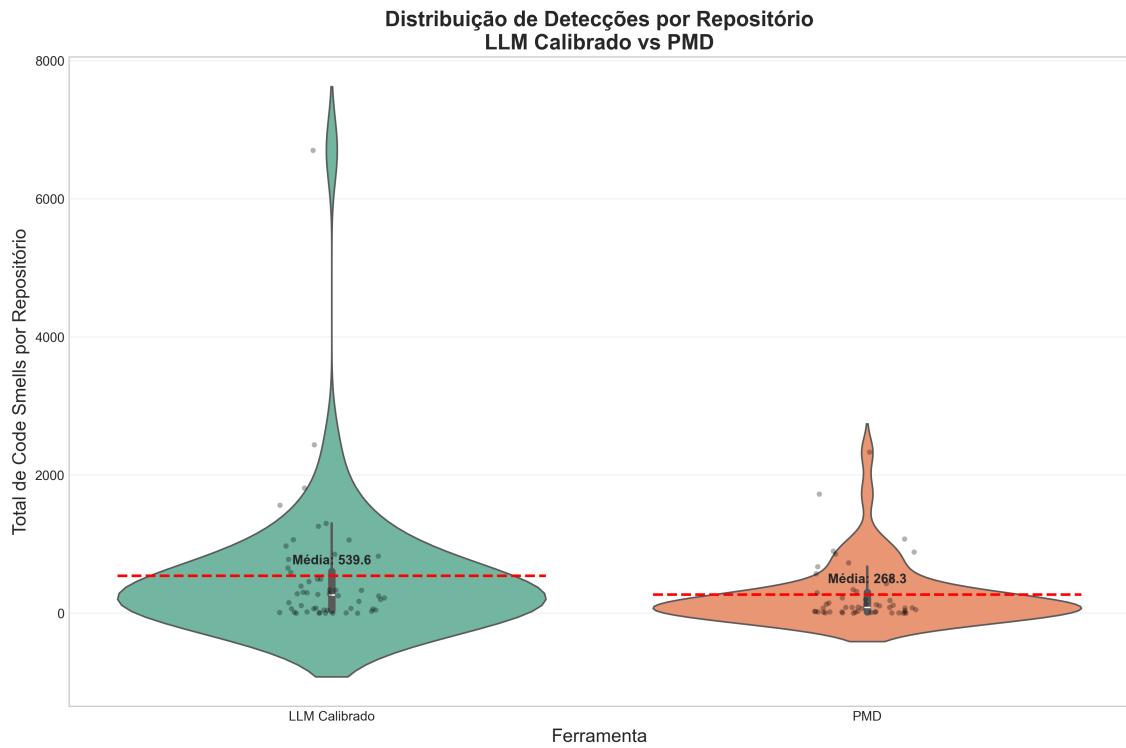
A comparação entre o LLM Calibrado (com *prompt* baseado nas regras do PMD) e a ferramenta PMD revelou:

- **Comparação por Tipo de *Code Smell*:** A figura a seguir mostra o gráfico de comparação por tipo de *code smell*. O gráfico indica que o PMD ainda supera o LLM Calibrado na detecção de algumas categorias de *code smells*, como "Cyclo-matic Complexity". No entanto, para outras categorias, o LLM Calibrado mostrou detecções em maior número.



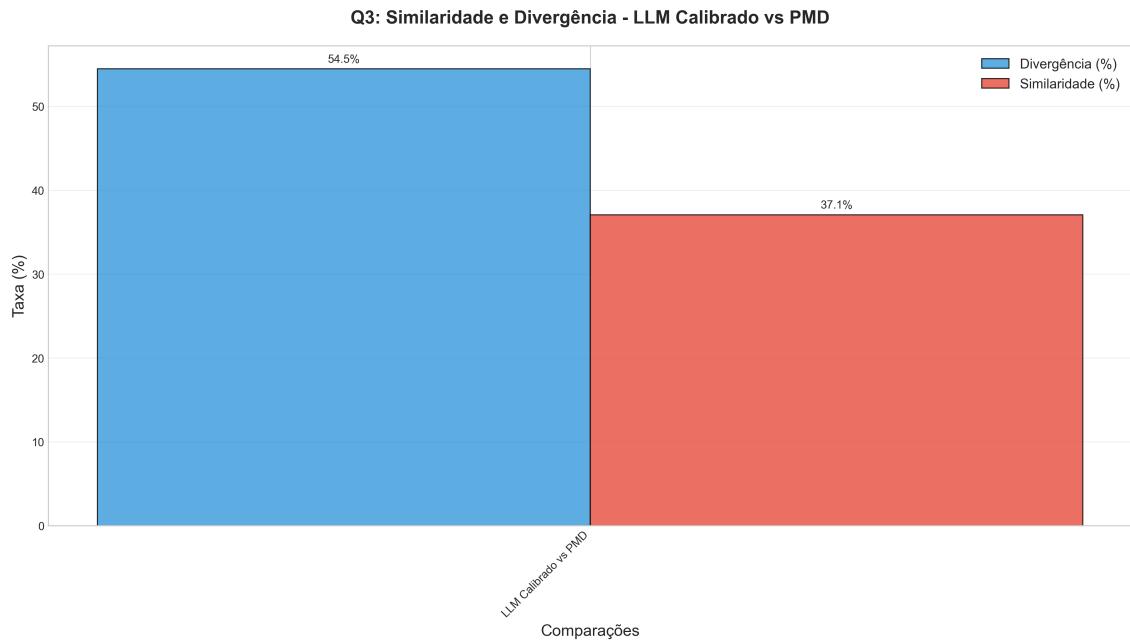
**Figure 10. Comparação por Tipo de Code Smell**

- **Distribuição de Detecções por Repositório:** A figura a seguir ilustra a distribuição de detecções. A média de detecções para LLM Calibrado foi de 539.6, enquanto para PMD foi de 268.3.



**Figure 11. Distribuição de Detecções por Repositório**

- **Similaridade e Divergência:** A figura abaixo detalha as taxas de similaridade e divergência. A taxa de similaridade entre os *code smells* detectados pelo LLM Calibrado e PMD foi de 37.1%, e a taxa de divergência foi de 54.5%.



**Figure 12. Similaridade e Divergência (LLM Calibrado - PMD)**

## References

- [ZHA 2024] (2024). Code smell detection based on supervised learning models: A survey. *Neurocomputing*, 565:127014.
- [Cordeiro et al. 2024] Cordeiro, J., Noei, S., and Zou, Y. (2024). An empirical study on the code refactoring capability of large language models. *arXiv preprint arXiv:2411.02320*.
- [dos Reis et al. 2022] dos Reis, J. P., e Abreu, F. B., d. F. Carneiro, G., and Anslow, C. (2022). Code smells detection and visualization: A systematic literature review. *Archives of Computational Methods in Engineering*, 29:47–94.
- [Lenarduzzi et al. 2021] Lenarduzzi, V., Lujan, S., Saarimaki, N., and Palomba, F. (2021). A critical comparison on six static analysis tools: Detection, agreement, and precision.
- [Mesbah et al. 2025] Mesbah, D., El Madhoun, N., Al Agha, K., and Chalouati, H. (2025). Leveraging prompt-based large language models for code smell detection: A comparative study on the mlcq dataset. In Barolli, L., editor, *Advances in Internet, Data and Web Technologies*, pages 444–454, Cham. Springer Nature Switzerland.
- [Silva et al. 2024] Silva, L. L., Silva, J., Montandon, J. E., Andrade, M., and Valente, M. T. (2024). Detecting code smells using chatgpt: Initial insights. page 7 pages, Belo Horizonte, Minas Gerais, Brasil.