Avaliação de Pagamento de Dívida técnica Automatizado com LLM em Projetos Python

Bernardo Oliveira Pires[1]
[1]**Pontifícia Universidade Católica de Minas Gerais (PUC Minas) – Praça da Liberdade**
Caixa Postal 14248 – CEP 30535-901 – Belo Horizonte – MG – Brasil
bernardoopires14@gmail.com

1

Abstract. *This study investigates the effectiveness of automated payment of self-admitted technical debt (SATD) using Large Language Models (LLMs), focusing on GPT-4o. Although LLMs are increasingly applied to software maintenance, there is little empirical evidence on how closely their edits resemble real developer actions. The study evaluates whether GPT-4o can approximate human SATD repayment when removing comments such as TODO, FIXME, and HACK. Popular Python repositories from GitHub were mined, SATD-removal commits were identified using PyDriller, and the "before" code of each case was submitted to GPT-4o under zero-shot and few-shot prompts. The generated variants were compared with the human commits using structural metrics and diff-based similarity measures. The results provide practical evidence about the usefulness and limitations of GPT-4o as a support tool for automated technical-debt repayment.*

Resumo. *Este trabalho investiga a efetividade do pagamento automatizado de Dívida Técnica Auto-Admitida (SATD) com modelos de linguagem de grande porte, focando no GPT-4o. Apesar do uso crescente de LLMs na manutenção de software, ainda faltam evidências empíricas sobre o quão próximas suas modificações estão das ações de desenvolvedores reais. O estudo avalia se o GPT-4o consegue aproximar o pagamento humano de SATD ao remover comentários como TODO, FIXME e HACK. Repositórios Python do GitHub foram minerados, commits de remoção de SATD foram identificados com PyDriller e o código "before" foi submetido ao GPT-4o em modos zero-shot e few-shot. As versões geradas foram comparadas aos commits humanos usando métricas estruturais e de similaridade. Os resultados oferecem evidências práticas sobre o potencial e as limitações do GPT-4o como apoio ao pagamento automatizado de dívida técnica.*

Bacharelado em Engenharia de *Software* - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientador de conteúdo (TCC I): João Pedro Oliveira Batisteli - 1019080@sga.pucminas.br
Orientador de conteúdo (TCC I): Joana Souza - joanasouza@pucminas.br
Orientador de conteúdo (TCC I): Leonardo Vilela - leonardocardoso@pucminas.br
Orientador acadêmico (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientador do TCC II: Jose Laerte Pires Xavier Junior - jlpxjunior@sga.pucminas.br

Belo Horizonte, 23 de Novembro de 2025

1. Introdução

O uso crescente da Inteligência Artificial (IA) tem transformado o desenvolvimento de software, especialmente na automação de tarefas como o pagamento de dívida técnica de código [Midolo and Di Penta 2025]. Neste trabalho, o foco recai sobre um subtipo específico dessa dívida, a Dívida Técnica Auto-Admitida (*SATD, Self-Admitted Technical Debt*), na qual o próprio desenvolvedor registra em comentários do código (por exemplo, TODO, FIXME, HACK) a existência de problemas, limitações ou atalhos a serem tratados futuramente [Sheikhaei et al. 2025]. Embora essa automação traga benefícios como agilidade e redução de custos, estudos recentes alertam para um possível efeito colateral da sua adoção: o aumento da dívida técnica, especialmente quando utilizada sem critérios adequados ou ferramentas apropriadas de avaliação contínua da qualidade [Zhang et al. 2022]. Nesse cenário, o pagamento de dívida técnica automatizado com modelos de linguagem como o GPT-4o (um LLM, *Large Language Model*) vêm sendo investigadas como alternativas promissoras para mitigar a tais problemas ao longo do ciclo de vida do software [Shirafuji et al. 2023].

Apesar do avanço das ferramentas de IA no suporte à engenharia de software, ainda persistem incertezas sobre sua efetividade prática na melhoria da qualidade, conforme sugerem estudos como Guo et al. (2024), que destacam que, embora modelos como o ChatGPT consigam propor alterações estruturais plausíveis, muitas dessas sugestões não resultam em benefícios claros em termos de redução de dívida técnica. Além disso, pesquisas apontam que o pagamento de dívida técnica automatizado pode introduzir inconsistências ou padrões não idiomáticos, dificultando a manutenção [Zhang et al. 2022, Midolo and Di Penta 2025].

Permanece, portanto, incerta a eficácia prática de modelos como o GPT-4o na redução da dívida técnica em códigos escritos em Python quando avaliados por critérios objetivos, como complexidade ciclomática (CC, *Cyclomatic Complexity*). Mesmo quando as modificações propostas são plausíveis, não há garantia de que se aproximem das melhorias de fato aplicadas por desenvolvedores humanos em seus commits. Diante disso, torna-se necessário verificar até que ponto os pagamentos de dívida técnica produzidos pelo modelo realmente contribuem para pagar a dívida técnica, aproximando-se dos padrões de edição observados em projetos reais. A verificação dessa eficácia, bem como da sua aderência ao pagamento efetuado por desenvolvedores humanos, é essencial para compreender os benefícios e limitações do uso de IA na manutenção e na mitigação da dívida técnica ao longo do ciclo de vida dos projetos.

A dívida técnica impacta diretamente a manutenibilidade e o custo operacional de sistemas de software. Embora ferramentas de IA venham sendo adotadas para apoiar o desenvolvimento, ainda há incertezas sobre sua contribuição real para redução de complexidade e melhoria de sustentabilidade [Guo et al. 2024]. O acúmulo de dívida técnica decorre, muitas vezes, de decisões rápidas e tende a comprometer a evolução do sistema, exigindo soluções proativas para sua gestão [Zhang et al. 2022]. Nesse contexto, a automação apoiada por IA é promissora; entretanto, sem critérios claros pode introduzir comportamentos inesperados e falhas sutis [Midolo and Di Penta 2025]. Avaliar o pagamento de dívida técnica com GPT-4o constitui, assim, questão de qualidade de software e de gestão eficiente de recursos.

Este trabalho tem como objetivo geral avaliar a eficácia do modelo GPT-4o na

automação do pagamento de Dívida Técnica Auto-Admitida (SATD, *Self-Admitted Technical Debt*) em projetos Python, e sua proximidade em relação ao pagamento de dívida técnica humano, observadas em commits. O estudo é conduzido de forma empírica: são identificados commits em repositórios populares que removem comentários SATD (`TODO`, `FIXME`, `HACK`); para cada caso, o código anterior é submetido ao GPT-4o com *prompts* pré-definidos nos modos *zero-shot* e *few-shot*, e o código gerado é comparado ao commit real. Os objetivos específicos são: (i) identificar e aplicar métricas objetivas para mensurar a qualidade estrutural do código antes e depois do reparo de dívida técnica. (ii) avaliar a capacidade do GPT-4o em propor pagamentos de dívida técnica eficazes e próximos dos realizados por humanos a partir de trechos com SATD; e (iii) comparar, de forma quantitativa e qualitativa, pagamentos de dívida técnica automatizados e pagamentos de dívida técnica humanos observados nos commits.

Os resultados obtidos mostram que o GPT-4o não reproduz o caminho de edição seguido pelos desenvolvedores humanos ao pagar SATD: as similaridades de diff permanecem baixas, com valores medianos de F1 LEMOD entre 0,32 e 0,35 e CrystalBLEU-diff entre 0,20 e 0,25. Apesar dessa baixa convergência em nível de edição, as variantes geradas pelo modelo apresentaram redução de complexidade estrutural mais intensa e consistente do que os commits reais, com medianas de ΔCC entre -1 e $-1{,}5$. Assim, o LLM não atua como um replicador do pagamento humano, mas como um gerador de soluções alternativas que, embora estruturalmente mais simples, divergem significativamente do diff aplicado no projeto.

Por fim, este artigo está organizado da seguinte forma: a Seção 2 apresenta a fundamentacao teórica sobre dívida técnica, limitações de LLMs e as metricas ultilizadas; a Seção 3 discute os trabalhos relacionados; a Seção 4 descreve os materiais, o conjunto de dados e os métodos utilizados no estudo; a Seção 5 apresenta os resultados obtidos; a Seção 6 discute os resultados; e a Seção 7 traz as conclusões e direções para trabalhos futuros.

2. Fundamentação Teórica

2.1. Dívida Técnica

O conceito de dívida técnica foi proposto como uma metáfora por Ward Cunningham para descrever os custos futuros associados a decisões de projeto e código apressadas, sendo posteriormente formalizado e difundido por trabalhos como o de Kruchten et al. [Kruchten et al. 2012]. Em termos gerais, a ideia é que escolhas de curto prazo que simplificam o desenvolvimento hoje geram uma dívida a ser pago no futuro, na forma de esforço adicional de manutenção e risco de defeitos.

Martin Fowler (2009) propôs uma categorização da dívida técnica em quatro quadrantes, a partir de dois eixos: intencionalidade (deliberada ou inadvertida) e prudência (prudente ou imprudente). Essa taxonomia ajuda a entender as motivações por trás da introdução da dívida técnica e a planejar estratégias de pagamento e gerenciamento [Fowler 2009].

Um subtipo importante é a *dívida técnica auto-admitida* (SATD, do inglês *Self-Admitted Technical Debt*), em que os próprios desenvolvedores registram a existência de dívida por meio de comentários como *TODO*, *FIXME* e *HACK*. Esses marcadores

oferecem uma oportunidade para estudos empíricos, pois permitem identificar de forma explícita pontos do código em que a equipe reconheceu a necessidade de melhoria. O pagamento dessa dívida pode ser observado em commits que removem tais comentários, servindo como fonte de dados para pesquisas que investigam práticas reais de manutenção.

2.2. Inteligência Artificial para Engenharia de Software

Modelos de linguagem como o GPT-4o são poderosos aliados na automação de tarefas de engenharia de software, mas apresentam limitações importantes. Em primeiro lugar, o modelo funciona como uma *black box*, ou seja, o raciocínio por trás das respostas geradas não é transparente [Guo et al. 2024]. Isso dificulta a validação do pagamendos de dívida técnica gerados.

Além disso, esses modelos podem cometer erros sutis, como sugerir mudanças que alteram o comportamento funcional do código, mesmo que mantenham a sintaxe correta. Guo et al. (2024) destacam que esse tipo de falha pode comprometer seriamente a confiabilidade do pagamento da dívida técnica gerado por LLMs. Outra limitação recorrente é a falta de contexto: como o modelo responde a partir de trechos isolados, ele não compreende o projeto como um todo, o que pode levar à perda de dependências importantes ou à violação de regras arquiteturais [Midolo and Di Penta 2025].

A qualidade dos prompts utilizados também é um fator crítico. Estudos mostram que a estrutura e a clareza do prompt influenciam diretamente a qualidade do código gerado [Shirafuji et al. 2023]. Estratégias como *zero-shot prompting* (sem exemplos) e *few-shot prompting* (com exemplos) impactam diretamente a legibilidade, a eficiência e a adesão às boas práticas no código gerado por modelos de linguagem. Shirafuji et al. (2023) mostram que a inclusão de exemplos melhora significativamente a qualidade estrutural do código refatorado, reduzindo complexidade e aumentando a manutenibilidade [Shirafuji et al. 2023].

Por fim, modelos como o GPT-4o podem gerar código com padrões não idiomáticos, ou seja, que funcionam, mas não seguem o estilo convencional da linguagem, o que prejudica a manutenção a longo prazo [Midolo and Di Penta 2025]. Esses fatores reforçam a importância de avaliar os resultados com métricas objetivas, como complexidade ciclomática, bem como com medidas de similaridade em nível de *diff*, como CrystalBLEU-diff e LEMOD, para comparar pagamento de dívida técnica humano e automático.

Com base nessas limitações, o desenho dos *prompts* adotados neste trabalho segue recomendações recentes sobre o uso de LLMs para manutenção e refatoração de código [Shirafuji et al. 2023, Sheikhaei et al. 2025]. O objetivo é explicitar a tarefa de pagamento de Dívida Técnica Auto-Admitida (SATD), restringir o espaço de respostas para evitar alterações irrelevantes e, ao mesmo tempo, permitir uma comparação justa com o desenvolvedor humano em nível de *diff*.

São usadas duas variantes de *prompting*: *zero-shot* e *few-shot*. Em ambas, o modelo recebe uma mensagem de sistema que o caracteriza como assistente de pagamento de dívida técnica em Python e exige que devolva o trecho completo de código, copiando as linhas não modificadas e produzindo apenas código bruto, sem explicações. Na variante *zero-shot*, a mensagem de usuário combina metadados (comentário de SATD, caminho do arquivo, trecho da mensagem de *commit*) com instruções concisas para pagar a dívida

técnica registrada no comentário a partir do código *before*, devolvendo o trecho inteiro sem texto explicativo. Na variante *few-shot*, esse mesmo esqueleto é precedido por um exemplo curto de pagamento de SATD (um caso *BEFORE/AFTER* simples), que ilustra o padrão esperado de transformação.

2.3. LLMs e Manutenção de Software

Modelos de linguagem de grande porte (LLMs) vêm sendo aplicados em tarefas de manutenção e evolução de software, como migração de bibliotecas, modernização de APIs e ajustes de compatibilidade [Almeida et al. 2024]. Esses estudos mostram que, com *prompts* adequados e validação posterior, os modelos são capazes de propor transformações estruturais úteis em projetos reais, mas também podem introduzir padrões não idiomáticos ou alterações de comportamento [Sheikhaei et al. 2025]. Este trabalho se insere nesse cenário ao investigar especificamente o pagamento automatizado de SATD em código Python, comparando pagamentos humanos e automatizados sob a ótica de métricas estruturais e de similaridade em nível de *diff*.

2.4. Métricas de avaliação de similaridade para avaliar pagamento de divída tecnica

A avaliação do pagamento de dívida técnica, seja humano ou automatizado, demanda métricas complementares que capturem tanto os aspectos estruturais do código quanto a proximidade entre as mudanças propostas e aquelas efetivamente aplicadas em projetos reais.

2.4.1. Métricas estruturais

Neste trabalho, as métricas estruturais são usadas para quantificar o impacto do pagamento de SATD sobre a qualidade interna do código.

A complexidade ciclomática (CC) é aproximada por meio de uma heurística de contagem de condições, baseada em uma adaptação prática do método de McCabe. O método formal estabelece que a CC é equivalente ao número de pontos de decisão (predicados) mais um.

O cálculo heurístico deste trabalho considera a soma de tokens de fluxo de controle (`if`, `while`, `for`, `try`, etc.), acrescida da ponderação de 0,5 para cada operador lógico (`and`, `or`) presente em predicados compostos; em seguida, o resultado é normalizado pela contagem de funções no trecho, gerando a medida avg cc (complexidade média por trecho).

Além da complexidade, considera-se uma métrica de tamanho para dimensionar o volume de código modificado. O número total de linhas (LOC) é a medida mais básica; no entanto, pode ser enganoso ao incluir comentários e elementos não executáveis. Por isso, este trabalho utiliza a métrica Linhas de Código Fonte (SLOC), que mede o volume de código após a remoção de comentários e *docstrings*, fornecendo um indicador mais fiel do código que o desenvolvedor precisa analisar e manter.

A combinação de complexidade média por trecho e **SLOC** permite comparar, de forma objetiva, a complexidade e o tamanho dos trechos modificados, tanto para o desenvolvedor humano quanto para as variantes geradas pela LLM.

2.4.2. Métricas de Similaridade Baseadas em *Diff*

Conforme discutido na Seção 2.4.1, este estudo utiliza duas famílias de métricas para avaliar o pagamento automatizado de SATD: (i) métricas estruturais (como CC e SLOC), que capturam o impacto das modificações na qualidade interna do código resultante; e (ii) métricas de similaridade entre o pagamento automatizado e o pagamento humano. Nesta subseção, o segundo grupo é detalhado, composto por métricas orientadas a *diff*, que são aplicadas diretamente sobre os trechos de código efetivamente modificados e são utilizadas nas análises empíricas deste trabalho para quantificar o quão próximo o reparo gerado pelos LLMs está do reparo observado no histórico de commits.

Medidas de similaridade orientadas a *diff* comparam o pagamento de dívida técnica produzido por um LLM com o pagamento de dívida técnica humano observado no commit. Essa abordagem é fundamental no contexto de Dívida Técnica Auto-Admitida (SATD), pois foca a avaliação estritamente nas linhas de código que foram alteradas, mitigando o ruído de métodos *whole-code* [Sheikhaei et al. 2025] e permitindo uma comparação mais fiel entre "o que o desenvolvedor realmente mudou" e "o que o modelo decidiu mudar".

No presente trabalho, são utilizadas duas métricas baseadas em *diff*: **CrystalBLEU-diff** e **LEMOD F1**. A primeira captura similaridade textual entre as mudanças; a segunda captura acerto em nível de linhas modificadas.

O CrystalBLEU-diff adapta algoritmos de similaridade textual baseados em *n-grams* para focar apenas no texto das diferenças entre o código de entrada e as versões pós-reparo (candidata e referência). Intuitivamente, quanto maior a sobreposição de *n-grams* entre o *diff* humano e o *diff* gerado pelo modelo, mais o pagamento automatizado se aproxima do pagamento observado no projeto.

Diferentemente do BLEU tradicional, *CrystalBLEU-diff* emprega listas de parada para atenuar a influência de *n-grams* pouco informativos (como pontuações e tokens muito frequentes), concentrando a pontuação em termos mais relevantes para descrever a modificação [Sheikhaei et al. 2025]. No contexto deste trabalho, essa métrica fornece uma visão quantitativa do quanto as alterações propostas pelos LLMs se alinham, em nível textual, às alterações efetivamente aplicadas pelos desenvolvedores.

O Line-Level Exact Match on Diff (LEMOD F1) é uma métrica proposta especificamente para o reparo de SATD [Sheikhaei et al. 2025] que avalia a qualidade do reparo em termos de **Precisão**, **Recall** e **F1** em nível de linha. Em vez de olhar apenas para a similaridade textual, o LEMOD foca na sobreposição dos conjuntos de linhas modificadas (*diff*) entre a verdade fundamental (reparo humano) e o código gerado pelo modelo, oferecendo uma medida mais interpretável de acerto sobre as linhas que "deveriam" ter sido alteradas.

As componentes do LEMOD são definidas como:

- $reference\_diff$: conjunto de linhas alteradas na verdade fundamental (reparo humano).
- $candidate\_diff$: conjunto de linhas alteradas no código gerado pelo modelo.

$$\text{LineP} = \frac{\text{count}(\text{intersection}(reference\_diff, candidate\_diff))}{\text{count}(candidate\_diff)} \quad \text{(Precisão em Linha)} \tag{1}$$

$$\text{LineR} = \frac{\text{count}(\text{intersection}(reference\_diff, candidate\_diff))}{\text{count}(reference\_diff)} \quad \text{(Recall em Linha)} \tag{2}$$

$$\text{LineF1} = \frac{2 \cdot \text{LineP} \cdot \text{LineR}}{(\text{LineP} + \text{LineR})} \quad \text{(F1 em Linha)} \tag{3}$$

Neste trabalho, o foco está na métrica **LineF1** (LEMOD F1), que sintetiza o equilíbrio entre a precisão e o recall em nível de linha e é utilizada como indicador principal de acerto estrutural do pagamento automatizado de SATD.

Em conjunto, CrystalBLEU-diff e LEMOD F1 fornecem sinais complementares ao longo das análises empíricas: enquanto as métricas estruturais (CC e SLOC) avaliam o impacto dos pagamentos de dívida técnica na qualidade do código resultante, as métricas de *diff* medem o quão próximo o pagamento de dívida técnica automatizado está daquele praticado pelos desenvolvedores no histórico do projeto, tanto em termos de *conteúdo* das mudanças (CrystalBLEU-diff) quanto em termos de *linhas* efetivamente modificadas (LEMOD F1).

3. Trabalhos Relacionados

Nesta seção, são discutidos trabalhos relacionados ao tema deste estudo, com foco na aplicação de inteligência artificial e modelos de linguagem para pagamento de dívida técnica automático de código em projetos de software, especialmente na linguagem Python. Esses trabalhos tratam de problemas correlatos aos definidos na introdução deste trabalho, e servem de base para comparação e aprofundamento da proposta aqui apresentada.

Uma revisão sistemática conduzida por Pandi et al. (2023) realiza uma revisão sistemática da literatura para investigar como técnicas de inteligência artificial podem ser aplicadas na identificação e gestão da dívida técnica em projetos de software. O estudo analisou quinze artigos relevantes e categorizou as principais abordagens de uso da IA, incluindo análise de código, testes automatizados, refatoração, manutenção preditiva e documentação automática. Os autores destacam que ferramentas como SonarQube e CAST apresentam potencial significativo, mas ainda enfrentam desafios relacionados à necessidade de dados de alta qualidade e às implicações éticas do uso de algoritmos automáticos. Este artigo oferece uma base teórica valiosa sobre as possibilidades e limitações do uso de IA na redução da dívida técnica, servindo como apoio para o presente estudo, que avança ao aplicar experimentalmente um modelo de linguagem natural (GPT-4o) na refatoração de código Python.

Lambert et al. (2024) investigam o uso de *Large Language Models* para identificação de Dívida Técnica Auto-Admitida (SATD) em *issues* de projetos de software, comparando três modelos generalistas (Claude 3 Haiku, GPT-3.5-turbo e Gemini

1.0 Pro) com o classificador baseado em *transformers* proposto por Skryseth et al. O estudo utiliza um conjunto de 8 663 *issues* rotuladas como TD ou Not_TD em quatro repositórios e avalia diferentes estratégias de *prompt engineering* (*zero-shot*, *think step-by-step*, *few-shot* e *chain-of-thought*) por meio de métricas como F1-score, precisão, revogação e MCC, mostrando que Claude 3 alcança desempenho competitivo em relação ao modelo especializado, embora com maior tendência a falsos positivos, e que o uso de *few-shot prompting* melhora significativamente o equilíbrio das predições. Esse trabalho relaciona-se ao presente estudo ao também empregar LLMs e técnicas de *prompting* no contexto de SATD, mas focando na tarefa de identificação em artefatos textuais, enquanto o estudo atual aborda o pagamento automatizado de SATD diretamente no código-fonte e analisa o impacto estrutural das modificações geradas.

Sheikhaei et al. (2025) investigam a eficácia de modelos de linguagem no pagamento automático de Dívida Técnica Auto-Admitida (SATD) e propõem *benchmarks* e métricas específicas para esse problema. Os autores constroem conjuntos de dados de pagamento de SATD para Java e Python a partir de históricos de commits do GitHub e aplicam um *pipeline* de filtragem para remover falsos positivos. O estudo argumenta que métricas tradicionais aplicadas ao código completo (como BLEU e CrystalBLEU) são pouco informativas nesse contexto e introduz variantes baseadas em *diff* (BLEU-diff, CrystalBLEU-diff) e a métrica LEMOD, que mede precisão, revocação e F1 em nível de linhas modificadas. Com esses artefatos, comparam modelos menores ajustados com LLMs de grande porte acessados por *prompting*, mostrando que estes últimos obtêm melhor desempenho nas métricas finas de similaridade de *diff* e qualidade das alterações. Este trabalho relaciona-se diretamente ao presente estudo ao fornecer o arcabouço conceitual e as métricas (BLEU-diff, CrystalBLEU-diff e LEMOD) usadas aqui para avaliar os pagamentos de SATD gerados pelo GPT-4o em código Python.

Outra abordagem relevante foi apresentada por Shirafuji et al. (2023), que propõem o uso do *GPT-3.5* para refatorar automaticamente códigos Python, com foco na redução da complexidade ciclomática. A abordagem utiliza *few-shot prompting*, com exemplos cuidadosamente selecionados para orientar o modelo a gerar versões menos complexas dos programas. A avaliação foi conduzida com 880 programas reais extraídos do Aizu Online Judge. Os resultados mostram que o modelo produziu refatorações corretas em 95,68% dos casos, considerando como base de comparação a versão original dos programas. Em média, as versões geradas apresentaram uma redução de 17,35% na complexidade ciclomática e de 25,84% no número de linhas de código, evidenciando o potencial do modelo em gerar versões estruturalmente mais simples sem comprometer a funcionalidade.

Wehaibi et al. (2016) investigam o impacto da Dívida Técnica Auto-Admitida (SATD) na qualidade de software em cinco projetos open source (Chromium, Hadoop, Spark, Cassandra e Tomcat). Os autores comparam arquivos com e sem SATD e analisam mudanças que tocam esses arquivos, utilizando SZZ para identificar *defect-inducing changes* e métricas de dificuldade de mudança, como *churn*, número de arquivos e diretórios modificados e entropia. Os resultados mostram que não há um padrão claro de maior defeito em arquivos com SATD em relação aos demais, mas que a taxa de correções aumenta após a introdução da SATD e que mudanças em arquivos com SATD são, em média, mais difíceis de realizar, embora induzam menos defeitos futuros do que mudanças sem SATD.

Esse trabalho se aproxima do presente estudo ao analisar empiricamente a relação entre SATD e qualidade, mas foca em histórico de defeitos e dificuldade de manutenção, enquanto esta pesquisa avalia o pagamento automatizado de SATD por LLMs e o impacto estrutural das refatorações geradas no código Python.

4. Materiais e Metodos

O presente estudo caracteriza-se como uma pesquisa aplicada, de natureza experimental e quantitativa. O foco reside na avaliação da eficácia do pagamento de dívida técnica automatizado com modelos de linguagem de grande porte (LLM), como o GPT-4o, em cenários reais de projetos Python de código aberto.

Nesta seção, inicialmente se define o ambiente de pesquisa, explicando o uso de ferramentas e a escolha da plataforma para coleta dos repositórios. Em seguida, são discutidas as etapas de definição e mineração da base de dados com SATD. Posteriormente, a amostragem e a curadoria manual dos casos são detalhadas. A seguir, aborda-se a aplicação do LLM para geração dos pagamentos automatizados de dívida técnica em diferentes configurações de *prompting*. Por fim, apresenta-se o conjunto de métricas utilizado para comparar a qualidade e a similaridade entre as soluções humanas e automatizadas.

4.1. Ambiente de pesquisa

O ambiente de pesquisa adota uma infraestrutura local baseada em Python 3.11, na qual são implementados os scripts responsáveis pela mineração de repositórios, extração de SATD, interação com o modelo GPT-4o e cálculo das métricas de avaliação. A escolha de Python se deve à sua ampla utilização em análise de dados e automação de pipelines experimentais, bem como à disponibilidade de bibliotecas para mineração de repositórios, manipulação de *diffs* e integração com APIs. A coleta de dados a partir do histórico de desenvolvimento ocorre sobre projetos hospedados no GitHub, em função da popularidade da plataforma, do grande número de repositórios disponíveis e da transparência do histórico de commits.

Para reduzir ruídos operacionais, o estudo mantém uma cópia local dos repositórios analisados e registra os *hashes* dos commits utilizados em cada etapa. As chamadas ao GPT-4o são realizadas via API com parâmetros fixos ao longo de todo o experimento, garantindo que as diferenças observadas entre as variantes se devam ao conteúdo dos *prompts* e ao código de entrada, e não a variações da configuração do modelo. Essa padronização do ambiente contribui para a reprodutibilidade dos resultados e para a comparabilidade entre os casos analisados.

4.2. Base de dados e mineração de SATD

A base de dados é composta por repositórios de código aberto com predominância da linguagem Python, selecionados a partir da API GraphQL do GitHub. A seleção prioriza projetos ativos, com volume significativo de código e número de estrelas, de forma a refletir contextos reais de desenvolvimento. Diferentemente de abordagens que filtram previamente arquivos ou módulos específicos, a estratégia adotada segue o princípio de coleta *all-in*: a árvore de commits de cada repositório é percorrida integralmente, sem restrição inicial de mensagens, autores ou datas, de modo a capturar o máximo possível de ocorrências de SATD ao longo da evolução do projeto.

Durante essa mineração, o estudo identifica commits que removem explicitamente marcadores de Dívida Técnica Auto-Admitida em comentários do código, tais como `TODO`, `FIXME` e `HACK`. Sempre que um desses marcadores desaparece entre uma revisão e a seguinte, o commit correspondente é interpretado como um pagamento de dívida técnica realizado pelo desenvolvedor humano. Para cada caso, são extraídos o trecho de código anterior à remoção da SATD e o trecho posterior, incluindo um pequeno contexto de linhas ao redor da mudança, de forma a preservar a estrutura local onde a dívida é inicialmente declarada e, em seguida, paga.

4.3. Amostragem e curadoria dos pares de código

A aplicação da mineração *all-in* resulta em um conjunto inicial de 6.968 casos candidatos a pagamento de SATD. Esse conjunto bruto inclui desde dívidas bem definidas até comentários ambíguos ou alterações cosméticas. Para possibilitar uma análise mais cuidadosa e alinhada ao escopo do trabalho, o estudo adota uma etapa de amostragem seguida de curadoria manual.

Primeiro, é sorteada uma amostra de 365 casos, cada um representando um par de trechos de código relacionados a um mesmo pagamento de SATD (antes e depois da remoção do comentário). Em seguida, cada par é inspecionado manualmente com base em três critérios. O primeiro critério verifica se o comentário removido corresponde de fato a uma SATD, isto é, se aponta explicitamente para uma limitação conhecida, um *hack* provisório ou uma solução que o próprio desenvolvedor reconhece como técnica e temporária. O segundo critério avalia se a alteração realizada no código realmente paga a dívida descrita no comentário, em vez de apenas remover o texto sem modificar o comportamento relevante. O terceiro critério examina se o pagamento se concentra no próprio trecho analisado, sem depender de modificações espalhadas por múltiplos arquivos, o que inviabilizaria uma comparação justa com uma proposta gerada a partir de um único contexto.

Os casos que não atendem a esses critérios são descartados por não representarem pagamentos de SATD claros ou analisáveis no recorte local desejado. Após essa curadoria, permanece um conjunto de 151 pares considerados válidos, que serve de base para a aplicação do LLM e para o cálculo das métricas estruturais e de similaridade nas etapas subsequentes.

4.4. Geração automatizada dos pagamentos de dívida técnica

Com a amostra curada de 151 casos, o modelo GPT-4o é aplicado para gerar pagamentos automatizados de dívida técnica a partir dos trechos *before*, isto é, o código que ainda contém o comentário de SATD e o contexto imediato ao redor dos pares de código.

Para cada par de código são produzidas duas variantes, seguindo as estratégias de prompting descritas na Seção 2.2: *zero-shot* e *few-shot*. Em ambas, o modelo recebe uma mensagem de sistema que o configura como assistente de pagamento de dívida técnica em Python e impõe requisitos estruturais fixos, como devolver o trecho completo no mesmo escopo do código de entrada, ecoar as linhas não modificadas, não usar *markdown* e produzir apenas código Python bruto.

Na configuração *zero-shot*, a mensagem de usuário combina o texto do comentário de SATD, o caminho do arquivo e o código *before*, acompanhados de um pequeno bloco

de requisitos que instruem o modelo a pagar a dívida técnica registrada no comentário, devolvendo o trecho inteiro e sem texto explicativo. Na configuração *few-shot*, o mesmo esqueleto é mantido, mas antecedido por um exemplo curto em formato BEFORE/AFTER, no qual um TODO simples é removido e substituído por uma implementação concreta. Os prompts permanecem fixos ao longo de todo o experimento, de modo que as diferenças observadas nos resultados decorrem apenas do conteúdo dos trechos de entrada e das variantes de prompting (*zero-shot* vs. *few-shot*).

4.5. Métricas de qualidade e similaridade

A avaliação dos pagamentos de dívida técnica produzidos pelo GPT-4o se baseia na comparação sistemática, para cada par de código, entre três variantes de código: o pagamento humano observado no commit, a versão automatizada gerada na configuração *zero-shot* e a versão gerada na configuração *few-shot*. As métricas apresentadas na Seção 2.4 são aplicadas aqui em dois grupos: métricas estruturais e métricas de similaridade em nível de *diff*.

No grupo estrutural, o script de análise aplica, para cada variante, a heurística de complexidade ciclomática descrita na Seção 2.4.1, gerando a medida avg_cc por trecho, bem como as contagens de SLOC. A partir desses valores absolutos, o estudo deriva diferenças intrínsecas entre variantes, como Δavg_cc, ΔSLOC, sempre tomando o pagamento humano como referência. Essas diferenças quantificam se cada pagamento automatizado torna o trecho estruturalmente mais simples, mais complexo ou apenas diferente em relação ao que foi feito no projeto real.

No grupo de métricas baseadas em *diff*, a avaliação se restringe às linhas efetivamente alteradas em relação ao código original com SATD. Para cada par "LLM vs. humano", o script extrai, a partir do mesmo `before`, os *diffs* do pagamento humano e da variante LLM, concatena o texto das linhas modificadas e calcula CrystalBLEU-diff sobre esses textos, aplicando, nesse caso, a *stoplist* de tokens sintáticos para atenuar *n-grams* pouco informativos. Em seguida, os conjuntos de linhas alteradas em cada *diff* são comparados pelo LEMOD, produzindo LineP, LineR e LineF1 em nível de linha. Todas essas métricas são calculadas de forma padronizada para os 151 casos, permitindo comparar, de maneira consistente, o comportamento das variantes *zero-shot* e *few-shot* em relação ao pagamento de dívida técnica observado nos commits.

5. Resultados

Esta seção apresenta os resultados obtidos a partir das métricas de qualidade estrutural e de similaridade de *diff* para os 151 pares de codigo de código analisados, comparando o desempenho do GPT-4o (nas variantes de *prompt zero-shot* e *few-shot*) com o pagamento de dívida técnica realizado pelos desenvolvedores humanos. Inicialmente, é feita uma caracterização geral do conjunto de dados utilizado; em seguida, são discutidos os resultados de similaridade entre os *diffs* do LLM e dos humanos e, por fim, o impacto estrutural dos pagamentos de dívida técnica na complexidade ciclomática média.

5.1. Caracterização do conjunto de dados

O conjunto de dados é composto por 151 pares de código de pagamento de Dívida Técnica Auto-Admitida (SATD) extraídos de repositórios públicos em Python no GitHub, conforme descrito na Seção 4.2. Para cada caso, são consideradas quatro variantes de código:

(i) a versão original antes do pagamento (*before*); (ii) o pagamento humano observado no commit (*ground truth*); (iii) a variante gerada pelo GPT-4o em configuração *zero-shot*; e (iv) a variante gerada em configuração *few-shot*. Todas as métricas estruturais e baseadas em *diff* são calculadas a partir dessas variantes.

5.2. Similaridade e alinhamento com o pagamento de dívidas técnicas humano

Para avaliar em que medida o processo de pagamento de SATD automatizado reproduz o caminho de edição observado nos commits, foram calculadas as métricas F1 LEMOD e CrystalBLEU-diff entre as variantes geradas pelo LLM e a solução humana, bem como a variação no número de linhas de código modificadas (ΔSLOC).

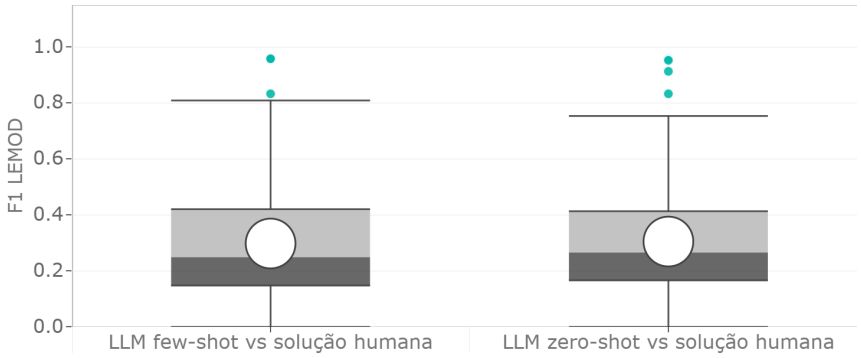

Figura 1. Distribuicao do F1 LEMOD para as variantes LLM *few-shot* e *zero-shot* em relacao ao pagamento humano.

Para quantificar o alinhamento entre o caminho de edição do LLM e o pagamento de SATD humano, foi calculada a métrica F1 LEMOD em nível de linha. A Figura 1 apresenta a distribuição desses valores para as duas variantes de prompting. Para a configuração *few-shot*, a mediana do F1 LEMOD é de aproximadamente $0{,}32$, enquanto o primeiro quartil se situa em torno de $0{,}16$ e o terceiro quartil em torno de $0{,}42$. Na configuração *zero-shot*, observa-se uma mediana ligeiramente maior, em torno de $0{,}35$, com primeiro quartil próximo de $0{,}18$ e terceiro quartil também em torno de $0{,}42$. Em ambas as variantes, portanto, pelo menos 75% dos pagamentos de dívida técnica automatizados permanecem com F1 abaixo de $0{,}5$, indicando que o conjunto de linhas modificadas pelo modelo converge apenas parcialmente para o conjunto de linhas alteradas pelo desenvolvedor humano.

A respeito da similaridade do CrystalBLEU-diff, a Figura 2 mostra a distribuição desses valores para as duas variantes de prompting. Na configuração *few-shot*, a mediana do CrystalBLEU-diff é de aproximadamente $0{,}20$, com primeiro quartil em torno de $0{,}08$ e terceiro quartil próximo de $0{,}32$. Já na configuração *zero-shot*, a mediana situa-se em torno de $0{,}25$, com o primeiro quartil aproximado em $0{,}11$ e o terceiro quartil em torno de $0{,}36$. Em ambas as configurações, pelo menos 75% dos casos permanecem abaixo de $0{,}40$.

5.3. Impacto na qualidade estrutural do código

O impacto estrutural dos pagamentos de SATD foi avaliado por meio da variação da complexidade ciclomática média (ΔCC) e da diferença no número de linhas de código

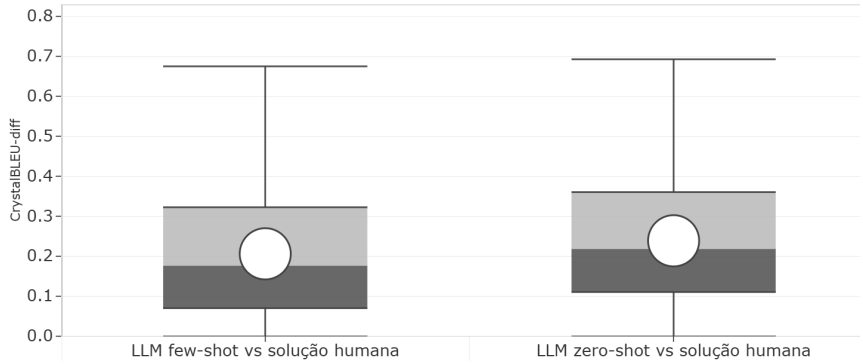

Figura 2. Distribuição da similaridade CrystalBLEU-diff entre as variantes LLM e o pagamento humano.

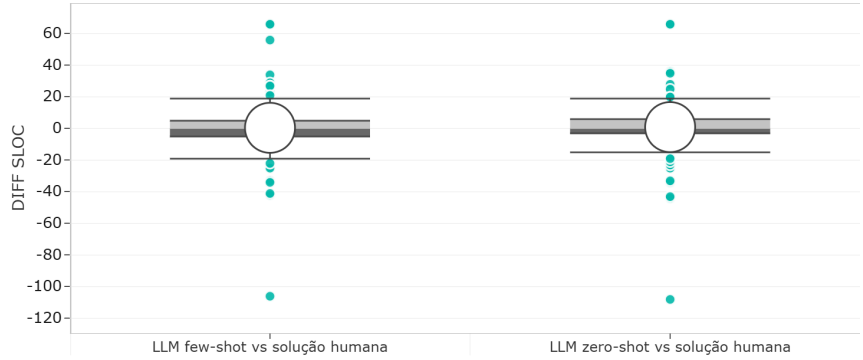

Figura 3. Distribuição da similaridade SLOC entre as variantes LLM e o pagamento humano.

(ΔSLOC) em relação ao código original e da comparação direta entre as variantes humanas e automatizadas.

Na Figura 3, observa-se a distribuição da diferença de linhas de código ($\Delta\mathrm{SLOC}$) entre as soluções geradas pelo LLM e o pagamento humano de dívida técnica. Na variante *few-shot*, a mediana de $\Delta\mathrm{SLOC}$ é igual a 0, enquanto o primeiro quartil se situa em aproximadamente -3 e o terceiro quartil em torno de 4, de forma que metade dos casos permanecem dentro desse intervalo simétrico em torno de zero. Na variante *zero-shot*, a mediana é aproximadamente 0, com primeiro quartil em cerca de -2 e terceiro quartil próximo de 4.

Na Figura 4, apresenta-se a distribuição da variação da Complexidade Ciclomática Média ($\Delta\mathrm{CC}$) em relação ao código original para as três variantes consideradas. No pagamento humano, a mediana de $\Delta\mathrm{CC}$ fica proxima de $-0,5$, com primeiro quartil em torno de -2 e terceiro quartil próximo de 0, indicando que metade dos casos se concentra nesse intervalo. Na variante *few-shot*, a mediana situa-se em cerca de $-1,5$, com primeiro quartil em aproximadamente -2 e terceiro quartil em torno de 0. Já na variante *zero-shot*, a mediana é próxima de -1, enquanto o primeiro quartil se encontra em torno de -2 e o terceiro quartil em 0. Em todas as variantes, observam-se ainda valores atípicos fora desses intervalos interquartis, indicando casos em que a variação de complexidade se afasta de forma mais acentuada do comportamento central.

Na Figura 5, apresenta-se a distribuição da variação da Complexidade Ciclomática

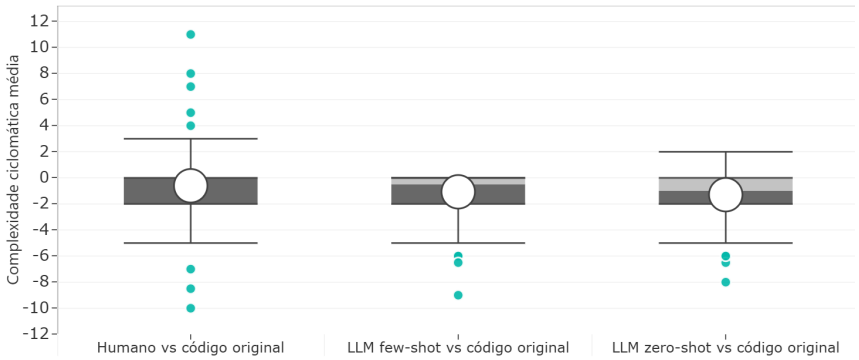

Figura 4. Variação de complexidade ciclomática média (ΔCC) em relação ao código original para as três variantes de pagamento.

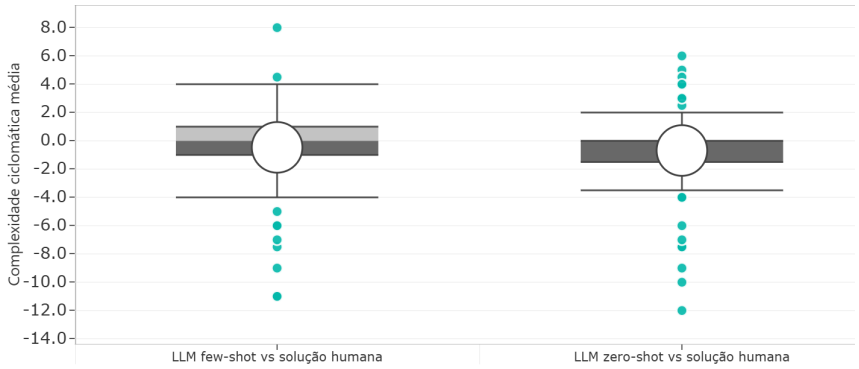

Figura 5. Comparação direta da distribuição de ΔCC entre as variantes do LLM e a solução humana.

Média (ΔCC) entre as variantes geradas pelo LLM e o pagamento humano. Na configuração *few-shot*, a mediana de ΔCC situa-se em aproximadamente $-0{,}5$, com o primeiro quartil em torno de $-1{,}0$ e o terceiro quartil próximo de $1{,}0$, concentrando metade dos casos nesse intervalo. Na configuração *zero-shot*, a mediana encontra-se em cerca de $-0,5$, enquanto o primeiro quartil está em torno de $-1,7$ e o terceiro quartil em aproximadamente 0.

6. Discussão dos Resultados

Esta seção discute como os resultados obtidos se relacionam com os objetivos do estudo. De um lado, as métricas estruturais (avg_cc e SLOC) permitem observar o efeito das edições na complexidade e no tamanho do código; de outro, as métricas de similaridade em nível de *diff* (LEMOD F1 e CrystalBLEU-diff) indicam o grau de alinhamento entre os pagamentos humanos e automatizados, ainda que não capturem aspectos semânticos mais finos.

No eixo da similaridade de *diff*, tanto o F1 LEMOD quanto o CrystalBLEU-diff indicam que o GPT-4o não reproduz o caminho de edição observado nos commits humanos. As medianas baixas em ambas as métricas, mesmo quando o ΔSLOC é próximo de zero, mostram que o modelo altera linhas diferentes e produz conteúdo lexical distinto, sugerindo que a divergência está na escolha das linhas e no tipo de modificação, e não na quantidade de mudanças aplicadas. No aspecto estrutural, o GPT-4o demonstra um

padrão consistente de simplificação mais intensa que a solução humana, com reduções de complexidade ciclomática mais agressivas. Enquanto o pagamento humano tende a pequenas reduções, o modelo frequentemente entrega versões estruturalmente mais simples, reforçando um papel de "otimizador estrutural" do código.

Do ponto de vista qualitativo, a combinação desses padrões sugere que o GPT-4o não atua como um reprodutor do pagamento de SATD realizado em projetos reais, mas como um gerador de soluções alternativas: pouco alinhadas em nível de *diff*, porém com efeito estrutural mais forte. Na prática, isso o torna promissor como apoio para propor pagamentos de dívida técnica e melhorias de qualidade, desde que suas edições passem por validação e revisão criteriosas antes de serem integradas ao código.

7. Conclusão

O objetivo deste trabalho foi avaliar se o *Large Language Model* (LLM) GPT-4o é capaz de pagar Dívida Técnica Auto-Admitida (SATD) de forma comparável aos desenvolvedores humanos em projetos *Python*. Para isso, foram analisados 151 pares de código de código extraídos de repositórios populares do GitHub, gerando, para cada caso, uma solução humana e duas variantes automatizadas (*zero-shot* e *few-shot*), avaliadas por meio de métricas estruturais e de similaridade de *diff*. Os resultados indicam que o GPT-4o não reproduz o caminho de edição seguido pelos desenvolvedores, apresentando baixa similaridade em nível de diff, mas produz codigos que, em geral, simplificam mais a estrutura do que os pagamentos humanos. Assim, o modelo se comporta menos como um replicador do commit original e mais como um gerador de soluções alternativas com foco em melhoria estrutural, o que o torna promissor para apoiar a manutenção, mas com risco adicional de realizar alterações além das que são espesificadas no comentario *SATD*.

Como desdobramentos futuros, este estudo aponta a necessidade de incorporar verificação semântica e testes automatizados sobre os pares de código avaliados, de modo a verificar se a simplificação estrutural promovida pelo LLM preserva o comportamento funcional do código. Também se mostra relevante investigar variações de engenharia de prompt que possa afetar a forma como a LLM realiza o pagamento, explorar a replicação do experimento em outras linguagens e domínios para ampliar a validade externa dos achados e, por fim, conduzir estudos empíricos com desenvolvedores para avaliar o custo-benefício prático de revisar diffs de baixa similaridade, porém com ganho estrutural, em cenários reais de manutenção de software.

Pacote de replicação

O pacote de replicação deste trabalho encontra-se disponível em:

`https://github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2025-1-tcci-0393100-pes-bernardo-pires`

Referências

Almeida, A., Xavier, L., and Valente, M. T. (2024). Automatic library migration using large language models: First results. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24)*, pages 1–7. ACM.

Fowler, M. (2009). Technical debt quadrant. https://martinfowler.com/bliki/TechnicalDebtQuadrant.html. Acessado em 25 jun. 2025.

Guo, Q., Cao, J., Xie, X., Liu, S., Li, X., Chen, B., and Peng, X. (2024). Exploring the potential of chatgpt in automated code refinement: An empirical study. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE '24)*.

Kruchten, P., Nord, R. L., and Ozkaya, I. (2012). Technical debt: from metaphor to theory and practice. *IEEE Software*, 29(6):18–21.

Lambert, P., Ishitani, L., and Xavier, L. (2024). On the identification of self-admitted technical debt with large language models. In *Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software (SBES 2024)*, pages 651–657. Sociedade Brasileira de Computação.

Li, Z., Avgeriou, P., and Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101:193–220.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320.

Midolo, G. and Di Penta, M. (2025). Automated refactoring of non-idiomatic python code: A differentiated replication with llms. *arXiv preprint arXiv:2501.17024*.

Pandi, S. B., Binta, S. A., and Kaushal, S. (2023). Artificial intelligence for technical debt management in software development. *arXiv preprint arXiv:2306.10194*.

Sheikhaei, M. S., Tian, Y., Wang, S., and Xu, B. (2025). Understanding the effectiveness of llms in automated self-admitted technical debt repayment. *arXiv preprint arXiv:2501.09888*.

Shirafuji, T., Hido, S., Iio, R., and Matsumoto, K.-i. (2023). Refactoring programs using large language models with few-shot examples. *arXiv preprint arXiv:2311.11690*.

Skryseth, D., Shivashankar, K., Pilán, I., and Martini, A. (2023). Technical debt classification in issue trackers using natural language processing based on transformers. In *Proceedings of the 2023 ACM/IEEE International Conference on Technical Debt (TechDebt)*, pages 92–101. IEEE.

Wehaibi, S., Shihab, E., and Guerrouj, L. (2016). Examining the impact of self-admitted technical debt on software quality. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 179–188. IEEE.

Zhang, Z., Xing, Z., Xia, X., Xu, X., and Zhu, L. (2022). Making python code idiomatic by automatic refactoring: Non-idiomatic python code with pythonic idioms. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22)*, pages 696–708. ACM.