UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Antonio Marcos Barbosa de Sá

Uso de Large Language Models (LLMs) para Auxílio na Correção de *Bugs*

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Antonio Marcos Barbosa de Sá

Uso de Large Language Models (LLMs) para Auxílio na Correção de *Bugs*

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Adriano Mendonça Rocha

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2024

Antonio Marcos Barbosa de Sá

Uso de Large Language Models (LLMs) para Auxílio na Correção de *Bugs*

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção do título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 19 de novembro de 2024:

Prof. Dr. Adriano Mendonça Rocha
Orientador

Prof. Dr. Carlos Cesar Mansur Tuma

Profa. Dra. Fabíola Souza Fernandes Pereira

Uberlândia, Brasil
2024

Resumo

Bugs são um problema frequente enfrentado por desenvolvedores, tornando a identificação e compreensão deles uma tarefa árdua, especialmente para aqueles que estão aprendendo uma nova tecnologia de programação. A complexidade de localizar e corrigir *bugs* aumenta conforme a linguagem, e o nível de experiência do desenvolvedor. Nesse cenário, os LLMs (Modelos de Linguagem de Grande Escala) surgem como uma nova tecnologia, que pode ser aplicadas em diversas tarefas do desenvolvimento de *software*, incluindo a detecção e explicação de *bugs*. O presente estudo visa avaliar e analisar a capacidade dos LLMs, especialmente o Gemini e o GPT (*Generative Pre-trained Transform*) na identificação e explicação de *bugs* de programação. Além de desenvolver e avaliar uma ferramenta para auxiliar desenvolvedores na identificação e correção de *bugs* em seus códigos-fonte. Além disso, esse estudo tem como objetivo realizar uma análise crítica da qualidade das explicações geradas pelos LLMs. Para realizar essa análise, foram utilizados 40 exemplos de códigos contendo *bugs*, obtidos do StackOverflow, sendo 10 exemplos para cada uma das seguintes linguagens: Java, Python, C e Kotlin. Esses códigos foram processados pelos LLMs, e avaliou-se se os bugs foram identificados, e se as explicações fornecidas sobre os *bugs* foram satisfatórias. No processo de avaliação das explicações, foram considerados critérios como clareza, e coerência lógica no esclarecimento das causas dos *bugs*. Além disso, foi desenvolvida uma ferramenta *web* em Flask, que visa identificar e explicar os *bugs* presentes no código fonte fornecido pelo usuário, que a partir da intenção do usuário e o código fornecido, a correção é feita por meio de uma chamada à API do Gemini. Essa tecnologia foi escolhida por ser gratuita e também pela eficiência da mesma, conforme os resultados deste estudo. Também foi elaborado um questionário no qual foi aplicado a usuários que utilizaram a ferramenta para avaliarem a eficiência da mesma. Com isso, chegaram-se aos seguintes resultados: todos os LLMs obtiveram 100% de acertos na identificação dos *bugs*. No entanto, quanto à satisfação com as explicações dos *bugs*, observou-se que o GPT foi satisfatório em todos os problemas nas linguagens Java, C e Kotlin; já em Python, 8 dos 10 exemplos de códigos apresentaram explicações satisfatórias. Com o Gemini, as explicações foram satisfatórias em 8 exemplos de códigos nas linguagens Java e Python, enquanto em Kotlin e C, todas as explicações foram satisfatórias. Com relação ao questionário aplicado, a ferramenta foi bem avaliada pelos usuários.

Palavras-chave: LLMs, *bugs*, identificação de *bugs*, desenvolvimento de *software*, código fonte.

Lista de ilustrações

Lista de abreviaturas e siglas

LLM	*Large Language Model*
GPT	*Generative Pre-trained Transformer*
IA	Inteligência Artificial
PLN	Processamento de Linguagem Natural

Sumário

1 Introdução

Durante o processo de desenvolvimento de *software* a ocorrência de *bugs* de programação é inevitável (ALBRECHT; GRABOWSKI, 2020). Essa ocorrência é por diversas razões, incluindo erros de lógica no código, falta de familiaridade com a linguagem de programação, imperfeição de compreensão dos requisitos com a questão ou projeto, adversidades de sintaxe e entre outros. A partir do momento que um desenvolvedor começa a atuar em uma linguagem de programação nova, os *bugs* tendem a ser mais frequentes, pois o mesmo ainda não está totalmente familiarizado com a sintaxe e as características da nova linguagem. Quando acontece esses *bugs*, frequentemente o desenvolvedor fica frustrado, pois pode consumir um tempo significativo por parte do mesmo ao tentar identificar e compreender o ocorrido. No entanto, a tecnologia LLM (*Large Language Model*) oferece uma solução promissora para auxiliar os desenvolvedores na identificação e correção de *bugs* de programação.

A respeito da relevância deste tema para a sociedade, pode considerar que é considerável, pois, a medida que a tecnologia desempenha um papel cada vez mais central em nossas vidas cotidianas e em setores críticos, como a saúde, finanças, segurança , transporte e similares. Alguns fatos e estatísticas ilustram a importância desse tópico, segundo Jackson (2022) a falta de colaboração entre desenvolvedores de segurança prejudica o sucesso do programa, que acarreta em vulnerabilidades de Segurança ao *software*. A exploração de *bugs* de *software* e o desenvolvimento tecnológico é uma das principais vias de ataques cibernéticos (PUPO, Vinicius Milani Del, 2023). À medida que os programas de *software* se tornam mais complexos e interconectados, a tarefa de identificar e corrigir *bugs* se torna ainda mais desafiadora. A inteligência artificial aliada a tecnologia LLM como o GPT, pode ajudar a lidar com essa crescente complexidade. (DANTAS; ROCHA; MAIA, 2023).

1.1 Objetivos

Nesta seção serão apresentados o objetivo geral e os objetivos específicos da pesquisa.

1.1.1 Objetivo Geral

O principal objetivo desta pesquisa é investigar e demonstrar como os LLMs (*Large Language Models*) podem ser utilizados para auxiliar desenvolvedores de *software* na identificação e compreensão de *bugs* de programação.

1.1.2 Objetivos Específicos

Os objetivos específicos desta pesquisa são os seguintes:

- Avaliar o potencial dos LLMs Gemini e GPT para identificar e explicar ocorrências de *bugs* em códigos de programação.
- Desenvolver uma página *web* destinada a auxiliar desenvolvedores na identificação e compreensão de *bugs* em códigos. Essa ferramenta será um importante suporte para usuários que enfrentarem dificuldades durante o desenvolvimento de software em diferentes linguagens de programação.
- Disponibilizar todos os artefatos resultantes da pesquisa, permitindo que outros pesquisadores possam contribuir e expandir os estudos nesta área.

1.2 Justificativa

Logo, o tema busca analisar desafios e limitações associados ao uso da tecnologia LLM nesse contexto, bem como fornecer orientações práticas para desenvolvedores que desejam adotar essa abordagem. Assim, espera-se que este estudo contribua para o avanço da área de correção de *bugs* e inspire novas pesquisas e aplicações no campo da Inteligência Artificial aplicada à area de detecção e correção de *bugs*.

1.3 Organização da Monografia

Nos próximos capítulos a organização desta monografia está feita da seguinte maneira: O Capítulo 2 apresenta os conceitos fundamentais abordados neste trabalho, como LLMs, *bugs* e linguagens de programação. O Capítulo 3 aborda os trabalhos relacionados com o tema proposto. No Capítulo 4, é mostrado o desenvolvimento, a metodologia e a avaliação da ferramenta proposta. Por fim, o Capítulo 5 apresenta a conclusão da monografia, ressaltando as principais contribuições propostas e sugerindo direções para pesquisas futuras.

2 Fundamentação Teórica

Neste capítulo serão abordados temas importantes como o processamento de linguagem natural, apoiado ao avanço dos LLMs, como o modelo GPT baseado na tecnologia *Transformer*, e também o modelo Gemini. Além disso serão argumentados conceitos fundamentais de linguagens de programação, com ênfase na compreensão de *bugs*, que podem ocorrer durante o desenvolvimento de *software*. Tais temas proporcionam o embasamento necessário para o entendimento de como essas tecnologias contribuem para a detecção e correção de *bugs* no código.

2.1 Processamento Linguagem natural

O Processamento de Linguagem Natural é um campo interdisciplinar que combina conhecimentos de linguística, ciência da computação e inteligência artificial para permitir que os computadores entendam, interpretem e gerem linguagem humana de forma semelhante aos seres humanos. O PLN é a base do funcionamento dos LLMs e de muitas outras aplicações baseadas em texto. Os avanços no processamento de linguagem natural resultaram em grandes modelos de linguagem (LLMs) que podem gerar código e explicações de código (MACNEIL et al., 2023).

2.2 Arquitetura *Transformer* e GPT

A arquitetura *Transformer* é uma rede neural desenvolvida para processar dados sequenciais, como texto ou séries temporais. Ela foi proposta inicialmente em um artigo intitulado "*Attention is All You Need*" (VASWANI et al., 2017). A principal inovação dos *Transformers* é o mecanismo de atenção, que permite que a rede se concentre em diferentes partes da entrada durante o processo de aprendizado.

A arquitetura *Transformer* é composta por blocos de atenção e camadas de feed-forward. Cada bloco de atenção permite que o modelo se concentre em diferentes partes da sequência de entrada, capturando relações de longo alcance de forma mais eficaz do que modelos sequenciais anteriores, como redes recorrentes. A Figura 1 mostra uma ilustração da arquitetura *Transformer*.

O GPT (*Generative Pre-trained Transformer*) é um LLM e uma aplicação específica da arquitetura *Transformer* em tarefas de linguagem natural. Ele utiliza uma arquitetura de codificador-decodificador, mas na maioria das aplicações de GPT, apenas o codificador é usado para tarefas como modelagem de linguagem, tradução automática

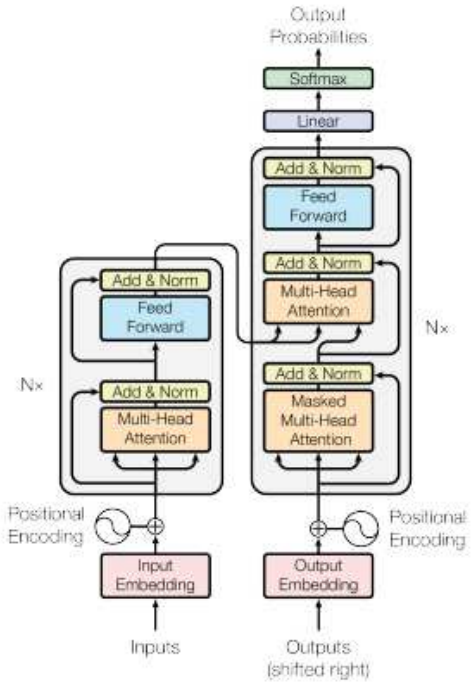

Figura 1 – Demonstração da arquitetura *Transformer*. Fonte: (VASWANI et al., 2017).

e geração de texto.

Assim, a principal semelhança entre a tecnologia *Transformer* e o GPT está na arquitetura de atenção. Ambos utilizam o mecanismo de atenção para processar sequências de dados de maneira eficaz, permitindo que o modelo capture relações contextuais em diferentes partes da entrada. O GPT é uma implementação específica do *Transformer* adaptada para tarefas de linguagem natural, e tornou-se notável por seu sucesso em modelos de linguagem pré-treinados em larga escala.

O desenvolvimento dos modelos GPT começou com o lançamento do GPT-1 pela OpenAI, seguido por iterações sucessivas, como o GPT-2 e o GPT-3. Cada nova versão trouxe melhorias significativas na geração de texto e compreensão da linguagem natural. O lançamento do GPT-4 enfatizou amplamente melhorias notáveis nas capacidades relacionadas ao tratamento de avaliações originalmente projetadas para participantes humanos (SAVELKA et al., 2023). Como dito anteriormente os modelos GPT recentes demonstraram habilidades impressionantes na geração de texto coerente, tradução automática, resumo de texto e até mesmo na resolução de problemas de programação. Exemplos de aplicação do GPT incluem assistentes virtuais, chatbots, resumos automáticos, tradução de idiomas, entre outros.

2.3 Gemini

O Gemini, anteriormente conhecido como Bard, é um marco importante na evolução dos LLMs, fruto de um enorme investimento em pesquisa em inteligência artificial feito pela Google AI. Embora a Google mantenha alguns detalhes técnicos em sigilo, sabe-se que o Gemini foi desenvolvido sobre uma rede neural robusta e treinado com uma vasta quantidade de dados variados. Essa base permite que ele realize uma série de tarefas, como redigir textos, traduzir idiomas, responder perguntas com precisão e até resolver problemas complexos, como identificar e corrigir *bugs* de código — tudo isso enquanto continua a aprender e se adaptar, ficando mais eficiente a cada uso.

Uma característica notável do Gemini é sua capacidade multimodal, o que significa que ele consegue trabalhar com diferentes tipos de dados, como texto, imagens e código, tornando-o ainda mais versátil.

Essa tecnologia tem um potencial imenso para transformar áreas como educação, saúde e atendimento ao cliente. Com o tempo, espera-se que modelos como o Gemini se tornem ainda mais sofisticados, prontos para realizar tarefas que hoje parecem quase impossíveis.

2.4 Linguagem de Programação

Linguagem de programação é um conjunto de regras e símbolos usados para escrever programas que instruem um computador sobre como executar determinadas operações. A escolha da linguagem de programação é crucial para o desenvolvimento de *software*, influenciando a eficiência, legibilidade, desempenho e manutenção do código, o impacto da linguagem de programação também é importante para enriquecer a programação e as habilidades técnicas (AMIN et al., 2022).

Desde as linguagens de baixo nível como Assembly até as modernas linguagens de alto nível como Python (LINDSTROM, 2005) e JavaScript, a evolução das linguagens de programação tem facilitado o desenvolvimento de *software*.

O surgimento de linguagens de programação mais seguras, eficientes e versáteis, bem como frameworks que simplificam tarefas de programação, têm contribuído para a melhoria no processo de desenvolvimento de *software* e fornece novas ferramentas para automatizar tarefas rotineiras de programação e permitir que ações programáveis sejam invocadas com base em determinados eventos (KURNIAWAN; ABRAMSON, 2008).

2.5 Bugs

Bugs são problemas no código de *software* que podem causar mau funcionamento ou falhas no programa. Eles podem variar em complexidade, desde *bugs* simples de sintaxe até problemas de lógica mais complexos. No trabalho *Security versus performance bugs: a case study on Firefox* (ZAMAN; ADAMS; HASSAN, 2011) mostra como compreendemos diferentes tipos de *bugs* que afetam o programa e causa essas falhas.

A ocorrência de *bugs* é uma parte particular na programação desde o início da computação. Com o aumento da complexidade dos sistemas de *software*, a importância de identificar e corrigir *bugs* se tornou mais relevante. Com isso deve-se entender e corrigir essas falhas de *software*, explica o trabalho *Development and Application Security Collaboration* (JACKSON, 2022)

A automação e o uso de técnicas de IA, como o GPT, têm sido explorados para acelerar o processo de identificação e correção de *bugs* de programação. (DANTAS; ROCHA; MAIA, 2023). Ferramentas e abordagens recentes para a detecção e correção de *bugs* incluem *linters*, análise estática, testes automatizados e o uso de IA para sugestões de correção.

A abordagem proposta nesse trabalho, tem como objetivo integrar as tecnologias de LLMs em um sistema web que utiliza Flask, com intuito de auxiliar desenvolvedores no processo de identificação e correção de problemas de código.

3 Trabalhos Relacionados

Referente a utilização de LLMs para correção de *bugs* no código, foi relatado por (CIPRIANO; ALVES, 2023) a experiência na utilização do GPT-3 para resolver 6 tarefas do mundo real utilizadas em um curso de Programação Orientada a Objetos, as observações, baseadas em uma avaliação objetiva do código, realizada por uma Ferramenta de Avaliação Automática de código aberto, mostram que o GPT-3 é capaz de interpretar e lidar com requisitos funcionais diretos, porém tende a não dar a melhor solução em termos de objeto. O resultado é uma análise qualitativa da produção do GPT-3 e reunindo um conjunto de recomendações para educadores de ciências da computação, uma vez que espera-se que os alunos usem e abusem desta ferramenta em seus trabalhos acadêmicos. Nota-se que esse artigo se relaciona diretamente com o tema proposto, que é o uso da tecnologia GPT para o auxílio de correções de *bugs*. Nesta proposta, será realizada uma análise em códigos de programação, com o objetivo de identificar possíveis *bugs* com o auxílio da tecnologia GPT. Espera-se que o GPT possa auxiliar o desenvolvedor na tarefa de correção dos *bugs* encontrados no código.

A tecnologia GPT é favorável em diversas área da vida cotidiana, o autor (BOU et al., 2023). Neste artigo ele destaca as limitações existentes na detecção e alerta de discurso de ódio *online* e introduz uma abordagem inovadora chamada BOU-Guard (*Behavior Observation Unit - Guard*), que utiliza a tecnologia GPT-3.5-Turbo para detectar e filtrar conteúdos preconceituosos ou ofensivos. Aliado ao tema, essa tecnologia do GPT tem grande importância também, pois poderá ser usada no tema proposta, na detecção de possíveis *bugs* de código que podem surgir no desenvolvimento de *software*.

No artigo *How do fixes become bugs?* (YIN et al., 2011), é destacado como a correção de *bugs* pode introduzir novos problemas, resultando em *patches* defeituosos que afetam usuários e fornecedores. O estudo analisa correções incorretas em sistemas como *Linux*, *OpenSolaris*, e *FreeBSD*, mostrando que uma parcela significativa dessas correções é falha, especialmente em *bugs* de simultaneidade. A falta de conhecimento dos desenvolvedores sobre o código envolvido é apontada como uma das principais causas. Para mitigar esses problemas, o artigo propõe melhorias no processo de revisão e correção de *bugs*, enfatizando a importância de práticas mais eficazes no desenvolvimento de *software*.

Os *bugs* de programação são comuns no desenvolvimento de *software* (ALBRECHT; GRABOWSKI, 2020) e podem ocorrer por diversas razões, como problemas de lógica, sintaxe, ou falta de familiaridade com a linguagem. Esses *bugs*, embora frustrantes, podem ser mitigados com o uso de tecnologias como os LLMs, que oferecem suporte na identificação e correção de *bugs* de código. Segundo (ALASMARI; SINGER; ADA,

2024), sistemas de codificação *online* carecem de recursos para ajudar programadores iniciantes, e o ChatGPT, com precisão de 93,3% na identificação de *bugs* (FWA, 2024), tem sido amplamente utilizado para aprendizado.

Estudantes enfrentam barreiras no início da programação, muitas vezes negligenciando práticas de desenvolvimento orientadas a testes (OLIVEIRA et al., 2024). Competências de programação são essenciais no ensino superior (FIGUEIREDO; nALVO, 2021), e atividades como a competição *Bug Battles* (ALHUMUD et al., 2023) tem mostrado que 93,26% dos participantes melhoram suas habilidades em testes de *software*.

Embora os LLMs apresentem inovações como geração de código e detecção de *bugs* (ALSHAHWAN et al., 2024), também podem gerar códigos incompletos sem o contexto adequado (BLINN et al., 2024). Comparações entre modelos, como Gemini e GPT, mostram bons resultados na documentação de código (DVIVEDI et al., 2024) e na detecção de vulnerabilidades em APIs (YLDRM; AYDN; CETIN, 2024). Estudos destacam que LLMs melhoram mensagens de *bugs*, tornando-as mais claras e úteis para iniciantes (LEINONEN et al., 2023), além de apresentar desempenho avançado em geração e explicação de código (MACNEIL et al., 2024), evidenciando sua relevância no contexto desta pesquisa.

4 Desenvolvimento

Neste capítulo, será avaliado o desempenho dos LLMs na identificação e explicação de *bugs* de programação enfrentados por desenvolvedores. Além disso, serão detalhadas as etapas de desenvolvimento da ferramenta criada para essa finalidade.

Por fim, será apresentado o formulário aplicado aos participantes do estudo, bem como os resultados obtidos a partir do uso da ferramenta proposta.

4.1 Métodos para Avaliação e Resultados

No que se refere à abordagem proposta neste estudo, destaca-se que ela se baseia em duas frentes de análise distintas. Inicialmente, para embasar de forma teórica o tema em questão, empregou-se duas abordagens, a primeira é a avaliação dos LLMs para verificar suas capacidades de identificar e explicar *bugs* dos códigos de programação, que são coletados com *bugs* de usuários reais do *Stack Overflow*. A segunda será das informações coletadas de uma forma de pesquisa quali-quantitativa, com a aplicação por meio de um formulário com os usuários sobre o uso da ferramenta proposta, com questões abertas e de múltipla escolha, visto que muitos desenvolvedores enfrentam problemas com *bugs* de código ao atuarem com a programação tanto no âmbito universitário quanto em projetos pessoais ou profissonais. O formulário aplicado em questão encontra-se, ao final das referências, na parte do Apêndice A.

4.2 Avaliação dos LLMs

Com relação as avaliações dos LLMs para a detecção e a explicação de *bugs* de programação, foi empregado uma análise do quanto os LLMs pode beneficiar para essa adversidade, como o aumento da criatividade, tomada de decisões mais rápida e inteligente.

Por esse lado, foi feita um balanço para testar o desempenho de dois LLMs existentes, que são o GPT da *OpenAI* e o Gemini do *Google*, tais testes foram feitos da seguinte forma: primeiro foi feita a escolha das linguagens de programação, que foram Java, Linguagem C, Python e Kotlin, tais linguagens de programação foram escolhidas por serem populares e também por causa que programadores costumam ter o primeiro contato com essas linguagens na universidade. Após a seleção das linguagens de programação, foram coletados 10 exemplos de códigos com *bugs* de usuários reais no *Stack Overflow*. Essa escolha foi feita de forma aleatória, selecionando os primeiros casos encontrados para

cada linguagem. Em seguida, cada exemplo foi avaliado tanto no GPT quanto no Gemini usando o seguinte *prompt* no LLM: *"Analise o seguinte código em X: Y. Identifique o bug* e me explique a sua causa." Onde X foi substituído por Java, C, Python ou Kotlin, e Y foi substituído pelo código obtido no *Stack Overflow*.

Após a coleta dos dados, foram selecionados 40 exemplos de código com *bugs* de programação (10 de cada linguagem). Em seguida, um avaliador humano analisou a capacidade dos LLMs GPT e Gemini de identificar e explicar os *bugs* presentes nesses exemplos.

O gráfico presente na Figura 2, mostra os resultados das análises dos 10 exemplos de códigos de cada linguagem, Java, C, Python e Kotlin, o mesmo mostra o desempenho tanto do GPT quanto do Gemini se ambos conseguiram identificar o *bug* de todos os codigos testados nos LLMs. É possível notar que obteve-se 100% de aproveitamento para todas as linguagens, ou seja, o GPT e o Gemini conseguiram identificar os *bugs* presentes em todos os exemplos de código de todas as linguagens de programação.

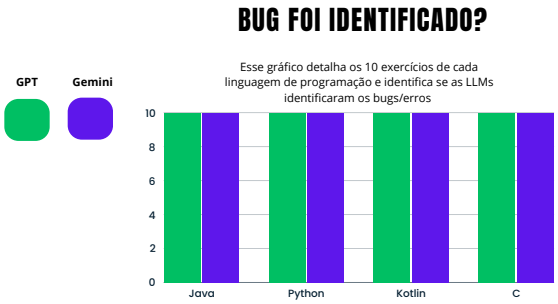

Figura 2 – Resultado da análise sobre a capacidade dos LLMs em identificar *bugs*

O gráfico presente na Figura 3 mostra os resultados da capacidade de explicação da causa dos *bugs* para os mesmos 40 exemplos de código. Observa-se que, para as linguagens Kotlin e C, ambas os LLMs apresentaram desempenho satisfatório. Em Python, ambas tiveram um acerto de 80%. Por fim, na linguagem Java, o GPT teve aproveitamento total, enquanto o Gemini apresentou explicações satisfatórias em 8 dos exemplos de código.

4.3 Avaliação da Ferramenta Proposta

Nessa seção será apresentado a ferramenta desenvolvida com o objetivo de auxiliar e ajudar os desenvolvedores a identificarem e compreenderem os *bugs* presentes em seus códigos. A ferramenta foi densenvolvida por meio da implementação de uma página *web*

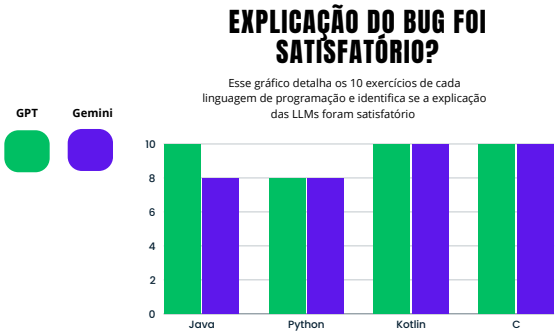

Figura 3 – Resultado da análise da capacidade dos LLMs em explicar as causas dos *bugs*

utilizando linguagem de programação python e o *framework* Flask aliado ao HTML. A ferramenta foi armazenada no PythonAnymore e utilizou uma chave da API do Gemini obtida na plataforma *Google Ai Studio*, que oferece ferramentas para desenvolver e implementar soluções com Inteligência Artificial. Foi usado uma chave GitHub, o qual é um token funcionando no lugar da senha da conta do usuário, que facilita o controle de acesso ao repositório. O público alvo são discentes de graduação, ou graduados na área de tecnologia que enfrentam problemas com *bugs* durante a programação de códigos.

A Figura 4 apresenta a interface do protótipo da ferramenta proposta. O usuário interage com a ferramenta selecionando uma das seguintes linguagens de programação: Java, Kotlin, Python ou C. Em seguida, o usuário insere sua intenção relacionada ao código que está apresentando um *bug*. Por fim, ele insere o código conténdo o *bug* na ferramenta e clica no botão "Enviar". A partir desse momento, a ferramenta retorna informações sobre a identificação do *bug*, juntamente com uma explicação sobre sua causa. No final da página, há um link para a avaliação da ferramenta.

[1] A ferramenta foi avaliada por meio da aplicação de um formulário contendo questões fechadas, nas quais foram avaliadas características como a usabilidade da interface gráfica, a facilidade de uso, a eficácia da ferramenta em identificar e explicar a causa do *bug*, e a clareza da explicação sobre a causa do *bug*. Além disso, o formulário incluiu uma questão aberta para que os usuários pudessem escrever suas sugestões e/ou críticas sobre a ferramenta proposta. O formulário completo encontra-se no Apêndice A deste manuscrito.

[1] https://github.com/antoniosa28/Flask

Figura 4 – Interface gráfica do protótipo da ferramenta proposta

4.3.1 Avaliação dos resultados do questionário sobre a ferramenta

Essa seção irá mostrar os resultados da avaliação realizada por usuários por meio de um formulário aplicado. O público alvo da pesquisa são discentes da área da tecnologia da informação.

Na coleta de dados, 11 discentes responderam o questionário. Na primeira questão foi perguntado qual curso eles(as) fazem ou fizeram, e obteve-se que 9 alunos cursam Sistema de Informação e 2 Ciências da computação, coforme apresentado na Figura 5.

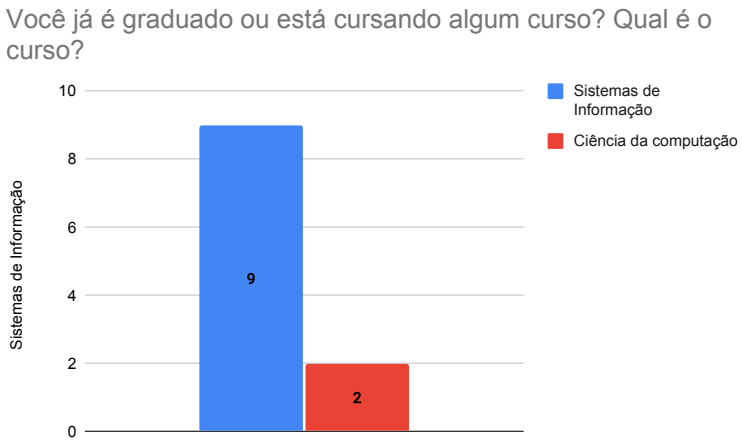

Figura 5 – Resultado da primeira questão do questionário.

Na segunda questão, foi perguntado sobre qual período os(as) alunos(as) estão,

obteve-se um resultado variado. Como mostrado na Figura 6, apenas 9,1% estão no 1º período, assim como quem está no 6ª período, que obteve a mesma porcentagem. Atingiu 18,2% estudantes do 7º e 8º período, um caso interessante que a mesma porcentagem aplicou-se para casos em que "Não se aplica", que são alunos já formandos ou que preferiram não responder a nenhuma das opções. O formulário nos trouxe um dado que, a maior parcela de alunos(as) estão no 2ª período que atingiu 27,3%.

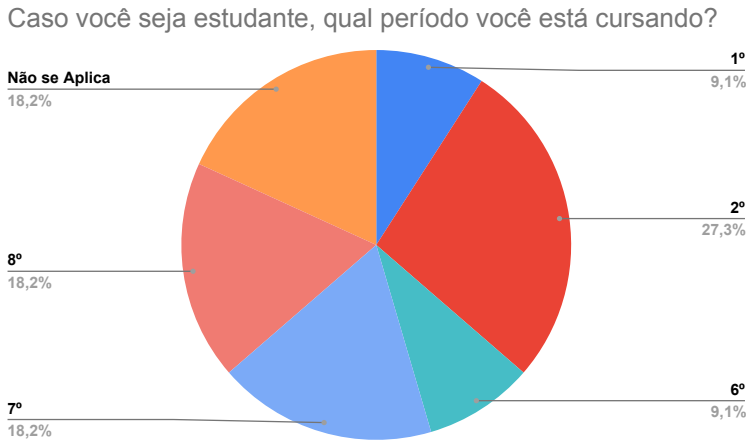

Figura 6 – Resultado da segunda questão do questionário.

Na questão sobre experiência na área de programação, mostrada na Figura 7, é possível notar que 4 alunos se encaixaram de 0 a 1 ano, diferente de quem obteve 2 a 3 anos de experiência que foram 5 estudantes, por fim apenas 2 alunos possuem 4 anos ou mais de experiência.

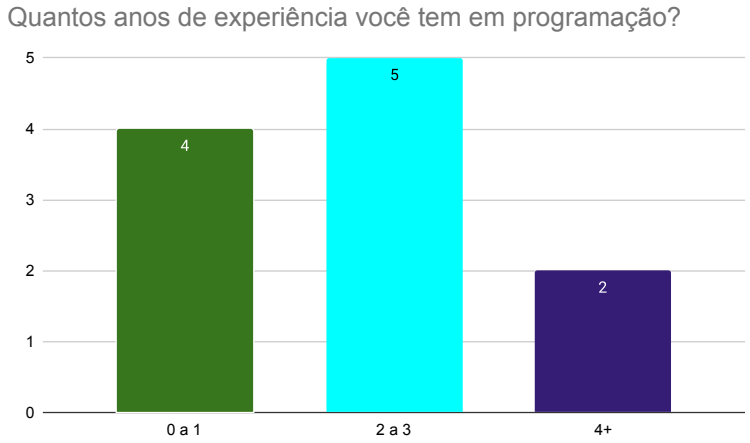

Figura 7 – Resultado da terceira questão do questionário.

O gráfico da Figura 8 mostra as linguagens de programação testadas pelos usuário ao utilizarem a ferramenta. Notou-se que a maioria utilizou a linguagem C, que é uma linguagem que normalmente os estudantes tem o primeiro contato, o Python é a segunda

mais utilizada e o Java a terceira com 18,20%, destaca-se a não utilização do Kotlin por nenhum dos estudantes.

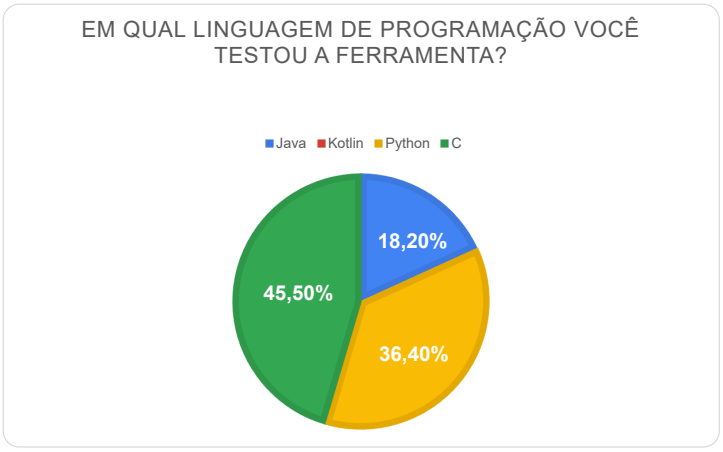

Figura 8 – Resultado da quarta questão do questionário.

Sobre a avaliação das funcionalidades da ferramenta, no que se refere sobre as caixas de entrada e saída, que respectivamente são "Intenção do usuário" e "Código", a Figura 9 mostra que, um pouco mais da metade (54,50%) acharam **Muito Adequada** o texto digitado e exibido, já os outros 45,50% julgaram **Adequada**. Esses resultados indicam que as caixas de entrada e saída da ferramenta são adequadas para o uso.

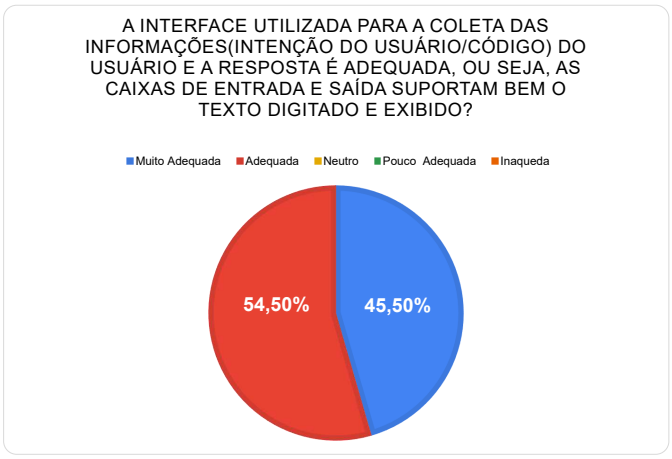

Figura 9 – Resultado da quinta questão do questionário.

A Figura 10 mostra os resultados da questão sobre a recomendação da ferramenta para outros usuários. Nota-se que 72,70% definitivamente recomendaria a ferramenta para outros usuários e 27,30% provavelmente recomendaria. Esses resultados mostram que os usuários gostaram da ferramenta e a indicariam a mesma para outros usuários.

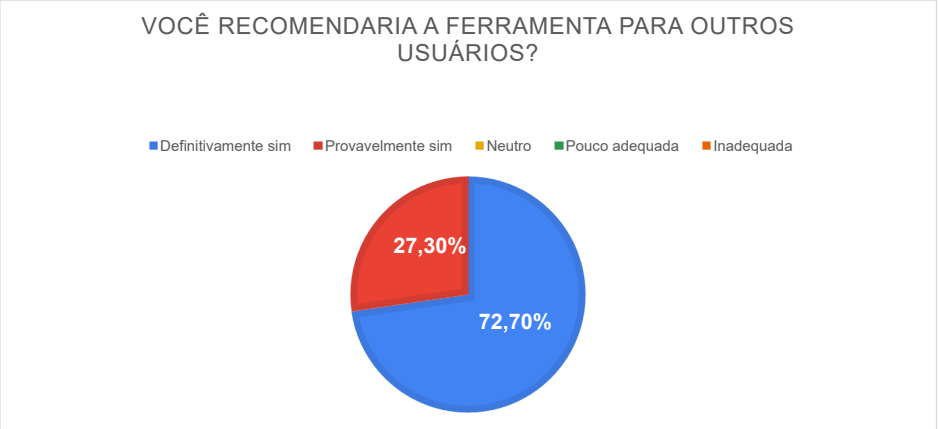

Figura 10 – Resultado da sexta questão do questionário.

Tratando-se da dificuldade de utiliza-se a ferramenta, a Figura 11 mostra que todos os usuários classificaram sua utilização como **Muito fácil**.

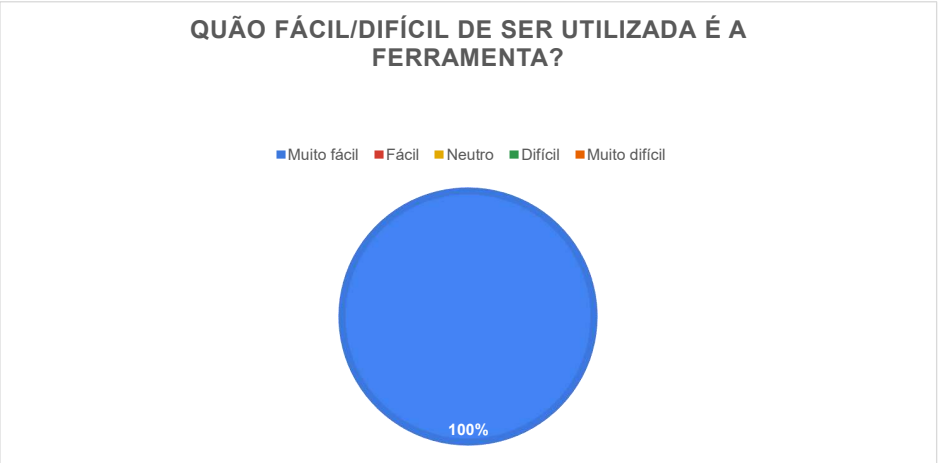

Figura 11 – Resultado da sétima questão do questionário.

A Figura 12 mostra a coleta de dados da utilidade da ferramenta na correção do *bugs* presente no código, foi apresentado que 72,70% dos usários acharam muito útil a

correção realizada, 18,20% avaliaram como útil e apenas uma pessoa teve o posicionamento neutro.

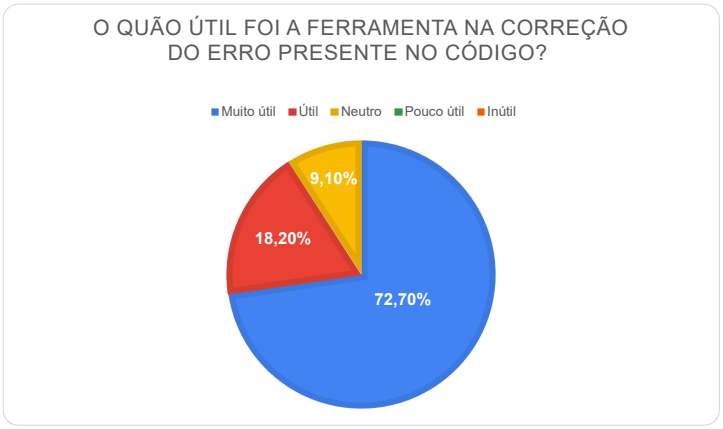

Figura 12 – Resultado da oitava questão do questionário.

No que se refere sobre a ferramenta conseguir identificar o *bugs* presente no código, a Figura 13 mostra que 1 estudante relatou que não foi útil, porém todo o restante respondeu que "Sim". Esses resultados mostram a efetividade da ferramenta em identificar os *bugs* presentes no código fonte.

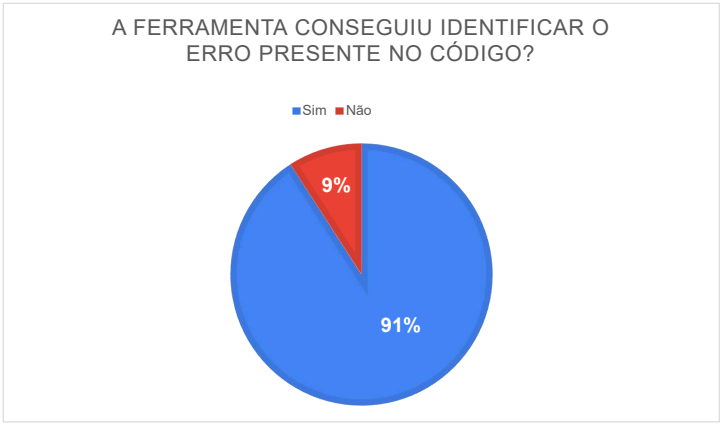

Figura 13 – Resultado da nona questão do questionário.

Relativo a resposta fornecida pela ferramenta de o quão claro foi a explicação do *bugs* presente no código, os estudantes julgaram com uma porcentagem de 72,70% que a resposta foi Muito Claro, 18,20% avaliaram que foi Claro identificar o *bugs* presente e apenas 9,10% responderam neutro, conforme a Figura 14. Logo chegou-se a uma conclusão que a ferramenta forneceu uma resposta clara sobre as causas dos *bugs* presentes no código.

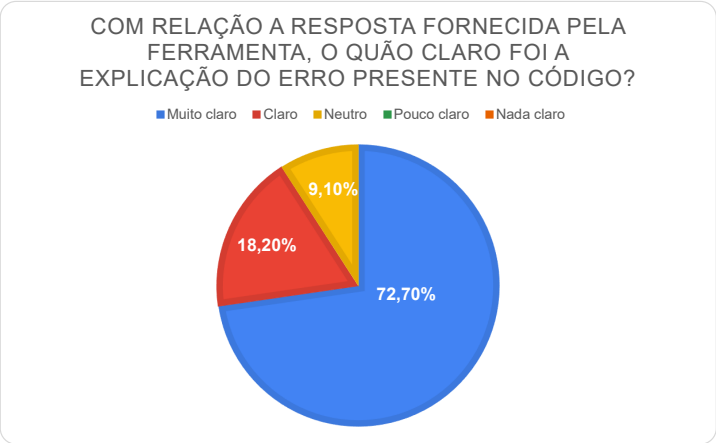

Figura 14 – Resultado da décima questão do questionário.

Diante desses resultados obtidos, é possível notar que a grande maioria dos estudantes considerou a ferramenta muito positiva para a identificação e explicação dos códigos que continham *bugs* presentes. Logo a ferramenta tem uma alta capacidade de comunicar de maneira efetiva as causas de *bugs* no código, facilitando o entendimento dos usuários.

As sugestões fornecidas pelos usuários destacaram pontos positivos e oportunidades de melhoria da ferramenta. Entre os elogios, mencionou-se a utilidade para correção e entendimento de *bugs* em códigos, bem como a clareza das explicações fornecidas, que tornam o aprendizado intuitivo e eficaz. Algumas sugestões de aprimoramento incluem melhorias na interface, ainda que reconheçam que o foco atual não seja a experiência do usuário, e a apresentação dos códigos refatorados com a devida indentação e separação em linhas distintas, para facilitar a compreensão. De modo geral, os participantes consideraram a ferramenta bem estruturada e eficaz em cumprir seu propósito.

5 Conclusão

Esse trabalho de conclusão de curso teve como próposito de avaliar e fornecer percepções sobre a capacidade dos LLMs na detecção e explicação de *bugs* presentes em códigos, contribuindo para o avanço da compreensão de suas potenciais aplicações na área de desenvolvimento de *software*.

Além disso, o presente trabalho também avaliou através de um questionário aplicado a usuários a viabilidade de oferecer uma ferramenta para auxiliar desenvolvedores na identificação e compreensão de *bugs* presentes em seus códigos. Observou-se que uma significativa maioria dos estudantes não apenas teve uma boa aceitação da ferramenta, mas também expressou aprovação em diversos aspectos, como sua clareza, facilidade de uso, e a forma como a recomendariam para outros usuários. Além do mais, muitos ressaltaram sua capacidade de identificar-se com as funcionalidades oferecidas, destacando sua utilidade prática no contexto acadêmico e sua contribuição para facilitar a compreensão e resolução de problemas.

Considerando o contexto atual, notou-se que o estudo pode contribuir para o avanço da compreensão de suas potenciais aplicações na área de desenvolvimento de *software* e correção de *bugs* de programação.

Para trabalhos futuros, considera-se a possibilidade de aplicar a abordagem desenvolvida diretamente em IDEs ou como extensões para navegadores. Essa integração pode tornar a ferramenta ainda mais acessível e eficiente, facilitando a detecção e compreensão de *bugs* no código em ambientes de desenvolvimento amplamente utilizados. Além disso, a disponibilização da ferramenta em diferentes plataformas poderia ampliar seu alcance e beneficiar um número maior de desenvolvedores, promovendo uma experiência de desenvolvimento mais fluida e interativa.

Referências

ALASMARI, O. A.; SINGER, J.; ADA, M. B. Do current online coding tutorial systems address novice programmer difficulties? In: **Proceedings of the 15th International Conference on Education Technology and Computers**. New York, NY, USA: Association for Computing Machinery, 2024. (ICETC '23), p. 242–248. ISBN 9798400709111. Disponível em: <https://doi.org/10.1145/3629296.3629333>. Citado na página 14.

ALBRECHT, E.; GRABOWSKI, J. Sometimes it's just sloppiness - studying students' programming errors and misconceptions. In: **Proceedings of the 51st ACM Technical Symposium on Computer Science Education**. New York, NY, USA: Association for Computing Machinery, 2020. (SIGCSE '20), p. 340–345. ISBN 9781450367936. Disponível em: <https://doi.org/10.1145/3328778.3366862>. Citado 2 vezes nas páginas 7 e 13.

ALHUMUD, W.; ALENZI, A.; BRYCE, R.; LI, Y.; ALSHAMMARI, N. Bug battles: A competition to catch bugs in different programming languages. **J. Comput. Sci. Coll.**, Consortium for Computing Sciences in Colleges, Evansville, IN, USA, v. 38, n. 7, p. 36–45, abr. 2023. ISSN 1937-4771. Citado na página 14.

ALSHAHWAN, N.; HARMAN, M.; HARPER, I.; MARGINEAN, A.; SENGUPTA, S.; WANG, E. Assured offline llm-based software engineering. In: **Proceedings of the ACM/IEEE 2nd International Workshop on Interpretability, Robustness, and Benchmarking in Neural Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2024. (InteNSE '24), p. 7–12. ISBN 9798400705649. Disponível em: <https://doi.org/10.1145/3643661.3643953>. Citado na página 14.

AMIN, M. F. I.; RAHMAN, M. M.; WATANOBE, Y.; DANIEL, M. M. Impact of programming language skills in programming learning. In: **2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)**. [S.l.: s.n.], 2022. p. 271–277. Citado na página 11.

BLINN, A.; LI, X.; KIM, J. H.; OMAR, C. Statically contextualizing large language models with typed holes. **Proc. ACM Program. Lang.**, Association for Computing Machinery, New York, NY, USA, v. 8, n. OOPSLA2, out. 2024. Disponível em: <https://doi.org/10.1145/3689728>. Citado na página 14.

BOU, G.; ROCHA, A. M.; QUINCOZES, V. E.; QUINCOZES, S. E.; KAZIENKO, J. F. Bou-guard: Uma abordagem para detecção de conteúdo impróprio na internet. In: **Proceedings of the XVII Workshop de Trabalhos de Iniciação Científica e de Graduação (WTICG)**. Juiz de Fora, MG, Brasil: [s.n.], 2023. (SBSeg '23). Citado na página 13.

CIPRIANO, B. P.; ALVES, P. Gpt-3 vs object oriented programming assignments: An experience report. In: **Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1**. New York, NY, USA: Association for Computing Machinery, 2023. (ITiCSE 2023), p. 61–67. ISBN

9798400701382. Disponível em: <https://doi.org/10.1145/3587102.3588814>. Citado na página 13.

DANTAS, C.; ROCHA, A.; MAIA, M. Assessing the readability of chatgpt code snippet recommendations: A comparative study. In: **Proceedings of the XXXVII Brazilian Symposium on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2023. (SBES '23), p. 283–292. ISBN 9798400707872. Disponível em: <https://doi.org/10.1145/3613372.3613413>. Citado 2 vezes nas páginas 7 e 12.

DVIVEDI, S. S.; VIJAY, V.; PUJARI, S. L. R.; LODH, S.; KUMAR, D. A comparative analysis of large language models for code documentation generation. In: **Proceedings of the 1st ACM International Conference on AI-Powered Software**. New York, NY, USA: Association for Computing Machinery, 2024. (AIware 2024), p. 65–73. ISBN 9798400706851. Disponível em: <https://doi.org/10.1145/3664646.3664765>. Citado na página 14.

FIGUEIREDO, J.; nALVO, F. G.-P. A tool help for introductory programming courses. In: **Ninth International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'21)**. New York, NY, USA: Association for Computing Machinery, 2021. (TEEM'21), p. 18–24. ISBN 9781450390668. Disponível em: <https://doi.org/10.1145/3486011.3486413>. Citado na página 14.

FWA, H. L. Experience report: Identifying common misconceptions and errors of novice programmers with chatgpt. In: **Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training**. New York, NY, USA: Association for Computing Machinery, 2024. (ICSE-SEET '24), p. 233–241. ISBN 9798400704987. Disponível em: <https://doi.org/10.1145/3639474.3640059>. Citado na página 14.

JACKSON, J. Development and application security collaboration. In: _____. **Corporate Cybersecurity: Identifying Risks and the Bug Bounty Program**. [S.l.: s.n.], 2022. p. 133–141. Citado 2 vezes nas páginas 7 e 12.

KURNIAWAN, D.; ABRAMSON, D. An ide framework for grid application development. In: **2008 9th IEEE/ACM International Conference on Grid Computing**. [S.l.: s.n.], 2008. p. 184–191. Citado na página 11.

LEINONEN, J.; HELLAS, A.; SARSA, S.; REEVES, B.; DENNY, P.; PRATHER, J.; BECKER, B. A. Using large language models to enhance programming error messages. In: **Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1**. New York, NY, USA: Association for Computing Machinery, 2023. (SIGCSE 2023), p. 563–569. ISBN 9781450394314. Disponível em: <https://doi.org/10.1145/3545945.3569770>. Citado na página 14.

LINDSTROM, G. Programming with python. **IT Professional**, v. 7, n. 5, p. 10–16, 2005. Citado na página 11.

MACNEIL, S.; DENNY, P.; TRAN, A.; LEINONEN, J.; BERNSTEIN, S.; HELLAS, A.; SARSA, S.; KIM, J. Decoding logic errors: A comparative study on bug detection by students and large language models. In: **Proceedings of the 26th Australasian Computing Education Conference**. New York, NY, USA: Association for Computing Machinery, 2024. (ACE '24), p. 11–18. ISBN 9798400716195. Disponível em: <https://doi.org/10.1145/3636243.3636245>. Citado na página 14.

MACNEIL, S.; TRAN, A.; HELLAS, A.; KIM, J.; SARSA, S.; DENNY, P.; BERNSTEIN, S.; LEINONEN, J. Experiences from using code explanations generated by large language models in a web software development e-book. In: **Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1**. New York, NY, USA: Association for Computing Machinery, 2023. (SIGCSE 2023), p. 931–937. ISBN 9781450394314. Disponível em: <https://doi.org/10.1145/3545945.3569785>. Citado na página 9.

OLIVEIRA, G. Silva de; GAO, Z.; HECKMAN, S.; LYNCH, C. Exploring novice programmers' testing behavior: A first step to define coding struggle. In: **Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1**. New York, NY, USA: Association for Computing Machinery, 2024. (SIGCSE 2024), p. 1251–1257. ISBN 9798400704239. Disponível em: <https://doi.org/10.1145/3626252.3630851>. Citado na página 14.

PUPO, Vinicius Milani Del. **Atividade probatória nos cibercrimes**. Trabalho de Conclusão de Curso (Graduação em Direito) — Universidade Federal de Uberlândia, 2023. Citado na página 7.

SAVELKA, J.; AGARWAL, A.; AN, M.; BOGART, C.; SAKR, M. Thrilled by your progress! large language models (gpt-4) no longer struggle to pass assessments in higher education programming courses. In: **Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1**. New York, NY, USA: Association for Computing Machinery, 2023. (ICER '23), p. 78–92. ISBN 9781450399760. Disponível em: <https://doi.org/10.1145/3568813.3600142>. Citado na página 10.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; POLOSUKHIN, I. Attention is all you need. In: **Proceedings of the 31st International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2017. (NIPS'17), p. 6000–6010. ISBN 9781510860964. Citado 2 vezes nas páginas 9 e 10.

YLDRM, R.; AYDN, K.; CETIN, O. Evaluating the impact of conventional code analysis against large language models in api vulnerability detection. In: **Proceedings of the 2024 European Interdisciplinary Cybersecurity Conference**. New York, NY, USA: Association for Computing Machinery, 2024. (EICC '24), p. 57–64. ISBN 9798400716515. Disponível em: <https://doi.org/10.1145/3655693.3655701>. Citado na página 14.

YIN, Z.; YUAN, D.; ZHOU, Y.; PASUPATHY, S.; BAIRAVASUNDARAM, L. How do fixes become bugs? In: **Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2011. (ESEC/FSE '11), p. 26–36. ISBN 9781450304436. Disponível em: <https://doi.org/10.1145/2025113.2025121>. Citado na página 13.

ZAMAN, S.; ADAMS, B.; HASSAN, A. E. Security versus performance bugs: A case study on firefox. In: **Proceedings of the 8th Working Conference on Mining Software Repositories**. New York, NY, USA: Association for Computing Machinery, 2011. (MSR '11), p. 93–102. ISBN 9781450305747. Disponível em: <https://doi.org/10.1145/1985441.1985457>. Citado na página 12.

Apêndices

APÊNDICE A – Formulário Proposto

Em relação ao formulário proposto, ele foi aplicado com o intuito de analisar os dados de acordo com a ferramenta utilizada.

A partir desses dados obtidos no apêndice, futuras pesquisas poderão ser realizadas, permitindo a averiguação dos resultados, em que possibilita a reprodução de estudos voltados a melhoria da presente ferramenta ou ao desenvolvimento de novos recursos, como aplicativo, programas ou até mesmo uma nova ferramenta.

A. Formulário sobre Ferramenta Web para correção de *bugs*

1. Você já é graduado ou está cursando algum curso? Qual é o curso?
2. Caso você seja estudante, qual período você está cursando?

   ☐ 1º

   ☐ 2º

   ☐ 3º

   ☐ 4º

   ☐ 5º

   ☐ 6º

   ☐ 7º

   ☐ 8º

   ☐ 9º

   ☐ 10º

   ☐ Não se Aplica
3. Quantos anos de experiência você tem em programação?
4. Em qual linguagem de programação você testou a ferramenta?
5. A interface utilizada para a coleta das informações(Intenção do usuário/Código) do usuário e a resposta é adequada, ou seja, as caixas de entrada e saída suportam bem o texto digitado e exibido?

   ☐ Muito adequada

   ☐ Adequada

☐ Neutro

☐ Pouco adequada

☐ Pouco adequada

☐ Inadequada

6. Você recomendaria a ferramenta para outros usuários?

   ☐ Definitivamente sim

   ☐ Provavelmente sim

   ☐ Neutro

   ☐ Provavelmente não

   ☐ Definitivamente não

7. Quão fácil/difícil de ser utilizada é a ferramenta?

   ☐ Muito fácil

   ☐ Fácil

   ☐ Neutro

   ☐ Díficil

   ☐ Muito díficil

8. O quão útil foi a ferramenta na correção do erro presente no código?

   ☐ Muito útil

   ☐ Útil

   ☐ Neutro

   ☐ Pouco útil

   ☐ Inútil

9. A ferramenta conseguiu identificar o erro presente no código?

   ☐ Sim

   ☐ Não

10. Com relação a resposta fornecida pela ferramenta, o quão claro foi a explicação do erro presente no código?

    ☐ Muito claro

    ☐ Claro

    ☐ Neutro

☐ Pouco claro

☐ Nada claro

11. Escreva suas sugestões e/ou críticas sobre a ferramenta avaliada.

APÊNDICE B – TCLE - Formulário

TCLE

Termo/Registro de Consentimento Livre e Esclarecido (TCLE) está sendo obtido de forma virtual antes do início da sua participação na pesquisa e coleta de dados. Antes de concordar em participar da pesquisa, você pode entrar em contato por email com o aluno Antonio Marcos Barbosa de Sá cujo email é o: antonio.sa@ufu.br, para discutir as informações do estudo.

Você tem o tempo que for necessário para decidir se quer ou não participar da pesquisa (conforme item IV da Resolução nº 466/2012 ou Capítulo. III da Resolução nº 510/2016).

A data de início e término da coleta de dados é: 21 de Agosto a 21 de Setembro.

Na sua participação, você responderá algumas perguntas. A pesquisa será feita através de um formulário do Google, sua participação será anônima, os resultados dessa pesquisas serão usados apenas para fins de pesquisa. A pesquisa é separada em 10 perguntas sobre a ferramenta de correções de *bugs* proposta pela atual pesquisa. O tempo mínimo estimado é de, aproximadamente, 10 minutos.

Você tem o direito de não responder qualquer questão, sem necessidade de explicação ou justificativa para tal. No questionário haverá opções de preferência para não serem respondidas.

Você não terá nenhum gasto e nem ganho financeiro por participar na pesquisa.

Nós, pesquisadores, atenderemos as orientações das Resoluções nº 466/2012, Capítulo XI, Item XI.2: f e nº 510/2016, Capítulo VI, Art. 28: IV - manter os dados da pesquisa em arquivo, físico ou digital, sob sua guarda e responsabilidade, por um período mínimo de 5 (cinco) anos após o término da pesquisa.

Para minimizar alguns riscos do ambiente virtual, é importante que você tenha todo o cuidado com a segurança e privacidade do local quando realizar o acesso às etapas virtuais da pesquisa para que sejam garantidos o sigilo e a confidencialidade necessários.

Não haverá benefícios, o intuito da pesquisa é a obtenção de dados para finalizar a pesquisa de graduação.

Havendo algum dano decorrente da pesquisa, você terá direito a solicitar indenização através das vias judiciais (Código Civil, Lei 10.406/2002, Artigos 927 a 954 e Resolução CNS nº 510 de 2016, Artigo 19).

Você é livre para deixar de participar da pesquisa a qualquer momento sem qual-

quer prejuízo ou coação. No questionário não será solicitado a identificação dos dados pessoais dos participantes, portanto esteja informado a impossibilidade de exclusão dos dados da pesquisa.

Em qualquer momento, caso tenha qualquer dúvida ou reclamação a respeito da pesquisa, você poderá entrar em contato com Antonio Marcos Barbosa de Sá - antonio.sa@ufu.br - (34) 998960228 ou Adriano Mendonça Rocha - adriano.rocha@ufu.br.

Você poderá também entrar em contato com o Comitê de Ética na Pesquisa com Seres Humanos – CEP, da Universidade Federal de Uberlândia, localizado na Av. João Naves de Ávila, nº 2121, bloco A, sala 224, campus Santa Mônica – Uberlândia/MG, 38408-100; pelo telefone (34) 3239-4131 ou pelo e-mail cep@propp.ufu.br. O CEP/UFU é um colegiado independente criado para defender os interesses dos participantes das pesquisas em sua integridade e dignidade e para contribuir para o desenvolvimento da pesquisa dentro de padrões éticos conforme resoluções do Conselho Nacional de Saúde.