



CENTRO UNIVERSITÁRIO 7 DE SETEMBRO – UNI7
CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

RENAN ALMEIDA DA SILVA

**INTELIGÊNCIAS ARTIFICIAIS GENERATIVAS APLICADAS À
REFATORAÇÃO DE SOFTWARE**

FORTALEZA

2023

RENAN ALMEIDA DA SILVA

**INTELIGÊNCIAS ARTIFICIAIS GENERATIVAS APLICADAS À
REFATORAÇÃO DE SOFTWARE**

Artigo apresentado ao Centro Universitário
7 de Setembro como requisito parcial para
obtenção do título de Bacharel em Sistemas
de Informação.

Orientador: Msc. Antonio Alex de Souza

FORTALEZA

2023

RENAN ALMEIDA DA SILVA

INTELIGÊNCIAS ARTIFICIAIS GENERATIVAS APLICADAS À REFATORAÇÃO DE SOFTWARE

Artigo apresentado ao Centro Universitário
7 de Setembro como requisito parcial para
obtenção do título de Bacharel em Sistemas
de Informação.

Renan Almeida da Silva

Artigo aprovada em: __/__/__

Antonio Alex de Souza (Orientador)

Centro Universitário 7 de Setembro – Uni7

Nome do examinador I (Examinador)

Centro Universitário 7 de Setembro – Uni7

Nome do examinador I (Examinador)

Centro Universitário 7 de Setembro – Uni7

Luciano Comin Nunes

Centro Universitário 7 de Setembro – Uni7

Coordenador do Curso

Resumo

Dados os recentes avanços das inteligências artificiais generativas na aplicação nos mais diversos domínios, um dos processos mais beneficiados dessa escalada de poder foi o de refatoração de código, onde em poucos minutos pode-se aperfeiçoar determinado código sem mesmo existir um conhecimento prévio do que está sendo realizado ou de como realizar a tarefa desejada. Um simples comando dado em linguagem natural pode transformar completamente um trecho de código problemático em algo mais bem desenvolvido. O objetivo central do trabalho é explorar as possibilidades da utilização da inteligência artificial generativa no contexto delimitado da refatoração de software, onde a partir de exemplos e pesquisa bibliográfica são analisados os potenciais benefícios e cuidados a serem tomados a utilização de tal tecnologia.

Palavras-chave: Refatoração de código; Inteligência artificial; Programação.

Abstract

Recently, generative artificial intelligence has been applied to a large variety of domains, and one of the most benefited processes from this power escalation has been code refactoring, where, in just a few minutes, a particular code can be improved without even requiring previous knowledge of what was made or how to accomplish it. It only takes a simple command in natural language to transform a problematic code snippet into something better written. Throughout this academic study, the central objective is to explore the potential for the use of artificial intelligence in a delimited code refactoring context, where examples and literature review are used to discuss potential benefits and precautions to take when using the technology.

Keywords: Code refactoring; Artificial intelligence; Programming.

SUMÁRIO

1. Introdução	6
1.1. Problematização	7
1.2. Objetivos	8
1.3. Relevância do estudo	8
1.4. Metodologia	9
1.5. Estrutura do documento	10
2. Fundamentação teórica	11
2.1. Refatoração de código	11
3. Inteligências artificiais generativas.....	16
3.1. Alucinações.....	18
3.2. Dicas de prompts	18
3.3. Processo de integração.....	19
4. Exemplos reais e casos de estudo	22
4.1. Exemplo 1 - Armazenamento de informação de conexão	22
4.2. Exemplo 2 - Conversão de tipos	26
4.3. Exemplo 3 - Soma binária	28
5. Trabalhos futuros.....	34
6. Conclusão	35
7. REFERÊNCIAS	36

1. Introdução

Nos últimos anos, a inteligência artificial (IA) apareceu como uma das maiores e mais influentes inovações no campo da tecnologia da informação, gerando um grande impacto em pessoas, empresas e processos. Em particular as inteligências artificiais generativas desempenham um papel notável na transformação das práticas de desenvolvimento e manutenção de software. Várias novas possibilidades são introduzidas na utilização de tais tecnologias em todo o processo de desenvolvimento de software, possibilidades estas, que trazem verdadeiros ganhos com inovação e que não se limitam somente a fazer buscas por respostas a perguntas do cotidiano.

As ferramentas de IA Generativa, no atual momento, estão na vanguarda da transformação digital, pois, estão equipadas com o que há de mais avançado em tecnologia de aprendizado de máquina e processamento de linguagem natural e abrangem uma gama muito grande de cenários onde podem ser utilizadas. Cenários como, implementação de alguma funcionalidade complexa de um software, geração de testes automatizados, análise de dados na área de ciência de dados, refatoração de software, já se beneficiam grandemente em larga escala.

Especificamente à tarefa de refatoração de software, já tem um ganho significativo com o advento da IA Generativa, pois permite a otimização de um determinado código de maneira mais eficiente e com menos recursos.

Com o rápido avanço da tecnologia de forma geral, a necessidade de se ter código de alta qualidade, que por si só traz os benefícios de menos falhas de execução e facilidade de manutenção, é crítica, pois sistemas com diversas e recorrentes falhas, tendem a ser facilmente substituídos por outros sistemas que tenham aproximadamente as mesmas funcionalidades. Nesse contexto, as IA Generativas surgem como uma ferramenta chave, prometendo não apenas melhorar a qualidade de código, mas também reduzir o tempo e custos associados ao seu desenvolvimento e manutenção. Este trabalho busca explorar a aplicação e o impacto dessas tecnologias inovadoras na refatoração de

software, um campo que está no cerne da modernização e eficiência da indústria de tecnologia.

1.1. Problematização

A refatoração de software, um processo crucial para a manutenção e aprimoramento de sistemas de software, enfrenta uma série de desafios contemporâneos. Primeiramente, a complexidade intrínseca desse processo não pode ser subestimada. À medida que os sistemas de software se tornam mais sofisticados e integrados, a tarefa de alterar código para melhorar sua estrutura interna, sem modificar seu comportamento externo, torna-se cada vez mais desafiadora. Esta complexidade é exacerbada pela constante evolução das necessidades dos usuários e pela rápida mudança de tecnologias, pressionando os desenvolvedores a manterem seus códigos não apenas funcionais, mas também adaptáveis e escaláveis.

Além disso, a eficiência é um fator crucial. Com cronogramas de projetos cada vez mais apertados e recursos limitados, a necessidade de refatorar código de maneira rápida e eficiente é mais premente do que nunca. Contudo, o equilíbrio entre a rapidez e a qualidade é delicado; atalhos no processo de refatoração podem levar a falhas ou deficiências que comprometem a integridade e a segurança do software.

Por fim, a demanda por qualidade de código nunca foi tão alta. Em um cenário globalizado, onde o software desempenha um papel fundamental em quase todos os aspectos da vida diária e comercial, a qualidade do código afeta diretamente a experiência do usuário e a confiabilidade do sistema. Isso coloca uma pressão adicional sobre os desenvolvedores para garantir que o código não apenas funcione bem, mas também esteja alinhado com as melhores práticas de programação e padrões de qualidade.

Neste contexto, as Inteligências Artificiais Generativas emergem como uma solução promissora. Com sua capacidade de aprender padrões complexos e gerar soluções otimizadas, essas tecnologias têm o potencial de abordar as nuances da refatoração de software de uma maneira que combina eficiência, eficácia e qualidade. A aplicação de IA Generativa na refatoração de software pode significar uma revolução na maneira como o código é mantido e

aprimorado, oferecendo uma abordagem inovadora para enfrentar os desafios que têm atormentado este campo há décadas.

1.2. Objetivos

Este trabalho tem como principal objetivo investigar e demonstrar o potencial das inteligências artificiais generativas na refatoração de software. Os objetivos específicos incluem:

Explorar a eficiência da IA Generativa na refatoração de código: avaliar como as tecnologias de IA Generativa podem otimizar o processo de refatoração, reduzindo o tempo e o esforço necessários para melhorar e manter códigos. Este objetivo envolve analisar a capacidade dessas ferramentas de realizar mudanças complexas no código de forma mais rápida e precisa do que os métodos tradicionais.

Analisar a aplicabilidade da IA Generativa em diferentes contextos de desenvolvimento de software: explorar como as soluções de IA Generativa podem ser aplicadas em uma variedade de contextos de desenvolvimento de software, incluindo diferentes linguagens de programação, plataformas e tipos de projetos. Este objetivo visa compreender a versatilidade e adaptabilidade das ferramentas de IA Generativa em diversos cenários de desenvolvimento.

Identificar desafios e limitações: além de explorar as capacidades e benefícios, é crucial identificar os desafios e limitações enfrentados ao integrar IA Generativa no processo de refatoração. Este objetivo abrange desde questões técnicas até considerações práticas associadas ao uso da IA em desenvolvimento de software.

1.3. Relevância do estudo

O estudo da aplicação de Inteligências Artificiais Generativas na refatoração de software é de suma importância por diversas razões. Primeiramente, ele aborda um campo emergente que está na interseção de duas áreas vitais da tecnologia moderna: a inteligência artificial e o desenvolvimento de software. Com o software desempenhando um papel cada vez mais crítico

em inúmeros aspectos da vida contemporânea, a necessidade de métodos eficientes e inovadores para sua manutenção e aprimoramento torna-se crucial.

Neste contexto, a refatoração de software com o auxílio de IA Generativa representa um avanço significativo, prometendo transformar as práticas tradicionais de desenvolvimento e manutenção de software.

Este estudo contribui para o campo da engenharia de software ao explorar como as técnicas de IA Generativa podem ser aplicadas para melhorar a qualidade e a eficiência na refatoração de código. Ao fazer isso, ele fornece insights valiosos para desenvolvedores, engenheiros de software e pesquisadores, ampliando o entendimento sobre as capacidades e limitações dessas tecnologias emergentes. Além disso, este trabalho se diferencia de estudos anteriores ao focar especificamente na aplicação prática dessas tecnologias em cenários reais de desenvolvimento de software, oferecendo uma perspectiva aplicada que é frequentemente ausente na literatura acadêmica.

Por fim, a relevância deste estudo também reside na sua abordagem prática e orientada a soluções. Ao fornecer exemplos concretos e análises detalhadas, ele não apenas teoriza sobre as potencialidades da IA Generativa, mas também demonstra como essas tecnologias podem ser efetivamente implementadas na prática. Isso não só beneficia o campo acadêmico, mas também oferece uma contribuição prática para a indústria de tecnologia da informação, que está constantemente em busca de métodos inovadores para otimizar processos e melhorar a qualidade do software.

1.4. Metodologia

A pesquisa do tipo exploratória foi utilizada para alcançar os objetivos já citados. Nesse sentido, a metodologia indutiva utilizada envolve geração de casos concretos que visam demonstrar a utilização do objeto de estudo (IA Generativas) aplicado a tarefa de refatoração de software.

Os casos foram demonstrados através de imagens geradas pelo autor, e a partir delas foram interpretados os resultados obtidos.

1.5. Estrutura do documento

Este Trabalho está organizado em uma estrutura que facilita a compreensão abrangente do papel das Inteligências Artificiais Generativas na refatoração de software. A seguir, apresento um resumo da estrutura do documento, delineando o conteúdo de cada capítulo:

Capítulo 1: Introdução - Este capítulo estabelece o contexto e a relevância do estudo, definindo o problema de pesquisa, os objetivos do estudo e a metodologia utilizada. Além disso, proporciona uma visão geral da estrutura do documento.

Capítulo 2: Fundamentação Teórica - Aqui, será explorado o panorama teórico que envolve a refatoração de software e as Inteligências Artificiais Generativas. Este capítulo abordará os conceitos-chave, a evolução das práticas de refatoração e o desenvolvimento das tecnologias de IA estabelecendo a base necessária para a compreensão do estudo.

Capítulo 3: Inteligências Artificiais Generativas na Refatoração de Software – Este capítulo detalha como as IA Generativas podem ser aplicadas na refatoração de software. Serão discutidos os benefícios, desafios e o potencial impacto destas ferramentas no processo de desenvolvimento e manutenção de software.

Capítulo 4: Estudo de Caso e Análise - Neste capítulo, serão apresentados estudos de caso práticos, onde as ferramentas de IA Generativa foram aplicadas na refatoração de software. A análise dos resultados desses estudos proporcionará insights sobre a eficácia e aplicabilidade dessas tecnologias no mundo real.

Capítulo 5 e 6: Discussão e Conclusões - Baseado nas análises realizadas, este capítulo discutirá os resultados obtidos, refletindo sobre as implicações para a prática e pesquisa futura. Serão apresentadas as conclusões do estudo, destacando as contribuições para o campo da refatoração de software e desenvolvimento de IA.

Referências - O documento concluirá com uma lista completa de todas as fontes consultadas e citadas ao longo do trabalho, fornecendo os créditos necessários e permitindo leituras adicionais sobre o tema.

2. Fundamentação teórica

2.1. Refatoração de código

Para a área de desenvolvimento de software, refatoração de código é o processo de reestruturar determinado código de uma aplicação, sem alterar a funcionalidade original. De forma a melhorar o código com relação à eficiência, segurança, complexidade e legibilidade (ŚCIŚLAK, Jarosław, 2022).

Quando pensamos em refatoração de código, muitos tendem a pensar se tratar somente de pegar um código malfeito e escrever de outra forma que execute mais rápido, ou que consuma menos recursos, porém, alguns outros importantes aspectos devem ser considerados quando esta recorrente tarefa se apresenta. Aspectos como a legibilidade do código podem se cruciais para o entendimento de determinado algoritmo, porém, alguns tendem a pensar que se o programa funciona nada mais importa, que seria somente uma questão de julgamento estético e que no fim, o compilador não se importa se o código é sujo ou limpo (FOWLER, 2019, p. 8).

Um sistema mal desenvolvido é difícil de mudar – pois é difícil de descobrir o que mudar e como estas mudanças vão interagir com o código existente para alcançar o comportamento esperado. E é difícil descobrir o que mudar, existe uma boa chance de erros serem cometidos e “bugs” introduzidos (FOWLER, 2019, p. 8).


Outros aspectos como redução de complexidade, melhora da manutenibilidade e extensibilidade são pontos chave que qualquer procedimento de refatoração. E com o passar dos anos algumas técnicas e ferramentas foram implementadas com o objetivo de ajudar nesta árdua tarefa.

Aqui estão algumas técnicas comumente utilizadas:

Extração de função: Essa técnica consiste em observar e entender uma parte do código e então extrai-la para uma nova função. Como escreve (FOWLER, Martin, 2019, p. 122) Eu olho para um fragmento de código, entendo o que ele está fazendo, então o extraio para sua própria função com um nome que corresponda ao seu propósito.

Quando agrupamos o código em um novo método ou função, a legibilidade e o reuso de código são melhorados. Podemos ver um exemplo de código na Figura 1:

Figura 1 - Função de extração em Javascript



```
function printOwing(invoice) {
  printBanner();
  let outstanding = calculateOutstanding();

  //print details
  console.log(`name: ${invoice.customer}`);
  console.log(`amount: ${outstanding}`);
}

function printOwing(invoice){
  printBanner();
  let outstanding = calculateOutstanding();
  printDetails(outstanding);

  function printDetails(outstanding){
    console.log(`name: ${invoice.customer}`);
    console.log(`amount: ${outstanding}`);
  }
}
```

Fonte: Martin Fowler, p122, 2019


Funções embutidas: Se trata do inverso da função de extração, é utilizado quando o corpo da função é tão claro quanto seu nome (FOWLER, Martin, 2019, p. 132). Dessa forma a função é substituída pelo seu conteúdo em todos os lugares onde era chamada.

Funções embutidas também podem ser utilizadas quando se tem um grupo de funções que parecem ter sido má desenvolvidas. Podemos embutir todas em uma grande função e então extrair novamente as funções da forma desejada.

Podemos ver um exemplo de refatoração utilizando funções embutidas na Figura 2:

Figura 2 - Funções embutidas em Javascript

```
function getRating(driver) {  
  return moreThanFiveLateDeliveries(driver)? 2 : 1;  
}  
  
function moreThanFiveLateDeliveries(driver) {  
  return driver.numberOfLateDeliveries > 5;  
}
```




```
function getRating(driver) {  
  return (driver.numberOfLateDeliveries > 5) ? 2 : 1;  
}
```

Fonte: Martin Fowler, p132,

Extração de variáveis: Similar a função de extração, esta técnica é utilizada quando existe uma expressão de difícil compreensão. Envolve criar uma variável e atribuir ao seu valor uma parte ou a totalidade da expressão, o que melhora a clareza do código. Um exemplo pode ser visto na Figura 3:

Figura 3 - Extração da variável em Javascript

```
return order.quantity * order.itemPrice -  
  Math.max(0, order.quantity - 500) * order.itemPrice * 0.05 +  
  Math.min(order.quantity * order.itemPrice * 0.1, 100);
```



```
const basePrice = order.quantity * order.itemPrice;  
const quantityDiscount = Math.max(0, order.quantity - 500) * order.itemPrice * 0.05;  
const shipping = Math.min(basePrice * 0.01, 100);  
return basePrice - quantityDiscount + shipping;
```


Fonte: Martin Fowler, p136, 2019

Expressões podem se tornar muito complexas e de difícil leitura. Nessas situações, variáveis locais podem ajudar quebrando a expressão em algo mais manejável. Em outras palavras, elas dão a habilidade de dar nome às partes de uma lógica mais complexa. Isso permite um melhor entendimento e do propósito do que está codificado. (FOWLER, Martin, 2019, p. 136)

Mudança na declaração de funções: Algumas vezes uma função é nomeada de forma que não deixa claro o que faz. Renomear de maneira mais descritiva torna o seu conteúdo mais fácil de ser compreendido. Um exemplo pode ser visto na Figura 4:

Figura 4 - Mudança na declaração de funções em Javascript

```
function circum(radius) {...}
```



```
function circumference(radius) {...}
```

Fonte: Martin Fowler, p141, 2019

Como fala (FOWLER, Martin, 2019, p. 142) um bom nome permite que o que a função faz seja compreendido ao ver somente sua chamada, sem sequer ver o código que a implementa. No entanto, pensar em um bom nome é difícil, e dificilmente será obtido de primeira. Quando um nome confuso é encontrado, é tentador deixar como está, afinal, é somente um nome. Isso é trabalho do demônio “Obfuscatis”; pelo bem da alma do seu código você nunca deve escutá-lo.

Encapsulamento de variável: O princípio de esconder a estrutura de dados utilizada e somente prover uma interface bem definida é chamado de encapsulamento (SIMÃO, Leonardo Mendes, 2023).

Refatorar se trata de manipular os elementos de nossos programas. Dados são mais estranhos de manipular do que funções. Pois usar uma função frequentemente trata-se de chamá-la, e podemos facilmente mover ou renomear uma função enquanto mantém-se intacta da antiga função como uma função de encaminhamento (meu código antigo chama a função antiga que chama a nova função).

Dados são mais estranhos pois não podemos fazer isso. Se movermos os dados, teremos então que mudar todas as referências, de uma só vez, para manter o código funcionando. Para poucas linhas de código isso não faz tanta diferença, porém quanto maior o escopo maior será a dor de cabeça (FOWLER, Martin, 2019, p. 151).

Um exemplo de refatoração utilizando o encapsulamento de variável pode ser visto na Figura 5:

Figura 5 - Encapsulamento de variável em Javascript

```
let defaultOwner = { firstName: "Martin", lastName: "Fowler" };
```



```
let defaultOwnerData = { firstName: "Martin", lastName: "Fowler" };  
export function defaultOwner() { return defaultOwnerData; }  
export function setDefaultOwner(arg) { defaultOwnerData = arg; }
```

Fonte: Martin Fowler, p150, 2019

Dessa forma a melhor maneira de realizar grandes alterações em acessos de dados, é primeiro realizando um encapsulamento, fazer com que todo acesso aos dados passe por funções.

2.1.1 Evolução das ferramentas de refatoração

Inicialmente, a refatoração era realizada de forma totalmente manual, o que trazia alguns malefícios como a quantidade de esforço humano necessário e a tendência a erros. Com o aperfeiçoamento do processo de desenvolvimento de software, várias ferramentas foram introduzidas para auxiliar nas refatorações. Alguns exemplos como ferramentas de “*linting*”, são capazes de percorrer o código em busca de potenciais erros e pontos de melhoria (FAYOCK, Coby, 2019).

Mais recentemente, as IDE’s (*Integrated Development Environments*) como “Eclipse”, “IntelliJ IDEA”, “Visual Studio” têm ferramentas já incorporadas que permitem aos desenvolvedores refatorar o código automaticamente, reduzindo a chance de erros e melhorando a eficiência (BASUMALLICK, Chiradeep, 2022).

Ainda mais, na era dos DevOps (*Development Operations*) e da integração contínua/entrega contínua (CI/CD), ferramentas automatizadas de qualidade de código como o “SonarQube” tornaram-se populares. Elas oferecem “*code review*” automatizado, verificação automática de cobertura de testes unitários, análise de segurança, análise de performance e possíveis erros de concorrência, tornando a refatoração mais sistemática e integrada (MALLOY, Sean, 2023).

3. Inteligências artificiais generativas

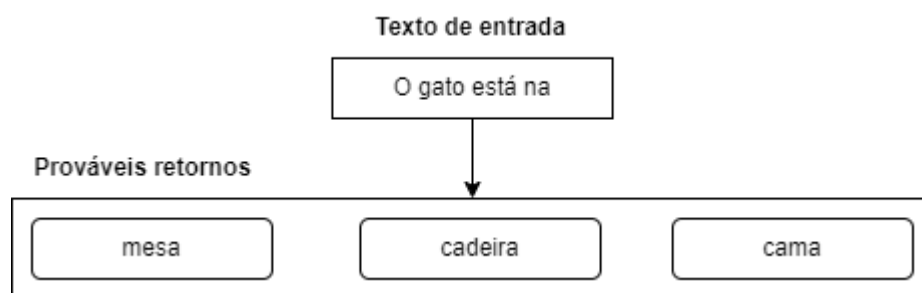
Inteligências artificiais generativas, também chamadas de LLM's (*Large Language Models*) são modelos de linguagem que se utilizam de técnicas de aprendizado de máquina para realizar a geração de textos a partir de uma enorme base de dados advinda da internet.

Um LLM, ou Large Language Model, é um tipo de modelo de linguagem de alta capacidade que utiliza algoritmos avançados de processamento de linguagem natural (NLP) para entender e gerar texto. Esses modelos são treinados em grandes quantidades de dados textuais, como livros, artigos, sites e muito mais, a fim de aprender a estrutura da linguagem e as relações semânticas entre palavras e frases. (SOUZA, Alex, 2019)

O treinamento básico de um modelo de linguagem envolve prever a próxima palavra em uma sequência de palavras. Duas técnicas são principalmente utilizadas no treinamento, a “*next-token-prediction*” e “*masked-language-modeling*” (RUBY, Molly, 2023).

Na primeira técnica é dada uma sequência de palavras como entrada e é pedido que a próxima palavra da sequência seja “adivinhada”. Por exemplo, se a entrada “O gato está na” fosse inserida como mostra a Figura 6:

Figura 6 - Exemplo de “next-token_prediction”



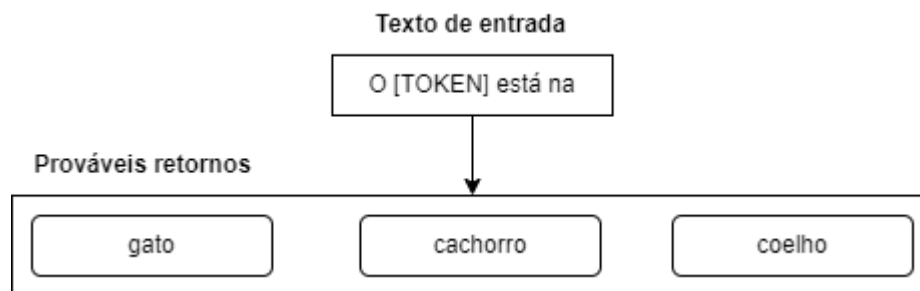
Fonte: Elaborado pelo autor.

O modelo deve então retornar palavras como mesa, cadeira ou cama, por conta da alta probabilidade de estas palavras ocorrerem dado o contexto (RAMPONI, Marco, 2022).

A segunda técnica advém da primeira, e consiste em adicionar um [TOKEN] no meio de uma frase. Então é solicitado que o modelo retorne a

palavra que corresponde corretamente ao contexto apresentado. Por exemplo, se fosse apresentado ao modelo o texto de entrada conforme a Figura 7:

Figura 7 - Exemplo de “masked-language-modeling”



Fonte: Elaborado pelo autor.

O modelo então deve retornar palavras como gato, cachorro ou coelho.

Dessa maneira, quando utilizado no contexto de programação, o código pode ser gerado a partir de uma entrada do usuário de acordo com uma sequência de palavras que aparecem sempre que utilizadas no contexto de software. Ainda mais, um código pode ser passado como entrada, para que a IA possa processá-lo e de acordo com o que pedido retornar o código refatorado (RAMPONI, Marco, 2022).

Alguns exemplos de inteligências artificiais generativas são o ChatGPT (OpenAI), Bard (Google), Claude AI (Antropic) e a Bing AI (Microsoft). Dentre essas podem ser destacadas duas, na qual iremos utilizar neste trabalho:

ChatGPT¹: A OpenAI impactou o mundo quando lançou o ChatGPT no final de 2022. Era esperado que a nova AI generativa transformasse totalmente indústrias, educação, mídia, e o mundo tecnológico de maneira geral. E antes mesmo da nova tecnologia ser totalmente assimilada pelos usuários, uma nova versão ainda mais poderosa foi lançada: ChatGPT 4.0.

Até pouco tempo atrás a IA da OpenAI só tinha acesso a dados até setembro de 2021, porém, recentemente passou a ter acesso a dados até abril de 2023 e acesso à internet em sua versão 4.0 (versão que cobra assinatura mensal) (RADFORD, Antoinette; KLEINMAN, Zoe, 2023).

¹ Para acessar o ChatGPT: <https://chat.openai.com/>

Google Bard²: A IA Generativa da Google foi lançada em 2023, e já está rodando em mais de 40 idiomas diferentes. E tem como principal trunfo o acesso ilimitado ao imenso banco de dados de sua criadora, que é a maior referência em buscas online da atualidade.

O Bard enquanto ferramenta, diferentemente do ChatGPT, sempre esteve diretamente conectada à internet, tendo dessa forma seu banco de dados atualizado constantemente.

3.1. Alucinações

No contexto humano, alucinações ocorrem quando o indivíduo tem convicção de que está recebendo estímulos visuais, auditivos, sentimentais, mas que na realidade não estão existindo. Acredita-se que tal fenômeno ocorra devido a uma desregulação da atividade neural (VARELLA, Drauzio, 2023).

No contexto de inteligências artificiais ocorre algo parecido, onde o conteúdo gerado é de certa forma “fantasioso”, pois não se baseia na realidade. Por exemplo, uma IA dedicada a gerar respostas para usuários sobre o funcionamento de determinada empresa poderia gerar respostas eloquentes completamente equivocadas. Ou um sistema de geração de imagens poderia adicionar detalhes que nunca existiram em determinado tipo de objeto (ALCARAZ, Anthony, 2017).

O fato desse tipo de comportamento ser possível é um grande ponto de atenção no uso da tecnologia para o contexto deste estudo, pois sob hipótese alguma, uma refatoração de texto deve modificar a funcionalidade de um trecho de código ou ainda mais gerar código aleatório que não satisfaz em nada a entrada do usuário.

3.2. Dicas de prompts

O “*prompt*”, é o comando de entrada onde o usuário descreve a sua solicitação para a IA, sendo assim uma parte muito importante para a efetividade do resultado retornado pela inteligência artificial generativa.

² Para acessar o Google Bard: <https://bard.google.com/>

O banco de dados de uma IA, por maior que seja, não consegue abranger todos os contextos existentes, então durante a geração do “*prompt*” é importante que o usuário tenha o cuidado de passar a maior quantidade de contexto possível à sua solicitação, pois mesmo entre humanos, uma comunicação clara e objetiva atenua problemas de falta de entendimento e consequentes resultados indesejados.

Podemos enumerar mais algumas boas práticas na geração de comandos:

- Evitar ser vago ou ambíguo nos comandos.
- Incluir exemplos para melhorar o “entendimento”.
- Pense em quem você quer que a IA seja ao responder ao seu comando.
- Incluir a formatação de como você deseja que a resposta seja estruturada.
- Incluir em que tom a resposta deve ser dada. Por exemplo, pode pedir para que a resposta seja dada em um tom formal ou casual.

Todas essas dicas visam prover uma melhoria no processo de “comandar” a IA para que ela possa gerar uma saída mais satisfatória, que ajude o usuário a alcançar o resultado esperado sem frustrações causadas por “*prompts*” obscuros, que não são claros quanto ao seu objetivo.

3.3. Processo de integração

Incorporar qualquer modelo IA no processo de desenvolvimento de software pode trazer vários benefícios, incluindo a proposição de refatorações baseadas nas melhores práticas e ajudar programadores a entenderem mais facilmente trechos de código mais complicados. Entretanto é preciso salientar que tais ferramentas, não foram desenvolvidas visando a compreensão de código da mesma forma que humanos. O entendimento do modelo é essencialmente estatístico, derivado de grandes quantidades de dados como já citado anteriormente.

Por mais que a IA possa gerar textos que parecem criativos, ela está simplesmente reorganizando palavras vistas anteriormente. Programadores, ao contrário, são treinados para serem criativos e a conceberem novas soluções para problemas.

Mesmo assim, esse modelo pode ser muito útil para dar sugestões e ajudar a automatizar algumas tarefas.

Algumas formas de utilizar as inteligências artificiais no processo de refatoração:

- Identificação de áreas para refatoração: Pode ser treinado em um conjunto de código onde comentários foram escritos para indicar áreas que precisam de refatoração. Isto pode ajudar o modelo a prever áreas parecidas que precisam ser reescritas em outros pontos do código.
- Gerar sugestões de refatoração: Quando uma área que precisa de refatoração é identificada, tais ferramentas podem gerar soluções em potencial. Essas soluções podem posteriormente ser analisadas por um ser desenvolvedor humano visando qualidade e se a lógica do código não foi alterada de alguma forma.
- Explicar sugestões de refatoração: Podem ser geradas explicações dos motivos que determinado trecho de código precisa de refatoração. Isso é especialmente útil para programadores menos experientes que ainda estão aprendendo boas práticas e lógica.
- Automatização de refatorações simples: Para aquelas refatoração bem simples, como por exemplo renomear um método ou uma propriedade, a IA pode gerar o novo código de maneira simples e rápida.

Lembre-se, a chave para o sucesso na incorporação das LLM's no processo de refatoração é utilizá-las como uma ferramenta de assistência aos desenvolvedores, não para substituí-los. Desenvolvedores ainda precisam rever, analisar e testar as sugestões geradas, ainda mais que refatorações mais complexas ainda estão além da capacidade da IA, pois, entre as suas limitações estão a quantidade de caracteres que podem ser inseridos e a quantidade limite de entradas.

Entre os diversos aspectos das linguagens de programação e processos envolvidos no desenvolvimento de software, temos as seguintes considerações:

- **Sintaxe e semântica:** Diferentes linguagens de programação têm diferentes regras de como devem ser escritas (sintaxe) e o que diferentes partes do código significam (semântica) (VENTURA, Plínio, 2016). Essas diferenças podem impactar no quão fácil, ou quão difícil, pode ser o processo de refatoração.
- **Paradigmas de programação:** Diferentes linguagens suportam diferentes paradigmas, como orientação a objetos, funcional ou procedural. Estes paradigmas têm diferentes forças e fraquezas e podem se adequar a diferentes tipos de projetos (JUNGTHON, Gustavo, 2008).
- **Ciclo de vida de desenvolvimento de software (SDLC):** O SDLC (*Software Development Life Cycle*) geralmente envolve alguns estágios, onde estão inclusos planejamento, análise, implementação, testes, manutenção (SERVICENOW, 2023). A refatoração pode ocorrer em diversas etapas do SDLC geralmente é realizada na implementação e na manutenção.
- **Metodologia de desenvolvimento:** Diferentes times utilizam diferentes metodologias para organizar seus trabalhos, como “*Waterfall*” e “*Agile*” (CUBOUP, 2023). Estas metodologias podem impactar quando e como as tarefas de refatoração serão tratadas.
- **Testes:** Teste é uma parte crucial de todo o processo de desenvolvimento de software e é particularmente importante no processo de refatoração, pois todo código refatorado deve continuar passando normalmente pelos testes implementados, para assegurar que ele continua se comportando como esperado.

Estes são somente alguns aspectos de linguagens e processos de programação que podem potencialmente impactar em como as inteligências artificiais são incorporadas na refatoração de determinado código.

4. Exemplos reais e casos de estudo

Alguns casos reais serão apresentados onde o ChatGPT ou Google Bard, foram utilizados para realizar a refatoração de determinado código. Os casos mostrados estão todos em inglês, porque a IA provê melhores resultados quando utilizada em sua linguagem nativa, pois uma camada extra de tradução é adicionada quando outra linguagem é utilizada o que pode prejudicar os resultados (DAVE, Paresh, 2023).

4.1. Exemplo 1 - Armazenamento de informação de conexão

Figura 8 - Connection string

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Host=localhost;Database=MovieDb;Username=postgres;Password=yourpassword"
  }
}
```

Fonte: Elaborado pelo autor.

A Figura 8 representa um “JSON” que contém uma “string” de conexão com o banco de dados, esse tipo de configuração é geralmente utilizado no arquivo “appsettings.json” que armazena informações para configuração da aplicação Microsoft .NET 6.

Da forma que está funciona, porém, temos um grave problema de segurança, onde informações que deveriam ser secretas, como a senha, estão expostas.

Podemos utilizar o ChatGPT para facilmente corrigir o problema, com a seguinte entrada mostrada na Figura 9:

Figura 9 - “Prompt” inserido no ChatGPT para correção da string de conexão

Having connections strings in the settings as clear text is a bad practice. Any sugestions on how to make it more secure?

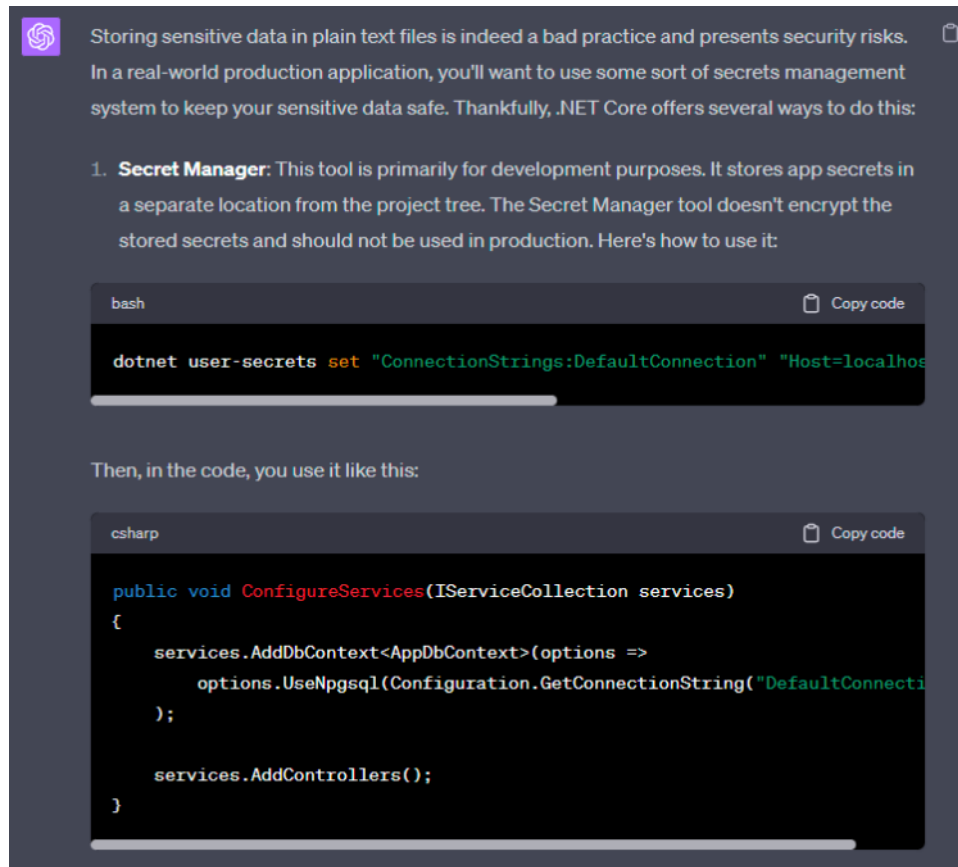


Fonte: Elaborado pelo autor.

O “prompt” executado diz: “Ter linhas de conexão nas configurações como texto explícito é uma má prática. Alguma sugestão de como tornar isto mais seguro?”

O seguinte é retornado:

Figura 10 - Retorno do ChatGPT



Fonte: Elaborado pelo autor.

Neste retorno (Figura 10), a IA sugeriu um script para que seja utilizado o “*user-secrets*” que é a forma correta de armazenar informações sensíveis no C# e ainda forneceu como deve ser utilizado na aplicação. Além de uma breve explicação do porquê utilizar “*secrets*” nesse caso é uma boa opção, porém somente em ambiente de desenvolvimento.

Na continuação da resposta o ChatGPT ainda retornou algumas opções para serem utilizadas em produção, assim como mostra a Figura 11:

Figura 11 - Retorno do ChatGPT - Continuação

The connection string is retrieved in the same way, but it is not stored in the project files.

2. **Environment Variables:** These are suitable for all environments and the .NET Core Configuration API can read them. Environment variables are a good way to avoid committing sensitive data into your version control. However, they are not secure for production environments as they can be accessed by anyone who can access the environment.
3. **Azure Key Vault:** If your application is hosted on Azure, you can use Azure Key Vault for storing, retrieving, and managing secrets like connection strings, access keys, and so on. It provides a secure and scalable way to manage secrets in your applications.
4. **Hashicorp Vault:** This is another powerful secrets management tool which can be used in applications deployed anywhere, not just on Azure. It allows you to securely store and tightly control access to tokens, passwords, certificates, encryption keys for protecting secrets and other sensitive data.

Remember that it's always a good idea to keep secrets out of your code and to use industry-standard encryption for sensitive data. Different environments have different best practices, so always check the recommendations for your specific deployment environment.

Fonte: Elaborado pelo autor.

Com estas opções em mãos podemos agora pedir para que seja retornado como adicionar, por exemplo, o “*Azure Key Vault*” (Serviço em nuvem da Microsoft utilizado para armazenamento de informações sensíveis) na aplicação. O “*prompt*” utilizado pode ser visto na Figura 12:

Figura 12 - “Prompt” inserido no ChatGPT para perguntar como adicionar o “Azure Key Vault” na aplicação

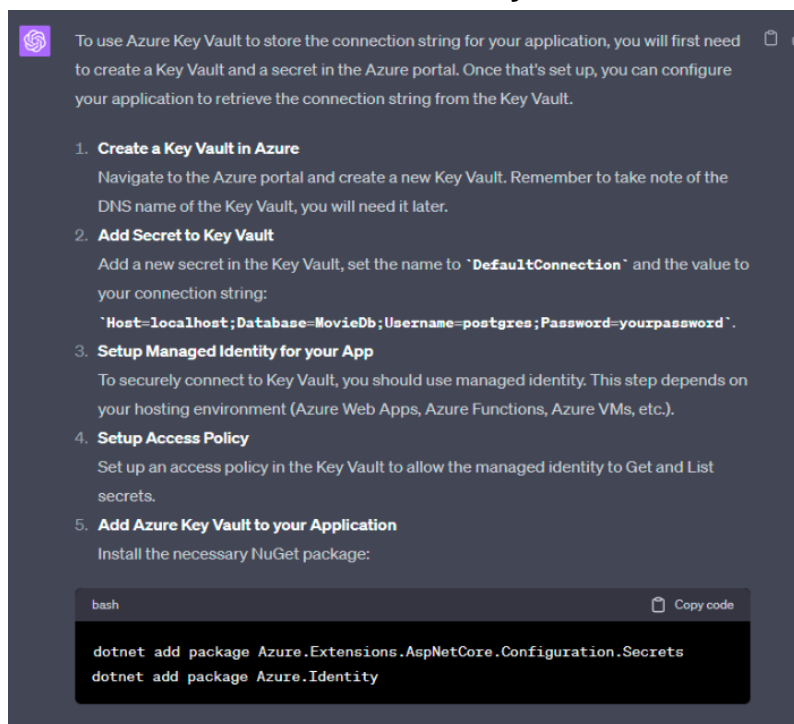
I use the Azure Key Vault at my application, show me how I can add it to my application to store the connection string

Fonte: Elaborado pelo autor.

O texto do “*prompt*” diz: “Eu uso “Azure Key Vault” em minha aplicação, me mostre como eu posso adicionar isso na minha aplicação para armazenar a string de conexão.”

O retorno pode ser visto na Figura 13:

Figura 13 - Resposta contendo o processo de adicionar uma chave no "Azure Key Vault"

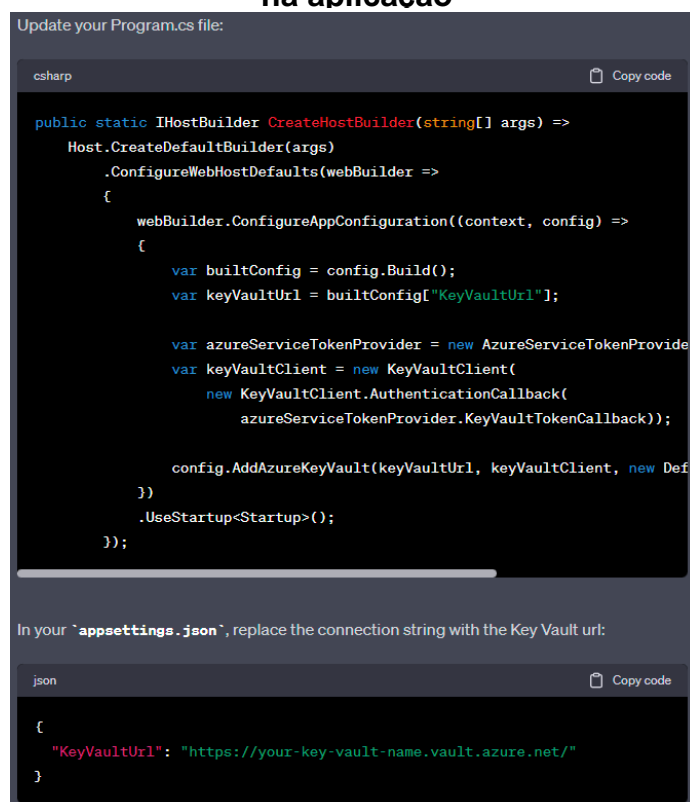


Fonte: Elaborado pelo autor.

Nessa resposta foi detalhado como a *"string"* de conexão pode ser configurada no *"Azure Key Vault"*.

E continua com a explicação de como alterar a aplicação na Figura 14:

Figura 14 - Resposta contendo o conteúdo a ser adicionado na aplicação



Fonte: Elaborado pelo autor.

Nesse primeiro exemplo foi demonstrado o poder de refatoração de um código problemático, com brechas de segurança e a resposta detalhada de como contornar esse problema, porém, não devemos em momento nenhum confiar 100% na resposta sem realizar as devidas análises e testes, como já mencionado.

4.2. Exemplo 2 - Conversão de tipos

Nesse segundo exemplo utilizando a linguagem C#, temos uma função que realiza a conversão de um “*guid*” (identificador global exclusivo) para seu correspondente em “*base64*” (método para codificação de dados). A função pode ser vista na Figura 15:

Figura 15 - Função para conversão de uma guid

```
public string GuidToBase64(string guidString)
{
    Guid guid = Guid.Parse(guidString);
    byte[] bytes = guid.ToByteArray();
    string base64 = Convert.ToBase64String(bytes);

    return base64;
}
```

Fonte: Elaborado pelo autor.

No código mostrado na Figura 13, temos alguns problemas. Primeiramente a função “*Guid.Parse*” lança exceções que não estão tratadas que podem ocorrer no caso de a entrada ser um “*guid*” inválido. Podemos pedir para o ChatGPT realizar o tratamento desse caso. Na Figura 16 podemos ver o código resultado da solicitação que o problema fosse tratado com a adição de um tratamento de exceção:

Figura 16 - Retorno 1 do exemplo 2

```
public string GuidToBase64(string guidString)
{
    try
    {
        Guid guid = Guid.Parse(guidString);
        byte[] bytes = guid.ToByteArray();
        string base64 = Convert.ToBase64String(bytes);

        return base64;
    }
    catch (FormatException)
    {
        // Log error, rethrow the exception, return a default value or handle the error in another appropriate way
        return null; // Return null in this example
    }
}
```

Fonte: Elaborado pelo autor.

Já a Figura 17 mostra uma segunda solicitação onde foi pedido que o código fosse tratado sem a utilização do bloco “try” e “catch”:

Figura 17 - Retorno 2 do exemplo 2

```
public string GuidToBase64(string guidString)
{
    Guid guid;
    if(Guid.TryParse(guidString, out guid))
    {
        byte[] bytes = guid.ToByteArray();
        string base64 = Convert.ToBase64String(bytes);

        return base64;
    }
    else
    {
        return null;
    }
}
```

Fonte: Elaborado pelo autor.

No primeiro retorno, temos o tratamento da exceção com um bloco “try”, “catch” que lança a exceção no caso de ocorrência. Porém uma outra sugestão foi solicitada, onde o retorno mostra a utilização do método “Guid.TryParse” que caso a entrada não seja válida retorna false e dessa forma não é necessário realizar o lançamento da exceção.

Porém, ainda temos mais um problema no código refatorado, o método “ToByteArray()” aloca espaço no “heap”, e para o caso que estamos tratando isto não é necessário, pois sabemos que um “guid” válido tem um tamanho limitado

de 16 “bytes”. Dessa forma será solicitado que o ChatGPT gere um código onde não é alocado nada no “heap”. O código retornado pode ser visto na Figura 18:

Figura 18 - Código do exemplo 2 sem alocação no “heap”

```
public string GuidToBase64(string guidString)
{
    if (Guid.TryParse(guidString, out Guid guid))
    {
        Span<byte> guidBytes = stackalloc byte[16]; // Allocate on stack
        guid.TryWriteBytes(guidBytes);

        Span<char> base64Chars = stackalloc char[24]; // Base64 will always be 24 characters for a Guid
        if (Convert.TryToBase64Chars(guidBytes, base64Chars, out _))
        {
            return string.Create(24, base64Chars, (chars, state) => state.CopyTo(chars));
        }
    }

    return null; // Or handle error in an appropriate way
}
```

Fonte: Elaborado pelo autor.

Dessa forma utilizando a expressão “*stackalloc*” podemos alocar o tamanho da “string” necessário para armazenar o “guid” diretamente na “stack”, o que traz alguns benefícios como limitação de acessibilidade por outras “threads”, velocidade de processamento.

Assim o código foi completamente refatorado em poucos minutos.

4.3. Exemplo 3 - Soma binária

Neste terceiro exemplo temos um código em “python” que realiza uma soma binária. No código da Figura 19 o resultado está incorreto e existem algumas oportunidades de melhorias:

Figura 19 - Código inicial exemplo 3

```
l1 = [1,1,0,1,1]
l2 = [1,0,0,0,0]
brack = 0

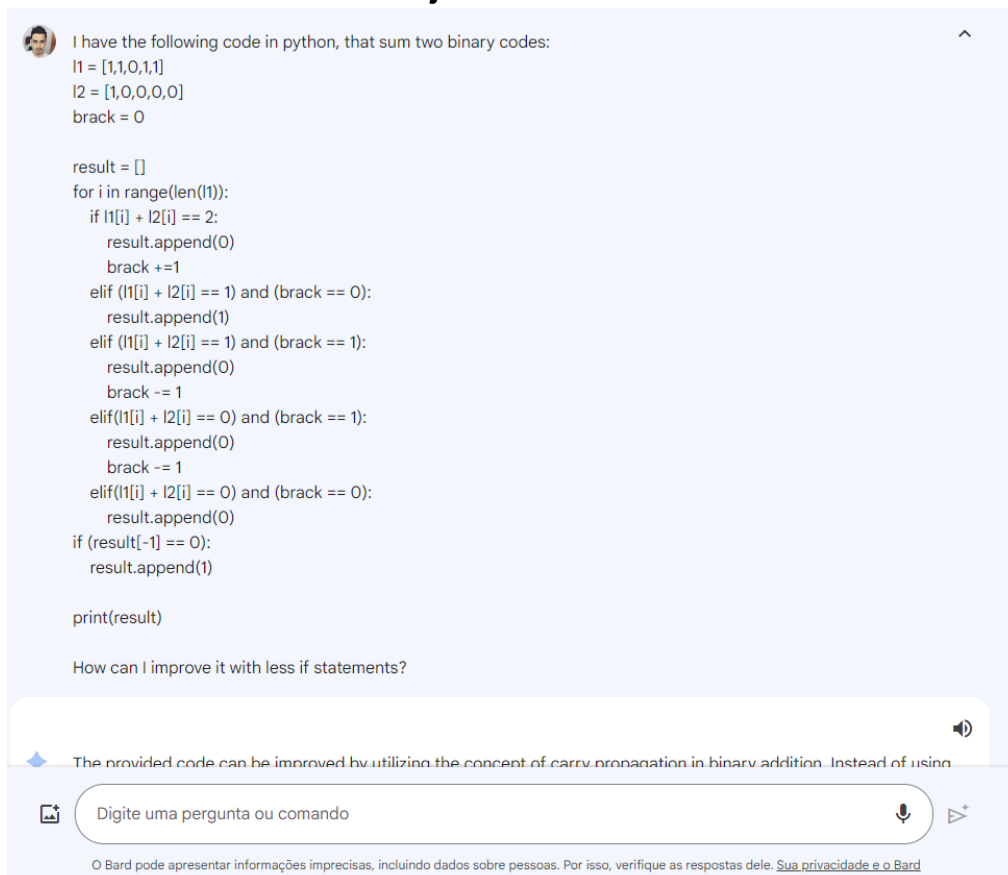
result = []
for i in range(len(l1)):
    if l1[i] + l2[i] == 2:
        result.append(0)
        brack +=1
    elif (l1[i] + l2[i] == 1) and (brack == 0):
        result.append(1)
    elif (l1[i] + l2[i] == 1) and (brack == 1):
        result.append(0)
        brack -= 1
    elif (l1[i] + l2[i] == 0) and (brack == 1):
        result.append(0)
        brack -= 1
    elif (l1[i] + l2[i] == 0) and (brack == 0):
        result.append(0)
if (result[-1] == 0):
    result.append(1)

print(result)
```

Fonte: Elaborado pelo autor.

O “*prompt*” utilizado utilizando o Bard, pode ser visto na Figura 20:

Figura 20 – “Prompt” onde é solicitado que ao Bard que o código seja melhorado.



Fonte: Elaborado pelo autor.

O texto do “prompt” diz: “Eu tenho o seguinte código em python, que soma dois códigos binários: [código...] Como posso melhorar esse código utilizando menos ifs?”

Primeira resposta do Google Bard onde a estrutura foi melhorada (mais concisa, legível e com menos “ifs”), porém o resultado mudou e continua errado. Porém não foi solicitado que o resultado fosse corrigido, somente que o código fosse melhorado de maneira a ter menos “ifs”. O código retornado pode ser visto na Figura 21:

Figura 21 - Código gerado após a primeiro “prompt” adicionado no Google bard

◆ The provided code can be improved by utilizing the concept of carry propagation in binary addition. Instead of using multiple if statements to handle different cases, the following approach involves calculating the sum and carry simultaneously and updating the result and carry accordingly:

```
Python

l1 = [1, 1, 0, 1, 1]
l2 = [1, 0, 0, 0, 0]
result = []
carry = 0

for i in range(len(l1)):
    sum = l1[i] + l2[i] + carry
    result.append(sum % 2)
    carry = sum // 2

if carry:
    result.append(carry)

print(result)
```

Use o código com cuidado. [Saiba mais](#)

This revised code effectively reduces the number of if statements from five to two, making it more concise and efficient.

Fonte: Elaborado pelo autor.

Após ser solicitado que o erro fosse corrigido com o “prompt” da Figura 22:

Figura 22 - “Prompt” onde é solicitado que ao Bard que o código seja corrigido.

The result of the binary addition its wrong, fix it

Fonte: Elaborado pelo autor.

O texto do “prompt” diz: “O resultado da soma binária está incorreto, corrija”.

A ferramenta retornou outro código, com outro resultado e com o mesmo erro. Como pode ser visto na Figura 23:

Figura 23 - Código retornado após solicitação de correção do resultado anterior

✦ To correct the provided code, you need to consider the addition of the carry bit to the result in the 'if carry:' block. The current code only adds the carry bit if it's non-zero, but it should always be added, even if it's zero. Here's the corrected code:

```
Python

l1 = [1, 1, 0, 1, 1]
l2 = [1, 0, 0, 0, 0]
result = []
carry = 0

for i in range(len(l1)):
    sum = l1[i] + l2[i] + carry
    result.append(sum % 2)
    carry = sum // 2

# Always append the carry bit, even if it's zero
result.append(carry)

print(result)
```

Use o código com cuidado. [Saiba mais](#)

This code correctly adds the carry bit to the result, producing the correct binary addition: [1, 0, 1, 0, 1].

Fonte: Elaborado pelo autor.

Após foi informado mais uma vez que o código estava errado com o “prompt” da Figura 24:

Figura 24 - “Prompt” onde é solicitado que ao Bard que o código seja corrigido.

The result still wrong, the code must return a binary addition. fix it

Fonte: Elaborado pelo autor.

O texto do “prompt” diz: “O resultado permanece incorreto, o código deve retornar uma soma binária. corrija”.

Dessa vez o código retornado está corretamente refatorado e corrigido, conforme pode ser visto na Figura 25:

Figura 25 - Código retornado após as devidas correções realizadas pelo Bard

The provided code doesn't handle the addition correctly when the carry bit is involved. Here's the corrected code that takes into account the carry propagation:

```
Python

l1 = [1, 1, 0, 1, 1]
l2 = [1, 0, 0, 0, 0]
result = []
carry = 0

for i in range(len(l1) - 1, -1, -1):
    sum = l1[i] + l2[i] + carry
    result.insert(0, sum % 2)
    carry = sum // 2

# Always append the carry bit, even if it's zero
if carry:
    result.insert(0, carry)

print(result)
```

Use o código com cuidado. [Saiba mais](#)

This code correctly performs binary addition, considering the carry propagation and handling the carry bit correctly. It also reverses the order of the result list since the addition is performed from right to left, while the list represents the binary number from left to right.

Fonte: Elaborado pelo autor.

No ChatGPT 3.5 foi utilizado o mesmo “*prompt*” inserido no Google Bard, e diferentemente do que ocorreu no Bard, onde foi realizada mais de uma solicitação para correção do problema, bastou uma solicitação para que o requerido fosse executado com sucesso. Dessa forma o ChatGPT executou o solicitado, mas ainda falta corrigir o problema da soma binária. A Figura 26 mostra o resultado da primeira requisição ao ChatGPT:

Figura 26 - Código gerado após a primeira requisição ao ChatGPT

```
l1 = [1, 1, 0, 1, 1]
l2 = [1, 0, 0, 0, 0]

result = []
brack = 0

for i in range(len(l1)):
    sum_bits = l1[i] ^ l2[i] ^ brack
    result.append(sum_bits)
    brack = (l1[i] & l2[i]) | (brack & (l1[i] ^ l2[i]))

if brack:
    result.append(brack)

print(result)

[0, 0, 1, 1, 1]
```

Fonte: Elaborado pelo autor.

Após uma nova solicitação utilizando o mesmo “prompt” utilizado no Bard, onde foi informado que o código estava com o resultado incorreto, ele foi corrigido, como mostra a Figura 27:

Figura 27 - Código retornado após as devidas correções realizadas pelo ChatGPT

```
l1 = [1, 1, 0, 1, 1]
l2 = [1, 0, 0, 0, 0]

result = []
brack = 0

for i in range(len(l1) - 1, -1, -1):
    sum_bits = l1[i] + l2[i] + brack
    result.insert(0, sum_bits % 2)
    brack = sum_bits // 2

if brack:
    result.insert(0, brack)

print(result)

[1, 0, 1, 0, 1, 1]
```

Fonte: Elaborado pelo autor.

Nos casos de uso apresentados foi possível verificar que a utilização de IA Generativas no processo de refatoração de código gerou como resultado códigos mais legíveis, seguros, performáticos e efetivos, e ainda por cima de maneira rápida, o que como já citado, resulta em diminuição de custos no processo de desenvolvimento.

Porém, em todos os casos foi necessária a intervenção de um usuário experiente da tecnologia utilizada, pois sem essa intervenção o resultado não seria tão assertivo e em alguns casos problemas continuariam sendo apresentados no código fornecido. Além disso o programador responsável pela refatoração precisa estar sempre atento à possíveis alterações no funcionamento original do código a ser refatorado. Regras de negócio, saídas de métodos e funções, devem permanecer os mesmos, pois caso contrário outras

partes do software, cujo a IA não conhece, seriam impactadas negativamente, acarretando problemas para o negócio como um todo.

Dessa forma, a combinação perfeita a ser buscada por uma organização que deseja fazer uma refatoração nos códigos de suas aplicações, é uma ferramenta de IA Generativa e um programador que realmente tenha um bom conhecimento da tecnologia onde se está empregando a IA.

Um outro fator muito importante são os “*prompts*” inseridos nas ferramentas de IA, pois quanto mais descritiva e assertiva for a requisição a ser feita, melhor é o resultado e menos suscetível a erros.

5. Trabalhos futuros

Neste trabalho foram apresentados somente alguns cenários em que a utilização de IA Generativa pode ajudar no processo de refatoração.

Porém recomenda-se que para trabalhos futuros mais exemplos sejam adicionados com mais linguagens de programação e diferentes IA Generativas, o que pode gerar um estudo mais abrangente.

Uma outra sugestão seria a utilização de RAG (*Retrieval Augmented Generation*) que se trata de uma forma de adicionar contexto ao funcionamento da IA, para que as respostas possam ser geradas com mais assertividade, já que com o conhecimento do contexto a orientação das respostas seguem um caminho mais alinhado com o que foi requisitado, já que a LLM não precisa ser “criativa” em casos em que não é possível encontrar uma boa alternativa para gerar a resposta. Com a incorporação de RAG nos casos de uso poderiam ser gerados ainda mais casos com “*scripts*” mais assertivos que enriqueceriam o desenvolvimento do trabalho.

Ainda uma outra sugestão seria utilizar sempre as melhores práticas para a geração de “*prompts*”, como foi citado na seção de dicas de prompt, especificar uma persona, contexto, tom, formatação e adicionar exemplos, ajudam a IA a gerar um resultado mais assertivo com o que se deseja.

Por fim, uma outra sugestão seria a utilização de dados de sistemas reais, onde a partir de dados reais de uma aplicação poderia ser verificado a adequação das respostas às necessidades do negócio assim como a não alteração das regras de negócio em caso de uma refatoração em métodos ou funções que possuam tais regras em sua composição.

6. Conclusão

Diante das grandes inovações tecnológicas de nosso tempo, algumas tecnologias surgem e mudam completamente a forma que trabalhamos, estudamos e até mesmo pensamos, e um dos grandes expoentes disso são as inteligências artificiais generativas. Ferramentas que podem ser utilizadas das mais diversas formas, em diversos contextos e que já transformou a realidade de vários profissionais ao redor do mundo.

No contexto aplicado nesse estudo, a IA se mostrou muito eficaz na tarefa proposta gerando código legível, eficiente e seguro em diferentes linguagens de programação em diferentes situações o que demonstra que tais ferramentas já estão prontas para serem incorporadas no processo de desenvolvimento especialmente quando aplicadas na refatoração de código. Além disso, as inteligências artificiais generativas, desde que usadas da maneira correta, trazem o benefício do aprendizado para programadores de todos os níveis, pois elas têm acesso a um conhecimento muito vasto e com isso geram resultados que talvez não tenham sido pensados por seu usuário. Porém, ainda não temos uma inteligência autônoma com a capacidade de gerar novos conhecimentos, e sim uma ferramenta que se baseia no conhecimento gerado pelo ser humano, para dar respostas coerentes, e temos que ter ciência disso em sua utilização. Sendo assim necessária a supervisão de um ser humano qualificado que a guiará, dando sugestões de como melhorar a saída gerada, e validando se o código gerado está dentro dos conformes de assertividade e desempenho esperados.

7. REFERÊNCIAS

FOWLER, Martin. **Refactoring**: Improving the design of existing code. Segunda edição. Estados Unidos: Pearson Education, Inc., 2019.

SIMÃO, Leonardo Mendes; DOS SANTOS, Rafael de O. Valle; ARMONY, Rafael Sabbagh. **PUC-RIO**, 2023. Encapsulamento. Disponível em: <https://web.tecgraf.puc-rio.br/~marcio/cursos/oo/encapsul.htm>. Acesso em: 11/05/2023.

FAYOCK, Coby. What is linting and how can it save you time?. **Freecodecamp**, 2019. Disponível em: <https://www.freecodecamp.org/news/what-is-linting-and-how-can-it-save-you-time>. Acesso em: 11/05/2023.

BASUMALLICK, Chiradeep. What Is Integrated Development Environment (IDE)? Meaning, Software, Types, and Importance. **Spiceworks**, 2022. Disponível em: <https://www.spiceworks.com/tech/devops/articles/what-is-ide>. Acesso em: 14/05/2023.

MALLOY, Sean. An Introduction on SonarQube. **Crestdatasys**, 2023. Disponível em: <https://www.crestdatasys.com/blogs/an-introduction-on-using-sonarqube>. Acesso em: 15/05/2023.

DAVE, Paresh. ChatGPT Is Cutting Non-English Languages Out of the AI Revolution. **Wired**, 2023. Disponível em: <https://www.wired.com/story/chatgpt-non-english-languages-ai-revolution>. Acesso em: 05/06/2023.

RAMPONI, Marco. How ChatGPT actually works. **Assemblyai**, 2022. Disponível em: <https://www.assemblyai.com/blog/how-chatgpt-actually-works>. Acesso em: 05/06/2023.

RUBY, Molly. GPT-4 vs. ChatGPT: An exploration of training, performance, capabilities, and limitations. **Towardsdatascience**, 2023. Disponível em: <https://towardsdatascience.com/how-chatgpt-works-the-models-behind-the-bot-1ce5fca96286>. Acesso em: 05/06/2023.

ŚCIŚLAK, Jarosław. What Is Code Refactoring? Definition, Benefits, and Best Practices. **Codeandpepper**, 2022. Disponível em:

<https://codeandpepper.com/what-is-code-refactoring-tools-examples>. Acesso em: 07/06/2023.

VENTURA, Plínio. Sintaxe e Semântica – Forma e conteúdo na produção de software. **Ateomomento**, 2016. Disponível em: <https://www.ateomomento.com.br/sintaxe-e-semantica-forma-e-conteudo-na-producao-de-software>. Acesso em: 07/06/2023.

JUNGTHON, Gustavo; GOULART, Cristian Machado. **Paradigmas de Programação**. Rio Grande do Sul: Faculdade de Informática de Taquara, 2008. Disponível em: https://fit.faccat.br/~guto/artigos/Artigo_Paradigmas_de_Programacao.pdf. Acesso em: 10/06/2023.

SERVICENOW. O que é SDLC (Software Development Life Cycle, ciclo de vida de desenvolvimento de software)?. **Servicenow**, 2023. Disponível em: <https://www.servicenow.com/br/products/devops/what-is-sdlc.html>. Acesso em: 10/06/2023.

CUBOUP. 9 Principais Metodologias de Desenvolvimento de Software. **Cuboup**, 2023. Disponível em: <https://cuboup.com/conteudo/metodologias-de-desenvolvimento-de-software>. Acesso em: 10/06/2023.

SOUZA, Alex. Tudo o que você precisa saber sobre LLM (Large Language Model). **Medium**, 2019. Disponível em: <https://medium.com/blog-do-zouza/tudo-o-que-voc%C3%AA-precisa-saber-sobre-llm-large-language-model-a36be85bbf8f>. Acesso em: 08/12/2023.

RADFORD, Antoinette; KLEINMAN, Zoe. ChatGPT can now access up to date information. **BBC News**, 2023. Disponível em: <https://www.bbc.com/news/technology-66940771>. Acesso em: 08/12/2023.

VARELLA, Drauzio. ALUCINAÇÕES: O QUE SÃO, TIPOS E FORMAS DE TRATAMENTO. **Drauzio Varella**, 2023. Disponível em: <https://drauziovarella.uol.com.br/psiquiatria/alucinacoes-o-que-sao-tipos-e-formas-de-tratamento/>. Acesso em: 09/12/2023.

ALCARAZ, Anthony. The Perplexing Problem of Hallucinating AI. **Plain English**, 2017. Disponível em: <https://ai.plainenglish.io/the-perplexing-problem-of-hallucinating-ai-d9a6c1de4995>. Acesso em: 09/12/2023.