Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

# Geração automática das mensagens de commit utilizando o GPT-4

**Thiago de Oliveira Abreu**

JUIZ DE FORA
SETEMBRO, 2024

# Geração automática das mensagens de commit utilizando o GPT-4

Thiago de Oliveira Abreu

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Gleiph Ghiotto Lima de Menezes

JUIZ DE FORA

SETEMBRO, 2024

# Geração automática das mensagens de commit utilizando o GPT-4

Thiago de Oliveira Abreu

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Gleiph Ghiotto Lima de Menezes
Doutor em Computação

André Luiz de Oliveira
Doutor em Ciência da Computação

Heder Soares Bernardino
Doutor em Modelagem Computacional

JUIZ DE FORA

27 DE SETEMBRO, 2024

*Aos meus amigos, pela companhia.*

*Ao meu irmão, pela motivação.*

*Aos meus pais, pelo apoio e carinho.*

# Resumo

As mensagens de *commit* são resumos de mudanças nos artefatos de *software* importantes para o entendimento e manutenibilidade do código. Entretanto, escrevê-las demanda tempo e uma boa elaboração na escolha de palavras que consigam descrever com precisão o motivo da modificação. Diversas abordagens já propuseram maneiras de gerar automaticamente as mensagens de *commit*. De maneira geral, as propostas utilizam inteligência artificial em conjunto com técnicas de processamento de linguagem natural. Também utilizam dados dos repositórios públicos do GitHub para estabelecer um conjunto de dados de treinamento para a inteligência artificial. Porém, a maioria das abordagens conta apenas com dados retirados de repositórios Java, tirando de foco outras linguagens de programação altamente utilizadas atualmente. Além disso, existem modelos de linguagem de grande porte capazes de serem explorados para a tarefa de geração de mensagens de *commit*, como o GPT-4. Portanto, neste trabalho avaliou-se o GPT-4 na geração automática das mensagens de *commit* e buscou-se avaliar como a inserção do histórico de mensagens de *commit* impacta a qualidade dos resultados. Como base de comparação, foi feita uma busca na literatura pelo modelo de geração de mensagens de *commit* com o melhor resultado e que fosse reprodutível, sendo o CoRec o modelo escolhido. A base de dados utilizada é composta por projetos escritos em JavaScript, uma linguagem de programação altamente utilizada. Os resultados mostraram que o GPT-4 (B-Norm de 12,63%) teve um desempenho superior ao modelo CoRec (B-Norm de 11,54%) e que a inclusão do histórico de mensagens de *commit* aumentou a eficácia do GPT-4, alcançando um B-Norm de 15%.

**Palavras-chave:** *Commit*, Geração automática de mensagens, GPT, Aprendizado de máquina, Rede neural recorrente.

# Abstract

Commit messages are summaries of changes in software artifacts crucial for understanding and maintaining code. However, writing them requires time and careful word selection to accurately describe the modification's purpose. Various approaches have proposed methods for automatically generating commit messages. Generally, these approaches use artificial intelligence combined with natural language processing techniques. They also leverage data from public GitHub repositories to establish a training dataset for AI. However, most approaches focus solely on data from Java repositories, ignoring other widely used programming languages. Moreover, large language models like GPT-4 can also be explored for commit message generation. Therefore, this work evaluates GPT-4 for automatic commit message generation and examines how incorporating commit message history affects the quality of the results. As a comparison, a search was conducted in the literature for the reproducible commit message generation model with the best result, leading to the selection of CoRec. The dataset used comprises projects written in JavaScript, a widely used programming language. The results showed that GPT-4 (B-Norm of 12.63%) outperformed the CoRec model (B-Norm of 11.54%) and that including commit message history enhanced GPT-4's effectiveness, achieving a B-Norm of 15%.

**Keywords:** Commit, Automatic generation of message, GPT, Machine learning, Recurrent neural network.

# Agradecimentos

Quero agradecer ao meu pai, à minha mãe e ao meu irmão, que me deram apoio, motivação, carinho e se fizeram presente em todos os momentos bons ou difíceis.

Também ao professor Gleiph Ghiotto pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Assim como aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

*"Without consistency, you never finish.*

*So do what you feel passionate about".*

*Denzel Washington*

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Lista de Abreviações

| | |
|---|---|
| DCC | Departamento de Ciência da Computação |
| UFJF | Universidade Federal de Juiz de Fora |
| PLN | Processamento de Linguagem Natural |
| ML | *Machine Learning* |
| BERT | *Bidirectional Encoder Representations from Transformers* |
| GLUE | *General Language Understanding Evaluation* |
| IDE | *Integrated Development Environment* |
| API | *Application Programming Interface* |
| URL | *Uniform Resource Locator* |
| WEB | *World Wide Web* |
| HTML | *Hyper Text Markup Language* |
| AST | *Abstract Syntax Tree* |
| SCV | Sistema de Controle de Versões |
| LLM | *Large Language Model* |

# 1 Introdução

As mensagens de *commit* são descrições de mudanças realizadas no código-fonte de um software, fundamentais para o entendimento das modificações sem a necessidade de leitura do código completo. Escrever uma mensagem de *commit* que resuma com precisão o propósito de uma alteração é uma prática comum que precede a submissão dessas mudanças para um Sistema de Controle de Versão (SCV). No entanto, assim como outras formas de documentação, as mensagens de *commit* são frequentemente prejudicadas pela pressão de tempo e pelas demandas do mercado de trabalho (ROEHM et al., 2012).

Estudos anteriores, como o de Maalej e Happel (2010), demonstraram que uma porcentagem significativa de mensagens de *commit* é inadequada, seja por serem vazias, excessivamente curtas ou sem sentido semântico. Diante disso, várias abordagens têm sido propostas para automatizar a geração dessas mensagens, visando melhorar a sua qualidade e reduzir o esforço dos desenvolvedores.

Visando automatizar a geração das mensagens de *commit*, Buse e Weimer (2010) e Linares-Vásquez et al. (2015) se baseiam em mapear o que fora modificado no código para uma mensagem descritiva através da criação de regras de transformação de texto. Liu et al. (2018b) e Huang et al. (2020) partem do pressuposto de que mudanças semelhantes possuem mensagens semelhantes e, portanto, assumem que as mudanças no código podem se apropriar da mensagem utilizada para outra mudança com características semelhantes. Entretanto, essas abordagens são, de maneira geral, efetivas produzir mensagens que resumem o que mudou no código-fonte, mas não o propósito da mudança.

Para se obter o "porquê"da mudança, é necessário um raciocínio humano capaz de julgar com base em dados complexos e contextos distintos. A hipótese da "naturalidade"do *software* pressupõe que, uma vez que criado por esforço humano, o *software* possui características repetitivas que, assim como a linguagem natural, podem ser previstas por modelos estatísticos. Isso possibilita que modelos de aprendizado de máquina possam ser utilizados para a tarefa de tradução, assumindo o código modificado como a linguagem de origem e a mensagem de *commit* como a linguagem de destino (HINDLE

et al., 2016).

Motivados pela "naturalidade"do *software*, Jiang, Armaly e McMillan (2017) utilizam a abordagem baseada na geração das mensagens, que se apoia no aprendizado de máquina, para gerar as mensagens de *commit* com base nas modificações realizadas no código-fonte, *i.e.*, os *diffs*. Entretanto, as abordagens baseadas em geração normalmente sofrem com a falta de semântica das mensagens geradas e, de acordo com Wang et al. (2021), transferir a semântica do código-fonte para a linguagem natural está no centro dos diversos desafios do aprendizado de máquina.

Neste trabalho, motivados pelo surgimento dos modelos de linguagem de grande porte, em especial, o GPT-4, acredita-se o modelo pode ser utilizado para na tarefa de geração automática das mensagens de *commit*. Sua alta capacidade de abstração de informação e contexto leva à hipótese de que o fornecimento de contexto, como o histórico das mensagens de *commit* anteriores, pode elevar ainda mais a qualidade das mensagens geradas. Desta forma, busca-se responder neste trabalho as seguintes questões de pesquisa:

- QP1: Como o fornecimento de histórico de mensagens de *commit* afeta a geração das mensagens de *commit* pelo GPT-4?
- QP2: Qual estratégia de prompt aplicada ao GPT-4 produz o melhor resultado?
- QP3: A estratégia com melhor resultado supera o modelo CoRec quanto a pontuação B-Norm?

## 1.1 Objetivos

O objetivo principal desta pesquisa é avaliar, através da métrica B-Norm, a geração automática das mensagens de *commit* utilizando o modelo GPT-4 quando fornecido histórico das mensagens de *commit* anteriores. A pesquisa também possui como objetivos secundários: avaliar o modelo GPT-4 na geração automática das mensagens de *commit* possuindo apenas o *diff* como informação; e a comparação dos resultados com um modelo de aprendizado de máquina conhecido já proposto para esta tarefa na literatura, o CoRec (WANG et al., 2021).

## 1.2 Principais Contribuições

Este trabalho trouxe contribuições relevantes para a tarefa de geração de mensagens de *commit* utilizando modelos de linguagem de grande porte. A avaliação do GPT-4, em comparação com o modelo CoRec, revelou que o GPT-4 foi capaz de gerar mensagens de *commit* mais precisas e contextuais, obtendo um B-Norm de 12,63%, superior ao resultado do CoRec (B-Norm de 11,54%).

Também foram desenvolvidas e testadas quatro estratégias de construção de *prompt*, incluindo abordagens com e sem histórico de mensagens. Dentre essas, a estratégia que utilizou o histórico de mensagens de *commit* do autor demonstrou o melhor desempenho, alcançando um B-Norm de 15%, evidenciando que a inclusão de contexto melhora consideravelmente os resultados na geração de mensagens com relação à estratégia que não utiliza contexto (B-Norm de 12,63%).

Este estudo contribui com a literatura ao demonstrar a viabilidade do uso do GPT-4 para a geração de mensagens de *commit*, ao mesmo tempo que sugere novas abordagens para investigações futuras, como o impacto de diferentes níveis de enriquecimento de *commits* e a comparação com outros modelos de linguagem.

## 1.3 Organização do Trabalho

Este trabalho está organizado em cinco capítulos, incluindo este de Introdução. No Capítulo 2, são introduzidos conceitos básicos sobre o que são Sistemas de Controle de Versões e como são utilizados nesta abordagem, as técnicas de processamento de linguagem natural, a ideia de aprendizado profundo e modelos de tradução de máquina, os mecanismos de atenção e os modelos de linguagem de grande porte e, por fim, os métodos de avaliação de modelos de tradução de máquina. No Capítulo 3 é apresentado o mapeamento sistemático da literatura e os trabalhos relacionados com o tema de geração automática das mensagens de *commit*. No Capítulo 4 são descritas as etapas de definição, planejamento, coleta de dados, execução e análise dos resultados do estudo experimental conduzido neste trabalho para avaliar a efetividade do GPT-4 em gerar as mensagens de *commit*. Por fim, no Capítulo 5 é feita a conclusão deste trabalho.

# 2 Fundamentação Teórica

Neste capítulo, são apresentados os conceitos necessários para entender as contribuições apresentadas deste trabalho. Na Seção 2.1 são apresentados os conceitos de sistemas de controle de versão. Na Seção 2.2 é abordado o conceito do processamento de linguagem natural. Na Seção 2.3 e 2.4 são abordados os conceitos acerca do tema da inteligência artificial, abrangendo aprendizado profundo, modelos de tradução de máquina, mecanismos de atenção e modelos de linguagem de grande porte. Por fim, na Seção 2.5 são apresentados os métodos de avaliação de modelos de tradução de máquina.

## 2.1 Sistemas de Controle de Versões

Os Sistemas de Controle de Versão (SCV) são fundamentais para acompanhar a evolução dos artefatos de *software*, como o código-fonte e a documentação (ZOLKIFLI; NGAH; DERAMAN, 2018). Por exemplo, os *commits* são registros individuais que documentam alterações realizadas em um repositório. Cada *commit* representa uma unidade de mudança, incluindo adição, remoção ou modificação de arquivos. Essas mudanças podem ser identificadas pelo mecanismo de *diff* que pode identificar as regiões dos artefatos que foram alteradas entre dois *commits*. Esses e outros conceitos possibilitam que uma pessoa ou um *software* possa identificar como os artefatos evoluíram ao longo do tempo.

De acordo com Zolkifli, Ngah e Deraman (2018), um SCV é responsável por gerenciar o desenvolvimento de um determinado *software*, isto é, manter o registro de quaisquer modificações realizadas por um desenvolvedor. O autor também afirma que, à medida que o número de versões de um projeto de *software* cresce, sua complexidade aumenta e isso dificulta o gerenciamento dos artefatos do projeto. Na ausência de um SCV, os desenvolvedores precisam fazer esse gerenciamento manualmente. Esse gerenciamento pode consistir em manter cópias de arquivos do projeto em seus computadores como forma de versionamento, tornando a manutenção das versões mais complexas, o que pode levar a erros irreversíveis, como a deleções ou modificações em arquivos indevidos.

Os SCV podem ser classificados em centralizados ou distribuídos (OTTE, 2009). Nos centralizados, existe apenas um repositório localizado em um servidor que contém todo o histórico de versões dos arquivos, exigindo que os usuários estejam conectados à rede para acessá-lo. Alguns exemplos de SCV centralizados são: CVS (FOGEL; BAR, 1999), Subversion (FITZPATRICK; PILATO; COLLINS-SUSSMAN, 2004) e Perforce (WINGERD, 2005). Por outro lado, os SCV distribuídos podem, além de serem gerenciados por um repositório central, estar replicados em diversas máquinas, permitindo que cada usuário possua um repositório local e possa acessá-lo sem estar conectado à rede. Desta forma, é necessário conectar-se à rede apenas quando for preciso sincronizar as alterações com outros usuários. São exemplos de SCV distribuídos o *Git* (CHACON; STRAUB, 2014), o *Mercurial* (MERCURIAL, 2023), o *Bazaar* (LTD., 2009) e o *BitKeeper* (BITKEEPER, 2023). Devido ao escopo deste trabalho, o restante desta seção mantém o enfoque apenas em SCV distribuídos e os exemplos são dados utilizando os comandos do Git.

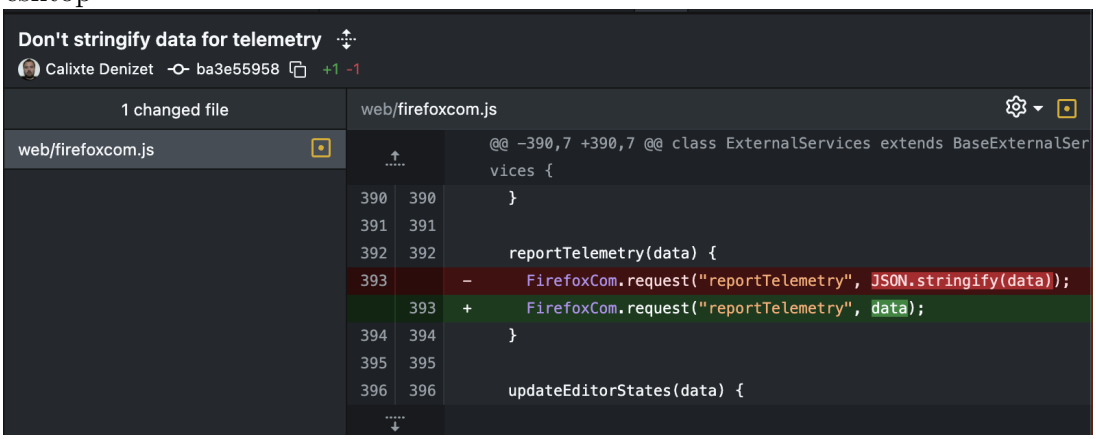
Após realizar as modificações desejadas no seu projeto, o desenvolvedor salva as modificações através do comando *git commit*, fornecendo uma descrição para as modificações, *i.e.* mensagem de *commit*, e armazena no repositório uma nova versão contendo informações do autor e a mensagem de *commit*. Na Figura 2.1 é ilustrado um *commit* do arquivo *firefoxcom.js* do repositório *mozilla*[1]. Na figura é possível identificar que o desenvolvedor *Calixte Denizet* alterou o arquivo *firefoxcom.js* colocando a mensagem "Don't stringify data for telemetry". Durante a alteração do arquivo, o autor alterou a linha 393, substituindo o conteúdo destacado em vermelho "JSON.stringify(data)"pelo conteúdo destacado em verde "data".

Os SCV permitem que o usuário liste as versões de um projeto e navegue entre elas. Para listar as versões em um repositório, o comando *git log* é utilizado. Esse comando permite visualizar os *commits* e as informações associadas a eles. A Listagem 2.1 ilustra parte da saída do comando *git log* referentes aos três últimos *commits* no repositório do projeto *pdf.js* mencionado anteriormente.

Os SCV são essenciais para a gestão de projetos de software, destacando-se pelos

[1]https://github.com/mozilla/pdf.js

Figura 2.1: Visualização de um *diff* do repositório "mozilla"através do aplicativo GitHub Desktop




Fonte: Elaborado pelo autor (2024).

recursos de mensagens de *commit* e *diffs*. A inclusão de informações do autor e mensagens de *commit* fornece uma narrativa sobre as alterações realizadas, possibilitando a análise retrospectiva de projetos. Ao adotar boas práticas, como o fornecimento de uma boa mensagem de *commit*, os desenvolvedores garantem transparência, colaboração eficiente e manutenção eficaz de projetos de software (REVELO, 2023).

Listagem 2.1: Saída do comando *git log* no repositório mozilla/pdf.js utilizado no experimento deste trabalho.

```
commit 6bb6ce6a5d133c5aa318e2fff9d1df5326a1ea96 (HEAD -> master, origin/
    master, origin/HEAD)
Merge: 9ee4c6528 6d0835dc5
Author: calixteman <calixte.denizet@gmail.com>
Date:   Wed Mar 6 14:39:58 2024 +0100

    Merge pull request #17767 from calixteman/automation_win_listener

    In the m-c automation, give the possibility to remove window
        listeners when a test ended

commit 6d0835dc5246c3ac64ef7cc5dbdfbafc173fc140
Author: Calixte Denizet <calixte.denizet@gmail.com>
Date:   Mon Mar 4 16:47:58 2024 +0100

    In the m-c automation, give the possibility to remove window
        listeners when a test ended

    The goal is to avoid to have some exceptions in the logs when a test
         finished.

commit 9ee4c6528d0f2f849803f7d73f950a2ecfcdd0d0
Merge: f634cb533 4e1b96c78
Author: calixteman <calixte.denizet@gmail.com>
Date:   Wed Mar 6 10:53:11 2024 +0100

    Merge pull request #17776 from calixteman/bug1883609

    [Annotations] Widget annotations must be in front of the other ones
        (bug 1883609)
```

## 2.2 Processamento de Linguagem Natural

De acordo com Jiang e Lu (2020), o Processamento de Linguagem Natural (PLN) representa uma vertente importante no campo do Aprendizado de Máquina. Sua finalidade é facilitar uma comunicação eficiente entre seres humanos e computadores, proporcionando ferramentas que desempenham um papel crucial nessa interação (BRANTS, 2003). Um exemplo notável é a tokenização, que estabelece uma representação numérica das palavras para que possam ser processadas pelo computador.

A tradução de máquina, reconhecimento sonoro e a recuperação de informação são possíveis aplicações em PLN. Sun, Luo e Chen (2017) descrevem algumas técnicas de PLN que são encontradas nas abordagens de geração automática de mensagens de *commit*. Alguns exemplos são: a tokenização, a marcação POS (Part-of-speech) e o *parsing*.

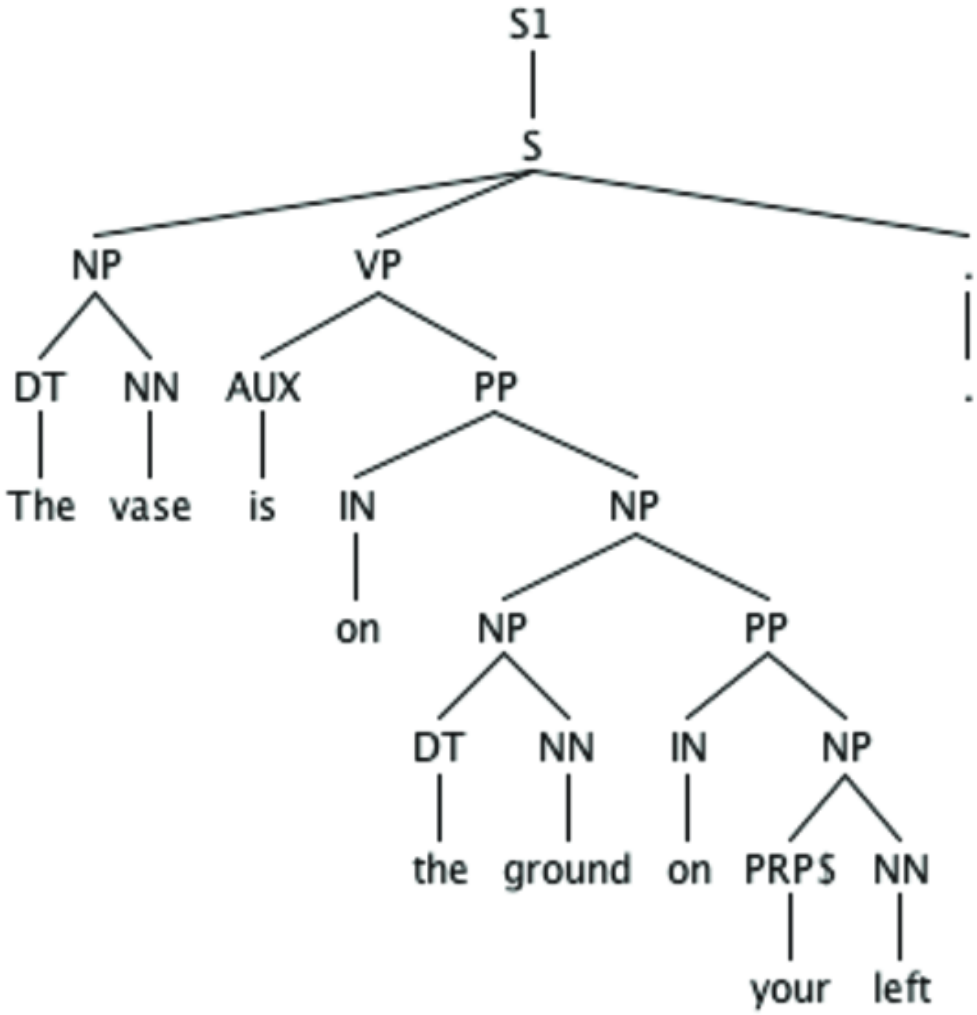
A **tokenização** é o processo de separar as palavras de um documento utilizando, normalmente, o espaço vazio como separador. Na tokenização também são removidas algumas palavras consideradas pouco significantes, chamadas de *stopwords*, que podem ser palavras curtas, com pouco significado ou com uma frequência muito alta (*e.g.*, "*the*"e "*a*"). A tokenização é uma técnica fundamental em PLN e, por isso, diversas ferramentas a implementa, por exemplo, *Stanford Tokenizer* (STANFORD, 2005) e *OpenNLP Tokenizer* (COMMUNITY, 2011).

A **marcação POS** é uma técnica do PLN que identifica e rotula cada palavra em um texto com sua classe gramatical. As classes gramaticais, também conhecidas como partes do discurso, definem a função que uma palavra assume na frase, como substantivo, verbo, adjetivo, etc.

O ***parsing*** produz uma árvore que representa a estrutura gramatical de uma frase, indicando as relações entre suas palavras. Em comparação com a marcação POS, o *parsing* fornece informações estruturais mais detalhadas. A Figura 2.2 ilustra uma árvore de análise para a frase "The vase is on the ground on your left"com as respectivas classes gramaticais em letra maiúscula, sendo elas: S1 e S (Oração), NP (Frase Nominal), VP (Frase Verbal), DT (Artigo Definido), NN (Substantivo), AUX (Verbo Auxiliar), PP (Frase Preposicional), IN (Preposição) e PRPS (Pronome Possessivo).

O processamento de linguagem natural é utilizado na etapa de pré-processamento

Figura 2.2: Árvore de análise da frase *"The vase is on the ground on your left"* com a classificação gramatical e relações de seus constituintes




Fonte: Kordjamshidi (2013).

dos dados. Isto ocorre porque é natural que a base de dados esteja repleta de informações irrelevantes. No contexto das mensagens de *commit*, a base pode conter mensagens muito pequenas ou muito extensas. Desta forma, o pré-processamento auxilia na filtragem dos dados irrelevantes e remoção de irregularidades para se obter uma melhor qualidade do resultado como em Jiang, Armaly e McMillan (2017), em que os autores utilizaram a ferramenta *Stanford CoreNLP* para filtrar as mensagens de *commit* pelo padrão "verbo + objeto direto", um padrão presente na maior parte das mensagens da base de dados analisada pelos autores.

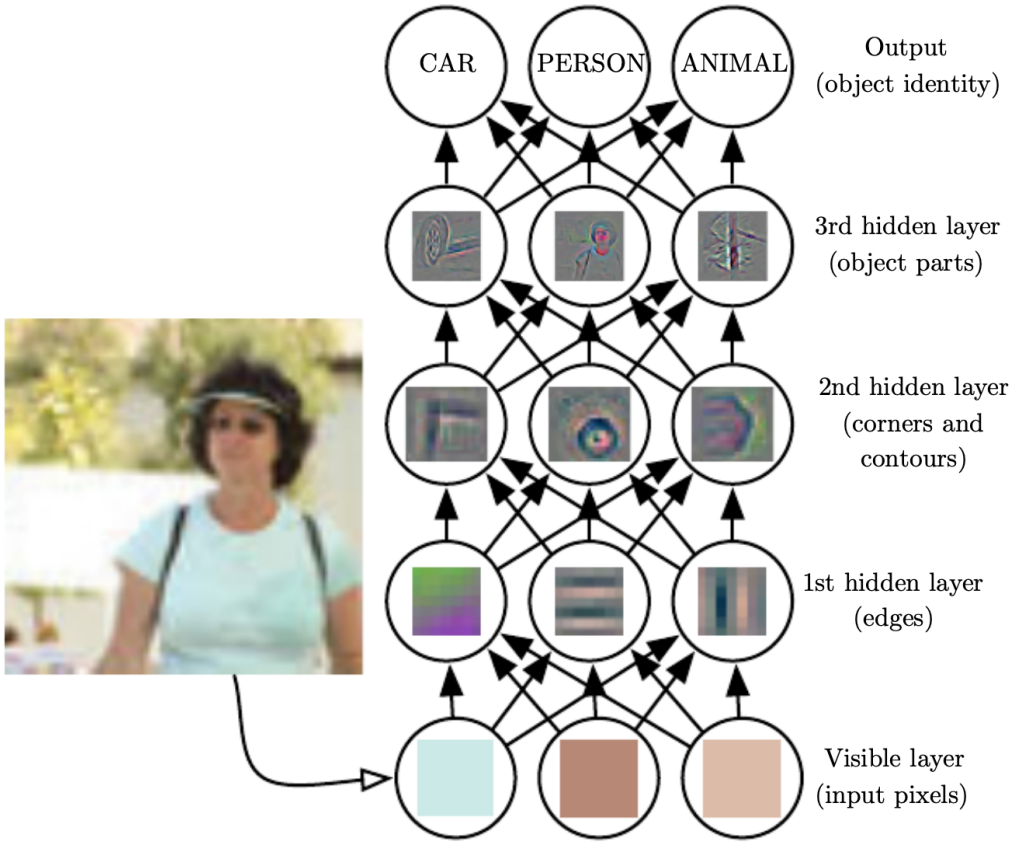
## 2.3 Aprendizado Profundo e Modelos de Tradução de Máquina

O aprendizado profundo é um subcampo da inteligência artificial (IA) e do aprendizado de máquina que se concentra no uso de redes neurais artificiais com múltiplas camadas para modelar padrões complexos e hierárquicos em dados. A principal característica que distingue o aprendizado profundo de outros métodos de aprendizado de máquina é a profundidade das redes neurais, ou seja, a quantidade de camadas ocultas entre a camada de entrada e a camada de saída (GOODFELLOW; BENGIO; COURVILLE, 2016).

Cada camada em uma rede neural profunda aprende uma representação dos dados de entrada de maneira progressiva, extraindo características de níveis mais baixos em camadas iniciais e características mais abstratas em camadas mais profundas. Isso permite que as redes profundas capturem relações complexas e não lineares entre os dados (GOODFELLOW; BENGIO; COURVILLE, 2016). A Figura 2.3 ilustra esse processo: na camada de entrada, a rede neural recebe os *pixels* de uma fotografia; nas camadas mais profundas ela captura bordas, contornos e partes de objetos; e, finalmente, na camada de saída, a rede identifica o objeto presente na imagem.

Nos métodos de aprendizado de máquina convencionais, definir quais características devem ser extraídas de uma base de dados e como extraí-las exige cautela e experiência no domínio do problema. Por outro lado, no aprendizado profundo isso deixa de ser uma preocupação, pois a máquina pode detectar e extrair essas características (LECUN; BEN-

Figura 2.3: Ilustração da hierarquia de conceitos em um modelo de aprendizado profundo




Fonte: Goodfellow, Bengio e Courville (2016).

GIO; HINTON, 2015). As redes neurais recorrentes e as redes neurais convolucionais são exemplos de redes utilizadas no aprendizado profundo.

As redes neurais recorrentes são comumente utilizadas nos casos em que os dados de entrada são sequenciais, como áudios e textos. Essas redes processam os elementos da entrada e mantêm um vetor de estado que contém o histórico de todos os elementos passados. Isso permite que a rede seja treinada para realizar atividades, sejam elas mais simples, como prever o próximo caractere de uma frase, ou mais mais complexas, como traduzir um texto de um idioma para outro (LECUN; BENGIO; HINTON, 2015).

Por outro lado, as redes neurais convolucionais possuem uma aplicação voltada para o processamento de dados que possuem uma topologia em grade, como no reconhecimento em imagens, em que os dados podem ser vistos como uma grade de *pixels* com duas dimensões (GOODFELLOW; BENGIO; COURVILLE, 2016). Na Figura 2.3 a imagem de uma mulher é inserida no modelo de aprendizado profundo que é capaz de classificar o conteúdo da imagem, a partir da distinção das partes específicas do objeto.

Os modelos de tradução de máquina são comuns na abordagem de geração automática das mensagens de *commit* e utilizam de redes neurais recorrentes para tratar o problema. Isso ocorre porque o problema é tratado como tradução de máquina *sequence-to-sequence* como afirmam Liu et al. (2019). Um problema do tipo *sequence-to-sequence* está relacionado com a geração de uma sequência de saída a partir de uma sequência de entrada, como na tradução de máquina.

No intuito de resolver o problema da tradução de máquina, os autores Cho et al. (2014) introduziram a arquitetura *Encoder-Decoder*. A arquitetura é composta por uma rede neural denominada **encoder** e outra denominada **decoder**, responsáveis por transformar uma sequência de entrada em uma sequência de saída. A rede **encoder** codifica uma sequência de entrada na linguagem de origem em uma representação vetorial de tamanho fixo que o algoritmo pode interpretar. Já a rede **decoder** transforma o vetor codificado na sequência de saída traduzida para a linguagem de destino (BAHDANAU; CHO; BENGIO, 2014; JIANG; ARMALY; MCMILLAN, 2017).

## 2.4 Mecanismos de Atenção e Modelos de Linguagem de Grande Porte

Bahdanau, Cho e Bengio (2014) introduziram o mecanismo de atenção nas redes neurais recorrentes para lidar com longas sequências de entrada. No mecanismo de atenção, a sequência de entrada é codificada em uma sequência de vetores, isto é, cada passo da codificação é mantido no vetor. Assim, durante a etapa de decodificação, a rede **decoder** decide para quais partes da sequência de entrada deve "prestar atenção", selecionando o subconjunto dos vetores gerados pelo **encoder** que apresenta maior semelhança com o passo da decodificação. O mecanismo de atenção foi utilizado na tarefa de tradução de máquina por Sennrich et al. (2017) e, posteriormente, aproveitado na geração de mensagens de *commit* pelos autores Jiang, Armaly e McMillan (2017) e Xu et al. (2019).

Em Vaswani et al. (2017) os autores introduziram a arquitetura *Transformer*, que dispensa o uso de redes neurais recorrentes e se baseia apenas nos mecanismos de atenção. Além de diminuir a complexidade da arquitetura, o desuso das redes neurais recorrentes (sequenciais) permite paralelizar a tarefa de treinamento do modelo, aumentando o seu desempenho. O modelo também alcançou o estado da arte nas tarefas de tradução de texto, superando os resultados obtidos até o momento da publicação.

A arquitetura deu origem aos Modelos de Linguagem de Grande Porte ($LLM$) que, ao serem treinados com conjuntos massivos de dados textuais, aprendem a utilizar o contexto das palavras. Esses modelos, escalados para bilhões de parâmetros, destacam-se pela capacidade única de realizar diversas tarefas, como, por exemplo, geração de código e processamento de linguagem natural, utilizando as informações contextuais aprendidas durante o treinamento (ELISEEVA et al., 2023). Um exemplo que demonstrou grande potencial em versatilidade é o *GPT-3.5* (LIU et al., 2023).

## 2.5 Métodos de Avaliação de Modelos de Tradução de Máquina

Para avaliar os resultados de problemas de tradução de máquina, Papineni et al. (2002) desenvolveram um método chamado *BLEU*, que calcula a semelhança entre o texto gerado e o texto esperado. No contexto das mensagens de *commit*, o cálculo da semelhança é feito entre a mensagem de *commit* gerada pelo modelo de inteligência artificial e a mensagem de *commit* originalmente fornecida pelo autor. O cálculo da semelhança é realizado através da contagem da quantidade de ocorrências de grupos de *n-grams* do texto candidato (gerado pelo modelo) no texto de referência.

Os *n-grams* são grupamentos de $n$ palavras, de forma que o BLEU-1 conta a ocorrência de cada palavra no texto de referência, o BLEU-2 conta a ocorrência de duas palavras consecutivas no texto de referencia, e assim por diante. Além disso, segundo os autores, baixos valores de $n$ estão relacionados com a medição de adequação das palavras, enquanto altos valores de $n$ estão relacionados com medição da fluência da frase.

A Listagem 2.2 ilustra o caso em que a precisão é calculada para uma frase denominada candidata com várias ocorrências de uma mesma palavra *"the"* enquanto a frase denominada referência possui apenas duas ocorrências da mesma palavra. É possível observar que o contador do número de vezes em que a palavra *"the"* aparece na candidata é ajustado para o máximo de vezes que ela aparece na referência, passando de 7 para 2. O resultado então é calculado pela divisão deste valor pelo número total de palavras da candidata, ou seja, 7, resultando no valor 2/7. Maiores detalhes de como o *BLEU* é calculado para *n-grams* com valores de $n$ superiores a um não serão abordados neste trabalho por não serem relevantes para o tema (PAPINENI et al., 2002).

Listagem 2.2: Resultado da precisão da frase de referência para a frase candidata. Fonte Papineni et al. (2002)

```
1    Candidate: the the the the the the the.
2    Reference 1: The cat is on the mat.
3    Reference 2: There is a cat on the mat.
4    Modified Unigram Precision = 2/7.
```

A métrica B-Norm é uma normalização da pontuação BLEU. Ela busca ajustar a pontuação BLEU para torná-la mais comparável em diferentes condições de teste, como tamanhos de texto variados e complexidade da linguagem. A métrica foi utilizada em (LOYOLA; MARRESE-TAYLOR; MATSUO, 2017) e foi considerada a mais adequada pelos autores em (TAO et al., 2022). Portanto, apesar de existirem outras métricas de avaliação como o METEOR (BANERJEE; LAVIE, 2005) e o ROUGE (LIN, 2004), amplamente utilizadas na literatura, elas não serão consideradas neste trabalho.

# 3 Trabalhos Relacionados

Neste capítulo são apresentados os trabalhos relacionados ao tema da geração de mensagens de *commit*. Na Seção 3.1 são apresentados os trabalhos que se baseiam em regras para gerar descrições de mudanças do código. Na Seção 3.2 são apresentados os trabalhos que se baseiam em inteligência artificial para gerar mensagens de *commit*.

A identificação dos trabalhos relacionados foi realizada por meio de uma revisão sistemática da literatura. O processo foi dividido em quatro etapas: formulação da questão de pesquisa, busca na base de dados, aplicação dos critérios de exclusão, e síntese dos resultados.

A pergunta de pesquisa que guiou a revisão foi: "Quais são as abordagens existentes para a geração automática de mensagens de *commit* utilizando técnicas de aprendizado de máquina e PLN (Processamento de Linguagem Natural)?"

A busca foi realizada na base de dados **Scopus**. A *string* de busca utilizada foi: *TITLE-ABS-KEY((“commit”OR “commits”) AND (“message”OR “messages”OR “summary”OR “summaries”OR “summarize”OR “summarized”) AND (“generation”OR “generate”OR “generated”) AND (“language”OR “lang”OR “natural”OR “NLP”OR “neural”) ) AND ( LIMIT-TO ( SUBJAREA, “COMP”) ) AND ( LIMIT-TO ( LANGUAGE, “English”) OR LIMIT-TO ( LANGUAGE, “Portuguese”) ).*

A execução da busca com a chave descrita acima foi realizada no dia 22 de novembro de 2022 e o resultado totalizou 45 publicações. Do momento de execução da busca até a publicação deste trabalho, outras 23 publicações foram realizadas na base Scopus. Entretanto, estas publicações não são tratadas neste trabalho devido à limitação quanto ao tempo disponível para acompanhamento das novas publicações.

Para complementar a busca inicial e identificar outros estudos relevantes, foi utilizado o método de *snowballing*, onde as referências dos artigos incluídos foram examinadas para identificar publicações adicionais relacionadas ao tema. Três artigos controle que mais se identificam com o tema foram utilizados como base para o *snowballing* e listados na Tabela 3.2.

Tabela 3.1: Artigos selecionados a partir da metodologia de busca na base de dados Scopus.

| Autor(es) | Título |
|---|---|
| Cortes-Coy et al. (2014) | On automatically generating commit messages via summarization of source code changes |
| Jiang e McMillan (2017) | Towards Automatic Generation of Short Summaries of Commits |
| Jiang, Armaly e McMillan (2017) | Automatically generating commit messages from diffs using neural machine translation |
| Loyola, Marrese-Taylor e Matsuo (2017) | A Neural Architecture for Generating Natural Language Descriptions from Source Code Changes |
| Liu et al. (2018a) | Neural-machine-translation-based commit message generation: How far are we? |
| Jiang (2019) | Boosting Neural Commit Message Generation with Code Semantic Analysis |
| Liu et al. (2019) | Generating Commit Messages from Diffs using Pointer-Generator Network |
| Xu et al. (2019) | Commit Message Generation for Source Code Changes |
| Etemadi e Monperrus (2020) | On the relevance of cross-project learning with nearest neighbours for commit message generation |
| Nie et al. (2021) | CoreGen: Contextualized Code Representation Learning for Commit Message Generation |
| Tao et al. (2021) | On the Evaluation of Commit Message Generation Models: An Experimental Study |
| Liu et al. (2022) | ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking |
| Dey et al. (2022) | Evaluating Commit Message Generation: To BLEU Or Not To BLEU? |
| Dong et al. (2022) | FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation |
| Shi et al. (2022) | RACE: Retrieval-Augmented Commit Message Generation |
| Tao et al. (2022) | A large-scale empirical study of commit message generation: models, datasets and evaluation |

Fonte: Elaborado pelo autor (2024).

Para evitar que o resultado possua como critério apenas a semelhança sintática entre os parâmetros de busca e o conteúdo, foram utilizados critérios de exclusão de maneira a garantir que os resultados obtidos possuam forte relação com o tema em pesquisa e a área de estudos. Esses critérios de exclusão são listados a seguir:

- CE1 - As publicações que descrevem e/ou apresentam *keynote speeches*, tutoriais,

Tabela 3.2: Artigos controle selecionados como base para o *snowballing*

| Autor(es) | Título |
|---|---|
| Jiang e McMillan (2017) | Towards automatic generation of short summaries of commits |
| Jiang, Armaly e McMillan (2017) | Automatically generating commit messages from diffs using neural machine translation |
| Liu et al. (2022) | ATOM: Commit message generation based on abstract syntax tree and hybrid ranking |

Fonte: Elaborado pelo autor (2024).

cursos e *workshops*;

- CE2 - As publicações que não estão disponíveis para *download*;
- CE3 - As publicações com títulos, resumos e conteúdo que não possuem relação com o tema em questão;

Os critérios de exclusão foram aplicados sobre as 45 publicações obtidas e esse número se reduziu para 15 publicações. A Tabela 3.3 contém os artigos que foram removidos bem como o critério de exclusão utilizado na remoção de cada. Já os artigos resultantes da remoção estão listados na Tabela 3.1.

Tabela 3.3: Artigos removidos de acordo com os respectivos critérios de exclusão

| Autor(es) | Título | Critério |
|---|---|---|
| Kuang, Sun e Lin (1997) | A partial evaluator for a parallel lambda language | CE3 |
| Kultima e Paavilainen (2007) | Creativity techniques in game design | CE3 |
| Rastkar e Murphy (2013) | Why did this code change? | CE3 |
| Não consta | ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering | CE1 |
| Tasnim e Rahman (2019) | Inferring bug patterns for detecting bugs in JavaScript by analyzing abstract syntax tree | CE3 |
| Loyola et al. (2018) | Content aware source code change description generation | CE3 |
| Moran et al. (2018) | Detecting and summarizing GUI changes in evolving mobile apps | CE3 |
| Peruma et al. (2018) | An empirical investigation of how and why developers rename identifiers | CE3 |
| Huang et al. (2018) | Cldiff: Generating concise linked code differences | CE3 |
| Zafar, Malik e Walia (2019) | Towards Standardizing and Improving Classification of Bug-Fix Commits | CE3 |
| Gharbi et al. (2019) | On the classification of software change messages using multi-label active learning | CE3 |
| Liu et al. (2019) | Automatic generation of pull request descriptions | CE3 |
| Awad e Nagaty (2019) | Commit Message Generation from Code Differences Using Hidden Markov Models | CE2 |
| Rantala e Mäntylä (2020) | Predicting technical debt from commit contents: reproduction and extension with automated feature selection | CE3 |
| Não consta | Raku Fundamentals: A Primer with Examples, Projects, and Case Studies, Second Edition | CE3 |
| Panichella e Zaugg (2020) | An Empirical Investigation of Relevant Changes and Automation Needs in Modern Code Review | CE3 |
| Chvojka, Jager e Kakvi (2020) | Offline Witness Encryption with Semi-adaptive Security | CE3 |

Os trabalhos relacionados possuem abordagens experimentais e analíticas acerca da geração de mensagens de *commit*. Os trabalhos analíticos como Tao et al. (2022), Dey et al. (2022), Tao et al. (2021), Etemadi e Monperrus (2020) e Liu et al. (2018a), não serão tratados nas próximas seções. As próximas seções descrevem os trabalhos relacionados às abordagens para geração de mensagem de *commit* utilizando abordagens baseadas em regras e abordagens baseadas em inteligência artificial.

| | | |
|---|---|---|
| Nath e Roy (2021) | Towards automatically generating release notes using extractive summarization technique | CE3 |
| Jung (2021) | Commitbert: Commit message generation using pre-trained programming language model | CE1 |
| Não consta | NLP4Prog 2021 - 1st Workshop on Natural Language Processing for Programming, Proceedings of the Workshop | CE3 |
| Lee e Chieu (2021) | Co-Training for Commit Classification | CE3 |
| Chakraborty e Ray (2021) | On Multi-Modal Learning of Editing Source Code | CE3 |
| Wang et al. (2021) | Context-Aware Retrieval-Based Deep Commit Message Generation | CE2 |
| Não consta | PROMISE 2022 - Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering, co-located with ESEC/FSE 2022 | CE3 |
| Xu et al. (2022) | Combining Code Context and Fine-Grained Code Difference for Commit Message Generation | CE2 |
| Sousa et al. (2022) | SysRepoAnalysis: A tool to analyze and identify critical areas of source code repositories | CE3 |
| Li et al. (2022) | Retrieve-Guided Commit Message Generation with Semantic Similarity And Disparity | CE2 |
| Hu et al. (2022) | Correlating Automated and Human Evaluation of Code Documentation Generation Quality | CE3 |
| Tian et al. (2022) | Is this change the answer to that problem? correlating descriptions of bug and code changes for evaluating patch correctness | CE3 |

Fonte: Elaborado pelo autor (2024).

## 3.1 Geração de descrição de mudanças baseada em regras

Em (CORTES-COY et al., 2014; BUSE; WEIMER, 2010), os autores se propuseram a sumarizar o que mudou no código em uma linguagem legível por humanos através de um conjunto de regras predefinidas. As abordagens envolvem uma análise estrutural do código modificado e possui foco em sumarizar o que mudou no código.

Em (CORTES-COY et al., 2014), os autores propõem uma estratégia de análise do *diff* para caracterizar os *commits* conforme os métodos modificados, adicionados e removidos. Com isso, os autores geram uma mensagem de *commit* composta de três elementos: uma etiqueta indicando o propósito do *commit*, *i.e.*, se conserta um erro ou implementa um novo recurso, uma descrição geral e uma descrição detalhada. Para gerar

estas descrições, os autores desenvolveram formatos de descrições com partes do texto que variam de acordo com o propósito do *commit*. A Listagem 3.1 ilustra uma descrição gerada pela técnica, para um dado *diff* do projeto Spring Social Java[2] que não se encontra disponível na publicação dos autores Cortes-Coy et al. (2014).

[2]Projeto Spring Social Java: http://goo.gl/a1q8Xh

Listagem 3.1: Exemplo de mensagem de *commit* listando o impacto de uma modificação de um metodo nas classes afetadas

```
1  Diff: Nao disponivel
2
3  Descricao:
4  BUG - FEATURE: <type-ID>
5  This is a small modifier commit that does not change the system
       significantly. This change set is mainly composed of:
6  1. Changes to package org.springframework.social.oauth2:
7  1.1. Modifications to AccessGrant.java:
8  1.1.1. Add a constructor method
9  The added/removed methods triggered changes to
       OAuth2ProviderSignInAccount class
10
11 2. Changes to package org.springframework.social.web.signin:
12 2.1. Modifications to OAuth2ProviderSignInAccount.java:
13 2.1.1. Modify arguments list when calling connect method at con- nect(
       Serializable) method
```

Os resultados demonstraram que a abordagem dos autores é capaz de enriquecer a mensagem de *commit* com informações essenciais. Embora seja rica em informações, os autores observaram que as mensagens de *commit* geradas precisam ser menores e mais sucintas. Desta forma, concluiu-se que a estratégia pode ser útil como assistência para desenvolvedores na escrita das mensagens de *commit* ou para gerar mensagens de *commit* quando elas não existem ou possuem baixa qualidade.

Em Buse e Weimer (2010), os autores desenvolveram uma técnica baseada na execução simbólica do software e a divide em três partes. Na primeira parte, os autores extraem os caminhos dos predicados de cada método para cada versão do código, *i.e.*, antes e depois da modificação. Na segunda parte, os autores registram os métodos que foram adicionados, removidos ou possuem mudanças nos predicados. Na parte três, os autores aplicam uma série de transformações de sumarização com perdas de informação até que a descrição esteja concisa. A Listagem 3.1 demonstra uma descrição gerada para um determinado *diff* retirada da publicação.

Listagem 3.2: Descrição gerada pelo algoritmo DeltaDoc para um determinado *diff*

```
Diff:
19c19,22
<    else return "";
---
>    else return pageParts [0];
>    // else return "";

Descricao:
When calling LastPage format(String s)
    If s is null
        return ""
    If s is not null and
            s.split("[-]+").length != 2
        return s.split("[-]+")[0]
            Instead of return ""
```

Os resultados obtidos mostraram que a abordagem é adequada para substituir cerca de 89% das mensagens de registro. Isso sugere que a descrição de "o que"mudou no código pode ser produzida automaticamente, reduzindo o esforço humano e permitindo que os desenvolvedores foquem no "porquê"da mudança.

## 3.2 Geração de mensagens de *commit* baseada em inteligência artificial

As gerações de mensagem de *commit* que utilizam abordagens baseadas em inteligência artificial compõem maior parte dos trabalhos relacionados. O ponto em comum entre este tipo de abordagem é que seguem, de maneira geral, os mesmos passos para execução do experimento, sendo eles: a obtenção e o pré-processamento da base de dados, o treinamento do modelo de inteligência artificial e a geração das mensagens de *commit*. Já as diferenças residem na seleção da base de dados, na maneira como é pré-processada e no modelo de inteligência artificial utilizado.

Algumas abordagens, como de Loyola, Marrese-Taylor e Matsuo (2017), Liu et al.

(2018a), Nie et al. (2021), Liu et al. (2022) e Shi et al. (2022) introduzem o próprio modelo de inteligência artificial para a geração de mensagens de *commit*. Esses experimentos não seguem uma padronização quanto a seleção da base de dados e, de maneira geral, utilizam apenas projetos escritos em Java, sendo Loyola, Marrese-Taylor e Matsuo (2017) e Shi et al. (2022) exceções à este caso pois utilizam uma base composta por múltiplas linguagens de programação. Os resultados obtidos pelos autores são promissores, entretanto existe uma dificuldade de comparação entre os trabalhos devido à falta de padronização quanto à base de dados.

Em outras abordagens, os autores optaram por realizar o experimento sobre um modelo já existente, como na abordagem de Jiang, Armaly e McMillan (2017) em que os autores utilizaram o modelo (SENNRICH et al., 2017) e na abordagem de Xu et al. (2019), em que os autores utilizaram o modelo (BAHDANAU; CHO; BENGIO, 2014). Os autores também utilizam de bases de dados contendo projetos escritos em Java, e se diferenciam principalmente em tamanho e na maneira como os dados são pré-processados. Jiang, Armaly e McMillan (2017) utilizam um conjunto de mil *commits* enquanto Xu et al. (2019) utilizam um conjunto de dez mil. Seus resultados contribuem com o estado-da-arte e com as técnicas utilizadas para a tarefa de geração de mensagens de *commit*.

Os autores das abordagens envolvendo inteligência artificial para a geração de mensagens de *commit* utilizam, de maneira geral, métricas de desempenho amplamente aplicadas na avaliação das tarefas de tradução de máquina. As métricas mais comuns são: BLEU (PAPINENI et al., 2002), ROUGE (LIN, 2004) e METEOR (BANERJEE; LAVIE, 2005). Apesar de não existir uma padronização das métricas que devem ser utilizadas, os autores Tao et al. (2022) apontam a variante B-Norm da métrica BLEU como sendo a mais apropriada para este tipo de tarefa. De acordo com os autores, B-Norm é uma métrica mais correlacionada com o julgamento humano, especialmente quanto à precisão e concisão do conteúdo.

## 3.3 Considerações finais

Os trabalhos relacionados estudados apresentam diferentes abordagens para solucionar o problema da geração das mensagens de *commit* com base no *diff*. Entretanto, a grande

maioria não considera a diversidade das linguagens de programação na avaliação dos modelos. Além disso, as bases de dados utilizadas na avaliação dos modelos se diferenciam para cada modelo. De maneira geral, a falta de uma comparação efetiva e justa entre os modelos dificultam o entendimento da situação atual acerca da capacidade de gerar mensagens de *commit* com base nos *diffs* (TAO et al., 2022).

Em Tao et al. (2022), o autor buscou reduzir esta distância entre os estudos e levantou conhecimentos acerca do processo de geração de mensagens de *commit* que se revelam importantes para este trabalho. Estes conhecimentos são teóricos e se resumem em técnicas que aumentam o desempenho dos modelos de geração de mensagens de *commit*. Para isso, os autores reuniram os modelos da literatura e os submeteram à mesmas condições de experimento, estabelecendo um método de comparação entre os mesmos.

Com isso, os autores reduziram a distância existente entre os modelos de geração de mensagens de *commit* e ofereceram os métodos mais eficazes de geração de mensagens de *commit* da literatura. A base de comparação entre os diferentes modelos, que antes não seguiam um padrão em comum em seus experimentos, estimulou este trabalho a selecionar o melhor modelo reprodutível como base de comparação para a abordagem implementada com o GPT-4.

# 4 Geração e análise das mensagens de commit utilizando o GPT-4

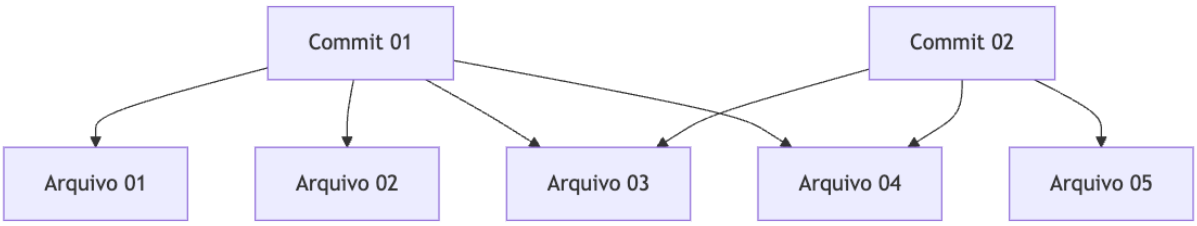
O GPT-4 é um modelo de linguagem de grande porte capaz de processar e responder a comandos textuais, denominados *prompts* (ACHIAM et al., 2023). Neste trabalho foram analisadas as respostas fornecidas pelo GPT-4 ao se solicitar uma mensagem de *commit* para um dado *diff* em cenários com e sem histórico de mensagens de *commit*. Para isso foram estabelecidas quatro estratégias na construção dos *prompts*: utilizando apenas o *diff*, sem histórico de mensagens de *commit*; utilizando o *diff* com o histórico de mensagens de *commit* do autor; utilizando o *diff* com o histórico de mensagens de *commit* dos arquivos; e utilizando o *diff* com o histórico de mensagens de *commit* do autor nos arquivos.

Cada estratégia que utiliza o histórico de mensagens de *commit* consiste em agrupar os *commits* anteriores à data do *commit* desejado e extrair suas mensagens. Na estratégia que utiliza o histórico do autor, são agrupados os *commits* anteriores desde que tenham sido feitos pelo mesmo autor, identificado pelo *e-mail*. Na estratégia que considera o histórico dos arquivos, os *commits* são agrupados se modificarem pelo menos 50% dos arquivos presentes no *commit* alvo. O percentual foi escolhido de forma arbitrária, sendo considerado uma ameaça à validade, uma vez que não foi explorado neste trabalho. Já na estratégia que combina o histórico do autor e dos arquivos, o agrupamento ocorre pela interseção dessas duas condições.

A Figura 4.1 ilustra dois *commits* e os arquivos que modificam, o "Commit 01" modifica quatro arquivos, enquanto o "Commit 02" modifica três arquivos, dos quais dois também são modificados pelo "Commit 01". Neste caso, o "Commit 02" seria eleito pela estratégia de histórico de mensagens de *commit* do arquivo, pois modifica pelo menos 50% dos arquivos modificados pelo "Commit 01" (ou seja, dois dos quatro arquivos).

Para avaliar a eficácia do GPT-4 na tarefa de geração de mensagens de *commit* e o quanto cada estratégia de *prompt* pode melhorar os resultados das mensagens geradas, é

Figura 4.1: Ilustração de um *commit* selecionado pela estratégia de histórico de mensagens de *commit* do arquivo




Fonte: Elaborado pelo autor (2024).

necessário uma base de comparação para os resultados. Por isso, este trabalho baseou-se na literatura para fazer a seleção de um modelo de geração de mensagem de *commit* e mostrar que o mesmo pode ser reproduzido e utilizado no experimento. Desta forma, este capítulo está organizado em quatro seções, nas quais a Seção 4.1 apresenta as questões de pesquisa. A Seção 4.2 aborda o design do experimento realizado. A Seção 4.3 discute os resultados obtidos pelo experimento usando o CoRec e o GPT-4, respondendo às questões de pesquisa e, por fim, a Seção 4.4 trata a respeito das ameaças a validade desta pesquisa.

## 4.1 Questões de Pesquisa

Para compreender os fatores que impactam a geração de mensagens de *commit* de qualidade, foram formuladas questões de pesquisa que exploram e analisam como diferentes estratégias de *prompt* podem aprimorar o desempenho do modelo GPT-4. Além disso,

essas questões investigam a eficácia do modelo proposto em comparação com o CoRec, no que se refere à pontuação B-Norm. A seguir, cada questão será discutida, permitindo uma melhor análise dos fatores que podem contribuir para a melhoria na geração automatizada de mensagens de *commit*.

*QP1: Como o fornecimento de histórico de mensagens de commit afeta a geração das mensagens de commit pelo GPT-4?*

Acredita-se na hipótese de que o fornecimento do histórico de mensagens de *commit* para o GPT-4 como forma de contexto pode melhorar os resultados obtidos na geração dessas mensagens. Desta forma, a QP1 busca validar essa hipótese a partir da compreensão de como o histórico pode impactar a geração das mensagens pelo GPT-4.

*QP2: Qual estratégia de prompt aplicada ao GPT-4 produz o melhor resultado?*

Acredita-se que, ao fornecer o histórico das mensagens de *commit* anteriores através da aplicação de diferentes estratégias de *prompt*, os resultados das mensagens geradas podem ser aprimorados. Esta pergunta busca entender como a qualidade das mensagens de *commit* geradas é impactada pela variação das estratégias de *prompt*, considerando o histórico de mensagens do autor, do arquivo, e do autor no arquivo.

*QP3: A estratégia com melhor resultado supera o modelo CoRec quanto a pontuação B-Norm?*

Ao utilizar uma abordagem inovadora que fornece contexto com base nas mensagens de *commit* anteriores, surge a necessidade de compreender se o modelo proposto produz resultados melhores ou piores em comparação ao modelo dominante na literatura CoRec. Essa pergunta busca determinar a melhor estratégia comparativa, utilizando a pontuação B-Norm.

## 4.2 Design do Experimento

Esta seção está organizada em subseções para separar as diferentes etapas do experimento. A Subseção 4.2.1 explica como foi selecionado o modelo de geração de mensagens de *commit* utilizado no experimento. A Subseção 4.2.2 aborda a reprodução do modelo CoRec,

enquanto a Subseção 4.2.3 trata do processo de obtenção da base de dados utilizada com o CoRec e o GPT-4. Em seguida, a Subseção 4.2.4 apresenta a execução do experimento para o modelo CoRec com a base de dados MCMD. Finalmente, a Subseção 4.2.5 define as estratégias implementadas para o GPT-4 e descreve a execução do experimento com esse modelo.

### 4.2.1 Seleção do Modelo

Para selecionar o melhor modelo de geração de mensagens de *commit* como base comparativa para o GPT-4, este trabalho se baseou na classificação dos melhores modelos disponibilizada pelos autores Tao et al. (2022), conforme ilustrado na Tabela 4.1. Os modelos na tabela foram treinados e avaliados com base em uma estratégia de separação dos conjuntos de treinamento e teste por tempo, onde os *commits* do conjunto de treinamento possuem datas anteriores às do conjunto de teste. De acordo com os autores, essa estratégia aproxima o modelo de um cenário mais real, pois ele não sofre influência de *commits* futuros ao gerar novas mensagens, assim como um desenvolvedor só tem conhecimento dos *commits* passados ao escrever uma nova mensagem de *commit*.

Tabela 4.1: Modelos de geração de mensagens de *commit* e suas respectivas avaliações medidas em B-Norm

| Modelo | B-Norm (%) |
|---|---|
| PtrGNCMsg | 12.98 |
| CoRec | 12.17 |
| CoDiSum | 11.17 |
| NNGen | 9.86 |
| NMT | 8.41 |
| CmtGen | 6.82 |

Fonte: Tao et al. (2022).

O PtrGNCMsg (LIU et al., 2019) foi o modelo com o melhor resultado B-Norm médio de 12,98%. Entretanto, não foi possível executar o experimento para o modelo devido à falta de suporte e compatibilidade das bibliotecas. Tentou-se contato com os autores, mas nenhuma resposta foi obtida, implicando na escolha de um segundo melhor modelo, o CoRec, com resultado B-Norm médio de 12,17% para a geração das mensagens de *commit*.

### 4.2.2 Reprodução do CoRec

A reprodução do experimento com o modelo CoRec permite validar que os resultados obtidos pelos autores podem ser alcançados, desde que os mesmos passos para a execução do experimento sejam seguidos. Para a reprodução, foi utilizada a base de dados disponibilizada pelos autores do modelo. Os resultados obtidos na reprodução podem não ser exatamente iguais aos divulgados pelos autores, uma vez que o ambiente de execução teve de ser compatibilizado com o computador utilizado para o experimento.

Durante o processo de reprodução do CoRec, foram encontradas incompatibilidades entre as bibliotecas utilizadas pelos autores e a GPU empregada neste experimento. Assim, as correções necessárias foram realizadas seguindo as documentações oficiais das bibliotecas, e o código atualizado está disponível no GitHub[3].

O modelo híbrido de geração de mensagens de *commit* CoRec (WANG et al., 2021) foi reproduzido sem modificações no projeto original dos autores. Inicialmente, surgiram dificuldades na reprodução do experimento devido a diferenças no driver da placa de vídeo utilizada. Para compatibilizar o projeto com a placa de vídeo NVIDIA RTX 3060Ti, foi necessário utilizar a versão mais recente do *driver* da NVIDIA (CUDA 11.8) em conjunto com a biblioteca PyTorch na versão 2.1.0[4].

Os autores Wang et al. (2021) propuseram duas bases de dados, denominadas top1000 e top10000, que contêm dados dos mil e dos dez mil repositórios Java com mais estrelas no GitHub, respectivamente. Neste trabalho, ambas as bases foram reproduzidas para validar a reprodutibilidade do experimento. O tempo de treinamento para a base top1000 foi de aproximadamente 3 horas e 36 minutos, enquanto para a base top10000 foi de cerca de 16 horas e 17 minutos.

Para comparar os resultados dos autores com os obtidos neste experimento, foram utilizadas as métricas BLEU e B-Norm. A métrica B-Norm, não utilizada pelos autores do modelo CoRec, foi calculada apenas para os resultados deste experimento. A Tabela 4.2 apresenta os valores de BLEU fornecidos pelos autores Wang et al. (2021) em comparação com os valores de BLEU e B-Norm obtidos neste estudo.

[3]Repositório do GitHub: https://github.com/abreuthrj/ufjf-tcc

[4]As instruções para instalação foram obtidas no site oficial da PyTorch: https://pytorch.org/get-started/locally

Tabela 4.2: Tabela dos valores de BLEU e B-Norm obtidos para o modelo CoRec

| Autor | Base | BLEU (%) | B-Norm (%) |
|---|---|---|---|
| Wang et al. (2021) | top1000 | 19,80 | -/- |
| | top10000 | 41,26 | -/- |
| Esta abordagem | top1000 | 19,40 | 25,56 |
| | top10000 | 40,85 | 41,88 |

Fonte: Elaborado pelo autor (2024).

### 4.2.3 Seleção da Base de Dados

Para escolher a base de dados utilizada nos experimentos, foi necessário buscar uma que contivesse as informações adequadas para a implementação das abordagens deste trabalho. A busca nos trabalhos relacionados revelou que a única base com as informações necessárias, como a identificação do *commit* (nome do repositório e identificador do *commit*), era a base de dados MCMD (TAO et al., 2022). A presença dessas informações permitiu extrair, através da API do GitHub, o autor e os arquivos envolvidos no *commit*, essenciais para a implementação das abordagens propostas utilizando o GPT-4 (Seção 4.2.4).

A base MCMD contém *commits* de cinco linguagens de programação, totalizando 450 mil *commits* por linguagem, separados pelos autores em 360 mil para o treinamento, 45 mil para a validação e 45 mil para o teste. Os autores também disponibilizaram conjuntos já ordenados por *timestamp* ou separados por projeto. Neste trabalho, foi utilizado o conjunto de *commits* em *JavaScript*, uma vez que é uma linguagem amplamente utilizada, e os dados foram ordenados por *timestamp*, uma abordagem mais coerente com o cenário real em que o modelo não tem acesso a *commits* futuros, conforme mostrado em Tao et al. (2022).

O conjunto de dados em *JavaScript* escolhido da base precisou ser reduzido devido à limitação de memória da placa de vídeo do ambiente experimental. Assim, os tamanhos dos conjuntos de treinamento, validação e teste foram ajustados para se aproximarem dos utilizados pelos autores do CoRec. Para evitar qualquer tipo de viés na redução, foram selecionados aleatoriamente 100 mil *commits* (96.704 no CoRec) para o conjunto de treinamento, 5.400 *commits* (5.373 no CoRec) para o conjunto de teste e 5.400 *commits*

(5.372 no CoRec) para o conjunto de validação. Para facilitar a compreensão, o conjunto de dados foi denominado como "base reduzida".

### 4.2.4 Geração utilizando o CoRec

Para garantir uma base de comparação entre o modelo CoRec e a geração de mensagens de *commit* pelo GPT-4, é necessário que ambos utilizem a mesma base de dados. Nesta etapa do experimento, o modelo CoRec foi treinado utilizando os *diffs* e mensagens de *commit* da base reduzida. Após o treinamento, a base reduzida também foi usada para testar o modelo e gerar as mensagens de *commit*.

Na Subseção 4.2.5, o conjunto de testes da base reduzida foi submetido ao GPT-4 até que, no *commit* de número 672, os créditos disponíveis para o experimento se esgotaram. Dessa forma, foi necessário realizar o teste do modelo CoRec limitado a esses *commits*, para manter a igualdade entre os experimentos. O conjunto limitado de *commits* foi denominado "conjunto experimental" para fins de entendimento.

Os resultados foram avaliados utilizando as métricas BLEU e B-Norm, e comparados com os resultados obtidos pelos autores Tao et al. (2022). No entanto, os autores não disponibilizaram o valor do BLEU para o experimento realizado na base de dados ordenada por *timestamp*. Assim, a Tabela 4.3 apresenta apenas o valor de B-Norm fornecido pelos autores, junto com os valores de BLEU e B-Norm obtidos neste experimento, tanto para a base reduzida (5.400 *commits*) quanto para o conjunto experimental (672 *commits*).

Tabela 4.3: Tabela comparativa dos valores de BLEU e B-Norm para os experimentos em Tao et al. (2022) e neste trabalho

| Experimento | BLEU (%) | B-Norm (%) |
|---|---|---|
| Tao et al. (2022) | -/- | 15,94 |
| Neste trabalho (5.400 *commits*) | 9,01 | 13,65 |
| Neste trabalho (672 *commits*) | 6,08 | 11,54 |

Fonte: Elaborado pelo autor (2024).

### 4.2.5 Geração utilizando o GPT-4

Para gerar as mensagens de *commit* utilizando o GPT-4, foi empregada a mesma base de dados reduzida mencionada na Seção 4.2.3, a qual também foi usada para avaliar o modelo CoRec na Subseção 4.2.2. Para implementar as quatro estratégias de construção do *prompt* apresentadas no início do Capítulo 4 com a base reduzida, foi necessário enriquecer os *commits* com o histórico de mensagens de *commit* anteriores. Para isso, foi construído um algoritmo que percorre os *commits* da base de dados e, para cada *commit*, consulta e armazena informações como o *e-mail* do autor e os nomes dos arquivos.

Cada estratégia de construção de *prompt* consiste em agrupar os *commits* anteriores ao *commit* que se deseja enriquecer com o histórico de mensagens de *commits*, utilizando uma chave em comum. Para cada estratégia, percorreu-se os *commits* da base de dados reduzida buscando os *commits* de data inferior e agrupando estes *commits* por *e-mail* do autor, pelo nome dos arquivos ou por ambos. Os *commits* de data inferior tiveram suas mensagens de *commit* extraídas e armazenadas juntas do *commit* como ilustrado na Figura 4.2, compondo assim uma nova base, denominada base enriquecida.

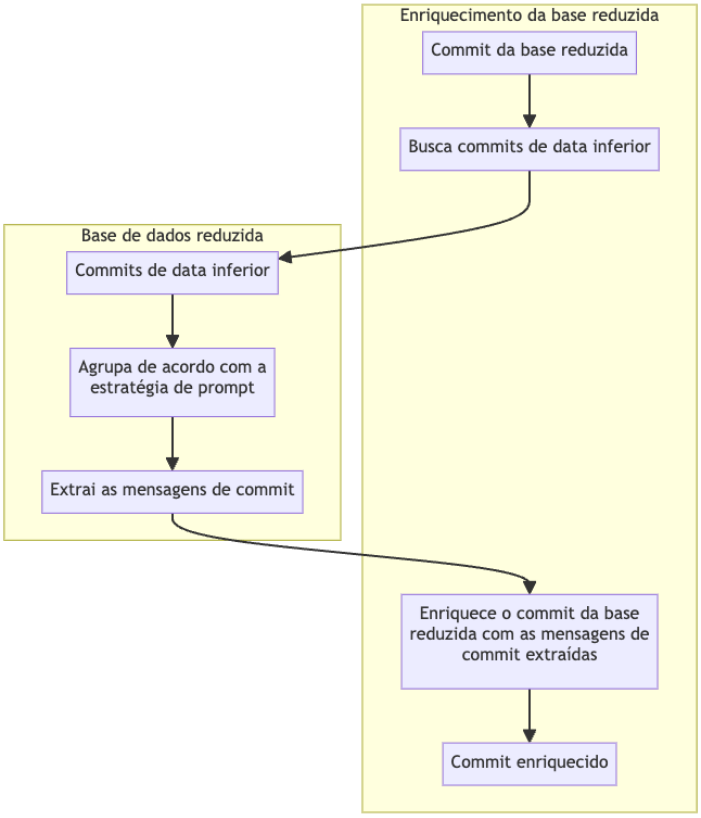
#### ***Diff* sem histórico de mensagens de *commit***

Para implementar a estratégia de *prompt* sem histórico de mensagens de *commit*, foi definido um texto base, conforme ilustrado na Listagem 4.1. Para cada *commit* da base de dados enriquecida, a palavra-chave "*$diff*" no texto base é substituída pelo *diff* correspondente ao *commit*, obtido a partir da mesma base de dados. A Listagem 4.2 mostra um *diff* obtido da base de dados enriquecida, pertencente ao *commit* do projeto "next.js"(⟨https://github.com/vercel/next.js/commit/7d6fbe7fd62a3d1f7a918b40f6c9c777ca8730af⟩). A Listagem 4.3 apresenta o *prompt* final enviado para o GPT-4, já com o *diff* da Listagem 4.2 substituído no texto base.

Listagem 4.1: Texto base utilizado na estratégia de *prompt* sem histórico de mensagens de *commit*

```
1 Write a one-line commit message for the diff:
2 $diff
```

Figura 4.2: Processo de enriquecimento do *commit* da base reduzida




Fonte: Elaborado pelo autor (2024).

Listagem 4.2: Exemplo de *diff* retirado da base de dados enriquecida para um *commit*

```
1 mmm a / . github / ISSUE_TEMPLATE / 1 . Bug_report . md <nl> ppp b / .
    github / ISSUE_TEMPLATE / 1 . Bug_report . md <nl> If applicable ,
    add screenshots to help explain your problem . <nl> - OS : [ e . g .
    macOS , Windows ] <nl> - Browser ( if applies ) [ e . g . chrome ,
    safari ] <nl> - Version of Next . js : [ e . g . 6 . 0 . 2 ] <nl> + -
     Version of Node . js : [ e . g . 10 . 10 . 0 ] <nl> <nl> # #
    Additional context <nl> <nl>
```

Listagem 4.3: Exemplo de prompt para a estratégia de *prompt* sem histórico de mensagens de *commit*

```
Write a one-line commit message for the diff:
mmm a/.github/ISSUE_TEMPLATE/1.Bug_report.md <nl> ppp b/.
   github/ISSUE_TEMPLATE/1.Bug_report.md <nl> If applicable,
   add screenshots to help explain your problem. <nl> - OS: [e.g.
   macOS, Windows] <nl> - Browser (if applies) [e.g. chrome,
   safari] <nl> - Version of Next.js: [e.g. 6.0.2] <nl> + -
    Version of Node.js: [e.g. 10.10.0] <nl> <nl> # #
   Additional context <nl> <nl>
```

### *Diff* com histórico de mensagens de *commit* do autor

Para implementar a estratégia de *prompt* com o histórico de mensagens de *commit* do autor, foi definido um texto base, conforme ilustrado na Listagem 4.4. Para cada *commit* da base de dados enriquecida, as palavras-chave "*$diff*" e "*$history*" no texto base são substituídas pelo *diff* e histórico de mensagens de *commit* do autor, obtidos a partir da mesma base de dados. A Listagem 4.5 mostra um *diff* obtido da base de dados enriquecida, pertencente ao *commit* do projeto "react-native"⟨https://github.com/facebook/react-native/commit/92160f3144dcfa510ff14b5f2eb231643f107af9⟩. A Listagem 4.6 apresenta o histórico de mensagens do autor do *commit*. A Listagem 4.7 apresenta o *prompt* final enviado para o GPT-4, já com o *diff* da Listagem 4.5 e histórico da Listagem 4.6 substituídos no texto base.

Listagem 4.4: Texto base utilizado na estratégia de *prompt* com o histórico de mensagens de *commit* do autor

```
Formato do prompt:
Write a one-line commit message for the diff:
$diff

Previous commit messages of the author:
$history
```

Listagem 4.5: Exemplo de *diff* retirado da base de dados enriquecida para um *commit*

```
mmm a / Libraries / Components / TextInput / TextInput . js <nl> ppp b /
    Libraries / Components / TextInput / TextInput . js <nl> function
   InternalTextInput ( props : Props ) : React . Node { <nl> const style
    = [ props . style ] ; <nl> const autoCapitalize = props .
   autoCapitalize | | ' sentences ' ; <nl> let children = props .
   children ; <nl> - let childCount = 0 ; <nl> - React . Children .
   forEach ( children , ( ) = > + + childCount ) ; <nl> + const
   childCount = React . Children . count ( children ) ; <nl> invariant (
    <nl> ! ( props . value & & childCount ) , <nl> ' Cannot specify both
    value and children . ' , <nl>
```

Listagem 4.6: Histórico de mensagens de *commit* do autor

```
change onRefresh flow typing ( )
mention RNTester app in contributor guide ( )
add ripple config object to Pressable ( )
fix : ripple should be applied even when borderless = = false ( )
get ripple drawables by id ( )
```

Listagem 4.7: Exemplo de prompt para a estratégia de *prompt* com histórico de mensagens de *commit* do autor

```
Write a one-line commit message for the diff:
mmm a / Libraries / Components / TextInput / TextInput . js <nl> ppp b /
    Libraries / Components / TextInput / TextInput . js <nl> function
   InternalTextInput ( props : Props ) : React . Node { <nl> const style
    = [ props . style ] ; <nl> const autoCapitalize = props .
   autoCapitalize | | ' sentences ' ; <nl> let children = props .
   children ; <nl> - let childCount = 0 ; <nl> - React . Children .
   forEach ( children , ( ) = > + + childCount ) ; <nl> + const
   childCount = React . Children . count ( children ) ; <nl> invariant (
    <nl> ! ( props . value & & childCount ) , <nl> ' Cannot specify both
    value and children . ' , <nl>

    Previous commit messages of the author:
    change onRefresh flow typing ( )
    mention RNTester app in contributor guide ( )
    add ripple config object to Pressable ( )
    fix : ripple should be applied even when borderless = = false ( )
    get ripple drawables by id ( )
```

### *Diff* com histórico de mensagens de *commit* dos arquivos

Para implementar a estratégia de *prompt* com o histórico de mensagens de *commit* dos arquivos, foi definido um texto base, conforme ilustrado na Listagem 4.8. Para cada *commit* da base de dados enriquecida, as palavras-chave "*$diff*" e "*$history*" no texto base são substituídas pelo *diff* e histórico de mensagens de *commit* dos arquivos, obtidos a partir da mesma base de dados. A Listagem 4.9 mostra um *diff* obtido a partir da base de dados enriquecida, pertencente ao *commit* do projeto "30-seconds-of-code"⟨https://github.com/Chalarangelo/30-seconds-of-code/commit/2ddddfe7947587203c7408ffaf6438a9891b6f22⟩. A Listagem 4.10 apresenta o histórico de mensagens dos arquivos do *commit*. A Listagem 4.11 apresenta o *prompt* final enviado para o GPT-4, já com o *diff* da Listagem 4.9 e histórico da Listagem 4.10 substituídos no texto base.

Listagem 4.8: Texto base utilizado na estratégia de *prompt* com o histórico de mensagens de *commit* dos arquivos

```
Formato do prompt:
Write a one-line commit message for the diff:
$diff

Previous commit messages of the file:
$history
```

Listagem 4.9: Exemplo de *diff* retirado da base de dados enriquecida para um *commit*

```
mmm a / snippets / cartesianProduct . md <nl> ppp b / snippets /
    cartesianProduct . md <nl> <nl> mmm <nl> title : cartesianProduct <nl
    > - tags : array , beginner <nl> + tags : math , array , beginner <nl
    > mmm <nl> <nl> Calculates the cartesian product of two arrays . <nl>
```

Listagem 4.10: Histórico de mensagens de *commit* dos arquivos

```
Add cartesianProduct
```

Listagem 4.11: Exemplo de prompt para a estratégia de *prompt* com histórico de mensagens de *commit* dos arquivos

```
Write a one-line commit message for the diff:
mmm a / snippets / cartesianProduct . md <nl> ppp b / snippets /
    cartesianProduct . md <nl> <nl> mmm <nl> title : cartesianProduct <nl
    > - tags : array , beginner <nl> + tags : math , array , beginner <nl
    > mmm <nl> <nl> Calculates the cartesian product of two arrays . <nl>

     Previous commit messages of the file:
     Add cartesianProduct
```

### *Diff* com histórico de mensagens de *commit* do autor nos arquivos

Para implementar a estratégia de *prompt* com o histórico de mensagens de *commit* do autor nos arquivos, foi definido um texto base, conforme ilustrado na Listagem 4.12. Para cada *commit* da base de dados enriquecida, as palavras-chave "*$diff*"e "*$history*"no texto base são substituídas pelo *diff* e histórico de mensagens de *commit* do autor nos arquivos,

obtidos a partir da mesma base de dados. A Listagem 4.13 mostra um *diff* obtido a partir da base de dados enriquecida, pertencente ao *commit* do projeto "pdf.js" ⟨https://github.com/mozilla/pdf.js/commit/0ae10bbacdb57f545f8e87439f71f59e19c8b481⟩. A Listagem 4.14 apresenta o histórico de mensagens do autor nos arquivos do *commit* citado. A Listagem 4.15 apresenta o *prompt* final enviado para o GPT-4, já com o *diff* da Listagem 4.13 e histórico da Listagem 4.14 substituídos no texto base.

Listagem 4.12: Texto base utilizado na estratégia de *prompt* com o histórico de mensagens de *commit* do autor nos arquivos

```
1 Write a one-line commit message for the diff:
2 $diff
3
4 Previous commit messages of the author in the file(s):
5 $history
```

Listagem 4.13: Exemplo de *diff* retirado da base de dados enriquecida para um *commit*

```
1 mmm a / README . md <nl> ppp b / README . md <nl> <nl> - # PDF . js [ !
    [ Build Status ] ( https : / / travis - ci . org / mozilla / pdf . js
     . svg ? branch = master ) ] ( https : / / travis - ci . org /
    mozilla / pdf . js ) <nl> + # PDF . js [ ! [ Build Status ] ( https :
     / / github . com / mozilla / pdf . js / workflows / CI / badge . svg
     ? branch = master ) ] ( https : / / github . com / mozilla / pdf .
    js / actions ? query = workflow % 3ACI + branch % 3Amaster ) <nl> <nl
    > [ PDF . js ] ( https : / / mozilla . github . io / pdf . js / ) is
    a Portable Document Format ( PDF ) viewer that is built with HTML5 .
    <nl> <nl>
```

Listagem 4.14: Histórico de mensagens de *commit* do autor nos arquivos

```
1 Merge pull request from Snuffleupagus / demo - link
2 Merge pull request from Snuffleupagus / README - gitpod
3 Merge pull request from Snuffleupagus / rm - gitpod - README
```

Listagem 4.15: Exemplo de prompt para a estratégia de *prompt* com histórico de mensagens de *commit* do autor nos arquivos

```
Write a one-line commit message for the diff:
mmm a / README . md <nl> ppp b / README . md <nl> <nl> - # PDF . js [ !
    [ Build Status ] ( https : / / travis - ci . org / mozilla / pdf . js
     . svg ? branch = master ) ] ( https : / / travis - ci . org /
    mozilla / pdf . js ) <nl> + # PDF . js [ ! [ Build Status ] ( https :
     / / github . com / mozilla / pdf . js / workflows / CI / badge . svg
     ? branch = master ) ] ( https : / / github . com / mozilla / pdf .
    js / actions ? query = workflow % 3ACI + branch % 3Amaster ) <nl> <nl
    > [ PDF . js ] ( https : / / mozilla . github . io / pdf . js / ) is
    a Portable Document Format ( PDF ) viewer that is built with HTML5 .
    <nl> <nl>

     Previous commit messages of the author in the file(s):
     Merge pull request from Snuffleupagus / demo - link
     Merge pull request from Snuffleupagus / README - gitpod
     Merge pull request from Snuffleupagus / rm - gitpod - README
```

Os *commits* da base de dados enriquecida foram percorridos e os *prompts* foram montados de acordo com cada estratégia citada nas subseções anteriores. Os *prompts* foram então submetidos ao GPT-4, e esse processo se repetiu até que todo o crédito disponível para o experimento fosse consumido, totalizando 672 *commits* analisados. Por fim, os valores de BLEU e B-Norm foram avaliados para as mensagens de *commit* geradas pelo GPT-4 e organizados de acordo com cada estratégia na Tabela 4.4.

Tabela 4.4: Tabela do valor de BLEU e B-Norm obtidos pela implementação das diferentes estratégias de *prompt* do GPT-4

| Estratégia | BLEU (%) | B-Norm (%) |
|---|---|---|
| *Diff* sem histórico de mensagens de *commit* | 1,24 | 12,63 |
| *Diff* com histórico de mensagens de *commit* do autor | 1,51 | 15,00 |
| *Diff* com histórico de mensagens de *commit* dos arquivos | 1,25 | 13,49 |
| *Diff* com histórico de mensagens de *commit* do autor nos arquivos | 1,15 | 13,50 |

Fonte: Elaborado pelo autor (2024).

## 4.3 Análise dos Resultados

Nesta Seção, são respondidas as questões de pesquisa que visam avaliar os resultados obtidos pelas abordagens utilizando o GPT-4 em comparação com o modelo de geração de mensagens de *commit* CoRec. Para isso, foram utilizados os resultados medidos em B-Norm das mensagens de *commit* geradas pelo CoRec e pelas abordagens com o GPT-4. O resultado com o maior valor de B-Norm indica que a abordagem foi capaz de gerar mensagens de *commit* mais semelhantes às originalmente escritas pelos autores dos *commits*.

*QP1: Como o fornecimento de histórico de mensagens de commit afeta a geração das mensagens de commit pelo GPT-4?*

O fornecimento do histórico de mensagens de *commit* para o GPT-4 baseia-se na hipótese de que o modelo pode produzir melhores resultados na geração de mensagens de *commit* quando recebe as mensagens anteriores como contexto. Os resultados apresentados na Tabela 4.4 mostram que a abordagem que não utilizou o histórico de mensagens de *commit* produziu um resultado inferior àquelas que utilizaram o histórico.

Quando se utiliza o histórico de mensagens de *commit* para gerar mensagens, acredita-se que o GPT-4 sofre um viés na seleção das palavras geradas. Entretanto, uma breve análise das mensagens de *commit* indicam que não há um aumento significativo do sentido semântico e descrevem o propósito da modificação do código tanto quanto a estratégia que utiliza apenas o *diff*. Dessa forma, a diferença no valor de B-Norm em relação às outras abordagens pode ser explicada pela variação nas palavras geradas pelo modelo.

*QP2: Qual das diferentes estratégias de prompt aplicadas ao GPT-4 produz o melhor resultado?*

A partir dos resultados obtidos na Tabela 4.4, é possível observar que o GPT-4 produziu melhores resultados com as estratégias que utilizaram contexto. A estratégia que obteve melhor desempenho foi a que utilizou do histórico de mensagens de *commit* do autor como contexto, com um valor de B-Norm igual a 15,00. Isso pode estar relacionado com o estilo particular do autor na redação das mensagens de *commit*, no entanto, essa é apenas uma hipótese e sua análise pode ser explorada em trabalhos futuros.

*QP3: Qual estratégia aplicada ao GPT-4 possui melhor desempenho que o resultado obtido pelo modelo CoRec para a métrica B-Norm?*

A comparação dos resultados nas Tabelas 4.3 e 4.4 revela que a estratégia que utiliza o contexto do autor com o GPT-4 (B-Norm = 15,00) superou o resultado obtido pelo modelo CoRec (B-Norm = 11,54). Portanto, em resposta à QP3, a estratégia que combina o *diff* com o contexto do autor gerou um melhor resultado em B-Norm. Uma hipótese para explicar esse resultado é que as mensagens de *commit* tendem a seguir um padrão específico no estilo de escrita de cada autor.

## 4.4 Ameaças a Validade

Embora os resultados obtidos sejam promissores ao indicar uma melhoria na pontuação do B-Norm utilizando o GPT-4, é importante destacar que existem ameaças à validade deste trabalho. Devido à limitação de recursos computacionais disponíveis, foi necessário reduzir o conjunto de dados utilizado para o treinamento do modelo CoRec. Isso pode refletir resultados inferiores, favorecendo o GPT-4, uma vez que o modelo não obteve toda a informação que poderia. Para mitigar este efeito, foi selecionada aleatoriamente da base de dados MCMD a mesma quantidade de *commits* utilizada pelos autores do CoRec.

No processo de escolha do melhor modelo, o CoRec foi selecionado para o experimento devido à dificuldade encontrada na reprodução do modelo PtrGNCMsg. Diferente do PtrGNCMsg, que é um modelo de geração, o CoRec é um modelo híbrido e, por isso, concorre com outros modelos híbridos da literatura. O modelo híbrido RACE (SHI et al., 2022) demonstrou uma melhor performance na tarefa de geração de mensagens de *commit* que o CoRec, entretanto, os recursos computacionais disponíveis não foram suficientes para realizar o experimento com o RACE.

Outra limitação que pode impactar os resultados finais obtidos pelo GPT-4 é a insuficiência de recursos financeiros. Ao todo, foram processados apenas 672 *commits*, pois os créditos disponíveis se esgotaram durante a execução do experimento. Isso pode afetar os resultados, pois uma quantidade menor de dados pode reduzir o poder de generalização dos resultados.

Durante o processo de enriquecimento do contexto com o histórico de mensagens,

outra ameaça à validade é observada. Para um determinado *commit*, busca-se os últimos *commits* que compartilham modificações em ao menos 50% dos arquivos para extrair a mensagem de *commit*. Entretanto, não se sabe exatamente o impacto desse percentual nos resultados, o que deve ser melhor explorado em trabalhos futuros.

Além disso, a origem dos *diffs* e das mensagens de *commit* utilizadas na implementação das estratégias de *prompt* também representa uma ameaça à validade. Esses dados, obtidos da base MCMD, se encontram pré-processados, ou seja, algumas palavras e caracteres foram inseridos, removidos ou modificados, como a troca de quebra de linha por "<nl>" nos *diffs*. Isso pode fazer com que os *diffs* e as mensagens de *commit* utilizadas nos *prompts* não estejam em suas formas originais encontradas do Git, e o impacto desse efeito nos resultados deve ser investigado em trabalhos futuros.

# 5 Conclusão

Neste trabalho, buscou-se avaliar se o modelo de linguagem de grande porte GPT-4 é capaz de melhorar os resultados na geração de mensagens de *commit* em comparação com um modelo existente na literatura. Além disso, investigou-se se o fornecimento de um histórico das mensagens de *commit* anteriores pode melhorar esses resultados. Para isso, foi necessário escolher o melhor modelo disponível na literatura, ou seja, o CoRec, para usar como base de comparação e definir a base de dados a ser utilizada no experimento.

A seleção da base de dados depende diretamente da possibilidade de se extrair o contexto necessário para a realização do experimento e, por isso, foi escolhida a base MCMD (TAO et al., 2022). Também foi escolhida a linguagem de programação JavaScript para a execução do experimento, devido à sua alta utilização no desenvolvimento de *software*.

Os experimentos podem ser divididos em duas partes. A primeira parte consistiu em treinar o modelo CoRec com a base de dados obtida. Para isso foi necessário ordenar os *commits* de acordo com a data e, em seguida, separar os conjuntos de treinamento, validação e teste. Após isso, o modelo foi treinado e o conjunto de testes foi utilizado para realizar a tradução dos *diffs* em mensagens de *commit*.

A segunda parte do experimento consistiu em, para cada *diff* do conjunto de teste da primeira parte, enriquecer o *diff* com informações do histórico de mensagens de *commit* e enviá-lo para o GPT-4. Foram elaboradas quatro estratégias de *prompt*: sem histórico de mensagens de *commit*, com histórico de mensagens de *commit* do autor, com histórico de mensagens de *commit* dos arquivos e com histórico de mensagens de *commit* do autor nos arquivos.

Os resultados mostraram que o GPT-4 obteve um desempenho superior na geração de mensagens de *commit*, especialmente quando recebeu o histórico de mensagens anteriores do mesmo autor, alcançando um B-Norm de 15,00%. Esse valor superou o resultado do modelo CoRec, que atingiu um B-Norm de 11,54%. Isso demonstra o potencial dos modelos de linguagem de grande porte, como o GPT-4, para a tarefa de geração de men-

sagens de *commit*. Portanto, o fornecimento do histórico de mensagens de *commit* provou ser uma forma eficaz de fornecer contexto ao modelo, contribuindo para a geração de mensagens mais próximas das mensagens reais escritas pelos autores, com um impacto significativo na qualidade dos resultados.

Apesar dos resultados promissores obtidos com o GPT-4, existem oportunidades para o aprimoramento e extensão deste trabalho. Futuras investigações poderiam explorar a utilização de modelos de linguagem ainda mais recentes ou especializados para comparar com o desempenho do GPT-4. Além disso, a análise aprofundada do impacto de diferentes quantidades e tipos de histórico de mensagens de *commit* sobre a qualidade das mensagens geradas poderia proporcionar uma compreensão mais profunda sobre como otimizar o fornecimento de contexto. Essas direções podem ajudar a consolidar o entendimento e a aplicação de modelos de linguagem em tarefas de geração de mensagens de *commit*.

# Bibliografia

ACHIAM, J.; ADLER, S.; AGARWAL, S.; AHMAD, L.; AKKAYA, I.; ALEMAN, F. L.; ALMEIDA, D.; ALTENSCHMIDT, J.; ALTMAN, S.; ANADKAT, S. et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

AWAD, A.; NAGATY, K. Commit message generation from code differences using hidden markov models. In: *ACM International Conference Proceeding Series*. [S.l.: s.n.], 2019. p. 96–99. Cited By 0.

BAHDANAU, D.; CHO, K.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

BANERJEE, S.; LAVIE, A. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. [S.l.: s.n.], 2005. p. 65–72.

BITKEEPER. *BitKeeper Scalable Version Control*. 2023. Acessado em: 26 de maio de 2023. Disponível em: ⟨bitkeeper.org⟩.

BRANTS, T. Natural language processing in information retrieval. *CLIN*, v. 111, 2003.

BUSE, R. P.; WEIMER, W. R. Automatically documenting program changes. In: *Proceedings of the 25th IEEE/ACM international conference on automated software engineering*. [S.l.: s.n.], 2010. p. 33–42.

CHACON, S.; STRAUB, B. *Pro git*. [S.l.]: Springer Nature, 2014.

CHAKRABORTY, S.; RAY, B. On multi-modal learning of editing source code. In: *Proceedings - 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021*. [S.l.: s.n.], 2021. p. 443–455. Cited By 0.

CHO, K.; MERRIËNBOER, B. V.; GULCEHRE, C.; BAHDANAU, D.; BOUGARES, F.; SCHWENK, H.; BENGIO, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

CHVOJKA, P.; JAGER, T.; KAKVI, S. Offline witness encryption with semi-adaptive security. v. 12146 LNCS, p. 231–250, 2020. Cited By 0.

COMMUNITY, A. O. D. *Apache OpenNLP Developer Documentation*. 2011. Acessado em: 05 de outubro de 2023. Disponível em: ⟨https://opennlp.apache.org/docs/2.3.0/manual/opennlp.html#tools.tokenizer⟩.

CORTES-COY, L.; LINARES-VASQUEZ, M.; APONTE, J.; POSHYVANYK, D. On automatically generating commit messages via summarization of source code changes. In: *Proceedings - 2014 14th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2014*. [S.l.: s.n.], 2014. p. 275–284. Cited By 99.

DEY, S.; VINAYAKARAO, V.; GUPTA, M.; DECHU, S. Evaluating commit message generation: To bleu or not to bleu? In: *Proceedings - International Conference on Software Engineering*. [S.l.: s.n.], 2022. p. 31–35. Cited By 0.

DONG, J.; LOU, Y.; ZHU, Q.; SUN, Z.; LI, Z.; ZHANG, W.; HAO, D. Fira: Fine-grained graph-based code change representation for automated commit message generation. In: *Proceedings - International Conference on Software Engineering*. [S.l.: s.n.], 2022. v. 2022-May, p. 970–981. Cited By 0.

ELISEEVA, A.; SOKOLOV, Y.; BOGOMOLOV, E.; GOLUBEV, Y.; DIG, D.; BRYKSIN, T. From commit message generation to history-aware commit message completion. *arXiv preprint arXiv:2308.07655*, 2023.

ETEMADI, K.; MONPERRUS, M. On the relevance of cross-project learning with nearest neighbours for commit message generation. In: *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. [S.l.: s.n.], 2020. p. 470–475.

FITZPATRICK, B. W.; PILATO, C. M.; COLLINS-SUSSMAN, B. *Version control with Subversion*. [S.l.]: O'Reilly Media, 2004.

FOGEL, K.; BAR, M. *Open source development with CVS*. [S.l.]: Coriolis Group Books, 1999.

GHARBI, S.; JENHANI, I.; MKAOUER, M.; MESSAOUD, M. On the classification of software change messages using multi-label active learning. In: *Proceedings of the ACM Symposium on Applied Computing*. [S.l.: s.n.], 2019. Part F147772, p. 1760–1767. Cited By 12.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. Cambridge, MA, USA: MIT press, 2016. ⟨http://www.deeplearningbook.org⟩.

HINDLE, A.; BARR, E. T.; GABEL, M.; SU, Z.; DEVANBU, P. On the naturalness of software. *Communications of the ACM*, ACM New York, NY, USA, v. 59, n. 5, p. 122–131, 2016.

HU, X.; CHEN, Q.; WANG, H.; XIA, X.; LO, D.; ZIMMERMANN, T. Correlating automated and human evaluation of code documentation generation quality. v. 31, n. 4, 2022. Cited By 0.

HUANG, K.; ZHOU, D.; CHEN, B.; WANG, Y.; ZHAO, W.; PENG, X.; LIU, Y. Cldiff: Generating concise linked code differences. In: *ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. [S.l.: s.n.], 2018. p. 679–690. Cited By 26.

HUANG, Y.; JIA, N.; ZHOU, H.-J.; CHEN, X.-P.; ZHENG, Z.-B.; TANG, M.-D. Learning human-written commit messages to document code changes. *Journal of computer science and technology*, Springer Singapore, Singapore, v. 35, n. 6, p. 1258–1277, 2020. ISSN 1000-9000.

JIANG, K.; LU, X. Natural language processing and its applications in machine translation: A diachronic review. In: *2020 IEEE 3rd International Conference of Safe Production and Informatization (IICSPI)*. [S.l.: s.n.], 2020. p. 210–214.

JIANG, S. Boosting neural commit message generation with code semantic analysis. In: *Proceedings - 2019 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019*. [S.l.: s.n.], 2019. p. 1280–1282. Cited By 4.

JIANG, S.; ARMALY, A.; MCMILLAN, C. Automatically generating commit messages from diffs using neural machine translation. In: *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. [S.l.: s.n.], 2017. p. 135–146.

JIANG, S.; MCMILLAN, C. Towards automatic generation of short summaries of commits. In: *IEEE International Conference on Program Comprehension*. [S.l.: s.n.], 2017. p. 320–323. Cited By 29.

JUNG, T.-H. Commitbert: Commit message generation using pre-trained programming language model. *arXiv preprint arXiv:2105.14242*, 2021.

KORDJAMSHIDI, P. *Structured Machine Learning for Mapping Natural Language to Spatial Ontologies*. Tese (Doutorado), 07 2013.

KUANG, H.; SUN, Y.; LIN, K. A partial evaluator for a parallel lambda language. *Journal of Computer Science and Technology*, Springer, v. 12, p. 441–457, 1997.

KULTIMA, A.; PAAVILAINEN, J. Creativity techniques in game design. In: *Proceedings of the 2007 conference on Future Play*. [S.l.: s.n.], 2007. p. 243–244.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group UK London, v. 521, n. 7553, p. 436–444, 2015.

LEE, J. Y. D.; CHIEU, H. L. Co-training for commit classification. In: *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*. [S.l.: s.n.], 2021. p. 389–395.

LI, Z.; CHENG, Y.; YANG, H.; KUANG, L.; ZHANG, L. Retrieve-guided commit message generation with semantic similarity and disparity. In: IEEE. *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*. [S.l.], 2022. p. 357–366.

LIN, C.-Y. Rouge: A package for automatic evaluation of summaries. In: *Text summarization branches out*. [S.l.: s.n.], 2004. p. 74–81.

LINARES-VÁSQUEZ, M.; CORTÉS-COY, L. F.; APONTE, J.; POSHYVANYK, D. Changescribe: A tool for automatically generating commit messages. In: IEEE. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. [S.l.], 2015. v. 2, p. 709–712.

LIU, Q.; LIU, Z.; ZHU, H.; FAN, H.; DU, B.; QIAN, Y. Generating commit messages from diffs using pointer-generator network. In: *IEEE International Working Conference on Mining Software Repositories*. [S.l.: s.n.], 2019. v. 2019-May, p. 299–309. Cited By 18.

LIU, S.; GAO, C.; CHEN, S.; NIE, L.; LIU, Y. Atom: Commit message generation based on abstract syntax tree and hybrid ranking. v. 48, n. 5, p. 1800–1817, 2022. Cited By 2.

LIU, Y.; HAN, T.; MA, S.; ZHANG, J.; YANG, Y.; TIAN, J.; HE, H.; LI, A.; HE, M.; LIU, Z. et al. Summary of chatgpt/gpt-4 research and perspective towards the future of large language models. *arXiv preprint arXiv:2304.01852*, 2023.

LIU, Z.; LO, D.; XIA, X.; XING, Z.; HASSAN, A.; WANG, X. Neural-machine-translation-based commit message generation: How far are we? In: *ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. [S.l.: s.n.], 2018. p. 373–384. Cited By 86.

LIU, Z.; XIA, X.; HASSAN, A. E.; LO, D.; XING, Z.; WANG, X. Neural-machine-translation-based commit message generation: how far are we? In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. [S.l.: s.n.], 2018. p. 373–384.

LIU, Z.; XIA, X.; TREUDE, C.; LO, D.; LI, S. Automatic generation of pull request descriptions. In: *Proceedings - 2019 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019*. [S.l.: s.n.], 2019. p. 176–188. Cited By 39.

LOYOLA, P.; MARRESE-TAYLOR, E.; BALAZS, J.; MATSUO, Y.; SATOH, F. Content aware source code change description generation. In: *Proceedings of the 11th International Conference on Natural Language Generation*. [S.l.: s.n.], 2018. p. 119–128.

LOYOLA, P.; MARRESE-TAYLOR, E.; MATSUO, Y. A neural architecture for generating natural language descriptions from source code changes. In: *ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)*. [S.l.: s.n.], 2017. v. 2, p. 287–292. Cited By 43.

LTD., C. *Bazaar*. 2009. Acessado em: 03 de outubro de 2023. Disponível em: ⟨https://web.archive.org/web/20230506175749/http://bazaar.canonical.com/en/⟩.

MAALEJ, W.; HAPPEL, H.-J. Can development work describe itself? In: IEEE. *2010 7th IEEE working conference on mining software repositories (MSR 2010)*. [S.l.], 2010. p. 191–200.

MERCURIAL. *Mercurial Source Control Management*. 2023. Acessado em: 26 de maio de 2023. Disponível em: ⟨https://www.mercurial-scm.org/⟩.

MORAN, K.; WATSON, C.; HOSKINS, J.; PURNELL, G.; POSHYVANYK, D. Detecting and summarizing gui changes in evolving mobile apps. In: *ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. [S.l.: s.n.], 2018. p. 543–553. Cited By 14.

NATH, S.; ROY, B. Towards automatically generating release notes using extractive summarization technique. In: *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*. [S.l.: s.n.], 2021. v. 2021-July, p. 241–248. Cited By 3.

NIE, L.; GAO, C.; ZHONG, Z.; LAM, W.; LIU, Y.; XU, Z. Coregen: Contextualized code representation learning for commit message generation. v. 459, p. 97–107, 2021. Cited By 3.

OTTE, S. Version control systems. *Computer systems and telematics*, Citeseer, p. 11–13, 2009.

PANICHELLA, S.; ZAUGG, N. An empirical investigation of relevant changes and automation needs in modern code review. v. 25, n. 6, p. 4833–4872, 2020. Cited By 7.

PAPINENI, K.; ROUKOS, S.; WARD, T.; ZHU, W.-J. Bleu: a method for automatic evaluation of machine translation. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. [S.l.: s.n.], 2002. p. 311–318.

PERUMA, A.; MKAOUER, M. W.; DECKER, M. J.; NEWMAN, C. D. An empirical investigation of how and why developers rename identifiers. In: *Proceedings of the 2nd International Workshop on Refactoring*. [S.l.: s.n.], 2018. p. 26–33.

RANTALA, L.; MäNTYLä, M. Predicting technical debt from commit contents: reproduction and extension with automated feature selection. v. 28, n. 4, p. 1551–1579, 2020. Cited By 5.

RASTKAR, S.; MURPHY, G. Why did this code change? In: *Proceedings - International Conference on Software Engineering*. [S.l.: s.n.], 2013. p. 1193–1196. Cited By 44.

REVELO. A importância do controle de versão para o desenvolvimento de software. *Blog Revelo*, 2023. Disponível em: ⟨https://blog.revelo.com.br/blog/controle-versao-software-como-usar/⟩.

ROEHM, T.; TIARKS, R.; KOSCHKE, R.; MAALEJ, W. How do professional developers comprehend software? In: IEEE. *2012 34th International Conference on Software Engineering (ICSE)*. [S.l.], 2012. p. 255–265.

SENNRICH, R.; FIRAT, O.; CHO, K.; BIRCH, A.; HADDOW, B.; HITSCHLER, J.; JUNCZYS-DOWMUNT, M.; LÄUBLI, S.; BARONE, A. V. M.; MOKRY, J. et al. Nematus: a toolkit for neural machine translation. 2017.

SHI, E.; WANG, Y.; TAO, W.; DU, L.; ZHANG, H.; HAN, S.; ZHANG, D.; SUN, H. Race: Retrieval-augmented commit message generation. *arXiv preprint arXiv:2203.02700*, 2022.

SOUSA, A.; RIBEIRO, G.; AVELINO, G.; ROCHA, L.; BRITTO, R. Sysrepoanalysis: A tool to analyze and identify critical areas of source code repositories. In: *ACM International Conference Proceeding Series*. [S.l.: s.n.], 2022. p. 376–381. Cited By 0.

STANFORD. *The Stanford Natural Language Processing Group*. 2005. Acessado em: 05 de outubro de 2023. Disponível em: ⟨https://nlp.stanford.edu/software/tokenizer.html⟩.

SUN, S.; LUO, C.; CHEN, J. A review of natural language processing techniques for opinion mining systems. *Information fusion*, Elsevier, v. 36, p. 10–25, 2017.

TAO, W.; WANG, Y.; SHI, E.; DU, L.; HAN, S.; ZHANG, H.; ZHANG, D.; ZHANG, W. On the evaluation of commit message generation models: An experimental study. In: *Proceedings - 2021 IEEE International Conference on Software Maintenance and Evolution, ICSME 2021*. [S.l.: s.n.], 2021. p. 126–136. Cited By 2.

TAO, W.; WANG, Y.; SHI, E.; DU, L.; HAN, S.; ZHANG, H.; ZHANG, D.; ZHANG, W. A large-scale empirical study of commit message generation: models, datasets and evaluation. v. 27, n. 7, 2022. Cited By 0.

TASNIM, A.; RAHMAN, M. Inferring bug patterns for detecting bugs in javascript by analyzing abstract syntax tree. In: *2018 Joint 7th International Conference on Informatics, Electronics and Vision and 2nd International Conference on Imaging, Vision and Pattern Recognition, ICIEV-IVPR 2018*. [S.l.: s.n.], 2019. p. 503–507. Cited By 0.

TIAN, H.; TANG, X.; HABIB, A.; WANG, S.; LIU, K.; XIA, X.; KLEIN, J.; BISSYANDÉ, T. F. Is this change the answer to that problem? correlating descriptions of bug and code changes for evaluating patch correctness. In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. [S.l.: s.n.], 2022. p. 1–13.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł.; POLOSUKHIN, I. Attention is all you need. *Advances in neural information processing systems*, v. 30, 2017.

WANG, H.; XIA, X.; LO, D.; HE, Q.; WANG, X.; GRUNDY, J. Context-aware retrieval-based deep commit message generation. v. 30, n. 4, 2021. Cited By 5.

WINGERD, L. *Practical perforce*. [S.l.]: "O'Reilly Media, Inc.", 2005.

XU, S.; YAO, Y.; XU, F.; GU, T.; TONG, H.; LU, J. Commit message generation for source code changes. In: *IJCAI International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 2019. v. 2019-August, p. 3975–3981. Cited By 31.

XU, S.; YAO, Y.; XU, F.; GU, T.; TONG, H. Combining code context and fine-grained code difference for commit message generation. In: *ACM International Conference Proceeding Series*. [S.l.: s.n.], 2022. p. 242–251. Cited By 0.

ZAFAR, S.; MALIK, M.; WALIA, G. Towards standardizing and improving classification of bug-fix commits. In: *International Symposium on Empirical Software Engineering and Measurement*. [S.l.: s.n.], 2019. v. 2019-Septemer. Cited By 13.

ZOLKIFLI, N. N.; NGAH, A.; DERAMAN, A. Version control system: A review. *Procedia Computer Science*, Elsevier, v. 135, p. 408–415, 2018.