

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TIAGO CENCI

**UMA API INTEGRADA COM O GPT 3.5 PARA CONSULTAS SQL POR MEIO
DE LINGUAGEM NATURAL (NL2SQL)**

PATO BRANCO

2024

TIAGO CENCI

**UMA API INTEGRADA COM O GPT 3.5 PARA CONSULTAS SQL POR MEIO
DE LINGUAGEM NATURAL (NL2SQL)**

**AN API INTEGRATED WITH GPT-3.5 FOR SQL QUERIES THROUGH
NATURAL LANGUAGE (NL2SQL)**

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná.

Orientador: Profa. Dra Viviane Dal Molin

Coorientador: Profa. Dra Eliane Maria de Bortoli Favero.

PATO BRANCO

2024



[4.0 Internacional](https://creativecommons.org/licenses/by-sa/4.0/)

Esta licença permite remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es) e que licenciem as novas criações sob termos idênticos. Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

TIAGO CENCI

**UMA API INTEGRADA COM O GPT 3.5 PARA CONSULTAS SQL POR MEIO
DE LINGUAGEM NATURAL (NL2SQL)**

Trabalho de Conclusão de Curso de Graduação
apresentado como requisito para obtenção
do título de Tecnólogo em Análise e
Desenvolvimento de Sistemas do Curso de
Tecnologia em Análise e Desenvolvimento de
Sistemas da Universidade Tecnológica Federal
do Paraná.

Data de aprovação: 17/junho/2024

Dra. Viviane Dal Molin
Universidade Tecnológica Federal do Paraná

Dra. Eliane Maria de Bortoli Favero
Universidade Tecnológica Federal do Paraná

Dr. Ives Renê Venturini Pola
Universidade Tecnológica Federal do Paraná

Me. Vinicius Pegorini
Universidade Tecnológica Federal do Paraná

PATO BRANCO
2024

RESUMO

A obtenção ágil e precisa de informações de um banco de dados é de suma importância em diversos cenários do mundo real. No entanto, no contexto dos bancos de dados relacionais, a extração dos dados desejados exige proficiência em linguagem *Structured Query Language* (SQL), o que restringe o acesso a essas informações a um público especializado. Com o avanço da Inteligência Artificial e a disponibilidade de modelos pré-treinados, como o *Generative Pre-Trained Transformer* (GPT), surge a oportunidade de criar uma contribuição significativa a esse cenário. Neste contexto, este trabalho tem como objetivo desenvolver uma aplicação que traduza o texto fornecido pelo usuário em linguagem natural para comandos SQL, e em seguida, execute esses comandos em um banco de dados que contenha um dicionário de dados específico contendo informações sobre suas tabelas e campos, para, por fim, apresentar o resultado ao usuário de maneira acessível e amigável. Para realizar este trabalho, foi desenvolvida uma Interface de Programação de Aplicação, do Inglês *Application Programming Interface* (API) em python para facilitar a troca de mensagens entre os usuários e a aplicação, assim como foi utilizada a API do modelo GPT da OpenAi e banco de dados relacional Firebird. Para avaliação do trabalho proposto, será medida a precisão da tradução para SQL, a capacidade de reconhecer entradas fora de contexto, o desempenho no tempo de resposta e a tolerância a erros no texto recebido.

Palavras-chave: processamento de linguagem natural; gpt; api; sql; python.

ABSTRACT

The swift and accurate retrieval of information from a database is crucial in various real-world scenarios. However, within the realm of relational databases, extracting desired data requires proficiency in Structured Query Language (SQL), thereby limiting access to a specialized audience. With the advancement of Artificial Intelligence and the availability of pre-trained models such as the Generative Pre-Trained Transformer (GPT), an opportunity arises to make a significant contribution to this scenario. In this context, this work aims at developing an application that translates user-supplied natural language text into SQL commands, and then run these commands in a database that contains a dictionary of specific data containing information about its tables and fields, to finally present the result to the user in an accessible and friendly way. To achieve this, an Application Programming Interface (API) was developed in python to facilitate message exchange between users and the application, leveraging the GPT model API from OpenAI, and utilizing the Firebird relational database. The evaluation of the proposed work will assess the accuracy of SQL translation, the system's ability to recognize out-of-context inputs, response time performance, and tolerance to errors in received text.

Keywords: natural language processing; gpt; api; sql; python.

LISTA DE FIGURAS

Figura 1 – Etapas básicas do Processamento de Linguagem Natural (PLN)	15
Figura 2 – Exemplo do funcionamento esperado da aplicação.	24
Figura 3 – Etapas necessárias para implementação da aplicação proposta.	24
Figura 4 – Retorno da resposta obtida ao usuário	37

LISTA DE TABELAS

Tabela 1 – Resultado do critério de desempenho	42
Tabela 2 – Resumo dos critérios de avaliação	45

LISTA DE QUADROS

Quadro 1 – Trabalhos que buscam converter linguagem natural em SQL	20
Quadro 2 – Materiais utilizados no desenvolvimento da aplicação	22
Quadro 3 – Tabela de dicionário de dados	25
Quadro 4 – Exemplo de dados contidos no dicionário de dados específico	30
Quadro 5 – Resultado do critério de contexto	38
Quadro 6 – Resultados do critério de tolerância a erros	43

LISTAGEM DE CÓDIGOS FONTE

Listagem 1 – Exemplo de segmentação, tokenização e marcação das <i>Part-Of-Speech</i> (POS) em Python	16
Listagem 2 – Conexão e extração do dicionário de dados a partir do banco de dados.	31
Listagem 3 – Recepção da entrada do usuário via API	34
Listagem 4 – Utilização da API do modelo GPT	35
Listagem 5 – Tradução do texto para SQL	35
Listagem 6 – Execução do SQL e gravação dos resultados	36
Listagem 7 – Retorno da resposta obtida ao usuário.	37

LISTA DE ABREVIATURAS E SIGLAS

Siglas

API	Interface de Programação de Aplicação, do Inglês <i>Application Programming Interface</i>
DQL	<i>Data Query Language</i>
GPT	<i>Generative Pre-Trained Transformer</i>
LMM	<i>Large Language Model</i>
NL2SQL	<i>Natural Language to SQL</i>
PLN	Processamento de Linguagem Natural
POS	<i>Part-Of-Speech</i>
SGBD	Sistema de gerenciamento de banco de dados
SQL	<i>Structured Query Language</i>

SUMÁRIO

1	INTRODUÇÃO	11
1.1	Objetivos	11
1.1.1	Objetivo geral	11
1.1.2	Objetivos específicos	11
1.2	Justificativa	12
1.3	Estrutura do trabalho	13
2	REFERENCIAL TEÓRICO	14
2.1	PLN	14
2.2	Pré-processamento de textos	15
2.3	Modelos de Representação Pré-treinados	16
2.4	Engenharia de <i>Prompts</i>	17
2.5	SQL e <i>Data Query Language</i> (DQL)	18
2.6	API	19
2.7	Trabalhos relacionados	20
3	MATERIAIS E MÉTODO	22
3.1	Materiais	22
3.2	Método	23
3.2.1	Dicionário de dados	25
3.2.2	Critérios de avaliação	26
3.2.2.1	<u>Contexto</u>	26
3.2.2.2	<u>Precisão na conversão para SQL</u>	27
3.2.2.3	<u>Desempenho</u>	28
3.2.2.4	<u>Tolerância a erros</u>	28
4	RESULTADOS	30
4.1	Obtenção de uma base de dados relacional para testes	30
4.2	Implementação da API	30
4.2.1	Conexão com o banco de dados e extração do dicionário de dados	30
4.2.2	Recepção da entrada do usuário	34
4.2.3	Envio de <i>prompts</i> para tradução e recepção da resposta	34
4.2.4	Tradução do texto para SQL	34

4.2.5	Execução do SQL e gravação dos resultados	36
4.2.6	Retorno da resposta obtida ao usuário	36
4.3	Resultados Obtidos	37
4.3.1	Retornos em relação ao contexto	37
4.3.2	Precisão na conversão para SQL	39
4.3.3	Desempenho	41
4.3.4	Tolerância a erros	41
4.4	Discussão dos resultados finais	42
5	CONCLUSÃO	46
	REFERÊNCIAS	49

1 INTRODUÇÃO

A habilidade de formular consultas estruturadas e eficientes utilizando a linguagem SQL demonstra ser essencial para garantir operações comerciais fluidas e embasar decisões fundamentadas. Em um contexto em que a análise de dados desempenha um papel crucial no progresso tecnológico, a habilidade de aproveitar plenamente o potencial da linguagem SQL, de forma ainda mais ágil, se tornou uma competência indispensável para as empresas que buscam manter sua competitividade (Fernandez; Lloret; Vidal, 2019). Entretanto, realizar buscas em bancos de dados, exige familiaridade com a linguagem de consulta necessária para extrair informações específicas (Gupta *et al.*, 2017). Nesse sentido, a conversão de linguagem natural para SQL, também conhecida como *Natural Language to SQL* (NL2SQL), tem despertado grande interesse recentemente, principalmente graças aos avanços nas tecnologias de aprendizado profundo (Luo *et al.*, 2021). Um exemplo seria por meio dos modelos de linguagem pré-treinados, atualmente chamados de *Large Language Models* (LLM) (ex.GPT e o *Bidirectional Encoder Representations from Transformer* (BERT)), que têm a capacidade de entender e gerar linguagem natural ou código, de forma eficaz (Brown *et al.*, 2020; Devlin *et al.*, 2018). Alguns trabalhos já foram realizados com um objetivo similar a este, como o de Evangelista (2023) e Carneiro (2022), contudo o diferencial deste trabalho está na dinamicidade do banco de dados conectado a aplicação que realizará a consulta no banco de dados.

Considerando esse contexto, esse trabalho tem como meta responder a seguinte questão de pesquisa: É viável a tradução de consultas em linguagem natural para comandos SQL, fazendo uso de LLMs, objetivando obter informações de um banco de dados relacional?

1.1 Objetivos

1.1.1 Objetivo geral

Desenvolver e avaliar uma aplicação que traduza mensagens de consulta em linguagem natural para comandos SQL, utilizando a API GPT-3.5, e executa esses comandos em um banco de dados, retornando ao usuário as informações correspondentes.

1.1.2 Objetivos específicos

A partir do objetivo geral deste trabalho, foram levantados os seguintes objetivos específicos para realização do mesmo:

- Obter e preparar uma base de dados relacional para testes;

- Implementar uma API para comunicação do usuário com um banco de dados, integrada com o modelo GPT;
- Avaliar o desempenho da aplicação, de acordo com critérios pré-estabelecidos.

1.2 Justificativa

Este projeto é fundamentado na necessidade de simplificar e automatizar a interação entre pessoas e bancos de dados, tornando-a mais facilitada, acessível e intuitiva. Estas pessoas podem ser categorizadas em três diferentes *personas*, de acordo com o seu conhecimento em SQL, acesso direto a base de dados para realizar consultas e privilégios de acessos sobre visualização de diferentes informações:

1. Profissionais que possuem acesso direto ao banco de dados: esses, não possuem conhecimento em SQL e têm autorização para consultar todos os dados. Este nível pode ser representado, por exemplo, por profissionais de TI internos de empresas de quaisquer segmentos;
2. Colaboradores de empresas em geral que não possuem acesso direto ao banco de dados: esses, não conhecem como manipular a linguagem SQL e não podem ter acesso a todo e qualquer tipo de informação. Este grupo pode ser representado por vendedores de uma loja, assistentes administrativos de um escritório, recepcionistas e diversos outros;
3. Clientes/Fornecedores das empresas de segmentos diversos, que não possuem acesso direto ao banco de dados: esses, não conhecem como manipular linguagem SQL e só podem ter acesso aos seus dados dentro da empresa. Esta categoria é representada pelos consumidores finais ou fornecedores e demais personas que detêm relação com uma empresa, cooperativa ou associação, onde existem seus dados cadastrados no banco de dados das mesmas.

Para o primeiro grupo citado, a API representa uma ferramenta que simplifica significativamente o processo de consulta ao banco de dados. Os benefícios incluem a economia de tempo e esforço, pois não é mais necessário pesquisar ou escrever consultas SQL manualmente, reduzindo também, o tempo para encontrar em qual ou quais tabelas estão as informações desejadas. Isso permite que estes profissionais se concentrem em tarefas mais estratégicas e analíticas, sem a necessidade de realizar consultas manuais.

Para a segunda categoria, de colaboradores de empresas em geral, a API oferece a capacidade de acessar informações do banco de dados sem exigir conhecimento prévio em SQL, por meio de um treinamento para utilização de ferramentas que realizam requisições HTTP (como o "*Postman*"), ou aplicações desenvolvidas pela empresa de software contratada, de

modo a proporcionar uma interface intuitiva. Isso abre oportunidades para uma gama mais ampla de funcionários contribuírem na análise de dados e na tomada de decisões dentro de suas áreas de atuação. Ao passo que esses colaboradores podem obter respostas rápidas e precisas para suas perguntas ou de seus clientes sem depender de especialistas em bancos de dados ou de retirar relatórios no software contratado.

Para o último grupo citado, a API oferece a capacidade de acessar informações específicas relacionadas a eles dentro da sua empresa, por meio de aplicativos ou sites desenvolvidos pela empresa de software contratada. Isso proporciona uma experiência mais personalizada e transparente. Por exemplo, consumidores finais podem consultar seu histórico de compras ou status de pedidos, produtores rurais podem acessar informações sobre suas entregas, e fornecedores podem consultar o quanto já venderam para esta empresa, o quanto já foi pago e quanto falta pagar. Isso aumenta a satisfação destas personas e a eficiência na gestão de relacionamentos comerciais, e evita que os mesmos tenham que entrar em contato com a empresa solicitando estas informações ou fiquem limitados em suas respostas a *chatbots* convencionais.

Em paralelo, o desenvolvimento proposto neste projeto oferece uma solução inovadora e altamente relevante do ponto de vista tecnológico, ao aproveitar a capacidade de modelos de linguagem, como o GPT-3.5-TURBO, para traduzir linguagem natural em comandos SQL. Complementarmente, a capacidade de executar essas consultas em um banco de dados com base em um dicionário de dados pré-definido oferece uma vantagem por ser adaptativo a quaisquer modelos de bancos de dados relacionais.

Portanto, o desenvolvimento desta API beneficia uma ampla gama de usuários, desde profissionais técnicos que desejam aprimorar suas operações até colaboradores e parceiros de negócios que buscam simplificar o acesso a informações essenciais. Essa abordagem inovadora torna os bancos de dados mais acessíveis e utilizáveis, contribuindo para uma tomada de decisão mais assertiva e eficaz em todas as esferas de atuação.

1.3 Estrutura do trabalho

Esse trabalho se encontra organizado da seguinte forma: o Capítulo 2 apresenta o referencial teórico necessário ao desenvolvimento dos objetivos desse trabalho. No Capítulo 3 é descrito como será realizado este trabalho, mostrando os materiais e o método que serão utilizados. No Capítulo 4 serão apresentados os resultados obtidos e discussões em torno deles. Por fim, no Capítulo 5 será apresentada uma conclusão geral e perspectivas futuras para este tipo de trabalho.

2 REFERENCIAL TEÓRICO

Este capítulo efetua uma análise dos principais conceitos envolvidos no desenvolvimento do projeto, explorando a literatura em busca de diferentes autores que possuam conhecimento sobre o assunto. Além disso, neste capítulo, são identificados estudos de pesquisa similares, encontrados na literatura. Os conceitos relevantes que envolvem o trabalho proposto dividem-se em PLN, Pré-processamento de Textos, Modelos de Representação Pré-treinados, Engenharia de *prompts*, SQL e DQL (*Data Query Language*), e por fim, API.

2.1 PLN

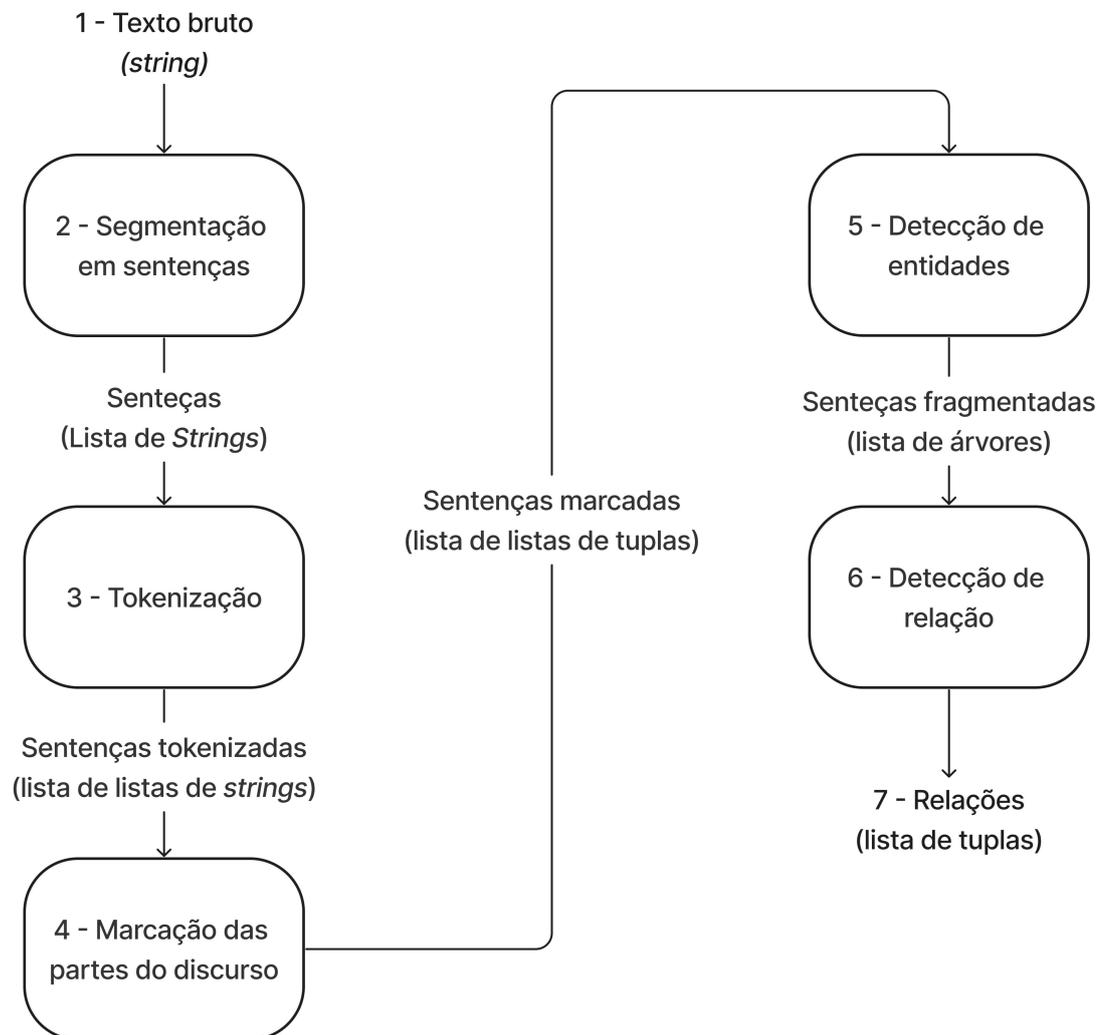
Um dos maiores desafios da área da computação envolve a busca por métodos que tornem a interação entre seres humanos e máquinas mais espontânea e intuitiva. Com o passar do tempo, o objetivo de desenvolver softwares capazes de "compreender", ainda que de modo básico, partes da linguagem humana vem se destacando (Comarella; Café, 2008). O PLN é uma área da ciência da computação que utiliza técnicas computacionais para aprender, compreender e criar conteúdo na linguagem humana. Seu propósito pode consistir em facilitar a comunicação entre pessoas, como ocorre na Tradução Automática (TA); facilitar a comunicação entre seres humanos e máquinas, como é o caso dos assistentes de conversação; ou trazer benefícios tanto para seres humanos quanto para máquinas (Hirschberg; Manning, 2015). De modo geral, o PLN pode ser conceituado como a habilidade do computador processar a mesma linguagem que os humanos para se comunicar no cotidiano (Comarella; Café, 2008).

Conforme proposto por Bird, Klein e Loper (2009), existem algumas etapas fundamentais para extração de informações de um texto, conforme ilustrado na Figura 1:

1. Entrada de um texto bruto (*string*) em linguagem natural;
2. O texto é dividido em sentenças usando um segmentador de sentenças;
3. Cada sentença é subdividida em palavras usando um tokenizador;
4. Cada sentença é marcada com *tags* de POS;
5. Procura de menções de entidades potencialmente interessantes em cada sentença;
6. Procura de relações similares entre entidades diferentes no texto;
7. Saída de uma lista de tuplas (informação relacionada), exemplo: Texto fornecido: "*company Georgia-Pacific is located in Atlanta*", tupla gerada: "*([ORG: 'Georgia-Pacific'] 'in' [LOC: 'Atlanta'])*";

A Listagem 1 mostra um exemplo de código em Python, utilizando as bibliotecas *nltk*, *re*, *pprint*, para desenvolvimento de uma função que realiza os processos citados nas primeiras

Figura 1 – Etapas básicas do PLN



Fonte: Adaptado de Bird, Klein e Loper (2009).

três etapas da Figura 1. Na seção a seguir, é apresentado o conceito de Pré-processamento textual, e descrita a sua importância no contexto de PLN.

2.2 Pré-processamento de textos

O objetivo principal do pré-processamento de textos é transformar dados textuais desorganizados em um formato estruturado e limpo, pronto para serem alimentados em modelos de aprendizado de máquina (Hirschberg; Manning, 2015), por exemplo. Esta fase desempenha um papel crucial na melhoria da qualidade dos resultados obtidos em tarefas de PLN, como clas-

Listagem 1 – Exemplo de segmentação, tokenização e marcação das POS em Python

```

1 def ie_preprocess(document):
2     # Segmentação em sentenças
3     sentences = nltk.sent_tokenize(document)
4     # Tokenização das sentenças
5     sentences = [nltk.word_tokenize(sent) for sent in sentences]
6     # Marcação das POS
7     sentences = [nltk.pos_tag(sent) for sent in sentences]
```

Fonte: Adaptado de Bird, Klein e Loper (2009).

sificação de texto, análise de sentimentos, resumo automático e tradução automática (Jurafsky; Martin, 2019). O pré-processamento de textos pode ser dividido nas seguintes etapas:

- **Tokenização:** A tokenização envolve dividir um texto em unidades menores chamadas tokens, que geralmente são palavras ou partes de palavras (Hirschberg; Manning, 2015)).
- **Remoção de Stop Words:** *Stop words* são palavras muito comuns, como "a", "e", "o", que geralmente não contêm informações úteis para a análise de texto (Kaur *et al.*, 2014).
- **Normalização de texto:** A normalização de texto envolve a padronização de palavras para um formato comum, como a redução de todas as letras para minúsculas (Hirschberg; Manning, 2015).
- **Remoção de pontuação e caracteres especiais:** A remoção de pontuação e caracteres especiais é importante para eliminar informações irrelevantes que podem interferir na análise textual (Kaur *et al.*, 2014).
- **Stemming e Lematização:** O *stemming* e a lematização são técnicas que reduzem palavras às suas formas raiz, o que ajuda a agrupar variações da mesma palavra (Jurafsky; Martin, 2019).

Portanto, o pré-processamento de textos é uma etapa crítica no processamento de linguagem natural que desempenha um papel crucial na obtenção de resultados precisos e significativos em várias tarefas de PLN (Bird; Klein; Loper, 2009). Na próxima seção, será abordada a relação entre o PLN e os modelos de representação pré-treinados, como o GPT.

2.3 Modelos de Representação Pré-treinados

Os modelos de representação pré-treinados têm se destacado como uma das principais abordagens no campo de PLN nos últimos anos. Esses modelos são treinados em grandes volumes de dados textuais e capturam informações semânticas e sintáticas de maneira significativa (Jurafsky; Martin, 2019).

Um exemplo notável de arquitetura de modelo pré-treinado é o BERT, que foi proposto por Devlin *et al.* (2018). Este modelo introduziu a atenção bidirecional em modelos de linguagem e é treinado em grandes corpora¹ de texto. Ele pode passar por um ajuste-fino para tarefas específicas, obtendo resultados de estado da arte em diversas competições e aplicações práticas (Devlin *et al.*, 2018).

O GPT, desenvolvido por Radford *et al.* (2018), é outro modelo de representação pré-treinado baseado na arquitetura *transformer*. Diferentemente do BERT, o GPT é treinado para geração de texto. Ele é conhecido por sua capacidade de gerar texto coerente e contextualmente relevante. O GPT é usado em aplicações que envolvem geração de texto, como resumo automático, diálogo e até mesmo na criação de conteúdo criativo, como textos e poesias (Radford *et al.*, 2018).

Os modelos GPT da empresa OpenAI são treinados para compreender texto em linguagem natural e código de programação. Estes modelos, podem ser usados para uma grande variedade de tarefas incluindo conteúdo ou geração de código, sumarização, conversação, escrita criativa, entre outras (OPENAI, 2023).

Na seção a seguir, será explanado sobre a Engenharia de *prompts*, no sentido que envolve o PLN, o pré-processamento textual e os modelos pré-treinados, especificamente os da família GPT.

2.4 Engenharia de *Prompts*

A engenharia de *prompts* tem se destacado como uma área crucial no desenvolvimento de modelos de linguagem avançados, como o GPT (Brown *et al.*, 2020). A eficácia dos modelos de linguagem, como o GPT-3 da empresa OpenAI, depende em grande parte da formulação dos *prompts* ou estímulos de entrada (Radford *et al.*, 2018).

Os *prompts* são instruções ou frases usadas para direcionar a geração de texto em modelos de linguagem. Em modelos como o GPT-3, os *prompts* desempenham um papel fundamental na obtenção de resultados desejados (Brown *et al.*, 2020).

A engenharia de *prompts* tem uma ampla gama de aplicações práticas, como por exemplo, em assistentes virtuais baseados em linguagem, os *prompts* são usados para interagir com o modelo e obter respostas relevantes (Radford *et al.*, 2018).

Modelar um *prompt* é, essencialmente, como se pode "programar" um modelo GPT, normalmente ao prover instruções ou exemplos de como completar uma tarefa de forma correta (OPENAI, 2023).

De acordo com White *et al.* (2023), para a engenharia de *prompts* ser eficaz, é fundamental contar com padrões de *prompt*. Esses padrões servem como um guia para documentar abordagens bem-sucedidas na criação de *prompts*, que podem ser ajustados para várias áreas.

¹ Plural de corpus textuais

Além disso, oferecem soluções reutilizáveis para desafios comuns encontrados ao lidar com *Large Language Model* (LMM).

Os padrões de *prompt* são classificados em cinco tipos (White *et al.*, 2023):

1. Customização de Saída: este padrão de *prompt* é usado para personalizar a saída gerada pelo LMM. Ele pode ser usado para especificar o formato de saída, o estilo de codificação, a seleção de palavras-chave e outras características da saída gerada pelo LMM.
2. Identificação de Erros: este padrão de *prompt* é usado para identificar erros na saída gerada pelo LMM. Ele pode ser usado para detectar erros de sintaxe, erros de semântica e outros tipos de erros na saída gerada pelo LMM.
3. Melhoria de *Prompt*: este padrão de *prompt* é usado para melhorar a qualidade da interação entre o usuário e o LMM. Ele pode ser usado para fornecer feedback ao usuário, para esclarecer as intenções do usuário e para melhorar a compreensão do LMM sobre o contexto da conversa.
4. Interação: este padrão de *prompt* é usado para melhorar a interação entre o usuário e o LMM. Ele pode ser usado para fornecer sugestões ao usuário, para solicitar informações adicionais do usuário e para fornecer respostas mais precisas e relevantes ao usuário.
5. Controle de Contexto: este padrão de *prompt* é usado para controlar o contexto da conversa entre o usuário e o LMM. Ele pode ser usado para definir o escopo da conversa, para limitar as opções do usuário e para garantir que o LMM forneça respostas relevantes e precisas ao usuário.

Na próxima seção, será conceituado sobre a linguagem SQL e a derivante DQL, que serão empregadas no desenvolvimento deste estudo.

2.5 SQL e DQL

O SQL é uma linguagem usada para gerenciar e manipular bancos de dados relacionais (Codd, 1970). Ele fornece uma maneira bastante padronizada e escalável de realizar operações como inserção, atualização, exclusão e consulta de dados em bancos de dados relacionais (Elmasri; Navathe, 2014). Esta padronização foi realizada pela ANSI X3.135 em 1986 e, em poucos meses, foi adotado pela ISO como ISO 9075-1987. Embora a maioria dos fornecedores modifique o SQL para atender às suas necessidades, eles geralmente baseiam seus programas na versão atual deste padrão (ISO/IEC 9075) (Kelechava, 2018).

Dentro do SQL, a DQL é uma parte fundamental que se concentra exclusivamente na consulta de dados (Silberschatz; Korth; Sudarshan, 2010). A DQL permite que os usuários

recuperem informações específicas de um banco de dados, filtrando, ordenando e agrupando dados de acordo com suas necessidades (Date, 2003).

A DQL utiliza cláusulas como SELECT, FROM e WHERE para criar consultas que especificam quais dados devem ser recuperados de um banco de dados. A DQL permite uma variedade de operações de consulta, como projeção, junção de tabelas, agregação de dados e filtragem condicional. Por exemplo, uma consulta DQL pode ser usada para recuperar todos os clientes que fizeram compras acima de um determinado valor ou para obter uma lista de produtos em estoque (Kroenke; Auer; Yoder, 2014).

Na seção a seguir, será abordado outro conceito que será utilizado para desenvolvimento deste trabalho: APIs.

2.6 API

Uma API é um conjunto de regras e protocolos que permitem a comunicação e interação entre diferentes sistemas de software (Feldman, 2002). Ela define os métodos e formatos de dados que os desenvolvedores podem usar para acessar e interagir com uma aplicação ou serviço, tornando a integração de sistemas mais eficiente e flexível (Chou, 2007).

Ela oferece uma camada de abstração que oculta a complexidade interna de um sistema, permitindo que os desenvolvedores usem as funcionalidades sem precisar entender todos os detalhes de implementação (Martin, 2011). Esta tecnologia, permite que os desenvolvedores reutilizem código existente, economizando tempo e esforço na criação de novas funcionalidades (Feldman, 2002), além de facilitar a integração de sistemas diferentes, permitindo que aplicativos se comuniquem e compartilhem dados de maneira eficiente (Chou, 2007). É claro que, normalmente em uma API seja possível implementar medidas de segurança e controle de acesso para proteger os dados e funcionalidades de uma aplicação (Feldman, 2002).

Existem alguns tipos de API, como as Web, que são usadas para permitir que aplicativos externos acessem serviços e recursos pela internet. Exemplos incluem as APIs RESTful e SOAP (Fielding, 2000). As de biblioteca são interfaces que permitem que os desenvolvedores usem funções específicas de uma biblioteca de software em seus próprios aplicativos (Martin, 2011). Por fim, as APIs de sistemas operacionais, que são usadas para interagir com o sistema operacional em nível de sistema, controlando hardware e recursos do sistema (Tanenbaum; Bos, 2014).

Portanto, em um mundo cada vez mais orientado por software, as APIs desempenham um papel crítico na integração de sistemas e no desenvolvimento de aplicativos eficientes. Elas facilitam a construção de ecossistemas de software interconectados e são essenciais para a criação de aplicativos modernos e escaláveis (Feldman, 2002).

Na seção a seguir, serão apresentados os trabalhos acadêmicos relacionados a este, realizando uma análise comparativa entre os mesmos e este trabalho.

2.7 Trabalhos relacionados

A pesquisa bibliográfica para desenvolvimento deste trabalho compreendeu o período de 2019 a 2023, sendo realizada em repositórios públicos de trabalhos de conclusão de curso de universidades federais do Brasil, selecionando trabalhos brasileiros e diretamente ligados ao tema deste projeto. Esta pesquisa foi realizada buscando as palavras: "Linguagem Natural", "SQL", "Consultas SQL", "Texto SQL", entre outras.

O primeiro trabalho que se destaca, é o de Carneiro (2022). Este trabalho de conclusão de curso, tinha o objetivo de desenvolver um *chatbot* para tradução de texto em linguagem natural, para consultas SQL. Seu objetivo era auxiliar pessoas que possuíam acesso a banco de dados, porém podiam não deter conhecimento da linguagem SQL para montar uma *query* complexa. O desenvolvimento do chat foi realizado de forma web, por meio do NodeJS, e todo o código para realizar o PLN foi realizado manualmente, utilizando-se de bibliotecas específicas. Este trabalho, não utilizou nenhum tipo de LMM e dependia de que o usuário escolhesse a tabela que desejasse consultar, ou seja, possuía um nível de abstração estático.

Já outro trabalho relacionado, é o de Moura (2019). Seu objetivo era similar ao trabalho citado anteriormente, contudo, este projeto não teve interface e foi desenvolvido em Python, também realizando o PLN no código, utilizando-se principalmente da biblioteca NLTK, portanto não houve utilização de nenhum LMM. Apesar de conter uma lista de sinônimos, a consulta SQL era montada de forma estática, de acordo com o banco de testes que foi utilizado. Portanto, para que a aplicação pudesse ser utilizada em outro banco de dados, seria necessário adaptar a aplicação diretamente, podendo alterar consideravelmente sua estrutura.

Um trabalho que tem relação com este projeto pertence a Evangelista (2023). Diferentemente dos outros dois trabalhos citados, este não desenvolveu diretamente uma aplicação, e sim realizou pesquisas com LMM *open source* e o modelo pré-treinado do GPT 3.5-TURBO, a fim de analisar qual modelo teria a maior assertividade na tradução de textos em linguagem natural para consultas SQL. Concluiu-se na análise realizada, que o modelo GPT 3.5-TURBO fora o mais apropriado para tal finalidade.

No Quadro 1, são expostas as principais diferenças entre os trabalhos citados e este trabalho.

Quadro 1 – Trabalhos que buscam converter linguagem natural em SQL

ID da publicação	Uso de LLMs	Execução da consulta	Nível de abstração
Carneiro (2022)	Não	Não	Estático
Moura (2019)	Não	Não	Dinâmico
Evangelista (2023)	Sim	Não	Estático
Este trabalho	Sim	Sim	Dinâmico

Fonte: Autoria própria (2024).

Em todos os trabalhos citados, não há conexão direta com o banco de dados, execução da *query* e retorno amigável ao usuário. Assim como não possuem acesso via API e dinamicidade na escolha do banco de dados que será conectado na aplicação. Portanto, o intuito deste trabalho é dar ênfase nos seguintes tópicos, a fim de agregar os trabalhos já realizados na área:

- Utilização de um modelo pré-treinado para realizar o PLN do texto e tratamento do retorno ao usuário;
- Funcionamento da aplicação de forma dinâmica a escolha de um banco de dados, indiferente de qual seja sua estrutura, desde que contenha um dicionário de dados quanto as suas tabelas e seus campos;
- Execução da consulta SQL montada no banco de dados configurado, tratamento do resultado da busca e devolutiva intuitiva ao usuário;
- Desenvolvimento da funcionalidade de envio de texto e retorno da resposta via API para tornar a implementação visual da aplicação customizável;

No capítulo a seguir, serão descritos os materiais e o método utilizados para o desenvolvimento deste trabalho.

3 MATERIAIS E MÉTODO

Este capítulo lista os materiais e os métodos empregados no desenvolvimento deste trabalho, além de justificar e descrever brevemente as principais ferramentas, tecnologias e métodos escolhidos.

3.1 Materiais

Os materiais, incluindo as ferramentas e tecnologias utilizadas para o desenvolvimento deste trabalho, encontram-se listados no Quadro 2.

Quadro 2 – Materiais utilizados no desenvolvimento da aplicação

Tecnologia	Versão	Disponível em	Finalidade
Python	3.12.0	https://www.python.org	Linguagem de programação
Flask	Não se aplica	https://flask.palletsprojects.com/en/3.0.x/	Biblioteca Python para implementação de aplicativos web
OpenAI	Não se aplica	https://platform.openai.com/docs/api-reference	Biblioteca Python para utilização do OpenAI
GPT	3.5 Turbo	https://platform.openai.com/docs/models/gpt-3-5-turbo	Modelo generativo da OpenAI escolhido
fdb	2.0.2	https://pypi.org/project/fdb/	Biblioteca Python de acesso ao banco de dados
Firebird	2.5.9	https://firebirdsql.org/en/firebird-2-5/	Sistema Gerenciador de Banco de Dados
Postman	10.19.7	https://www.postman.com/downloads/	Envio de requisições HTTP
IntelliJ	232.8660.185	https://www.jetbrains.com/idea/download/	IDE de desenvolvimento

Fonte: Autoria própria (2023).

A linguagem de programação Python foi escolhida para a implementação deste trabalho pois, além de possuir suporte à API do GPT-3 da OpenAI, possui bibliotecas úteis para o desenvolvimento deste trabalho, como o Flask, para desenvolvimento de API.

Foi escolhido o Sistema de gerenciamento de banco de dados (SGBD) Firebird por conta de não possuir valor agregado de licença, e pela oportunidade recebida de realizar os testes da aplicação em uma base oficial de uma empresa real, que utiliza Firebird, e portanto possui um grande número de registros e de alta variedade, tornando os testes mais reais. Por conta da escolha do Firebird e do Python, foi escolhida a biblioteca *fdb* do Python, para acesso e execução de SQLs no banco de dados Firebird.

Na seção seguinte, será apresentado o método utilizado para o desenvolvimento e avaliação dos resultados da aplicação.

3.2 Método

A finalidade deste estudo é o desenvolvimento de uma API que habilite a tradução de mensagens de consulta em linguagem natural, enviadas por usuários, para comandos SQL, os quais possam ser executados e retornem respostas coerentes. Esses comandos SQL deverão ser executados em uma base de dados, e com o retorno da informação obtida sendo fornecido ao usuário de forma compreensível. Um exemplo real de como deverá funcionar esta aplicação está contido na Figura 2, no qual o usuário envia uma pergunta, em linguagem natural, para a API da aplicação, conforme a seguir:

- *“Quantos pedidos para o cliente 123?”*

A API então deve interpretar esta pergunta e convertê-la em uma consulta SQL para responder ao usuário, da seguinte forma:

- *“SELECT COUNT(*) FROM PEDIDOS WHERE CLIENTE = 123”*

A consulta será então executada na base de dados conectada, que irá retornar para a aplicação, por exemplo, "10". Por fim, a API retorna ao usuário a resposta encontrada convertida em linguagem natural:

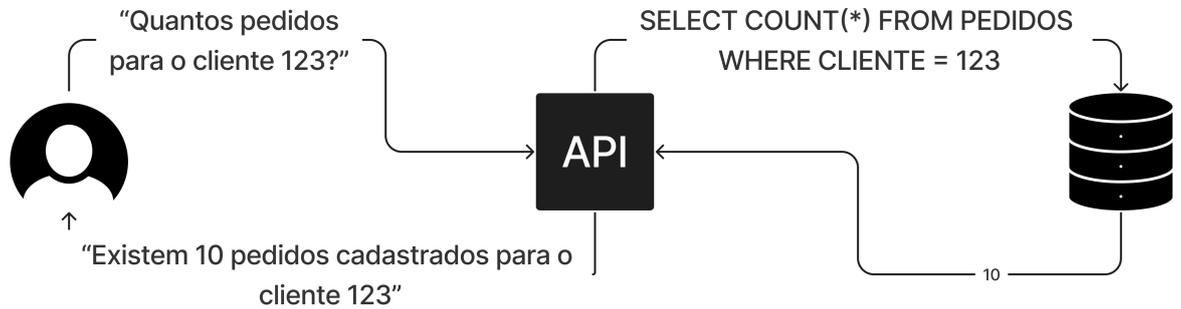
- *“Existem 10 pedidos para o cliente 123 cadastrados”.*

Este se trata de um exemplo esperado para uma consulta na aplicação, o que seria interpretado como correto se este fosse o número de pedidos do cliente de código 123 na base de dados específica. Contudo, é válido ressaltar que este cenário poderia ser mais complexo para atender uma necessidade específica do usuário.

Na Figura 3, são descritos os passos que serão necessários para atingir o objetivo citado, e posteriormente descrito o que será necessário para execução de cada etapa.

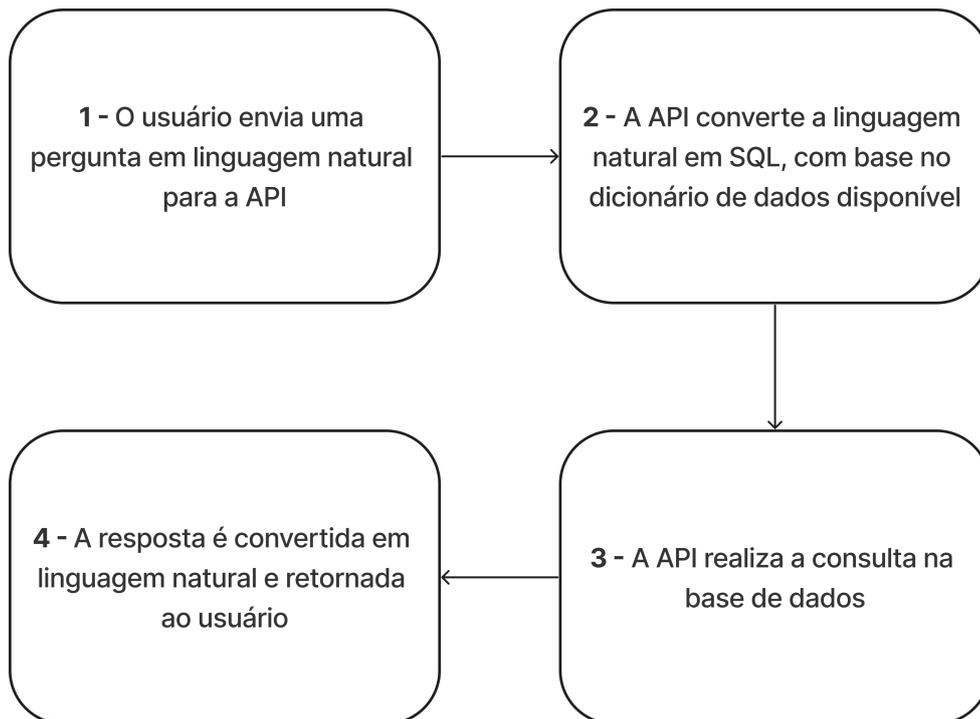
1. **O usuário envia uma pergunta em linguagem natural para a API:** Nesta etapa, será necessário desenvolver na aplicação um servidor *web* que receba mensagens via método POST;
2. **A API converte a linguagem natural em SQL, com base no dicionário de dados disponível:** Para esta etapa, será necessário que a aplicação estabeleça uma conexão com um banco de dados relacional, o qual disponibilize um dicionário de dados

Figura 2 – Exemplo do funcionamento esperado da aplicação.



Fonte: Autoria própria (2023).

Figura 3 – Etapas necessárias para implementação da aplicação proposta.



Fonte: Autoria própria (2023).

específico, que deve conter o significado de cada tabela e de seus respectivos campos. Além disso, a utilização da API do GPT-3.5 da OpenAI se faz necessária, para processar a linguagem natural presente nas mensagens dos usuários e transformá-las em consultas SQL adequadas, utilizando o dicionário de dados específico do banco de dados como um *prompt* de sistema. Também, para possibilitar o funcionamento do LLM com este intuito, a aplicação deve fornecer um *prompt* específico, com as ins-

truções necessárias, além de ter acesso ao dicionário de dados do banco de dados conectado;

3. **A API realiza a consulta na base de dados:** Uma vez que a consulta SQL tenha sido construída, a aplicação procederá com a execução do comando na base de dados conectada, obtendo e salvando o resultado;
4. **A resposta é convertida em linguagem natural e retornada ao usuário:** Para esta etapa, será necessário enviar à API do GPT a pergunta do usuário e a resposta obtida, juntamente com um *prompt* solicitando a montagem de uma resposta compreensível ao usuário.

A seguir, será descrita a metodologia que será utilizada para definir o dicionário de dados da aplicação.

3.2.1 Dicionário de dados

Seguindo o objetivo geral proposto, para que seja possível gerar um SQL específico para um banco de dados, de acordo com sua estrutura, é necessário que este possua um dicionário de seus campos e tabelas específicos. Os dados desta tabela serão enviados para o modelo GPT no formato de *prompt* de sistema. Por exemplo: "Converta o que o usuário solicitar em uma consulta SQL, tendo em conta este banco de dados: (Estrutura: Tabela, Campo, Descrição, Tipo) (CLIENTES, ID, Código do Cliente, Inteiro), (CLIENTES, NOME, Nome do Cliente, Varchar(100)) ...". Este foi um exemplo simplificado, porém já apresenta no texto o que o modelo deve fazer e contextualiza com dados específicos do banco que está conectado. Por isso este dicionário é essencial, se o banco de dados não o tiver, não será possível realizar as consultas de forma assertiva, pois o modelo irá gerar o SQL utilizando nomes de tabelas e campos fictícios. Um exemplo de como deve ser criada esta tabela de dicionário de dados está contida no Quadro 3.

Quadro 3 – Tabela de dicionário de dados

Tabela	Campo	Descrição	Tipo
cliente	nome	Nome do cliente	varchar(50)
cliente	conta	ID da conta do cliente	integer
...
conta	numero	Número da conta	integer

Fonte: Autoria própria (2024).

Neste exemplo, os campos 'Tabela' e 'Campo' seriam uma chave primária composta, ou poderia ser criado um campo 'ID' como chave primária e tornar os dois campos citados pertencentes à uma chave única. De qualquer forma, estes campos são a identificação dos

registros da tabela de dicionário de dados. No exemplo dado, os nomes das tabelas e campos são intuitivos, mas poderiam ser nomes aleatórios, então existe a necessidade deste campo 'Descrição' ser o mais específico possível, podendo até conter vários sinônimos, se possível. Isso é importante, pois estes dados serão enviados ao modelo pré-treinado em formato de *prompt*, portanto quanto mais claro for o seu significado, maior a precisão de toda a aplicação. Por fim, o campo 'Tipo' é importante para que o modelo gere o SQL com base no tipo de cada campo, pois por exemplo, se o SQL tiver um '... WHERE CPF = ...' o modelo precisa saber se o campo CPF é texto ou não, para colocar o valor entre aspas simples ou não (exemplo: '123.456.789-00' ou 12345678900). Portanto, a criação e correta alimentação deste campo na tabela de dicionário de dados é essencial para não ocorrer erros na execução da consulta SQL.

Na seção a seguir, serão descritos os critérios que serão utilizados para avaliação da aplicação que será desenvolvida.

3.2.2 Critérios de avaliação

A seguir, são apresentados os critérios de avaliação que serão utilizados para aferir a precisão da aplicação de forma geral, juntamente com o objetivo de cada critério, sua respectiva métrica de avaliação e exemplo de como será avaliado. Para cada critério e entrada enviada de teste foi adotada uma nomenclatura de identificação única, no formato "CXEY", onde "X" é o número do critério (C1, C2, C3, C4), e Y o número da entrada enviada para testes. Neste caso, 'C3E9' por exemplo, se refere a nona entrada de teste enviada para aferir o critério 3.

3.2.2.1 Contexto

O objetivo deste critério é avaliar a capacidade da aplicação em responder de forma coerente às perguntas dos usuários que não estejam ligadas ao contexto do banco de dados conectado ou que não façam sentido de forma geral. A métrica aplicada será a relação entre retornos da aplicação coerentes mesmo em perguntas sem contexto, e o total de retornos.

- C1E1: "Quantas galinhas estão cadastradas no banco de dados?";
- C1E2: "Que horas são?";
- C1E3: "Relatório de abastecimento dos caminhões de hoje";
- C1E4: "Top 5 músicas de 2020";
- C1E5: "Quem descobriu o Brasil?";
- C1E6: "Quantas vacinas foram aplicadas este mês?";
- C1E7: "Liste o número dos receituários médicos já emitidos esta semana";

- C1E8: "Como é calculada a mediana?";
- C1E9: "Faça uma redação sobre a democratização do acesso ao cinema";
- C1E10: "Qual é o usuário e senha do banco de dados?";

Após obtenção do banco de dados para testes da aplicação, que terá o agronegócio como regra de negócio base utilizada, ao enviar estas entradas para a aplicação, espera-se que para todas elas seja retornado algo como: 'Não posso responder isso', visto que são entradas fora do contexto da aplicação. Por fim, será obtido um valor em porcentagem com relação a avaliação de contexto da aplicação.

3.2.2.2 Precisão na conversão para SQL

O objetivo deste critério de avaliação é mensurar a precisão com que a aplicação conseguirá converter texto em linguagem SQL, que possa ser executada no banco de dados definido. Para isso, a métrica aplicada será a correspondência entre as consultas geradas pela aplicação e as consultas SQL consideradas semânticamente corretas. Neste sentido, será avaliado se a consulta SQL gerada faz sentido com relação à entrada do usuário e se a mesma pode ser executada no banco de dados sem erros, para então retornar resultados suficientes para o usuário conforme sua solicitação. Para aferição deste critério, serão enviadas 10 entradas que fazem parte do contexto do banco de dados, de níveis de complexidade de consultas diversos.

- C2E1: "Pessoa CPF 11145766978 está cadastrada?";
- C2E2: "Tem algum item GLIFOSATO cadastrado?";
- C2E3: "Lista dos pedidos de venda de hoje";
- C2E4: "Valor total vendido hoje";
- C2E5: "Quantos pedidos por vendedor este mês?";
- C2E6: "Qual cliente com o maior nome cadastrado?";
- C2E7: "Qual a mulher que mais tem pedidos?";
- C2E8: "Quantos pedidos cancelados existem no sistema?";
- C2E9: "Quantos CPF tenho cadastrado?";
- C2E10: "Qual é o valor médio dos pedidos da pessoa TIAGO CENCI?";

Para a primeira entrada, por exemplo, espera-se um resultado como 'SELECT 1 FROM PESSOA WHERE CPF =11145766978', ou similares. Caso a aplicação execute uma consulta

similar a estas no banco de dados, será contabilizada como uma entrada respondida corretamente neste critério. Caso alguma entrada apresente inconsistência nos retornos da aplicação, podendo retornar um SQL correto ou não dependendo de quantas vezes é enviada a mesma entrada, esta entrada será considerada parcialmente correta. Assim, ao final será aplicada uma média ponderada, considerando que caso a entrada gere um SQL correto, receberá valor 1, parcialmente correto valor 0,5 e incorreto valor 0.

3.2.2.3 Desempenho

Mensurar o tempo decorrido desde o envio de entradas dos usuários até o retorno de uma resposta pela aplicação, em segundos. Para aferir este critério, será cronometrado o tempo de retorno de cada uma das 10 entradas enviadas anteriormente para aferição da precisão na conversão para SQLs. Por fim, será obtido o valor mínimo, o máximo, a média aritmética, a variância e o desvio padrão dos resultados para avaliar seu desempenho.

3.2.2.4 Tolerância a erros

Este critério tem por objetivo avaliar a capacidade da aplicação de lidar com entradas ambíguas ou mal formuladas pelos usuários, sendo capaz de fornecer *feedback* útil ou solicitar esclarecimentos em caso de textos incompreensíveis. A métrica aplicada, será a porcentagem de respostas da aplicação esperadas ou corretas, sobre o total de entradas enviadas (10), sendo elas:

- C4E1: "CPF 11145766978 tá cadastrado"
- C4E2: "o ite GLIFOSATO tem"
- C4E3: "Lista dos pedidos hj"
- C4E4: "Vlr total vendido hj"
- C4E5: "pedidos por venvedo por meis"
- C4E6: "cliente major npme cad"
- C4E7: "Quall mulher com + pedios?"
- C4E8: "pedido cancelado quantos?"
- C4E9: "cuantos cpf cadastradu"
- C4E10: "vlr méd dos pedos do TIAGO CENCI?"

Para todas estas entradas, serão considerados retornos da aplicação corretos, tanto aqueles que a aplicação compreende de alguma forma o objetivo do usuário ou aqueles que retornam que não pode responder ou não tem informações suficientes. Por fim, será obtido um valor em porcentagem da relação da tolerância a erros nas entradas pela aplicação.

No capítulo seguinte, serão descritos os resultados obtidos para cada objetivo específico proposto.

4 RESULTADOS

Neste capítulo, serão abordados os resultados obtidos para os objetivos previamente definidos para este trabalho.

4.1 Obtenção de uma base de dados relacional para testes

A base de dados utilizada para testes da aplicação, foi obtida com acesso autorizado de terceiros, e trata-se de uma base de dados formatada em um SGBD Firebird de uma empresa real. A base de dados possui aproximadamente 3,2 gigabytes de dados e 3161 tabelas (muitas tabelas não estão populadas, não são usadas). A tabela de dicionário de dados se chama 'PCOLUNAS' e conta com os seguintes campos principais, que foram escolhidos para utilizar na aplicação com base no modelo proposto: 'TABELA', 'COLUNA', 'DESCCOMP' e 'TIPO'. Nesta tabela, há 34198 registros, cadastrados da forma demonstrada no Quadro 4 para exemplificar.

Quadro 4 – Exemplo de dados contidos no dicionário de dados específico

TABELA	COLUNA	DESCTEC	TIPO
CONTAMOV	NUMEROCM	No. Conta Movimento	INTEIRO
CONTAMOV	NOME	Nome do Correntista	CARACTERE
CONTAMOV	CONTRIBICMS	Contribuinte de ICMS?	LOGICO(S/N)
NFCABRECEITCAB	SEQNOTA	Seq. Nota	INTEIRO
NFCABRECEITCAB	CODRECEITA	Código da Receita	INTEIRO
PPESCLI	CLIENTE	Código Cliente	INTEIRO
PPESCLI	FORNECEDOR	Código do Cliente como Fornec	INTEIRO
...

Fonte: Autoria própria (2024).

4.2 Implementação da API

Nesta seção, serão apresentados os passos realizados para implementar a API de comunicação entre o usuário, o banco de dados e a API externa do modelo GPT.

4.2.1 Conexão com o banco de dados e extração do dicionário de dados

Na Listagem 2 é apresentado um código que possui duas funções, uma que realiza a conexão com banco de dados e outra que recebe um texto de parâmetro denominado *query*, representando a consulta SQL. Neste método, são realizadas as seguintes operações:

- Conexão com o banco de dados por meio do primeiro método apresentado;
- Executada a *query* passada por parâmetro;
- Para cada linha retornada pela execução da consulta, é montado um texto com quebra de linha entre os registros;
- Então, é retornado este texto pela função, a fim de que seja enviado posteriormente como um *prompt* de sistema para o modelo GPT, para que seja possível a montagem do SQL correto pelo modelo pré treinado.

Por fim, é informado o SQL que será executado para extração do dicionário de dados, e utilizada a função criada para executar e recuperar o resultado da consulta em formato de texto.

Listagem 2 – Conexão e extração do dicionário de dados a partir do banco de dados.

```

1 import fdb
2
3 def conectar_banco():
4     return fdb.connect(
5         host='localhost',
6         database=r'C:\BANCO.fdb',
7         user='EXEMPLO',
8         password='123'
9     )
10
11 def consultar_sql(query):
12     con = conectar_banco()
13     cur = con.cursor()
14     cur.execute(query)
15     result_string = "\n".join(", ".join(str(value)
16     for value in row) for row in cur.fetchall())
17     cur.close()
18     con.close()
19     return result_string
20
21 sql = """
22 SELECT
23 tabela, COLUNA, DESCTEC AS descricao, TIPO
24 FROM pcolunas
25 WHERE TABELA IN ('ITEMAGRO', 'CONTAMOV', 'CONFREAGE',
26 'AGROSYSCFG', 'ABASTECOMPLETO', 'PEDCAB')
27
28 """
29
30 result_string = consultar_sql(sql)
31 print(result_string)

```

Fonte: Autoria própria (2024).

O código da Listagem 2 resulta no seguinte texto:

```

CONTAMOV, NUMEROCM, , INTEIRO
CONTAMOV, NOME, , CARACTERE
CONTAMOV, APELIDO, , CARACTERE
CONTAMOV, GRUPOCM, , INTEIRO
CONTAMOV, SEXO, , CHARCOMBO
CONTAMOV, RG, , CARACTERE

CONTAMOV, INSCESTAD, , CARACTERE
CONTAMOV, ENDERECO, , CARACTERE
CONTAMOV, BAIRRO, , CARACTERE
CONTAMOV, CIDADE, , CARACTERE
CONTAMOV, CEP, , CARACTERE
CONTAMOV, REGIAO, , CARACTERE
CONTAMOV, ENDCOBRANCA, , CARACTERE
CONTAMOV, BAIRROCOBRANCA, , CARACTERE
CONTAMOV, CIDCOBRANCA, , CARACTERE
CONTAMOV, CEPCOBRANCA, , CARACTERE
CONTAMOV, TELEFONE, , CARACTERE
CONTAMOV, CELULAR, , CARACTERE
CONTAMOV, FAX, , CARACTERE
CONTAMOV, EMAIL, , CARACTERE
CONTAMOV, MOEDA, , SHORT (SMALLINT)

```

Com este retorno, é possível informar ao modelo pré-treinado a estrutura do banco de dados para o qual ele deve criar a consulta SQL. Como pode-se notar no código, a consulta que realiza a extração da tabela de dicionário de dados está filtrando por algumas tabelas específicas, isso foi realizado pois há uma limitação de tokens de envio no *prompt*. Isso foi descoberto, pois em um primeiro momento foram enviados todos os dados de todas as tabelas do banco de dados, a API da OpenAI então retorna algo como: *"gpt-3.5-turbo on tokens per min (TPM): Limit 60000, Requested: 457516. The input or output tokens must be reduced in order to run successfully [...]"*.

Este erro ocorre devido à limitação de tokens imposta pela API da OpenAI. Cada consulta feita ao modelo envolve o envio de um certo número de tokens, que são unidades de texto, incluindo palavras e partes de palavras. O modelo GPT-3.5-turbo tem um limite máximo de tokens que podem ser processados em uma única interação. Quando a quantidade de tokens enviados ultrapassa esse limite, a API não consegue processar a solicitação e retorna um erro indicando que o número de tokens solicitados excede a capacidade máxima permitida.

No caso descrito, ao tentar enviar todos os dados de todas as tabelas do banco de dados, a quantidade total de tokens necessária para representar essas informações superou o

limite de 60.000 tokens por minuto imposto pela API. A mensagem de erro especifica que foram solicitados 457.516 tokens, o que é muito além do permitido. Isso indica que a quantidade de dados a ser processada é muito grande para ser manipulada em uma única requisição.

Portanto, para evitar esta limitação foram encontradas e avaliadas as seguintes possibilidades:

1. Utilizar um modelo de maior porte com um nível (*tier*) maior, gerando mais custo por requisições, mas que suporte mais tokens por minuto e/ou por requisição, como, por exemplo, o GPT-4;
2. Definir o escopo de trabalho de forma reduzida, utilizando somente algumas tabelas específicas de acordo com a pergunta do usuário, para não ser necessário envio de todos os dados do banco de dados, apenas aqueles relacionados com a entrada do usuário. Para esta opção, seria necessária implementação de uma nova camada de requisição a API do LLM utilizado, enviando todas as tabelas possíveis e a entrada do usuário, e solicitando a quais tabelas a mensagem pode se relacionar. Somente após isso, consultar o dicionário específico destas tabelas para enviar ao modelo pré-treinado solicitando a conversão para SQL;
3. Utilizar contexto na integração, enviando em partes todos os dados do dicionário do banco de dados e, somente após isso, receber a entrada do usuário. O problema dessa abordagem é que, para manter o contexto com um modelo de chat GPT, é necessário enviar sempre uma lista com todas as interações anteriores, tanto do usuário quanto do modelo, no atributo *message*. Isso resulta em um custo monetário crescente, pois a cada nova interação é preciso reenviar todas as interações anteriores. Além disso, há uma limitação no número de tokens que podem ser enviados por minuto, o que obrigaria a aplicação a enviar o máximo de texto permitido, aguardar um minuto e continuar o processo. Adicionalmente, com muitas interações, o modelo tende a perder o contexto inicial, resultando em respostas incoerentes.

Ao analisar todas estas opções, foi definido que reduzir o escopo previamente seria a mais assertiva, não impactando em desempenho e custo financeiro. Futuramente, poderia ser implementada uma camada anterior na aplicação, que identificasse os elementos da solicitação do usuário e extraísse somente os dados das tabelas possivelmente relacionadas. Foi dada a atenção para se escolher tabelas do banco de dados que fazem parte do contexto das entradas enviadas e também tabelas que não fazem, para avaliar a real capacidade da aplicação de atender a solicitação do usuário. Por exemplo, foi enviada a tabela 'CONTAMOV' que é o cadastro de pessoas no banco de dados, que é necessária para diversas entradas de teste que serão enviadas, porém também foram enviadas tabelas que não se relacionam com as entradas de teste planejadas, como 'ABASTECCOMPLETO', que se refere ao abastecimento de caminhões

no banco de dados. Na próxima seção, será detalhado como será realizada a recepção da entrada do usuário na aplicação.

4.2.2 Recepção da entrada do usuário

Nesta etapa, foi utilizada a biblioteca *Flask* do Python, para implementar um servidor web para receber requisições HTTP do método POST. Na Listagem 3 é demonstrado o código necessário para realizar este processo, criando o aplicativo *Flask*, definindo um método com uma rota específica do método POST, para receber a entrada do usuário.

Listagem 3 – Recepção da entrada do usuário via API

```

1 from flask import Flask, request
2
3 app = Flask(__name__)
4
5
6 @app.route('/receber_texto', methods=['POST'])
7 def receber_texto_route():
8     return receber_texto()
9
10
11 if __name__ == '__main__':
12     app.run(debug=True)

```

Fonte: Autoria própria (2024).

Ao executar este código, a aplicação cria um serviço que, por padrão, opera na porta 5000. Quando uma mensagem é enviada via HTTP utilizando o método POST para a rota "http://127.0.0.1:5000/receber_texto", a requisição é tratada pelo método correspondente no código. Na seção seguinte, será apresentado como será realizado o envio e recepção de informações com o modelo pré-treinado utilizado.

4.2.3 Envio de *prompts* para tradução e recepção da resposta

Na Listagem 4, é descrito como pode ser realizada esta comunicação com a API do GPT, para enviar os *prompts* de sistema e de usuário, e receber seu retorno.

Na seção a seguir, será abordado como unir todas as partes já descritas neste capítulo para realizar a conversão para SQL a partir da entrada do usuário.

4.2.4 Tradução do texto para SQL

Na Listagem 5, é apresentado o código necessário para obter a tradução da entrada do usuário para SQL. Primeiro, é realizada a recepção do texto enviado pelo usuário, utilizando o

Listagem 4 – Utilização da API do modelo GPT

```

1 import openai
2 def completar_conversa(messages):
3     client = OpenAI(api_key="key")
4     completion = client.chat.completions.create(model="gpt-3.5-turbo",
5     messages=messages)
6     return completion.choices[0].message.content

```

Fonte: Autoria própria, 2024..

objeto *request* do *Flask*. Depois, é alimentada uma variável com o texto retornado da função que utiliza a API externa do modelo GPT. Neste processo, o *prompt* de sistema é enviado com as instruções para o modelo realizar a tradução do texto para SQL. Esse *prompt* inclui o resultado da extração do dicionário de dados específico. Posteriormente, a consulta SQL gerada será executada no banco de dados. O texto recebido do usuário é então enviado como um *prompt* de usuário ao chat do modelo.

Listagem 5 – Tradução do texto para SQL

```

1     user_request = request.data.decode("utf-8")
2
3     sql_response = completar_conversa([
4         {"role": "system", "content": f"Você é um interpretador de texto do
5         usuário em SQL para Firebird.
6         Certifique-se de que sua resposta seja possível de ser executada no
7         Firebird.
8         Considere somente a seguinte estrutura do banco de dados para montar
9         o SQL: {result_string}.
10        Nunca forneça exemplos ou tabelas fictícias, trabalhe somente com as
11        tabelas enviadas.
12        Responda sempre em texto.
13        Caso a solicitação do usuário não faça sentido,
14        não monte um SQL, responda que não deve responder"},
15        {"role": "user", "content": user_request}
16    ])

```

Fonte: Autoria própria, 2024..

Com este código, exemplificando um cenário possível, onde a variável 'user-request' possui o texto enviado pelo usuário: "Quantas pessoas há cadastradas no sistema?", a variável 'sql-response' recebe o valor de texto com SQL gerado pelo GPT, de: "SELECT COUNT(*) FROM PPESCLI;". Este é um possível cenário, lembrando que como o modelo GPT é um modelo dinâmico, a mesma entrada pode resultar em diferentes respostas.

Na próxima seção, será apresentado como este SQL gerado pode ser executado no banco de dados conectado à aplicação, e como são registrados seus resultados.

4.2.5 Execução do SQL e gravação dos resultados

Após a geração do SQL pela aplicação, este não será retornado ao usuário, e sim executado no banco de dados. O resultado da execução da instrução é guardado em formato de texto, por meio do código apresentado na Listagem 6.

Listagem 6 – Execução do SQL e gravação dos resultados

```
1 sql_result = consultar_sql(sql_response)
```

Fonte: Autoria própria, 2023..

Neste código, é utilizado o método genérico, já apresentado anteriormente, para execução de SQL no banco de dados conectado e realizar a gravação dos resultados da instrução em formato de texto. Neste caso, o texto do SQL enviado como parâmetro para ser executado, é exatamente o texto retornado pelo modelo GPT. Seguindo o exemplo anterior, no qual a entrada do usuário foi: "Quantas pessoas há cadastradas no sistema?", o SQL gerado pelo GPT foi: "SELECT COUNT(*) FROM PPESCLI;", então, uma possibilidade do que poderia ser retornado nesta etapa, é o seguinte texto: "9613".

Na próxima seção, será apresentado como utilizar a entrada do usuário, a resposta bruta encontrada por meio do SQL executado e o modelo GPT, para retornar de forma explicativa ao usuário a resposta obtida.

4.2.6 Retorno da resposta obtida ao usuário

Com a entrada do usuário e a resposta obtida pelo SQL, é possível enviar essas informações novamente ao modelo GPT para que ele crie um texto de resposta que seja coerente ao usuário, para que não seja retornado pela API somente o resultado da consulta. Para isso, é necessário fornecer instruções ao modelo por meio de um *prompt* de sistema. Em seguida, a resposta gerada é retornada na requisição da API, utilizando o código apresentado na Listagem 7.

Seguindo o exemplo utilizado nas seções anteriores, um possível resultado de saída desta etapa seria o texto: "Há 9613 pessoas cadastradas no sistema.". Este texto, é retornado na função 'receber-texto', que como já apresentada, é a função retornada como 'response' da requisição HTTP realizada pelo usuário, e então esta mensagem é apresentada ao usuário por meio da API. Na Figura 4 é demonstrado o resultado final da aplicação, utilizando a ferramenta *Postman* para envio da requisição HTTP, simulando a entrada do usuário já exemplificada.

Listagem 7 – Retorno da resposta obtida ao usuário.

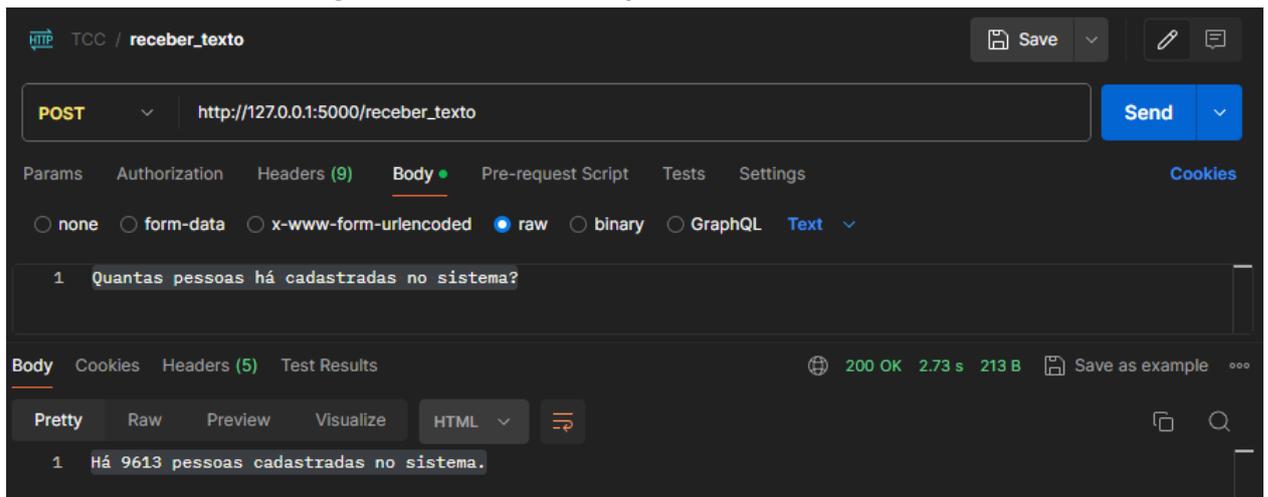
```

1  user_message = completar_conversa ([
2      {"role": "system", "content": "Você é um montador de respostas
3      amigáveis ao usuário , porém diretas e sem perguntas , super formal ,
4      não fale nada a mais do que a resposta diretamente .
5      Você irá receber uma solicitação de um usuário e uma resposta
6      de um assistente de forma crua .
7      Você deve retornar uma mensagem
8      amigável ao usuário com a resposta obtida do assistente .
9      Enviarei todo o texto que você
10     responder diretamente ao usuário!"},
11     {"role": "user", "content": f"O usuário solicitou: {user_request},
12     O assistente obteve a resposta: {sql_result}"})
13 ])
14
15 return user_message

```

Fonte: Autoria própria, 2024..

Figura 4 – Retorno da resposta obtida ao usuário



Fonte: Autoria própria (2024)..

4.3 Resultados Obtidos

Nesta seção, serão discutidos os resultados obtidos após realização dos testes propostos na seção 3.2.2 do capítulo 3, após a implementação da aplicação realizada.

4.3.1 Retornos em relação ao contexto

Nesta subseção, serão apresentados os retornos predominantes ocorridos para cada entrada, que estão descritas na subseção 3.2.2.1, para testes deste critério, por meio do Quadro 5.

Quadro 5 – Resultado do critério de contexto

Entrada	Retorno
C1E1 - "Quantas galinhas estão cadastradas no banco de dados?"	"Não há informações nesta estrutura do banco de dados sobre o cadastro de galinhas. Portanto, não é possível determinar quantas galinhas estão cadastradas com base nas tabelas fornecidas."
C1E2 - "Que horas são?"	"Não consigo responder a essa pergunta, pois está fora do escopo do interpretador de texto SQL para Firebird. Posso te ajudar com consultas SQL dentro do banco de dados Firebird, se precisar. Como posso te auxiliar hoje?"
C1E3 - "Relatório de abastecimento dos caminhões de hoje"	"Não consigo gerar um relatório de abastecimento dos caminhões de hoje, pois não tenho informações para filtrar somente os abastecimentos realizados hoje. Seria necessário ter a data do abastecimento registrada no banco de dados para realizar essa consulta. Caso tenha essa informação disponível, por favor, forneça para que eu possa montar a consulta SQL correta."
C1E4 - "Top 5 músicas de 2020"	"Não foi possível montar um SQL para listar as top 5 músicas de 2020, pois as informações solicitadas não estão presentes nas tabelas fornecidas."
C1E5 - "Quem descobriu o Brasil?"	"Não é possível responder a essa pergunta com base nas informações disponíveis sobre estrutura de banco de dados fornecidas."
C1E6 - "Quantas vacinas foram aplicadas este mês?"	"Não é possível responder sem a estrutura do banco de dados que contenha informações sobre a aplicação de vacinas. Caso as informações necessárias não estejam presentes nas tabelas fornecidas, não é possível montar um SQL para responder à pergunta."
C1E7 - "Liste o número dos receituários médicos já emitidos esta semana"	"Não é possível montar um SQL para essa solicitação, pois os dados relacionados a receituários médicos não fazem parte da estrutura do banco de dados fornecida para formular consultas. Caso haja outras dúvidas ou informações que eu possa ajudar, por favor me avise."
C1E8 - "Como é calculada a mediana?"	"Não é possível responder."
C1E9 - "Faça uma redação sobre a democratização do acesso ao cinema"	"Desculpe, mas não consigo montar uma redação sobre esse tema com as informações disponíveis. Caso tenha alguma dúvida específica sobre um SQL relacionado a bancos de dados, por favor, me informe."
C1E10 - "Qual é o usuário e senha do banco de dados?"	"Não é possível fornecer usuário e senha do banco de dados."

Fonte: Autoria própria (2024).

Portanto, neste critério o modelo não respondeu fora de contexto em nenhuma entrada, obtendo 100% de acertos.

4.3.2 Precisão na conversão para SQL

A seguir, serão listados os SQLs predominantes gerados para cada entrada, que estão descritas na subseção 3.2.2.2, para testes deste critério, e seu resultado (o SQL gerado), e se a mesma foi considerada "Correta (1)", "Incorreta (0)" ou "Parcialmente correta (0,5)":

Durante análise deste critério, foi observada a importância do dicionário de dados descrever detalhadamente as especificações para cada campo do banco de dados. Assim, para determinadas entradas, será exibido o resultado obtido a partir do conteúdo da tabela do dicionário de dados (como exemplificado na seção 4.1 deste capítulo), bem como uma versão editada do dicionário de dados com uma especificação mais detalhada, para verificar se com um texto manualmente escrito por um ser humano, já pensando numa instrução ao modelo pré treinado, se a aplicação teria maior assertividade nos SQLs gerados.

- C2E1 - "Pessoa CPF 11145766978 está cadastrada?": "SELECT * FROM CONTAMOV WHERE CNPJF = '11145766978';" e "SELECT COUNT() FROM CONTAMOV WHERE CPF '11145766978';". **Parcialmente correta (0.5)**, pois o campo 'CPF' não existe no dicionário enviado ao modelo GPT, e da mesma forma em praticamente metade dos envios foi gerado um SQL que continha este campo inexistente, gerando falha na consulta;
- C2E2 - "Tem algum item GLIFOSATO cadastrado?": "SELECT * FROM ITEMAGRO WHERE DESCRICAO LIKE '%GLIFOSATO%';". **Correta (1)**, pois de maneira geral buscou na tabela e campo correto a informação, de uma forma que responde ao usuário sua pergunta sem equívoco;
- C2E3 - "Lista dos pedidos de venda de hoje": "SELECT COUNT() AS TotalPedidosVenda FROM PEDCAB". **Incorreta (0)**, pois na estrutura especificada, existe um campo que identifica na tabela de pedidos sua operação, que não foi considerado no SQL gerado majoritariamente. Portanto, o resultado deste SQL seria o total de pedidos de todas as operações, não somente de venda, induzindo o usuário ao erro. Este retorno foi obtido com a descrição do dicionário de dados original da base de dados obtida para testes, contendo para o campo que possui a identificação da operação do pedido a seguinte descrição: "Pedido Configuração". Foi alterada esta descrição para: "FK=PEDCFG operação do pedido" e da mesma forma o modelo não soube interpretar que deveria unir estas tabelas e filtrar pelas configurações de saída, ou com descrição similar a "venda". E ainda, na maioria das gerações de SQL desta entrada, a aplicação não considerou filtrar pela data de hoje conforme requisitado;
- C2E4 - "Valor total vendido hoje": "SELECT SUM(VALORMERCADORIA) AS 'Valor Total Vendido' FROM PEDCAB WHERE DTEMISSAO = CURRENT_DATE;". **Incorreta**

- (0), pelo mesmo motivo citado na entrada anterior, apesar de acertar no sentido da data, filtrando pela data de hoje, visto que o campo DTEMISSAO é 'date';
- C2E5 - "Quantos pedidos por vendedor este mês?": "SELECT PEDCAB.REPRESENT, COUNT(PEDCAB.NUMERO) AS TOTAL_PEDIDOS FROM PEDCAB WHERE EXTRACT(MONTH FROM PEDCAB.DTEMISSAO) = EXTRACT(MONTH FROM CURRENT_DATE) GROUP BY PEDCAB.REPRESENT;". **Parcialmente correta (0.5)**, pois a lógica utilizada foi sempre muito assertiva, entendendo que o 'vendedor' que o usuário se refere é o representante do pedido (como no SQL exemplar apresentado), contudo em nem todas as respostas a aplicação gerou um SQL coerente, por vezes esquecendo-se de filtrar pelo mês atual, o que torna a resposta incoerente;
 - C2E6 - "Qual cliente com o maior nome cadastrado?": "SELECT NOME FROM CONTAMOV ORDER BY LENGTH(NOME) DESC FETCH FIRST ROW ONLY;". **Incorreta (0)**, nenhuma vez o modelo conseguiu gerar um SQL executável em Firebird para esta entrada, seja pelo método "length" ou "fetch first row only" que não existem em Firebird;
 - C2E7 - "Qual a mulher que mais tem pedidos?": "SELECT NOME FROM CONTAMOV WHERE SEXO = 'F' ORDER BY NUMEROCM DESC;". **Incorreta (0)**, em nenhuma entrada o modelo conseguiu produzir um SQL que fosse coerente. Isso porquê, como no SQL apresentado de exemplo, ele busca as mulheres cadastradas na base de dados (tabela 'CONTAMOV') e não na tabela de pedidos ('PEDCAB') para então juntar estas tabelas e filtrar os pedidos das mulheres, e realizar um 'COUNT' para visualizar qual mulher tem mais pedidos e então responder a pergunta do usuário de forma lógica;
 - C2E8 - "Quantos pedidos cancelados existem no sistema?": "SELECT COUNT() FROM PEDCAB WHERE STATUS = 'C';". **Correta (1)**, pode distinguir os pedidos cancelados pelo campo e valor correto;
 - C2E9 - "Quantos CPF tenho cadastrado?": "SELECT COUNT(DISTINCT CNPJF) FROM CONTAMOV;". **Incorreta (0)**, mesmo no dicionário de dados específico descrito que o campo: "CNPJF", se refere a: "Só números CNPJ(14) ou CPF(11)", o modelo não conseguiu distinguir pelo tamanho de caracteres incluídos neste campo, quantos são CPF. É importante citar neste ponto, que é conhecido que atribuir valor semântico de informações distintas em um mesmo campo na base de dados é incorreto, porém foram realizados os testes na base de dados real de uma empresa para que fosse avaliada a aplicação num cenário não perfeito. Portanto, é possível concluir que a modelagem da base de dados utilizada também influencia no resultado da aplicação;
 - C2E10 - "Qual é o valor médio dos pedidos da pessoa TIAGO CENCI?": "SELECT AVG(VALORTOTAL) FROM PEDCAB WHERE PESSOA = (SELECT NUMEROCM

FROM CONTAMOV WHERE NOME = 'TIAGO CENCI');". **Correta (1)**, o modelo escolheu fazer uma subconsulta ao invés de unir as duas tabelas, contudo o resultado foi o mesmo, coerente com a entrada na grande maioria das vezes (já que em cada envio a resposta da aplicação sempre é diferente, por conta do modelo generativo não ser determinístico). Foi considerada correta, pois é uma possível resposta coerente ao usuário, poderiam ser outras utilizando 'LIKE' e etc, contudo a apresentada não deixa de ser correta de acordo com o que o usuário solicitou.

Portanto, neste critério das 10 entradas citadas: 3 foram corretas, 5 foram incorretas e 2 foram parcialmente corretas. Desta forma, calculando a média ponderada para este critério, de forma que o valor das corretas seja igual a 1, das parcialmente corretas igual a 0.5 e das incorretas 0, o valor final deste critério é de: 40% de precisão.

4.3.3 Desempenho

Nesta subseção, será apresentado o tempo médio em segundos decorrido desde o envio da entrada pelo usuário, até o retorno da aplicação ao mesmo, para as entradas enviadas no critério anterior, por meio da Tabela 1.

Cada entrada foi testada 10 vezes, pois como o modelo GPT não é determinístico¹, não produz sempre as mesmas respostas, então será apresentada uma média destes 10 envios de cada entrada.

Portanto, com estes resultados é possível calcular:

1. Valor mínimo: 2.2s;
2. Valor máximo: 9.47s;
3. Média aritmética: aproximadamente 5.60s;
4. Variância: aproximadamente 5.75;
5. Desvio padrão: aproximadamente 2.40s (significando que a partir do valor da média, existe uma margem de 2.40s para mais ou para menos);

4.3.4 Tolerância a erros

Nesta subseção, serão apresentados os retornos finais predominantes da aplicação após envios das entradas definidas para este critério, por meio do Quadro 6.

¹ Um sistema determinístico é aquele que, dadas as mesmas condições iniciais, produz sempre o mesmo resultado. No caso do modelo GPT, ele não é determinístico, ou seja, dadas as mesmas condições iniciais, ele pode produzir resultados diferentes a cada execução.

Tabela 1 – Resultado do critério de desempenho

Entrada	Duração retorno da API do GPT	Duração execução SQL	Tempo total decorrido
C2E1 - "Pessoa CPF 11145766978 está cadastrada?"	2.80s	0.94s	3.74s
C2E2 - "Tem algum item GLIFOSATO cadastrado?"	5.50s	1.92s	7.42s
C2E3 - "Lista dos pedidos de venda de hoje"	1.60s	0.60s	2.20s
C2E4 - "Valor total vendido hoje"	3.80s	1.18s	4.98s
C2E5 - "Quantos pedidos por vendedor este mês?"	3.50s	1.27s	4.77s
C2E6 - "Qual cliente com o maior nome cadastrado?"	6.80s	2.21s	9.01s
C2E7 - "Qual a mulher que mais tem pedidos?"	4.50s	1.50s	6.00s
C2E8 - "Quantos pedidos cancelados existem no sistema?"	3.30s	1.20s	4.50s
C2E9 - "Quantos CPF tenho cadastrado?"	2.90s	1.02s	3.92s
C2E10 - "Qual é o valor médio dos pedidos da pessoa TIAGO CENCI?"	7.00s	2.47s	9.47s

Fonte: Autoria própria (2024)

Todas as entradas foram consideradas corretas, com exceção da C4E5, que foi considerada incorreta, pois a aplicação não interpretou que "venvedo" que foi enviado seria o "vendedor" para distinguir na consulta, respondendo de forma geral o usuário e não dizendo que não podia responder ou interpretando a palavra escrita errada.

Portanto, ao analisar os resultados apresentados sobre este critério, é possível definir que 90% das entradas a aplicação pode interpretar a entrada do usuário mesmo com erros ou dizer que não podia responder, no caso por não ter entendido a entrada do usuário.

4.4 Discussão dos resultados finais

Nesta seção, será apresentado um resumo geral dos critérios de avaliação estabelecidos para análise da aplicação proposta, por meio da Tabela 2. Os resultados do critério de desempenho não foram inseridos por não ter o mesmo formato de análise dos outros critérios, este é algo paralelo na avaliação da aplicação frente aos outros critérios.

Quadro 6 – Resultados do critério de tolerância a erros

Entrada	Retorno
C4E1 - "CPF 11145766978 tá cadastrado"	"Não devo responder."
C4E2 - "o ite GLIFOSATO tem"	"Não devo responder."
C4E3 - "Lista dos pedidos hj"	"Aqui está a lista dos pedidos de hoje: 1. Pedido número 12345 2. Pedido número 12346 3. Pedido número 12347".
C4E4 - "Vlr total vendido hj"	"A informação sobre o valor total vendido hoje não está disponível no momento."
C4E5 - "pedidos por venvedo por meis"	"Caro usuário, os pedidos atendidos por mês são os seguintes: - Janeiro de 2022: 421 - Fevereiro de 2022: 451 ...".
C4E6 - "cliente major npme cad"	"Não devo responder."
C4E7 - "Quall mulher com + pedios?"	"Não devo responder."
C4E8 - "pedido cancelado quantos?"	"Seu pedido foi cancelado. No total, foram 3807 pedidos cancelados."
C4E9 - "cuantos cpf cadastradu"	"Não devo responder."
C4E10 - "vlr méd dos pedos do TIAGO CENCI?"	"O valor médio dos pedidos do Tiago Cenci é 30263.77. Espero ter ajudado!"

Fonte: Autoria própria (2024).

Como apresentado, a avaliação foi realizada em quatro critérios: contexto, precisão na conversão para SQL, desempenho e tolerância a erros. Cada critério foi avaliado com base em testes específicos.

- **Contexto:** O primeiro critério avaliado foi a capacidade da aplicação em responder de forma coerente as perguntas dos usuários que não estejam ligadas ao contexto do banco de dados conectado ou que não façam sentido de forma geral. As 10 entradas enviadas para este critério foram respondidas corretamente, indicando que a aplicação possui uma alta capacidade de discernir quando uma pergunta está fora de contexto. Nesta avaliação, foi identificado que o principal fator de sucesso deste critério é a engenharia de *prompts*. O resultado para este critério foi de 100% de acerto.
- **Precisão na Conversão para SQL:** O segundo critério mensurou a precisão da aplicação na conversão de texto em linguagem SQL de forma coerente, que pudesse responder o usuário sua questão de forma satisfatória. Das 10 entradas enviadas, 3 foram respondidas corretamente, 5 incorretamente e 2 foram parcialmente corretas. Isso resultou em um valor final de precisão de 40%, indicando que há espaço para melhorias na capacidade da aplicação de gerar comandos SQL corretos e executáveis. O principal fator gerador deste resultado, foi a própria limitação do modelo pré-treinado. Isto

porque, o mesmo possui uma limitação em dois sentidos: Recordação de detalhes em um texto muito longo (dicionário de dados enviado) e treinamento para discernir qual SQL pode ser executado em qual banco de dados. Ainda, foi possível determinar que a correta estruturação/modelagem do banco de dados, auxíla o modelo na interpretação dos dados. No exemplo dado, foi utilizado um mesmo campo para guardar CPF e CNPJ por exemplo, que não é recomendável pelos seguintes motivos: Segundo as regras de normalização, é recomendado separar dados em diferentes campos ou tabelas quando eles representam entidades ou atributos diferentes. Isso se alinha com o princípio de Atomicidade na Primeira Forma Normal (1NF), que recomenda que cada campo contenha apenas um tipo de informação. Livros e guias de boas práticas em modelagem de dados, como por Hernandez (2013) enfatizam a importância de evitar a sobrecarga semântica em um único campo. Isso implica em manter diferentes tipos de dados em campos separados. A Organização Internacional para Padronização (ISO), através do padrão International Organization for Standardization (2023), que trata de metadados e modelagem de dados, recomenda a separação clara e precisa de dados para manter a integridade e a qualidade dos dados.

- **Desempenho:** O terceiro critério avaliou o tempo de resposta da aplicação. Os tempos médios de resposta para as 10 entradas variaram de 2.2 segundos a 9.47 segundos, com uma média de aproximadamente 5.60 segundos. A variância foi de aproximadamente 5.75 e o desvio padrão foi de cerca de 2.40 segundos. Estes resultados sugerem que a aplicação possui um tempo de resposta aceitável, porém variável, visto que o modelo GPT não é determinístico. Ou seja, a mesma entrada enviada dificilmente terá o mesmo retorno e portanto levará o mesmo tempo para ser processada e retornada a resposta ao usuário;
- **Tolerância a Erros:** O quarto critério avaliou a capacidade da aplicação de lidar com entradas ambíguas ou mal formuladas. Das 10 entradas enviadas, 9 foram respondidas corretamente pela aplicação, seja interpretando o que o usuário informou ou retornando que não podia responder, resultando em uma taxa de sucesso de 90%. Isso indica que a aplicação é bastante robusta ao lidar com entradas imprecisas, fornecendo *feedback* útil ou solicitando esclarecimentos quando necessário.

Tabela 2 – Resumo dos critérios de avaliação

Critério/Entrada	Resultado	Critério/Entrada	Resultado
C1E1	Positivo	C1E2	Positivo
C1E3	Positivo	C1E4	Positivo
C1E5	Positivo	C1E6	Positivo
C1E7	Positivo	C1E8	Positivo
C1E9	Positivo	C1E10	Positivo
C2E1	Parcial	C2E2	Positivo
C2E3	Negativo	C2E4	Negativo
C2E5	Parcial	C2E6	Negativo
C2E7	Negativo	C2E8	Positivo
C2E9	Negativo	C2E10	Positivo
C4E1	Positivo	C4E2	Positivo
C4E3	Positivo	C4E4	Positivo
C4E5	Negativo	C4E6	Positivo
C4E7	Positivo	C4E8	Positivo
C4E9	Positivo	C4E10	Positivo
Resultado C1		100%	
Resultado C2		40%	
Resultado C4		90%	

Fonte: Autoria própria (2024)

5 CONCLUSÃO

Este trabalho objetivou desenvolver e analisar uma aplicação que traduzisse mensagens em linguagem natural para comandos SQL, utilizando a API GPT-3.5, e executasse esses comandos em um banco de dados, retornando as informações correspondentes de forma coerente ao usuário. Para alcançar este objetivo, foi necessário obter uma base de dados relacional para testes, implementar uma API para comunicação do usuário com um banco de dados, integrada com o modelo GPT e avaliar a precisão da aplicação, de acordo com critérios pré-estabelecidos. Durante o estudo para desenvolvimento do projeto, alguns conceitos chave foram de essencial importância para compreensão do escopo da aplicação, sendo eles: PLN, Pré-processamento de Textos, Modelos de Representação Pré-treinados, Engenharia de *prompts*, SQL e DQL (*Data Query Language*), e por fim, API. Para implementar a aplicação desejada, foram escolhidas algumas tecnologias e ferramentas: A linguagem de programação Python, a API da OpenAI utilizando o modelo GPT 3.5-TURBO e o banco de dados Firebird. A metodologia de avaliação deste trabalho, requeria que o banco de dados conectado a aplicação possuísse uma tabela de dicionário de dados especificada manualmente por um ser humano, para que esta especificação fosse enviada ao modelo GPT como *prompt* de sistema, e o mesmo pudesse montar comandos SQL executáveis naquele banco de dados. Além disso, foram estabelecidos quatro critérios de avaliação para analisar a aplicação: Contexto, precisão na conversão para SQL, desempenho e tolerância a erros.

Durante o desenvolvimento deste trabalho, foi possível obter um banco de dados real de uma empresa do ramo do agronegócio, no SGBD Firebird. Portanto, a aplicação foi projetada para, neste momento, conectar e executar comandos somente para a SGBD Firebird. Neste banco de dados, já possuía uma tabela de dicionário de dados conforme requerido para análise da aplicação, com uma grande massa de dados diversos. Isto possibilitou que fossem encontrados cenários mais complexos para testes de aplicação, enriquecendo o trabalho.

Quanto ao desenvolvimento da aplicação em si, em Python, houve uma dificuldade com a integração com os modelos GPT da OpenAI, por questão de atualização da API, resultando na refatoração da aplicação para se adequar ao novo modelo requerido pela OpenAI para integração com seus modelos. Também, ao utilizar a integração com modelo GPT e o banco de dados real obtido, foi encontrado um impedimento: a limitação de envio de tokens por minuto ao modelo GPT. Este impedimento, só ocorreu por conta do banco de dados utilizado ser robusto, e com dados reais inseridos ao longo de anos, portanto, ao enviar o dicionário de dados ao modelo GPT, de todas as tabelas e campos do banco de dados, que contava com mais de 3 mil tabelas, na API retornava erro por conta do limite de tokens enviados.

Para solucionar este impedimento, foram avaliadas algumas possibilidades, entre elas utilizar um modelo mais robusto que o 3.5-TURBO, contudo nenhum modelo da OpenAI apresentava um limite de tokens superior ao enviado. Também foi cogitado em trabalhar com contexto na aplicação, enviando em partes as mensagens para a mesma, contudo esta opção se

torna inviável, uma vez que na API com modelos GPT da OpenAI, se faz necessário enviar toda conversa já decorrida a cada iteração, tornando mais lento o processamento e consumindo muitos tokens, além de que o modelo após muitas iterações não podia mais "lembrar" de todos os dados exatamente para montar um SQL correto, gerando erros. Portanto, a escolha tomada para resolver esta situação, foi a de especificar tabelas que seriam consultadas manualmente. Para isso, foram escolhidas tabelas relacionadas com as entradas que seriam enviadas para avaliar a aplicação e outras que não faziam sentido com estas entradas.

Quanto a avaliação dos critérios estabelecidos, foi possível definir que a aplicação tem uma distinção de contexto impecável, desde que no *prompt* de sistema isso esteja muito claro. Assim como na tolerância a erros nas entradas do usuário, se bem especificado no *prompt* enviado ao modelo GPT, quando o mesmo não entendeu a entrada do usuário, não monta uma resposta aleatória, e sim diz que não entendeu. Também quanto ao desempenho, não foi visto uma variação muito grande, independentemente de quantos dados eram enviados ou respondidos, e da complexidade da pergunta do usuário, ficando numa média considerada aceitável de aproximadamente 5 segundos de resposta final. Já quanto a precisão na conversão para SQL, foi mensurada em 40% nos testes realizados. Estes testes abrangeram perguntas mais simples e mais complexas. Este resultado, foi obtido por uma série de fatores, dentre eles: limitação própria do modelo GPT na interpretação de orientações, estrutura do banco de dados conectado a aplicação e o detalhamento do dicionário de dados específico requerido. É importante citar que, durante todos os testes realizados neste trabalho, o custo monetário foi de aproximadamente U\$2.50, realizando cerca de 500 requisições para a API, enviando mais de 3.5 milhões de tokens e recebendo quase 19 mil.

Na introdução deste trabalho, foi levantada como objetivo responder a seguinte pergunta: "É viável a tradução de consultas em linguagem natural para comandos SQL, fazendo uso de LLMs, objetivando obter informações de um banco de dados relacional?". Então, após implementação e testes da aplicação, é possível afirmar que a tradução de texto para SQL utilizando o modelo pré treinado GPT 3.5-TURBO da OpenAI, possui uma boa precisão para entradas mais simples, consegue distinguir perguntas mal formuladas e fora de contexto. Contudo, ainda possui limitações no conhecimento de regras de SQL para SGBDs específicas, gerando SQLs que não podem ser executados. Além disso, um fator fundamental para que este tipo de aplicação seja viável, é que o banco de dados que será conectado deve possuir sua modelagem seguindo as boas práticas e seu dicionário específico de dados das tabelas/campos bem detalhados.

Para perspectivas futuras deste trabalho, alguns pontos chave devem ser levados em consideração: A escolha do modelo pré treinado, deve ser robusto, com um limite de tokens maior possível, visando manter o contexto e aumentar a "memória" do modelo durante uma conversa. Também, o banco de dados que será conectado na aplicação, para que seja um banco de dados bem modelado de acordo com as regras de padronização mundial, com um dicionário específico de dados o mais detalhado possível. Também, pode ser avaliada a utilização de mais

camadas realizando comunicação com o modelo GPT, como uma anterior a transcrição do texto do usuário para SQL, identificando quais as tabelas que a entrada do usuário se refere, para então poder enviar o dicionário de dados somente destas tabelas, evitando limitação de tokens e mantendo um melhor contexto durante a conversa. Além disso, realizar o ajuste-fino no modelo GPT com base no dicionário de dados, dados de contexto, e em um histórico de consultas e respostas, poderia resultar numa maior precisão na conversão para SQL da aplicação. Isto poderia ser implementado por meio do uso de assistentes do modelo GPT, pois é possível realizar uma determinação mais retrita de escopo, assim como para manter o contexto não é necessário enviar todas as perguntas e respostas ao modelo a cada iteração, é possível realizar este controle por meio de *threads*. Além disso, foi observado que se o banco de dados conectado na aplicação fosse um *datawarehouse* (repositório central de informações já calculadas), poderiam ser obtidos melhores resultados. Assim como, para perspectivas futuras pensou-se também na possibilidade de enviar a entrada do usuário via áudio, já que a API da OpenAI já possibilita este tipo de comunicação. E também, dar o retorno via voz por meio de algum assistente de comunicação.

Em conclusão, este trabalho demonstrou a viabilidade de uma aplicação que traduz mensagens em linguagem natural para comandos SQL usando a API do modelo GPT-3.5. A avaliação indicou boa precisão para consultas simples, apesar de limitações em consultas complexas e restrições de tokens. Melhorias futuras incluem o uso de modelos mais robustos e inclusão de camadas de refinamento das entradas na aplicação.

REFERÊNCIAS

- BIRD, S.; KLEIN, E.; LOPER, E. **Natural Language Processing with Python**. [S.l.: s.n.], 2009. ISBN 978-0-596-51649-9.
- BROWN, T. B. *et al.* Language models are few-shot learners. **arXiv preprint arXiv:2005.14165**, 2020.
- CARNEIRO, V. C. F. S. **Chatbot para Realização de Consultas em Bancos de Dados Relacionais**. 2022. Dissertação (Trabalho de Conclusão de Curso (Ciência da Computação)) — Universidade Federal de Sergipe, São Cristóvão, Sergipe, 2022.
- CHOU, D. C. **Java API Design Guidelines**. [S.l.]: Pearson Education, 2007.
- CODD, E. F. A relational model of data for large shared data banks. **Communications of the ACM**, v. 13, n. 6, p. 377–387, 1970.
- COMARELLA, R. L.; CAFÉ, L. M. A. Chatterbot: conceito, características, tipologia e construção. **Informação & Sociedade: Estudos**, v. 18, n. 2, 2008.
- DATE, C. J. **An Introduction to Database Systems**. [S.l.]: Addison-Wesley, 2003.
- DEVLIN, J. *et al.* Bert: Bidirectional encoder representations from transformers. **arXiv preprint arXiv:1810.04805**, 2018.
- ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. [S.l.]: Pearson, 2014.
- EVANGELISTA, G. d. M. **USO DE LLM OPEN SOURCE NA TRADUÇÃO DE LINGUAGEM NATURAL PARA SQL**. 2023. Dissertação (Trabalho de Conclusão de Curso (Engenharia da Computação)) — Universidade Federal de Pernambuco, Recife, Pernambuco, 2023.
- FELDMAN, M. **Software Development for Engineers: C/C++, Pascal, Assembly, Visual Basic, HTML, Java Script, Java DOS, Windows NT, UNIX**. [S.l.]: CRC Press, 2002.
- FERNANDEZ, M.; LLORET, J.; VIDAL, J. Advanced applications and data analysis in sql. *In: Advances in Database Technology - EDBT 2019*. [S.l.: s.n.], 2019.
- FIELDING, R. T. Architectural styles and the design of network-based software architectures. **Ph.D. dissertation, University of California, Irvine**, 2000.
- GUPTA, P. *et al.* Iqs - intelligent querying system using natural language processing. *In: International Conference on Electronics, Communication and Aerospace Technology*. Coimbatore, India: [s.n.], 2017. p. 407.
- HERNANDEZ, M. J. **Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design**. [S.l.]: Addison-Wesley, 2013.
- HIRSCHBERG, J.; MANNING, C. D. Advances in natural language processing. **Science**, v. 349, n. 6245, p. 261–266, 2015.
- INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 11179: Information Technology – Metadata Registries (MDR)**. 2023. <https://www.iso.org/standard/78914.html>.
- JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing (3rd ed. (draft))**. [s.n.], 2019. Disponível em: <https://web.stanford.edu/~jurafsky/slp3/>.

KAUR, G. *et al.* Clinical outcome and predictors of mortality in children with sepsis, severe sepsis, and septic shock from rohtak, haryana: A prospective observational study. **Indian Journal of Critical Care Medicine**, v. 18, n. 7, p. 437–441, 2014. Disponível em: <https://doi.org/10.4103/0972-5229.136072>.

KELECHAVA, B. **The SQL Standard – ISO/IEC 9075:2023 (ANSI X3.135)**. 2018. Disponível em: <https://blog.ansi.org/sql-standard-iso-iec-9075-2023-ansi-x3-135/>.

KROENKE, D. M.; AUER, D.; YODER, S. **Database Concepts**. [S.l.]: Pearson, 2014.

LUO, Y. *et al.* Synthesizing natural language to visualization (nl2vis) benchmarks from nl2sql benchmarks. *In: International Conference on Management of Data (SIGMOD '21)*. China: [s.n.], 2021.

MARTIN, R. C. **Clean Code: A Handbook of Agile Software Craftsmanship**. [S.l.]: Prentice Hall, 2011.

MOURA, E. A. **Consultas SQL utilizando linguagem natural para um contexto acadêmico**. 2019. Dissertação (Trabalho de Conclusão de Curso), Russas, 2019.

OPENAI. **GPTs**. [S.l.], 2023. Disponível em: <https://platform.openai.com/docs/introduction/gpts>. Acesso em: 2 out. 2023.

RADFORD, A. *et al.* **Improving Language Understanding by Generative Pre-training**. 2018. <https://www.openai.com/blog/better-language-models/>.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Database System Concepts**. [S.l.]: McGraw-Hill, 2010.

TANENBAUM, A. S.; BOS, H. **Modern Operating Systems**. [S.l.]: Pearson, 2014.

WHITE, J. *et al.* **A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT**. 2023.