UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO" – UNESP

Instituto de Biociências, Letras e Ciências Exatas - Câmpus de São José do Rio Preto

JOÃO PEDRO QUADRADO

ARQUITETURA HÍBRIDA BASEADA EM MODELOS DE LINGUAGEM PARA VISUALIZAÇÃO INTELIGENTE DE DADOS

São José do Rio Preto

2026

JOÃO PEDRO QUADRADO

ARQUITETURA HÍBRIDA BASEADA EM MODELOS DE LINGUAGEM PARA VISUALIZAÇÃO INTELIGENTE DE DADOS

Dissertação apresentada à Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP), Instituto de Biociências, Letras e Exatas, São José do Rio Preto, para obtenção do título de Mestre em Ciência da Computação

Área de Concentração: Computação Aplicada

Orientador: Prof. Dr. Carlos Roberto Valêncio

São José do Rio Preto

2026

Quadrado, João Pedro
Q1a Arquitetura híbrida baseada em modelos de linguagem para visualização inteligente de dados / João Pedro Quadrado. -- São José do Rio Preto, 2026
142 f.

Dissertação (mestrado) - Universidade Estadual Paulista (UNESP), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientador: Carlos Roberto Valêncio

1. Ciência da computação. 2. Processamento de linguagem natural. 3. Inteligência artificial. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

IMPACTO POTENCIAL DESTA PESQUISA

Este trabalho propõe uma arquitetura híbrida com modelos de linguagem para interfaces de linguagem natural, capaz de gerar consultas SQL e ampliar para visualizações complexas. A abordagem é viável em baixo custo com modelos *open-source* e promove a democratização da ciência de dados, com potencial de aplicação em diversos setores.

POTENTIAL IMPACT OF THIS RESEARCH

This work proposes a hybrid architecture using language models for natural language interfaces, capable of generating SQL queries and enabling complex visualizations. The approach is cost-effective with open-source models and promotes the democratization of data science, with potential applications across various sectors.

JOÃO PEDRO QUADRADO

ARQUITETURA HÍBRIDA BASEADA EM MODELOS DE LINGUAGEM PARA VISUALIZAÇÃO INTELIGENTE DE DADOS

Dissertação apresentada à Universidade Estadual Paulista “Júlio de Mesquita Filho” (UNESP), Instituto de Biociências, Letras e Exatas, São José do Rio Preto, para obtenção do título de Mestre em Ciência da Computação.

Área de Concentração: Computação Aplicada

Data da defesa: 24/02/2026

Banca Examinadora:

_____
Prof. Dr. Carlos Roberto Valêncio
UNESP – Instituto de Biociências, Letras e Ciências Exatas - Câmpus de São José do Rio Preto
Orientador

_____
Prof. Dr. Geraldo Francisco Donegá Zafalon
UNESP – Instituto de Biociências, Letras e Ciências Exatas - Câmpus de São José do Rio Preto

_____
Prof. Dr. Pedro Luiz Pizzigatti Corrêa
Universidade de São Paulo – Escola Politécnica

AGRADECIMENTOS

Agradeço primeiramente a Deus pela benção e força durante toda a minha vida, aos meus pais João e Yolanda pelo apoio e amor incondicional, a minha namorada e futura esposa Priscila por me acompanhar nessa trajetória, ao Gustavo, meu melhor amigo, uma longa amizade desde a 5ª série até o final da vida, ao Wellington, um amigo que conheci no grupo e que passamos pela trajetória do mestrado, e a todos os membros do Grupo de Banco de Dados. Agradeço em especial ao professor Carlos Roberto Valêncio, por ter me acolhido no grupo, pelos ensinamentos para se tornar uma pessoa melhor, pela orientação e suporte tanto nos projetos quanto no mestrado, coisas que vou levar para o resto da vida, obrigado de coração!

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 88887.952374/2024-00.

RESUMO

A análise e visualização de dados permitem identificar padrões e apresentá-los por meio de representações visuais. Com o propósito de minimizar a necessidade de conhecimento em ferramentas especializadas, as interfaces de linguagem natural permitem interação por meio de perguntas feitas sobre um conjunto de dados, convertidas em visualizações apropriadas ao contexto. Recentemente, modelos de linguagem pré-treinados e os grandes modelos de linguagem tornaram-se a base para a construção de interfaces mais inteligentes. Este trabalho propõe uma arquitetura híbrida baseada em modelos de linguagem para interfaces de linguagem natural voltada à visualização de dados. A arquitetura realiza a combinação de dois modelos de linguagem com papéis complementares: um modelo pré-treinado, responsável por identificar tabelas e colunas relevantes no esquema do banco de dados, e um modelo de linguagem de grande porte, responsável pela conversão de perguntas em linguagem natural em consultas SQL. Ambos os modelos foram ajustados para suas tarefas e comparados a outros, em que os melhores foram integrados à interface. O sistema resultante suporta 17 tipos de visualizações, com inclusão de opções mais complexas, como mapas e diagrama de Sankey. Os resultados da validação mostraram uma acurácia de 91,67% na identificação de tabelas e colunas relevantes e 75% de respostas corretas nas consultas SQL, com maior dificuldade em questões complexas. Esses valores indicam que a arquitetura é eficaz na resolução de perguntas de níveis fácil e médio; assim, torna-se viável para a implantação em bancos de dados reais de pequeno e médio porte como uma ferramenta de apoio ao processo de ciência de dados. Como contribuição científica, tem-se uma arquitetura que explora o uso combinado de diferentes modelos de linguagem na construção de interfaces de linguagem natural, a qual amplia o suporte para uma maior variedade de visualizações, com inclusão das mais complexas.

Palavras–chave: interface de linguagem natural; visualização de dados; modelos de linguagem; processamento de linguagem natural; inteligência artificial.

ABSTRACT

Data analysis and visualization enable the identification of patterns and their presentation through visual representations. In order to reduce the need for expertise in specialized tools, natural language interfaces allow interaction through questions posed over a dataset and converted into context-appropriate visualizations. Recently, pre-trained language models and large language models have become the foundation for the development of more intelligent interfaces. This work proposes a hybrid architecture based on language models for natural language interfaces aimed at data visualization. The architecture combines two language models with complementary roles: a pre-trained model responsible for identifying relevant tables and columns in the database schema, and a large language model responsible for converting natural language questions into SQL queries. Both models were fine-tuned for their respective tasks and compared with alternative models, with the best-performing ones integrated into the interface. The resulting system supports 17 types of visualizations, including more complex options such as maps and Sankey diagrams. Validation results showed an accuracy of 91.67% in identifying relevant tables and columns and 75% correct responses in SQL query generation, with greater difficulty observed in complex questions. These results indicate that the architecture is effective in addressing easy- and medium-level questions, making it viable for deployment in small- and medium-sized real-world databases as a support tool for data science workflows. As a scientific contribution, this work presents an architecture that explores the combined use of different language models in the construction of natural language interfaces, expanding support for a wider variety of visualizations, including more complex representations.

Keywords: natural language interface; data visualization; language models; natural language processing; artificial intelligence.

LISTA DE FIGURAS

Apêndice A

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AC	Aprendizado em Contexto
AED	Análise Exploratória dos Dados
AM	Aprendizado de Máquina
AMC	Auto-atenção Multi-Cabeça
AP	Aprendizado Profundo
APA	Aprendizado de Poucas Amostras
AZA	Aprendizado por Zero Amostras
BERT	Bidirectional Enconder Representations from Transformers
CSV	Comma-Separated Values
EI	Engenharia de Instruções
GPT	Generative Pre-trained Transformer
GPU	Unidade de Processamento Gráfico
IA	Inteligência Artificial
ILN	Interface de Linguagem Natural
JSON	JavaScript Object Notation
LLM	Grandes Modelos de Linguagem
LN	Linguagem Natural
LoRA	Low Rank Adaptation
MLI	Modelos de Linguagem
MLM	Modelo de Linguagem Mascarado
PEFT	Parameter Efficient Fine Tuning
PLM	Modelos de Linguagem Pré-treinados
PLN	Processamento de Linguagem Natural
RAD	Rede de Alimentação Direta
RAG	Recuperação Aumentada por Geração
REN	Reconhecimento de Entidade Nomeada
RN	Redes Neurais
RNR	Redes Neurais Recorrentes
SGBD	Sistema Gerenciador de Banco de Dados

SL	Ligação de Esquemas
SQL	Structured Query Language

SUMÁRIO

CAPÍTULO

1

1. INTRODUÇÃO

A visualização de dados é uma importante aliada da ciência e da análise de dados, pois facilita a compressão e processamento de informações relevantes ao apresentá-las por meio de representações visuais, uma vez que indivíduos tendem a apresentar mais dificuldade na interpretação de dados complexos quando estão dispostos em textos e números, em comparação aos gráficos (Islam; Jin, 2019; Kavaz; Puig; Rodríguez, 2023; Shen et al., 2023). Além disso, a visualização pode ser aplicada em outras etapas do processo da ciência de dados, como na fase da coleta e limpeza, em que são detectados valores incomuns do conjunto de dados (Unwin, 2020).

A construção de boas visualizações para uma determinada coleção de registros ocorre quando há uma associação entre comunicação, ciência de dados e design (Islam; Jin, 2019). Contudo, a criação de representações visuais não é uma tarefa trivial pois necessita de conhecimento prévio dos analistas em relação as ferramentas de visualização, bem como um entendimento aprofundado de tal coleção, por exemplo, o número de dimensões e qual as suas relações, para que assim a visualização mais adequada possa ser escolhida (Kavaz; Puig; Rodríguez, 2023).

O avanço das tecnologias associadas ao Processamento de Linguagem Natural (PLN) possibilitou que o paradigma de Interface de Linguagem Natural (ILN) fosse utilizado para resolver tal problema, em que perguntas feitas em linguagem natural pelos usuários sobre um conjunto de dados são transformadas em visualizações apropriadas ao contexto. Como vantagens, pessoas podem focar mais nos gráficos em si em vez de manipular as operações específicas de dentro das ferramentas, ampliar a utilização para usuários sem conhecimentos técnicos avançados e tornar o processo de ciência de dados mais democrático (Islam; Jin, 2019; Shen et al., 2023).

A transformação de uma pergunta em linguagem natural para visualização baseia-se na técnica de tradução automática, que converte um texto de um idioma para outro (Wang et al., 2022). No contexto das ILNs, as perguntas de entrada geram como respostas códigos executáveis de linguagem de programação, linguagens de visualização ou expressões abstratas. Para casos em que se tem um banco de dados associada a questão do usuário, a saída da interface é um código na linguagem *Structured Query Language* (SQL). Posteriormente, essa consulta é executada na base de dados, e os dados obtidos como resultado são analisados para construir as representações gráficas correspondentes. Assim, a tarefa de conversão de texto para visualização engloba uma subtarefa de traduzir a questão para SQL (Yang et al., 2024).

As primeiras ILNs eram baseadas em um conjunto de regras, em que se tinha limitações de seguir operações específicas e engessadas. Com o avanço da Inteligência Artificial (IA), essas interfaces começaram a ser desenvolvidas por meio de técnicas de Aprendizado Profundo (AP). Recentemente, os Modelos de Linguagem Pré-treinados, do inglês *Pre-trained Language Model* (PLM), e os Grandes Modelos de Linguagem, do inglês *Large Language Model* (LLM), pré-treinados com grandes volumes de textos, se estabeleceram como abordagens importantes na construção de visualizações a partir de perguntas em linguagem natural (Yang et al., 2024). Dessa forma, se torna interessante a exploração desse paradigma na construção de arquiteturas para ILNs mais robustas, inteligentes e adaptáveis (Maddigan; Susnjak, 2023).

1.1 MOTIVAÇÃO E ESCOPO

A arquitetura Transformador, do inglês *Transformer*, se estabeleceu como o paradigma mais importante para a tradução automática na atualidade, além de estar presente em outras áreas da computação, como a visão computacional e o processamento de áudio (Lin et al., 2022). Modelos de linguagem pré-treinados e grandes modelos de linguagem, ambos baseados nessa arquitetura, apresentam capacidades que se complementam: enquanto PLMs se destacam na captura de informações semânticas, LLMs demonstram maior capacidade em resolver tarefas gerais do mundo real (Zhao et al., 2023). Deste modo, a tarefa de transformar uma

pergunta em visualização ou consulta SQL pode ser considerada uma extensão a tradução automática, porém com abertura para melhorias.

Inúmeros trabalhos da literatura têm explorado a abordagem de PLMs e LLMs no desenvolvimento de ILNs (Lee et al., 2024; Maddigan; Susnjak, 2023; Yang et al., 2024; Zhu et al., 2024). Uma estratégia comum para aumentar a eficácia das respostas é dividir a tarefa de visualização em subtarefas, em que cada uma resolve um problema em específico e a saída de uma subtarefa serve como a entrada para a próxima (Yang et al., 2024). No entanto, há uma carência na literatura em relação ao uso de diferentes modelos de linguagem na construção de ILNs, em que os modelos são combinados e cada um se especializa em uma subtarefa por meio de ajustes. Diante dessas limitações, torna-se relevante investigar abordagens que explorem a combinação de diferentes modelos em uma arquitetura integrada.

Além disso, os artigos pertencentes a área de ILNs se concentram em apresentar visualizações consideradas básicas, como gráficos de dispersão, barras e linhas. Devido a isso, abranger o escopo com a inclusão de representações mais complexas constitui um tópico importante a ser explorado. Exemplos dessas representações são o diagrama de Sankey, gráficos de redes e mapas (Kavaz; Puig; Rodríguez, 2023).

1.2 OBJETIVOS

Dada a importância das interfaces de linguagem natural que torna o processo da ciência de dados mais acessível e democrático, especialmente para pessoas sem conhecimento prévio em ferramentas de visualização, e os avanços recentes em inteligência artificial, com destaque para os modelos de linguagem pré-treinados e os grandes modelos de linguagem, este trabalho tem como objetivo o desenvolvimento de uma arquitetura híbrida para interfaces de linguagem natural baseada na combinação de dois modelos de linguagem, um PLM e um LLM, cada um especializado numa tarefa exclusiva, com suporte a visualizações diversificadas e complexas.

1.3 METODOLOGIA

Em relação a metodologia, inicialmente foi realizado um levantamento bibliográfico do estado da arte sobre visualização de dados, interfaces de linguagem

natural, tradução de texto para SQL e o uso de modelos de linguagem pré-treinados e grandes modelos de linguagem em ILNs. A partir dessa análise, foram selecionados os trabalhos correlatos mais relevantes, os quais direcionaram para a problematização e a definição dos objetivos do trabalho.

Em seguida, foi proposta uma arquitetura híbrida para ILNs voltadas à visualização de dados, baseada na combinação de dois modelos de linguagem, um PLM e um LLM, em que cada modelo desempenha uma subtarefa específica. O PLM foi empregado na tarefa de ligação de esquemas, a qual reconhece as tabelas e colunas mais importantes fornecida a pergunta e o contexto do banco de dados, e o LLM foi responsável por converter perguntas em linguagem natural em consultas SQL.

Ambos os modelos de linguagem foram ajustados para suas respectivas tarefas com o uso de um conjunto de dados de pares de pergunta-SQL, amplamente utilizado na literatura, e posteriormente avaliados por meio de métricas adequadas a cada subtarefa e comparados a outros modelos, com o objetivo de selecionar aqueles que apresentaram o melhor desempenho

Além disso, foi desenvolvido um algoritmo heurístico para seleção automática de visualizações, responsável por direcionar os dados resultantes da execução das consultas SQL para representações gráficas apropriadas ao contexto da pergunta.

Por último, foi desenvolvida uma interface de linguagem natural na qual a arquitetura proposta foi integrada e validada por meio de experimentos com um banco de dados do mundo real e um conjunto de perguntas com diferentes níveis de complexidade.

.

1.4 CONTRIBUIÇÕES

A contribuição científica deste trabalho é apresentar uma abordagem híbrida para visualização de dados, inserida no contexto da ciência de dados, que explora o uso de diferentes modelos de linguagem na construção de interfaces de linguagem natural, a qual amplia a variedade de visualizações, com incremento de complexidade.

1.5 ORGANIZAÇÃO DA DISSERTAÇÃO

Neste capítulo, foi apresentada a importância da visualização de dados no apoio à análise e intepretação de informações, com destaque para as interfaces de

linguagem natural que contribui para tornar o processo de ciência de dados mais acessível e aberto. Além disso, foi apresentado o impacto do avanço de novas tecnologias associadas a inteligência artificial na construção de ILNs, com uma nova área a ser explorada relacionada aos modelos de linguagem. Por último, foram mostrados os objetivos, a metodologia e as contribuições científicas deste trabalho.

No Capítulo 2 é exibida a fundamentação teórica do trabalho, com conceitos de visualização de dados, inteligência artificial, processamento de linguagem natural, aprendizado profundo, modelos de linguagem com suas derivações e os trabalhos correlatos pertencentes a área de ILNs desenvolvidas por meio de modelos de linguagem.

No Capítulo 3 é detalhada a proposta do trabalho, com a descrição da arquitetura proposta e a metodologia usada nos ajustes finos. No Capítulo 4 é apresentado os resultados efetuados para a validação do ajuste fino e da arquitetura em si. No Capítulo 5 são descritos a conclusão e contribuição científica do trabalho.

CAPÍTULO

2

2. VISUALIZAÇÃO DE DADOS E INTELIGÊNCIA ARTIFICIAL: FUNDAMENTOS E AVANÇOS

Neste capítulo são apresentados os principais conceitos do trabalho proposto, com os seguintes tópicos: visualização de dados e a sua importância de estar associada a ciência e a análise de dados (Seção 2.1), além de aplicações, técnicas, ferramentas e a escolha da melhor visualização na exibição um determinado conjunto de dados; inteligência artificial (Seção 2.2), a subárea de processamento de linguagem natural e a sua relação com a visualização de dados (Seção 2.3); aprendizado profundo e o avanço motivado por tal abordagem no processamento de linguagem natural com a criação da arquitetura *Transformer* e subsequentemente dos modelos de linguagem pré-treinados e grandes modelos de linguagem (Seção 2.4 e 2.5, respectivamente); ajuste fino para os modelos de linguagem (Seção 2.6), com métodos de otimização; aprendizado em contexto e suas principais técnicas (Seção 2.7); trabalhos correlatos na área de ILNs com uso de modelos de linguagem (Seção 2.8).

2.1 CIÊNCIA, ANÁLISE E VISUALIZAÇÃO DE DADOS

Com a crescente popularização e evolução da internet ao longo das últimas décadas, naturalmente o volume de informações aumentou e, independente da área, seja na esfera de negócios, indústria, saúde ou em outros setores, todas passaram a ser apoiadas pelos dados (Duan; Xiong, 2015). Nesse cenário, a ciência de dados surge como uma abordagem capaz de lidar com a grande variedade e quantidade de

informações disponíveis, com o propósito de transformá-las em descobertas que levam a tomada de decisões por parte das organizações (Cao, 2017).

A ciência de dados é considerada uma evolução dos conceitos de análise de dados, mineração de dados e aprendizado de máquina, que foram incorporados dentro de uma estrutura mais ampla e interdisciplinar (Sarker, 2021). Dessa forma, a descoberta de conhecimento nessa estrutura envolve um conjunto de atividades, como a coleta, preparação, exploração, análise, visualização e interpretação dos dados (Donoho, 2017).

Todas essas etapas podem ser englobadas como parte do processo de análise de dados, que possui um papel crucial ao possibilitar a identificação de padrões e estruturas que contribuem para a tomada de decisões pelas organizações, assim como ocorre na ciência de dados. Além disso, trata-se de um conceito multidisciplinar, com inclusão das áreas de computação e matemática, como estatísticas, reconhecimento de padrões e inteligência artificial (Runkler, 2020).

Um conceito a ser destacado e que se encontra em diferentes etapas da análise de dados é a visualização de dados. Tal abordagem possui como definição representações visuais interativas, geralmente na forma de gráficos dispostos por meio de computadores, que permitem a exploração de dados com o intuito de identificar padrões e correlações (Li, 2020). Além disso, a visualização pode ser utilizada em diversas etapas da análise de dados, desde a aquisição e seleção, com a verificação de valores incomuns do conjunto de dados, até as fases finais de análise e interpretação dos resultados. Esta fase preliminar é conhecida também como Análise Exploratória dos Dados (AED) (Unwin, 2020).

Na etapa final de um experimento de ciência de dados, a visualização gerada é chamada de análise visual, que envolve tanto a identificação de padrões e tendências quanto a validação dos modelos de saída com base na apresentação gráfica das informações (Unwin, 2020).

2.1.1 Aplicações

Assim como na ciência e análise de dados, a visualização é um campo multidisciplinar, aplicado em diversos setores. Por exemplo, uma das áreas que mais se beneficia é a da saúde, em que apresentar informações de forma clara é crucial para a avaliação de pacientes ou para o acompanhamento de dados epidemiológicos.

Nesse contexto, torna-se importante a criação de ferramentas inteligentes, como representações visuais que permitem avaliar mudanças drásticas de sinais vitais de um determinado paciente e assim intervir em tempo real ou gráficos voltados à análise de exames laboratoriais (Oeste; Borland; Hammond, 2015; Sadiku et al., 2016).

Outra esfera a ser destacada é a de negócios, que possibilita avaliar o desempenho da organização em relação ao mercado, como vendas ou lucros, além de contribuir para a detecção de fraudes por meio da identificação de padrões suspeitos. Ao mesmo tempo, o meio ambiente tem-se tornado cada vez relevante pois é possível comprovar, por meio de mapas, locais que perderam fauna e flora ao longo do tempo e assim promover o reflorestamento ou no monitoramento de ambientes que tiveram áreas afetadas por queimadas. Em conclusão, na área de educação, a visualização de dados contribuiu para o gerenciamento de informações complexas oriundas de diferentes fontes e, com isso, mostrá-las de forma criativa para o entendimento de alunos e professores (Sadiku et al., 2016).

2.1.2 Técnicas mais comuns

As técnicas de visualização possuem diversos formatos, em que cada uma proporciona um determinado efeito visual. A seleção de um método depende principalmente dos tipos de dados a serem analisados, e a escolha de um formato adequado reflete no quão eficaz será a análise e consequentemente na identificação de padrões escondidos (Li, 2020). Existem inúmeras maneiras de retratar as informações por meio de representações visuais, porém as mais comuns são mostradas a seguir (Li, 2020; Mohammed; Alhabshy; Eldahshan, 2022):

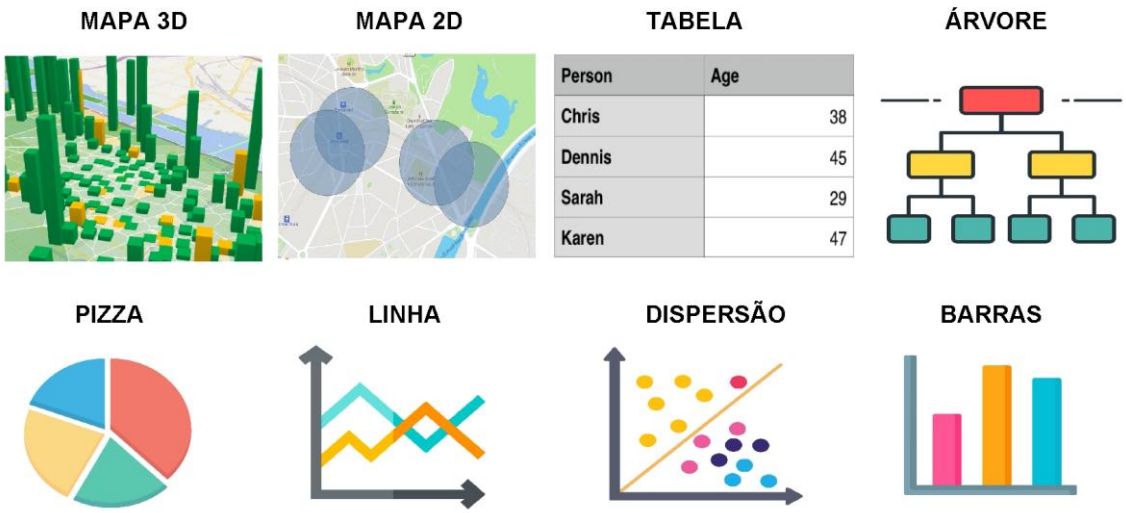
a) Mapas - interface de duas ou três dimensões que apresenta dados geográficos de um determinado espaço, em que são destacadas as relações entre objetos e regiões, geralmente por meio de cores ou bolhas.
b) Tabelas - representada por linhas e colunas, tem como intuito realizar comparações entre dados de diferentes categorias. Como vantagem, pode-se mostrar uma grande quantidade de informações e dimensões numa só tabela.
c) Árvores - estrutura que organiza os valores num formato hierárquico, em que os conteúdos estão dispostos por meio de nós. A árvore inicia-se

por um nó chamado raiz, localizado no topo, e a partir desse ponto, ramifica-se para os nós inferiores, considerados os filhos, até atingir os nós finais.

d) Gráficos e suas variações, como linhas, barras, pizza e dispersão – gráficos de linhas são escolhidos com objetivo de mostrar alterações ao longo do tempo, assim possibilita a verificação de períodos em que há picos/quedas. Gráficos de Barras realizam comparativos entre grupos ou apresentam as informações em ordem cronológica. São divididos em 3 categorias: barras horizontais, colunas e colunas empilhadas. Gráficos de pizza permitem expor a proporção de informações em relação ao conjunto como todo. E por último, gráficos de dispersão consistem em verificar se há correlação entre duas variáveis dispostas em um sistema cartesiano. Com base na maneira como os pontos estão espalhados pelo plano é possível afirmar se a correlação é positiva, negativa ou inexistente.

Desse modo, na Figura 1 é apresentado um exemplo de cada um dos métodos de visualizações mencionados.

Figura 1 – Exemplos de métodos de visualização de dados.



Fonte: Elaborado pelo autor.

2.1.3 Ferramentas

As ferramentas de visualização auxiliam os usuários tanto no processo da ciência de dados quanto na forma como essas informações são apresentadas. Atualmente, tais produtos evoluíram a um ponto em que o processo completo de análise se tornou cada vez mais automatizado. Em poucos passos, é possível conectar a uma base de dados, extrair as informações mais relevantes e apresentá-las por meio de um gráfico, inclusive o *software* recomenda a melhor representação visual (Islam; Jin, 2019).

Como exemplos de ferramentas neste contexto, tem-se o *Tableau* e *Microsoft Power BI*, amplamente usadas por empresas na área de negócios. Ambas possuem um sistema completo de análise e visualização, com inúmeras opções de importação de dados e gráficos. Além disso, a interface interativa possibilita a personalização das visualizações em tempo real. Outra ferramenta, num escopo menor, é o *Google Charts*, capaz de gerar gráficos interativos nativamente em navegadores, isto é, sem a necessidade de adicionar aplicativos externos. Possui uma gama de gráficos disponíveis além de permitir a integração a outras ferramentas do Google e banco de dados externos (Islam; Jin, 2019; Mohammed; Alhabshy; Eldahshan, 2022).

Por último, bibliotecas da linguagem *Javascript*, como *Chart.js*, *D3.js* e *Chartist.js* são boas opções para visualizações dinâmicas por meio de navegadores web. Suas principais características envolvem ser de código livre, personalizáveis, leves em relação ao armazenamento e oferecer uma extensa variedade de gráficos (Mohammed; Alhabshy; Eldahshan, 2022).

2.1.4 Escolha da melhor visualização

É válido afirmar que atualmente há inúmeras opções de visualizações disponíveis e torna-se um desafio escolher a melhor forma de mostrar visualmente um determinado conjunto de dados. Como comentado, a seleção de uma visualização está principalmente atrelada aos tipos de dados e, por conta disso, existe a necessidade de avaliar as propriedades destas informações.

Tal processo consiste em verificar o número de variáveis independentes e dependentes, e para cada uma, analisar suas principais características, como (Tory; Moller, 2004):

a) A variável é escalar, vetorial (com diferentes dimensões) ou uma estrutura mais complexa.

b) Se os valores apresentados são discretos ou contínuos, ou seja, valores finitos ou infinitos, respectivamente.
c) Situações em que as variáveis são nominais, ordinais, intervalos ou razões, isto é, variáveis nominais não possuem uma ordem natural; variáveis ordinais, de maneira contrária, apresentam uma ordem entre os valores; variáveis de intervalos contém uma métrica entre dois valores; e variáveis de razões têm uma variável de intervalo relevante, com um zero absoluto, que separa em dois grupos.

Diferentes trabalhos na literatura e no setor empresarial abordam o tema da taxonomia de gráficos, isto é, propõem uma classificação em que separa as representações visuais em diferentes setores. Por exemplo, o trabalho de Kucher, Paradis e Kerren (2018) categoriza técnicas de visualização de sentimentos com base em publicações da respectiva área, e com isso, divide-as em diversos grupos: comunicação, comparação, monitoramento, séries temporais, etc.

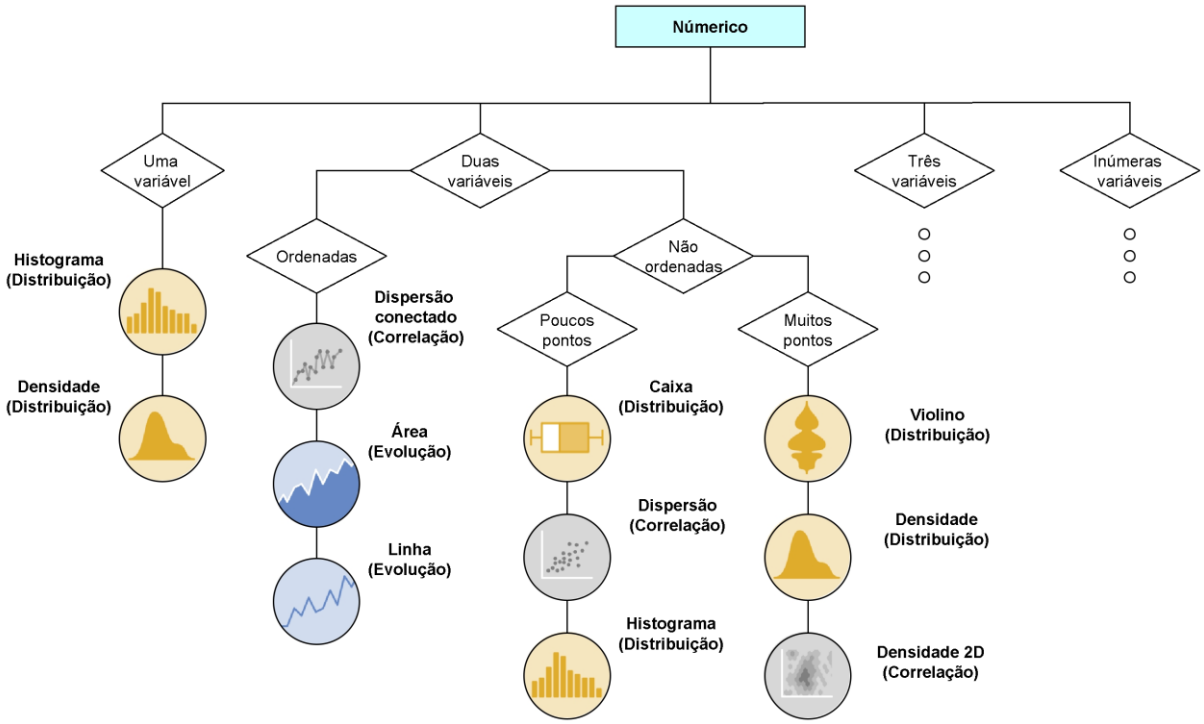
Outro trabalho, de Abela (2013), separa os gráficos em 4 domínios: distribuição, comparação, relacionamento e composição, e por meio de uma árvore de decisão, analisa as propriedades dos valores do conjunto de dados disponível, como quantidade de variáveis presentes, se as informações são estáticas ou há mudanças conforme o tempo, e direciona para a melhor representação visual.

Por último, o estudo de Healy e Holtz (2017) também usufrui de árvores de decisão e realiza uma análise das características dos dados para recomendar a visualização mais adequada. Essa análise é mais complexa em comparação a outras abordagens, além de abranger para uma quantidade maior de gráficos. O processo de recomendação de visualizações da pesquisa inicialmente verifica as propriedades gerais do formato dos dados, que são separadas em seis categorias: numéricos, categóricos, numéricos e categóricos, mapas, redes e séries temporais. Com base na escolha, a árvore de decisão se ramifica e, com isso, examina-se características mais internas, como o número de variáveis numéricas e categóricas, se estão ordenadas, se possuem subgrupos, entre outros, até chegar no nó final, que consiste na representação gráfica mais adequada para o conjunto de dados de entrada.

Na Figura 2 é mostrado a árvore de decisão do trabalho de Healy e Holtz (2017) para a categoria numérica. De início, verifica-se a quantidade de variáveis: para apenas uma variável numérica, a melhor representação são o histograma ou gráfico de densidade. Para duas variáveis, a árvore expande-se e analisa-se se os valores

estão ordenados ou não. Se estão ordenados, pode-se utilizar um gráfico de área, linha ou dispersão. Caso contrário, é feita mais uma checagem com base na quantidade de pontos, a partir da qual direciona para diferentes visualizações se há poucos ou muitos pontos. Outros tipos de análises são realizados para três ou mais variáveis numéricas.

Figura 2 – Árvore de decisão para o valor numérico.



Fonte: Adaptado de Healy e Holtz (2017).

A visualização de dados é uma área que está em constante evolução e cada vez mais há um interesse em associá-la às tecnologias de Inteligência Artificial. A junção desses dois escopos possibilita a automatização de processos, principalmente na geração de representações visuais. A recomendação é uma etapa crucial para automatizar a criação de gráficos, em que informações valiosas oriundas de banco de dados são analisadas visualmente de forma eficiente (Wu et al., 2022).

2.2 INTELIGÊNCIA ARTIFICIAL E APRENDIZADO DE MÁQUINA

A inteligência artificial tem ganhado cada vez mais destaque na era moderna, o que resulta em um impacto direto no cotidiano das pessoas. Com aplicações nas

áreas da indústria, educação, saúde, segurança e outras esferas, a tendência é que o mercado associado a esse conceito gere U$ 190 bilhões de dólares até 2025, a uma taxa anual de crescimento de cerca de 36% desde 2018 (Jiang et al., 2022).

Por definição, a inteligência artificial consiste no estudo de sistemas, também chamados de agentes computacionais, que operam de forma autônoma, compreendem o ambiente em que estão inseridos e realizam adaptações a fim de maximizar os resultados esperados (Russell; Norvig; Davis, 2010). Ao longo de décadas, diversos métodos e teorias associados a IA foram criados que possibilitaram avanços em vários campos de pesquisa (Jiang et al., 2022). Um dos campos que deve ser destacado é o Aprendizado de Máquina (AM), que consiste na utilização de técnicas para executar tarefas específicas com instruções mínimas fornecidas. Os algoritmos utilizam amostras, conhecidas como dados de treinamento, para aprender os padrões de uma determinada tarefa e, com isso, realizam previsões com base em entradas não passadas anteriormente (El Mrabet; El Makkaoui; Faize, 2021).

2.3 PROCESSAMENTO DE LINGUAGEM NATURAL

Uma outra subárea pertencente à inteligência artificial e que trabalha em conjunto com o aprendizado de máquina é o Processamento de Linguagem Natural. O objetivo principal deste conceito é estabelecer uma conexão entre o dialeto humano e computadores por meio de técnicas que realizam a análise e representação automática da linguagem (Bharadiya, 2023; Chowdhary, 2020).

A junção da PLN com o AM iniciou na década de 80, em que, até este momento, sistemas voltados à Linguagem Natural (LN) eram construídos baseados em regras escritas à mão. Com o desenvolvimento de métodos de aprendizado associados ao AM, a área de PLN passou por uma grande revolução (Bharadiya, 2023). Como consequência, houve um avanço nas tecnologias de PLN, com destaque para a tradução automática. Essa técnica abriu caminhos na exploração de sistemas de interação baseados em LN para visualização de dados (Shen et al, 2023).

2.3.1 Processamento de Linguagem Natural associada a Visualização de Dados

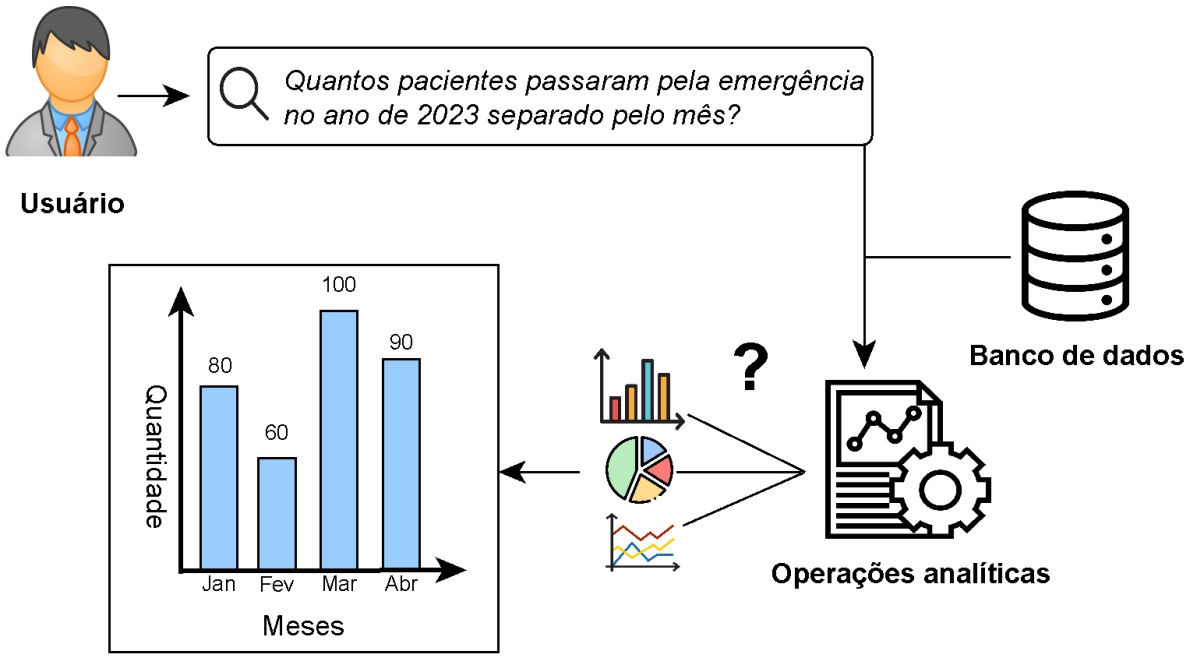
A tradução automática é um processo usado pelos computadores para traduzir um idioma para outro (Wang et al., 2022). É perceptível que tal definição não

especifica o tipo de linguagem, assim tem-se a possibilidade da integração de diferentes áreas. Uma dessas opções é a transformação de textos elaborados em linguagem natural para visualizações, tarefa conhecida como Interfaces de Linguagem Natural. As ILN permitem que usuários comuns realizem perguntas sobre um conjunto de dados e recebam como resposta representações visuais a respeito da questão (Narechania; Srinivasan; Stasko, 2021). Com isso, a ciência de dados e a descoberta de padrões se tornam mais fáceis, intuitivas e acessíveis, pois abrangem uma quantidade maior de usuários sem a necessidade de ter um conhecimento prévio sobre programação ou configurações técnicas de ferramentas, além da automatização do processo como um todo (Maddigan; Susnjak, 2023).

O fluxo de uma ILN começa ao receber, de forma amigável, a pergunta feita em linguagem natural e o conjunto de dados que deseja analisar. Esse conjunto pode ser, por exemplo, um arquivo com os dados tabulares anexado ou um SGBD relacional conectado a interface. Em seguida, as informações da entrada são modeladas em atributos que passam por tarefas analíticas, como filtros, correlação e identificação de tendências. Após a análise, associa-se a pergunta feita com a visualização mais apropriada, e no final, é gerado o gráfico correspondente (Maddigan; Susnjak, 2023; Narechania; Srinivasan; Stasko, 2021).

Na Figura 3 são apresentadas as etapas da ILN. No início tem-se a pergunta "*Quantos pacientes passaram pela emergência no ano de 2023 separado pelo mês?*" em conjunto com o banco de dados. A próxima fase recebe tais dados e realiza as operações analíticas. Por último, gera-se a representação que melhor condiz com a questão feita, no caso, um gráfico de barras.

Figura 3 – Etapas de uma Interface de Linguagem Natural.



Fonte: Elaborado pelo autor.

De forma geral, as abordagens que convertem texto para visualização fornecem como saída códigos executáveis de linguagem de visualização ou expressões abstratas, como, por exemplo, a linguagem de programação *Python* com suas bibliotecas de gráficos e a ferramenta *Vega-lite*, uma gramática de alto nível que possibilita a elaboração de representações visuais estatísticas. Com isso, tais códigos são interpretados e tem-se como resultado a geração automática das visualizações (Yang et al., 2024).

No entanto, quando a pergunta feita pelo usuário está atrelada a um SGBD, as tarefas analíticas devem voltar às operações próprias dos bancos de dados relacionais, no caso a linguagem SQL. Dessa forma, a conversão do texto para visualização envolve um subprocesso de traduzir o texto para SQL, executar a consulta no SGBD e subsequentemente realizar o mapeamento dos dados tabulares obtidos para a visualização mais apropriada (Yang et al., 2024).

Tal processo também pode ser representado como uma fórmula matemática, mostrado em (1). Dada a pergunta em linguagem natural, representada como P, e um conjunto de dados, definido como CD, ambos são combinados para a construção da instrução I, que é a função de entrada para o sistema. Essa instrução é processada por um algoritmo de PLN, denominado A. Como resultado, tem-se a conversão do

texto num código SQL, os dados de retorno (DR) após a execução do SQL e o processo de mapeamento (MAP) dos dados de retorno com o gráfico mais adequado a pergunta realizada (Yang et al., 2024).

$$A(I(P, CD)) \rightarrow \{SQL, DR, MAP\} \quad (1)$$

A tradução de texto para visualização, assim como de texto para SQL, é um paradigma que depende de diversos fatores para o funcionamento adequado. Por exemplo, os computadores precisam compreender as instruções passadas pelo usuário, porém a linguagem natural é ambígua, específica e complexa. Ao mesmo tempo, há um esforço considerável para o desenvolvimento de algoritmos que convertam textos em visualizações apropriadas, além da dificuldade de compreender o contexto à medida que novas instruções são transmitidas (Luo et al., 2022; Shen et al., 2023).

As primeiras ILN se basearam em técnicas clássicas de PLN para o entendimento da LN, como heurísticas, regras e gramática probabilística. Os métodos heurísticos utilizam de regras pré-definidas e é o que tem a menor precisão dentre os três métodos. Em relação as abordagens baseadas em regras, há a necessidade de pessoas com conhecimento na área para a elaboração de normas que compreendem a LN. Essas técnicas possuem uma melhor precisão, mas falham em ser flexíveis quando a consulta se torna complexa. Por último, a gramática probabilística supera as outras duas em termos de flexibilidade e precisão com o uso de regras gramaticais formais em conjunto com a probabilidade, porém necessita de computadores mais potentes. O avanço de tecnologias associadas a PLN possibilitou que novas abordagens surgissem na construção de ILN. Uma delas se destacou das demais e se consolidou como a mais relevante para este conceito: o aprendizado profundo (Maddigan; Susnjak, 2023).

2.4 APRENDIZADO PROFUNDO

Aprendizado profundo (AP), do inglês *Deep Learning*, é uma subárea do aprendizado de máquina que tem como principal característica o uso de Redes Neurais (RN) em sua composição (Shinde; Shah, 2018). Redes neurais são estruturas compostas por camadas de neurônios que recebem uma função matemática de

entrada e tem como objetivo aprender uma representação aproximada de tal função, que é gerada como saída.

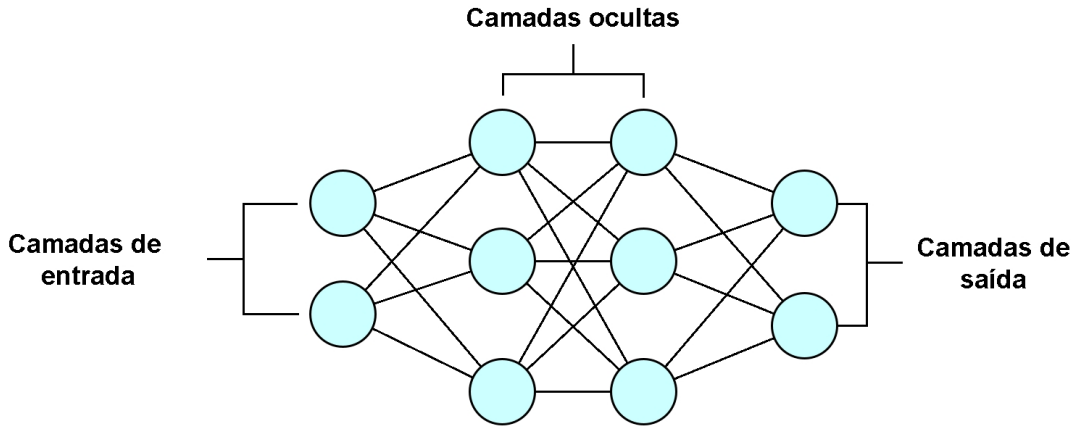
Um neurônio, também chamado de unidade neural, tem como base uma função linear, representada por (2), em que x representa a entrada do neurônio, w é o peso, *weight* em inglês, e b é o viés, *bias* em inglês. O peso e o viés realizam o controle, respectivamente, da inclinação e deslocamento da reta, e ambos são ajustados durante o treinamento da rede neural. Ao combinar todas as funções lineares obtidas dos neurônios, tem-se como resposta uma função que ainda é linear, e existem situações em que o problema a ser resolvido é uma função não linear, por exemplo, uma parábola (Dong; Wang; Abbas, 2021).

$$F(x) = w.x + b \tag{2}$$

Para isto, são utilizadas funções de ativação que introduzem a não linearidade ao modelo. Tais funções são baseadas em neurônios biológicos, se são ativados ou não. Ao utilizá-las para transformar em valores não lineares, em junção com as unidades neurais e as inúmeras camadas, tem-se a estrutura da rede neural, inspirada na organização do cérebro humano (Dong; Wang; Abbas, 2021). Além disso, os dados de entrada são passados de camada em camada, nas quais as mais próximas do início aprendem recursos considerados mais simples enquanto as camadas mais profundas, também chamada de camadas ocultas, realizam cálculos mais complexos (Shinde; Shah, 2018).

Na Figura 4 é apresentado um exemplo de uma rede neural com as camadas de entrada, as camadas intermediárias, chamadas de ocultas, e as camadas de saída. Geralmente as camadas de entrada não são contabilizadas como uma camada, pois não possuem neurônios, apenas servem para transmitir informações.

Figura 4 – Representação de uma rede neural.



Fonte: Elaborado pelo autor.

2.4.1 Redes neurais recorrentes

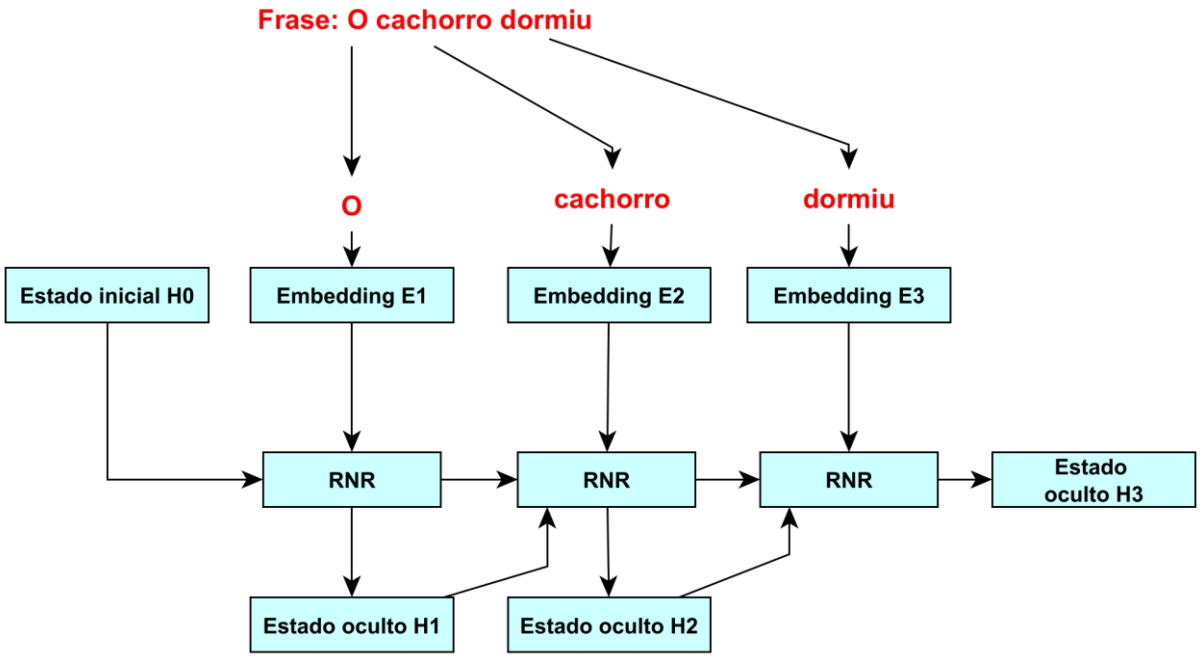
As Redes Neurais Recorrentes (RNR), um dos métodos de RN, foram consideradas uma das mais relevantes na área de tradução automática durante um período de tempo considerável. Seu funcionamento consiste em dada uma sequência de palavras ou caracteres como entrada, cada um desses dados é convertido em *tokens* e posteriormente são transformados em *embeddings,* isto é, vetores que representam numericamente a palavra/caractere da sequência (Tunstall; Werra; Wolf, 2022).

Cada *embedding* passa pelas camadas da rede e gera como saída uma estrutura do tipo vetor chamada estado oculto, que resume todas as informações propagadas até o momento. Tal estado retorna para a rede de maneira recorrente, ou seja, o próximo *embedding* a ser executado usa o estado oculto da saída anterior. Após a execução de toda a sequência, obtém-se o último vetor de estados ocultos, que sumariza todos os dados essenciais transmitidos pela rede. Com isso, é possível entender o contexto de tais informações e utilizar o vetor para as previsões futuras. Todo o processamento de transformar os tokens em *embbeddings* e posteriormente convertê-los no último vetor de estado ocultos é chamado de codificador (Tunstall; Werra; Wolf, 2022).

Na Figura 5 é apresentado um exemplo do funcionamento do codificador para a frase “O cachorro dormiu”. Inicialmente, cada palavra é transformada em *embbedings*, E1, E2 e E3, para “O”, “cachorro” e “dormiu”, respectivamente. Em

seguida, o primeiro *embbeding*, E1, é transmitido para uma célula da RNR, juntamente como estado oculto inicial, H0. Como resultado, tem-se a atualização do estado oculto, que se torna H1. Posteriormente, H1 retorna a rede combinado com o próximo *embbeding*, E2. Tal processo se repete para cada *embbeding* na sequência, no qual o estado oculto obtido na saída anterior é usado como entrada no próximo passo. No final do processo, é obtido o último estado oculto, H3, que resume toda a frase transmitida à rede.

Figura 5 – Exemplo de funcionamento do codificador.

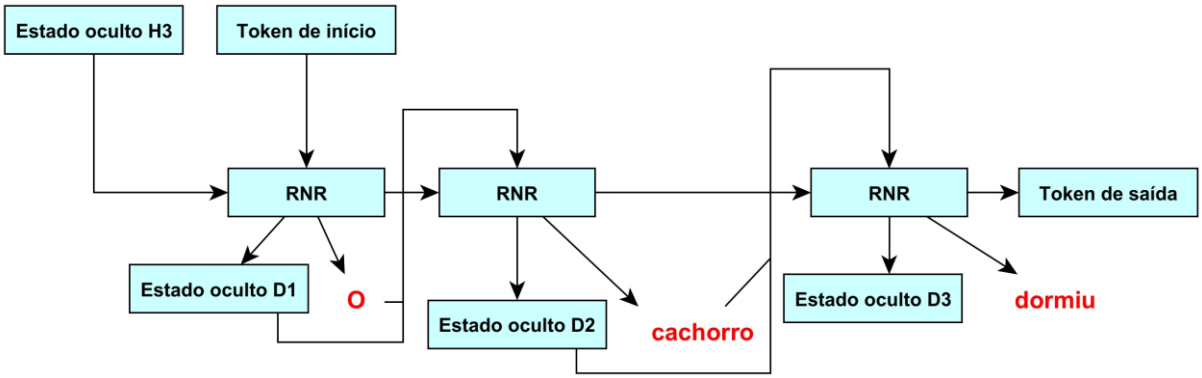


Fonte: Elaborado pelo autor.

Como papel inverso ao codificador tem-se o decodificador, o qual transforma o vetor de estados em *tokens* de palavras. A arquitetura completa do codificador-decodificador voltada aos textos sequenciais também é conhecida como sequência-a-sequência (Tunstall; Werra; Wolf, 2022). Na Figura 6 é mostrada a operação do decodificador para traduzir a frase “O cachorro dormiu”. De início, utiliza-se como entrada o último estado oculto H3 obtido no codificador e um *token* de início especial. Essas informações são enviadas a RNR, na qual gera duas saídas: a primeira palavra, “O” e o estado oculto atualizado, D1. Esse novo estado, em conjunto com a palavra “O”, retorna a rede como entrada, que assim elabora a palavra “cachorro”. O mesmo

processo se repete até a tradução estar completa, que se encerra quando um token de saída é gerado.

Figura 6 – Exemplo de funcionamento do decodificador.



Fonte: Elaborado pelo autor.

No entanto, quando o texto a ser traduzido contém uma grande quantidade de *tokens*, de maneira proporcional também existirá uma grande quantidade de estados ocultos. Como consequência, ao uni-los todos no último vetor resultante, há a possibilidade de informações iniciais e que são consideradas importantes serem perdidas no processo, pois tal vetor apenas resume os dados e não os armazena como um todo. Como solução, foi desenvolvido um mecanismo que atualmente se encontra em inúmeras redes neurais, conhecido como atenção (Tunstall; Werra; Wolf, 2022).

2.4.2 Mecanismo de atenção

A ideia por trás do método de atenção se baseia no pensamento humano de focar apenas no conhecimento importante da vida diária e ignorar os elementos irrelevantes. Ao aplicar este conceito às redes neurais, tem-se em prestar a atenção nos componentes mais significativos por meio de pesos (He et al., 2023). Em vez de usar apenas o último vetor de estado oculto, o mecanismo de atenção possibilita que o decodificador tenha acesso a qualquer estado construído pelo codificador. Porém, para não manipular todos os estados de uma só vez, o decodificador calcula um peso em cada etapa da sentença e prioriza apenas os estados mais importantes para

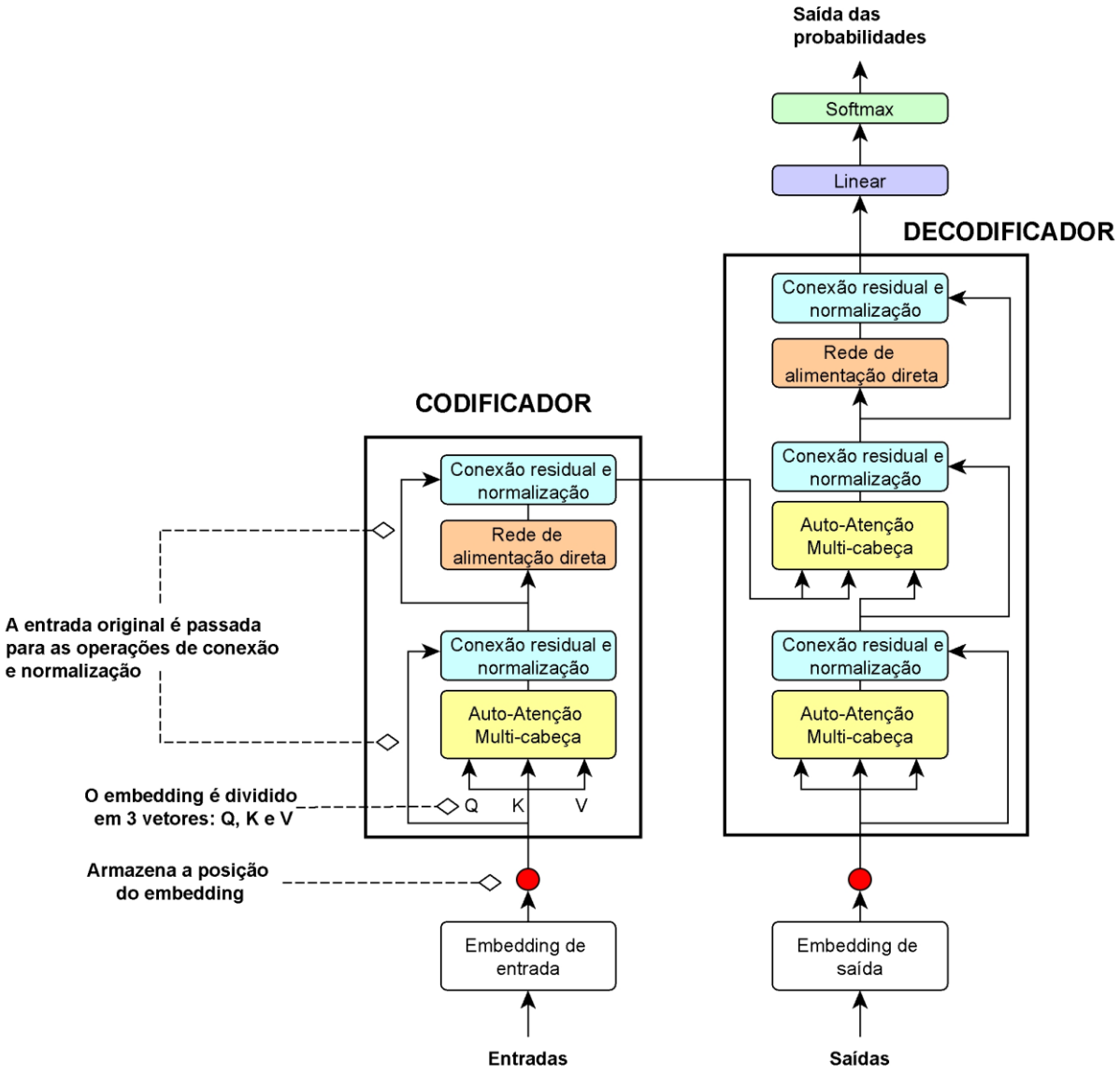
aquela fase. Assim, é possível conectar as palavras de origem com a suas respectivas traduções de saída de maneira mais eficiente (Tunstall; Werra; Wolf, 2022).

O mecanismo de atenção aprimorou os algoritmos voltados a tradução automática, mas os cálculos produzidos pelo codificador-decodificador das RNR ainda são feitos de forma sequencial. Neste contexto, um artigo publicado por Vaswani et al. (2017) introduziu um novo conceito chamado autoatenção que possibilitou executar de forma paralela as operações de atenção ao empilhá-las numa só camada, assim todos a sequência é processada de uma só vez. Tal avanço fez com que um novo paradigma surgisse para as tarefas de PLN: a arquitetura transformador, do inglês *Transformer* (He et al., 2023; Tunstall; Werra; Wolf, 2022)

2.4.3 Transformador

O *Transformer* pode ser considerado a grande revolução dos últimos anos na área de inteligência artificial. Além de se tornar a abordagem mais importante para as tarefas de PLN, principalmente para a tradução automática, tal conceito tem sido aplicado em diversas outras áreas de estudo, como a visão computacional e processamento de áudio (Lin et al., 2022). Sua composição é apresentada na Figura 7, que possui, assim como as RNR, o codificador e decodificador.

Figura 7 – Arquitetura *transformer*



Fonte: Adaptado de Vaswani et al. (2017).

No que se refere ao codificador, este contém duas subcamadas: a Auto-atenção Multi-Cabeça (AMC), do inglês *Multi-Head Attention* e uma Rede de Alimentação Direta (RAD), do termo em inglês *Feed-Forwad Network* (Lin et al., 2022). Inicialmente, a sequência de textos a ser analisada é transformada em um vetor de *embeddings*, em que cada um é transmitido para o codificador. Então este consiste em, primeiro, armazenar a posição absoluta do *embedding* em relação a sequência do texto, visto que o *transformer* não tem recorrência, e em seguida passar tal *embedding* para a AMC (Vaswani et al., 2017). A subcamada de AMC é o local onde ocorrem os cálculos dos pesos de autoatenção. A ideia é executar várias vezes esse

mecanismo paralelamente sobre o mesmo *embedding*, em que cada representação é conhecida como cabeça. Esta operação é realizada pois assim é possível concentrar ao mesmo tempo em diferentes características semânticas da sequência (Tunstall; Werra; Wolf, 2022).

Para o cálculo desta etapa, cada *embedding* é mapeado em 3 vetores: *Query* (Q), *Key* (K) e *Value* (V). Após isso, aplica-se o produto escalar de Q com K, ou seja, uma multiplicação de matrizes de *embeddings*, com o objetivo de encontrar o quanto a palavra de Q está relacionada com cada uma das palavras de K. Valores altos significam que Q e K são similares. Em seguida, para não existirem valores grandes, o produto escalar é dividido pela raiz quadrada da dimensão do vetor de K. Após, utiliza-se a função *Softmax* para normalizar os dados e transformá-los em probabilidades, entre 0 e 1 e, no final, o valor é multiplicado por V (Tunstall; Werra; Wolf, 2022). A função de atenção é apresentada em (3).

$$Atenção(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (3)$$

Como comentado, o processo de auto-atenção é realizado várias vezes, e no final, todos as cabeças são concatenadas e transformadas em um só vetor. Com isso, a operação da AMC se encerra. Após a subcamada de AMC, aplica-se ao vetor resultante uma conexão residual seguida de uma normalização, isto é, o *embedding* de entrada antes de passar para AMC é somado com a sua saída e, após esse cálculo, os valores são normalizados. Por último, a subcamada RAD é acionada, composta por duas operações lineares em conjunto com uma função de ativação. Deste modo, cada vetor de saída da AMC é transformado em uma representação não linear (Vaswani et al., 2017).

Em relação a decodificador, este contém duas subcamadas de auto-atenção, além da RAD. A primeira subcamada de auto-atenção, que recebe os *embeddings* de entrada após armazenar a sua posição absoluta, foi modificada para apenas focar nas saídas anteriores e evitar o acesso às sequências posteriores. A segunda subcamada de auto-atenção processa a saída do codificador e permite o uso de seu contexto. Em todas as subcamadas também são realizadas as conexões residuais e normalização, os mesmos processos descritos no codificador (Vaswani et al., 2017).

A arquitetura *transformer* se tornou a base para os Modelos de Linguagem Pré-treinados (PLM), e os Grandes Modelos de Linguagem (LLM), os quais

proporcionaram um avanço na inteligência das interfaces de linguagem natural (Shen et al., 2023). Esses paradigmas unem o entendimento da linguagem e o raciocínio, assim torna-se mais fácil o desenvolvimento de interfaces robustas e adaptáveis que relacionam a compreensão do contexto linguístico e a geração de visualizações (Maddigan; Susnjak, 2023).

2.5 MODELOS DE LINGUAGEM PRÉ-TREINADOS E GRANDES MODELOS DE LINGUAGEM

Modelos de linguagem (MLI) são sistemas que preveem a próxima palavra ou *token* baseada numa sequência de textos de entrada. A associação dos conceitos de pré-treinamento e MLI consiste na utilização de um grande volume de textos não rotulados, no qual o modelo aprende, por meio do aprendizado autossupervisionado, a estrutura geral da linguagem e seus padrões de uso sem direcionar para uma tarefa específica. Após a conclusão do pré-treinamento, o MLI pode ser modificado para um serviço específico, por exemplo a tradução da linguagem natural para SQL, por meio de ajustes finos (Sun; Luo; Luo; 2022; Wang et al., 2023).

Com a chegada do *Transformer* em substituição as RNR, os modelos de linguagem pré-treinados se tornaram rapidamente o padrão para as tarefas de PLN. Um dos PLMs mais populares atualmente é o *Bidirectional Enconder Representations from Transformers* (BERT), que tem como funcionamento o método de Modelo de Linguagem Mascarado (MLM). A ideia dessa técnica é a exclusão de uma determinada palavra pertencente a um texto e com base na análise bidirecional do contexto das palavras restantes, o modelo faz uma previsão e preenche com a informação mais adequada. O BERT é usado principalmente em tarefas de classificação e de perguntas/respostas. Atualmente existem diversas variações do modelo base, como o *RoBERTa*, que otimiza as técnicas de pré-treinamento em comparação ao BERT original e o *BERTimbau*, o qual foi pré-treinado para o idioma português-brasileiro (Sun; Luo; Luo; 2022).

A evolução dos PLMs ao longo dos anos possibilitou um aumento significativo no número parâmetros utilizados, o que levou ao surgimento dos grandes modelos de linguagem. Tais modelos usam a mesma arquitetura presente nos PLMs, porém devido a quantidade expressiva de atributos, geralmente na casa dos bilhões, o modelo tem um aprimoramento na capacidade de aprendizado. Como consequência,

compreendem melhor a complexidade da linguagem, geram respostas com mais detalhes e executam tarefas das mais diversas áreas. Por outro lado, os PLMs são mais eficazes em capturar recursos semânticos de uso geral, e devido a isso, conseguem solucionar várias tarefas de PLN, especialmente por meio do pré-treinamento seguido de ajustes finos para adaptação a tarefas específicas (Zhao et al., 2023).

Como exemplo de LLMs, tem-se como principais a família *Generative Pre-trained Transformer* (GPT), um conjunto de modelos de código fechado desenvolvido pela empresa *OpenAI* e o *Llama*, de código-aberto, construído pela empresa *Meta*. Em relação aos modelos do GPT, a cada nova versão lançada, há uma melhoria na capacidade do modelo de adquirir conhecimento, no suporte a longos textos de entradas e otimizações de desempenho, além do aumento do número de parâmetros. Atualmente o GPT-4 é a última versão disponível, com sua variação GPT-4 Turbo, que tem como diferencial em relação as versões anteriores a possibilidade de criar assistentes para atividades específicas e o aprimoramento da capacidade em outros campos sem ser a escrita, como visão, fala e audição (Zhao et al., 2023).

Sobre os sistemas de código-aberto, o *LLama* é um dos modelos mais populares devido à sua efetividade e liberdade de modificações. Seu lançamento permitiu a criação de inúmeros LLMs especializados em tarefas com um custo computacional reduzido em relação ao modelo base. Atualmente, a versão mais recente é a 3.1, que apresenta melhorias na quantidade de textos como entrada, no número de parâmetros e em novas ferramentas de segurança, além de alcançar resultados semelhantes em diferentes *benchmarks* em comparação ao GPT-4. (Zhao et al., 2023).

Como mencionado, ao realizar o pré-treinamento, tanto PLMs quanto os LLMs possuem um conhecimento geral na realização de tarefas, mas existe uma carência em resolver problemas para domínios específicos. Para solucionar essa limitação, realiza-se uma adaptação, chamada de ajuste fino ou do inglês *fine-tunnig*, que permite que tais modelos se tornem especialistas em uma determinada área (Zhao et al., 2023).

2.6 AJUSTE FINO

O ajuste fino dos modelos de linguagem tem como objetivo aumentar a precisão na execução de tarefas específicas por meio de técnicas que aprimoram a arquitetura do modelo e otimizam os parâmetros e os métodos de aprendizado (Wang et al., 2024). No contexto da tradução da LN para SQL, essa adaptação consiste em um treinamento adicional com o objetivo dos modelos entenderem as estruturas da atividade. Para os PLMs, são usados dados compostos por perguntas em linguagem natural combinadas com as tabelas e colunas do banco de dados. cujo intuito é reconhecer se uma tabela ou coluna é relevante para a questão, uma tarefa de classificação binária. Em relação aos LLMs, além dessas duas informações, também são fornecidos os códigos SQL referente a pergunta, assim o modelo aprenda a gerar consultas completas.

Diante desse cenário, são utilizadas base de dados de pares de pergunta-SQL que alimentam o modelo no processo de treinamento e teste. Os conjuntos *Spider* e o *WikiSQL* são considerados os primeiros corpus voltados à tarefa de pergunta-SQL, em que a complexidade é a principal diferença entre essas bases. O WikiSQL possui cerca de 80.000 pares de perguntas-SQL extraídos de aproximadamente 24.000 tabelas do *Wikipédia*, porém tais consultas são simples pois abordam apenas uma tabela por código (Zhong; Xiong; Socher, 2017). Em relação ao *Spider*, tem-se cerca de 10.000 questões com as respectivas consultas em SQL, oriundas de 200 bancos de dados de inúmeros domínios. Essas consultas contêm ligações de diferentes tabelas por meio do comando JOIN, além de outras operações matemáticas que aumentam sua complexidade (Yu et al., 2018).

Como avaliação do ajuste fino para os PLMs, devido ao fato de ser uma tarefa de classificação binária, são aplicadas as métricas Acurácia, Precisão, Revocação e F1-Score. A acurácia consiste em calcular a proporção de classificações corretas, ou seja, tabelas/colunas relevantes e não relevantes corretamente identificadas (verdadeiros positivos e verdadeiros negativos, respectivamente) em relação a todo o conjunto de dados classificados. Na precisão é apresentada a proporção de tabelas/colunas classificadas corretamente como relevantes (verdadeiros positivos) sob o conjunto total das tabelas/colunas classificadas como relevantes, com inclusão das tabelas/colunas classificadas incorretamente como relevantes (falsos positivos). A revocação, também chamada de sensibilidade, calcula as tabelas/colunas classificadas corretamente como relevantes em relação as tabelas/colunas que são de fato relevantes, isso inclui as tabelas/colunas classificadas como irrelevantes mas

são relevantes (falsos negativos). Por último, F1 é uma medida harmônica que relaciona precisão e revocação, e representa um equilíbrio entre essas duas métricas. Com isso, um valor alto de F1 depende também de valores altos de precisão e revocação, útil para conjuntos com classes desbalanceadas (Miller et al., 2024).

As fórmulas das métricas de Acurácia, Precisão, Revocação e F1-Score são apresentadas em (4), (5), (6) e (7), respectivamente. VP consiste nos verdadeiros positivos (tabelas/colunas classificadas corretamente como relevantes), FP são os falsos positivos (tabelas/colunas irrelevantes classificadas incorretamente como relevantes), VN representa os verdadeiros negativos (tabelas/colunas classificadas corretamente como irrelevantes) e FN simboliza os falsos negativos (tabelas/colunas relevantes classificadas incorretamente como irrelevantes).

$$Acurácia = \frac{VP + VN}{VP + VN + FP + FN} \tag{4}$$

$$Precisão = \frac{VP}{VP + FP} \tag{5}$$

$$Revocação = \frac{VP}{VP + FN} \tag{6}$$

$$F1 = 2 \times \frac{Precisão \times Revocação}{Precisão + Revocação} \tag{7}$$

Sobre os LLMs, a validação consiste na comparação das consultas geradas pelo modelo em relação ao conjunto de testes, processo realizado de forma manual. Além disso, a realização do ajuste fino no modelo inteiro demanda de uma grande quantidade de recursos computacionais, principalmente para a unidade de processamento gráfico (GPU). Dessa forma, técnicas eficientes de ajuste fino, do termo em inglês *Parameter Efficient Fine Tuning* (PEFT), desempenham um papel crítico na otimização desses recursos (Hu et al., 2023).

2.6.1 Ajuste fino de parâmetros de modo eficiente

A ideia por trás do PEFT é realizar o ajuste em apenas uma parte do LLM em vez de modificá-lo por inteiro, isto é, as camadas da rede neural são congeladas e um pequeno conjunto de parâmetros é alterado para adaptar a tarefa desejada. Como consequência, tem-se uma pequena queda de eficácia em comparação ao ajuste fino completo, porém, em troca, menos recursos computacionais são gastos além da possibilidade de evitar o esquecimento catastrófico, quando o sistema esquece o conhecimento adquirido anteriormente (Hu et al., 2023; J et al., 2024).

A técnica conhecida como *Low Rank Adaptation* é uma das abordagens PEFT mais utilizadas atualmente. Tal método consiste na adição de duas subcamadas de funções lineares para cada módulo de atenção do *Transformer*, em que o objetivo é reduzir a dimensão do vetor de entrada e, em seguida, retornar a dimensão original. Isso possibilita diminuir de forma expressiva a quantidade de parâmetros necessários para o ajuste fino, geralmente na ordem de milhões e, com isso, tem-se uma redução drástica do uso de memória e armazenamento durante o procedimento (J et al., 2024).

Ao mesmo tempo, o processo de quantização pode ser implantado para diminuir a quantidade de memória consumida pelos LLMs sem afetar significativamente a precisão do resultado. De maneira geral, cada parâmetro ou peso de um grande modelo de linguagem contém 32 bits de tamanho, dos quais 1 bit é reservado para o sinal, 8 bits são usados para o expoente e o restante para a mantissa do ponto flutuante. Neste cenário, um LLM considerado pequeno, de 7 bilhões de parâmetros, consome 28 GB de memória. Com a aplicação da quantização, que tem como papel transformar um conjunto de valores contínuos em discretos, os pesos são reduzidos a uma escala menor, com 16 bits, por exemplo. Com isso, tem-se 5 bits guardado para o expoente e 10 para a mantissa. Como exemplo desse processo no contexto do PEFT, o *Quantized LoRA*, a evolução do método *LoRA*, possibilita quantizar os parâmetros de um modelo para 4 bits, assim gasta-se cerca 3,5 GB de memória em troca de uma pequena queda de precisão (J et al., 2024).

Após o processo de ajuste fino, uma maneira adicional de aprimorar os LLMs é o uso de instruções ou *prompts* que focam numa tarefa específica. Tal abordagem chama-se Engenharia de Instruções, do inglês *Prompt Engineering* (EI), e seu principal objetivo é elaborar textos que explicam para o modelo a tarefa a ser realizada, por exemplo, a descrição da atividade, o contexto das informações de entrada e o formato desejado da saída (Zhao et al., 2023). O aprendizado em contexto (AC), também conhecido como *In-Context Learning* do inglês, é uma técnica de EI que

relaciona comandos e exemplos numa só instrução e, ao combiná-la com o ajuste fino, tem-se uma melhoria significativa na tradução da linguagem natural para SQL.

2.7 APRENDIZADO EM CONTEXTO

A grande vantagem do aprendizado em contexto, ao comparar com o ajuste fino, é que não há a necessidade de treinamento, ou seja, o LLM se adapta temporariamente ao *prompt* de entrada sem precisar ajustar os seus parâmetros. Dessa forma, há um gasto mínimo de recursos computacionais, além de ser possível aplicar os modelos de linguagem como serviços, os quais são utilizadas para as tarefas do cotidiano (Dong et al., 2023).

2.7.1 Técnicas de aprendizado em contexto de texto para SQL

Em relação ao cenário de texto para SQL, as instruções passadas para o LLM consistem essencialmente de 2 itens: o esquema do banco de dados (tabelas e colunas) e a pergunta. No entanto, quando o esquema atrelado a questão é complexo, com milhares de tabelas e colunas, uma série de problemas surgem. Por exemplo, o LLM possui um limite de *tokens* de entrada, então bancos de dados grandes não cabem num só contexto. Além disso, trabalhos na literatura demonstraram que inserir mais informações no prompt não aumenta a eficácia da consulta gerada. Inclusive, existe ainda a possibilidade de diminuir o desempenho pois um processamento adicional deve ser efetuado para gerar o resultado. Em complemento, mesmo para casos em que o esquema inteiro se encaixa nas instruções, a melhor maneira é selecionar as tabelas e colunas que estejam associadas com a pergunta do usuário (Floratou et al., 2024).

Diante dessa situação, o método pertencente a classe do AC chamado Ligação de Esquemas, do inglês *Schema Linking* (SL), atua na escolha das tabelas e colunas mais importantes da questão. Isso possibilita que o LLM enfatize os atributos relevantes e gere a consulta SQL com mais precisão (Maamari et al., 2024). Alguns trabalhos na literatura realizam o SL diretamente no LLM, como é o caso de Ge et al., (2023), que solicita ao modelo que identifique as tabelas e colunas significativas em um *prompt* e depois transfere tais dados para um novo contexto que constrói o SQL. Outros artigos abordam técnicas de similaridade, por exemplo, o trabalho de Zhang et

al., (2023) representa o esquema do banco de dados no formato Tabela.Coluna e, em seguida, calcula a semelhança da pergunta para cada tabela e coluna. Os atributos que possuírem os maiores valores condizem com as tabelas e colunas mais relevantes (Shi et al., 2024). Porém, há uma carência de estudos que combinem diferentes modelos de linguagem para o processo de SL e geração de SQL, em que cada um cumpre uma tarefa específica.

Como complemento, existem PLMs que foram desenvolvidos para relacionar a linguagem natural com dados tabulares, por exemplo o TABERT (Yin et al., 2020). Isso permite que tais ferramentas sejam usadas no processo de SL. No entanto, uma limitação presente é que esses modelos foram treinados apenas para o idioma inglês, então ao trabalhar com outras línguas observa-se uma queda de precisão.

Além da ligação de esquemas, outras técnicas atreladas ao AC podem ser utilizadas na tarefa de texto para SQL. O método de Aprendizado por Zero Amostras (AZA) ou do inglês *Zero-shot Learning* se baseia em operar sem exemplos de consultas no prompt, ou seja, o modelo usufrui do seu conhecimento prévio para resolver a atividade desejada. De maneira contrária, o Aprendizado de Poucas Amostras (APA), do termo em inglês *Few-shot Learning,* consiste em apresentar alguns exemplos de pergunta para SQL como entrada e depois solicita ao modelo a geração do código referente a questão feita com base no contexto transmitido anteriormente. Por último, a técnica da Cadeia de Pensamento, do inglês *Chain of Thought* (CP), direciona o LLM a dividir a tarefa em várias etapas por meio de comandos como “Vamos pensar passo a passo”. Dessa forma, o modelo releva o que é feito em cada etapa, com explicações do seu raciocínio e apenas na última fase que o código SQL é elaborado (Shi et al., 2024).

2.7.2 Recuperação aumentada por geração

Uma outra estratégia que expandiu paralelamente à evolução dos LLMs é a Recuperação Aumentada por Geração, do inglês *Retrieval Augmented Generation* (RAG). Tal paradigma consiste em incluir, antes da geração da resposta pelo modelo, uma etapa de recuperação de informações relevantes a partir de uma base externa de documentos associada a pergunta do usuário. Essas informações, posteriormente, são inseridas como contexto adicional para que o LLM elabore respostas com mais qualidade (Gao et al., 2024).

Assim como o aprendizado em contexto, o método RAG evita a necessidade de realização do ajuste fino para tarefas específicas. Nesse caso, apenas a disponibilização da base de dados externa é o suficiente, a qual fornece informações adicionais ao modelo e contribui para melhorar a precisão das respostas para perguntas específicas (Gao et al., 2024).

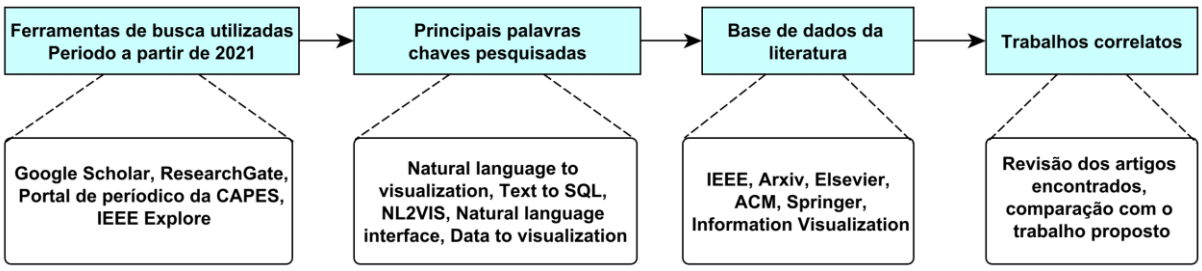
Além disso, tanto o RAG quanto as técnicas de ligação de esquemas possuem um objetivo em comum: identificar os elementos mais importantes baseado na pergunta do usuário. Porém, no cenário de texto para SQL, há uma diferença importante entre essas abordagens. Enquanto o RAG recupera trechos de texto ou documentos completos, as técnicas de ligação de esquemas realizam a seleção das estruturas do banco de dados, como tabelas, colunas relacionamentos e atributos.

Outra distinção relevante está na forma como o contexto é construído. O RAG busca enriquecer o prompt com informações adicionais. Por outro lado, a ligação de esquemas atua de forma contrária, com a filtragem do esquema do banco de dados para um contexto reduzido. Essa estratégia é importante quando se trabalha com banco de dados complexos, nos quais a inclusão de todas as tabelas e colunas no prompt ultrapassaria o limite de *tokens* no modelo ou poderia atrapalhar na geração da consulta SQL com informações irrelevantes sem relação com a pergunta.

2.8 TRABALHOS CORRELATOS

Neste subcapítulo são apresentados os trabalhos pertencentes a literatura que abordam a geração de visualização oriundas de perguntas realizadas em linguagem natural com utilização de um LLM ou PLM. A maioria dos estudos desenvolvem técnicas de engenharia de instruções em que são extraídos os atributos mais importantes da pergunta. Alguns artigos convertem o texto para consultas SQL, outros elaboram estruturas de dados específicas que facilitam a elaboração de gráficos e poucas pesquisas realizam o ajuste fino nos modelos para melhorar a eficácia na tarefa de texto para visualização. Na Figura 8 são descritas as etapas realizadas na escolha dos trabalhos correlatos.

Figura 8 – Etapas para a seleção dos trabalhos correlatos



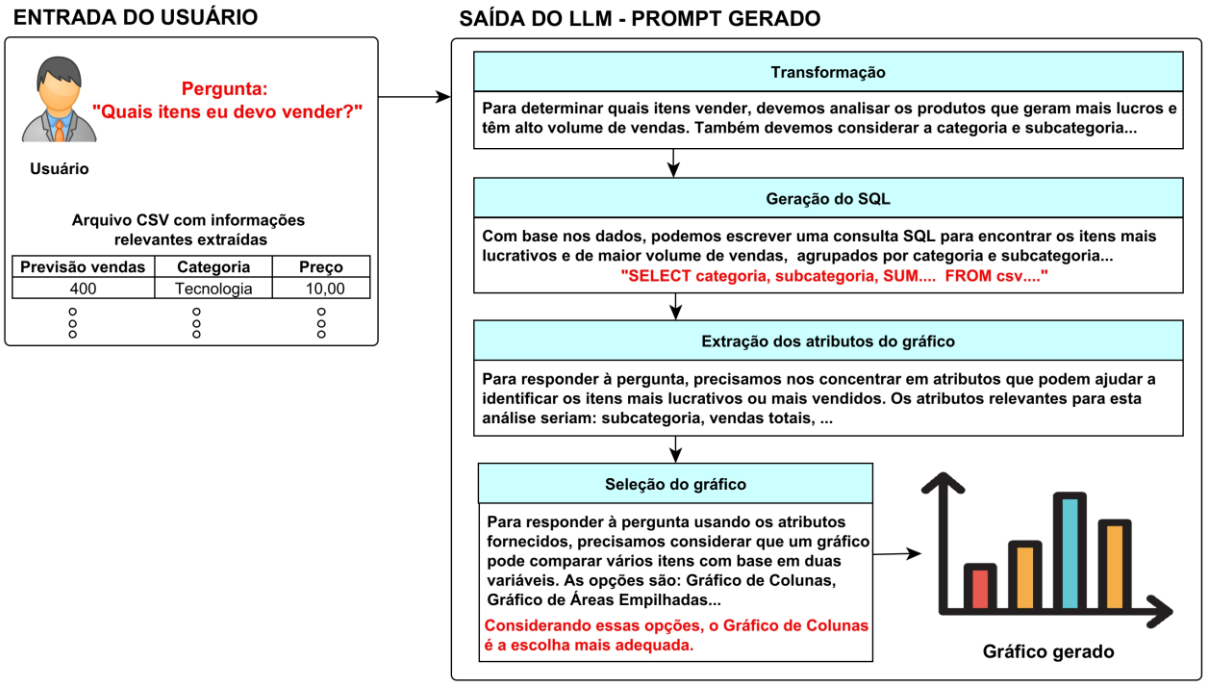
Fonte: Elaborado pelo autor.

2.8.1 Geração de gráficos a partir de instruções de alto nível

O trabalho elaborado por Lee et al., (2024) propõe um conjunto de etapas que transforma perguntas de usuários em gráficos por meio de um LLM combinado a técnicas de engenharia de instruções e a taxonomia de gráficos de Abela (2013). A arquitetura do trabalho recebe como entrada a pergunta e a base de dados, que são processadas em diferentes etapas conduzidas pelo modelo para extrair os atributos mais relevantes, gerar a consulta SQL correspondente e direcionar a construção da visualização com base na taxonomia adotada. Os resultados da pesquisa demonstraram um bom potencial para a geração automática de gráficos, porém foi apontado a necessidade de modificações na arquitetura para abranger diferentes conjuntos de dados e todos os gráficos possíveis da taxonomia utilizada.

Na Figura 9 são apresentados os passos na geração da visualização proposta por Lee et al., (2024). O fluxo inicia com a pergunta do usuário combinada com um arquivo *Comma-Separated Values* (CSV), que contém os dados tabulares. Ambos os dados são transmitidos ao LLM que analisa, pela técnica da CP, os itens mais relevantes associadas à pergunta. Em seguida, o modelo gera o código SQL e a consulta é executada no banco de dados. A partir dos resultados obtidos, o sistema identifica a visualização mais adequada e os atributos necessários para sua construção. Por último, a visualização é elaborada e exibida ao usuário.

Figura 9 – Passos para gerar a visualização do trabalho de Lee et al., (2024)



Fonte: Adaptado de Lee et al., (2024).

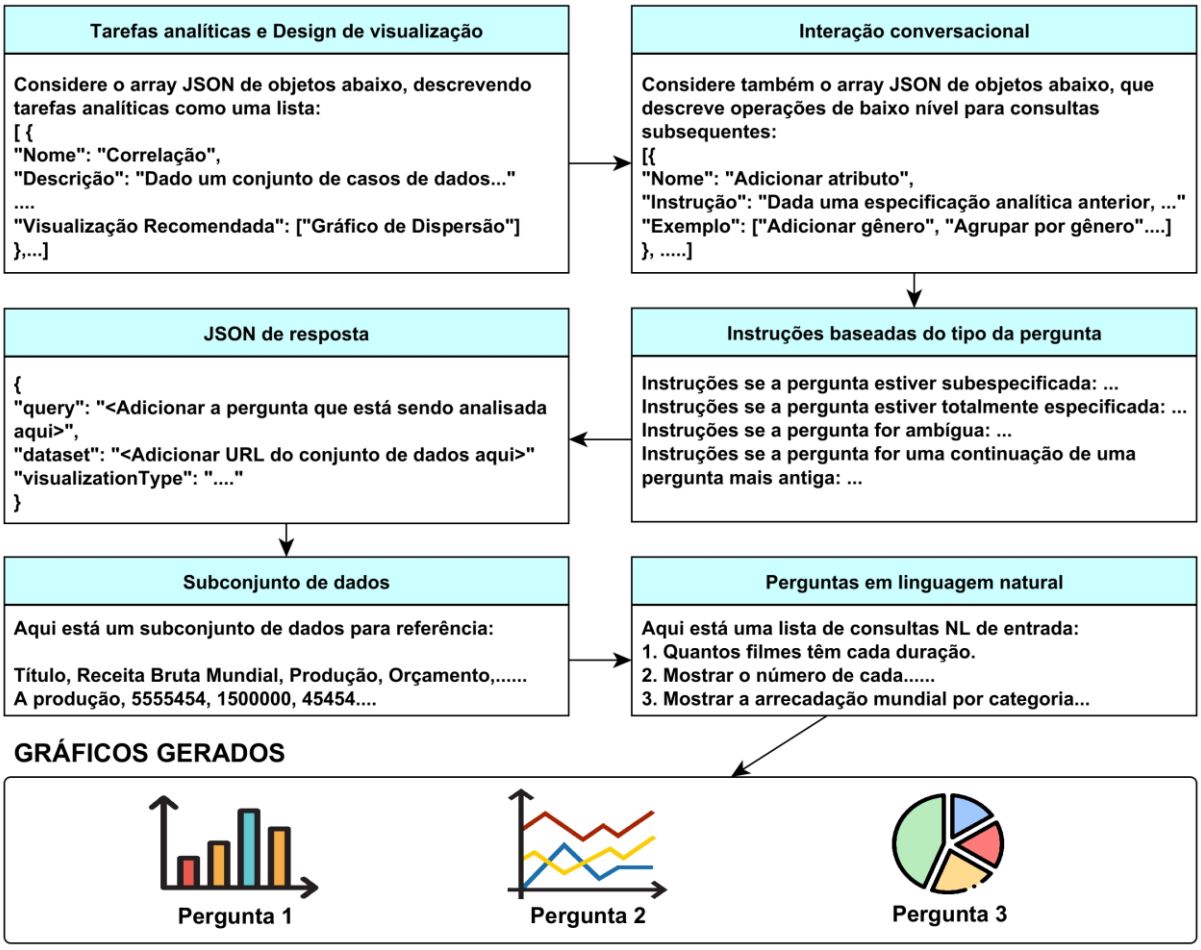
2.8.2 Elaboração de especificações analíticas para visualização de dados

Em relação ao trabalho de Sah et al. (2024), foi construído um conjunto de instruções de *prompts* para um LLM que gera visualizações baseada numa pergunta em linguagem natural sobre dados tabulares. Por meio do método de aprendizado em contexto, uma estrutura do tipo *JavaScript Object Notation* (JSON), a qual contém informações de nome, tipos de dados de atributos e recomendações de visualizações, foi transmitida para o GPT4 aprender seus padrões, além de instruções adicionais que lidam com perguntas ambíguas ou incompletas. Como saída, o sistema gera especificações de visualização na linguagem *Vega-Lite*, uma gramática de visualização de dados. Nos experimentos, foram avaliadas 740 consultas de um conjunto de dados pergunta-visualização, em que foi obtido uma acurácia de 87,02%.

Na Figura 10 são ilustradas as etapas do trabalho de Sah et al., (2024). O fluxo inicia com o envio de múltiplos blocos de instruções estruturados numa lista JSON, que definem o formato e as tarefas analíticas a serem seguidas. Após, são enviadas ao LLM instruções adicionais para lidar com diferentes tipos de pergunta e definir onde as informações devem ser posicionadas. Posteriormente, as perguntas do usuário e

o conjunto de dados são processados pelo modelo, que, ao final, gera as visualizações correspondentes a cada pergunta fornecida.

Figura 10 – Etapas do prompt do trabalho de Sah et al., (2024)



Fonte: Adaptado de Sah et al., (2024).

2.8.3 Sistema para análise de dados de exploração automática

O artigo publicado por Zhu et al., (2024) propõe um sistema de análise de dados de exploração automática que realiza o processamento de texto para SQL e a geração do gráfico por meio de técnicas de aprendizado de zero amostras e cadeia de pensamento aplicadas a um LLM. A arquitetura é composta por uma interface de usuário para inserção das perguntas e do conjunto de dados, e um módulo responsável por identificar palavras-chave da pergunta, extrair tabelas e colunas relevantes, gerar e otimizar consultas SQL e, no final, elaborar visualizações adequadas ao contexto da pergunta. Nos experimentos, a abordagem foi comparada aos modelos GPT-3.5 e GenSQL, tanto no conjunto de dados Spider quanto em bases

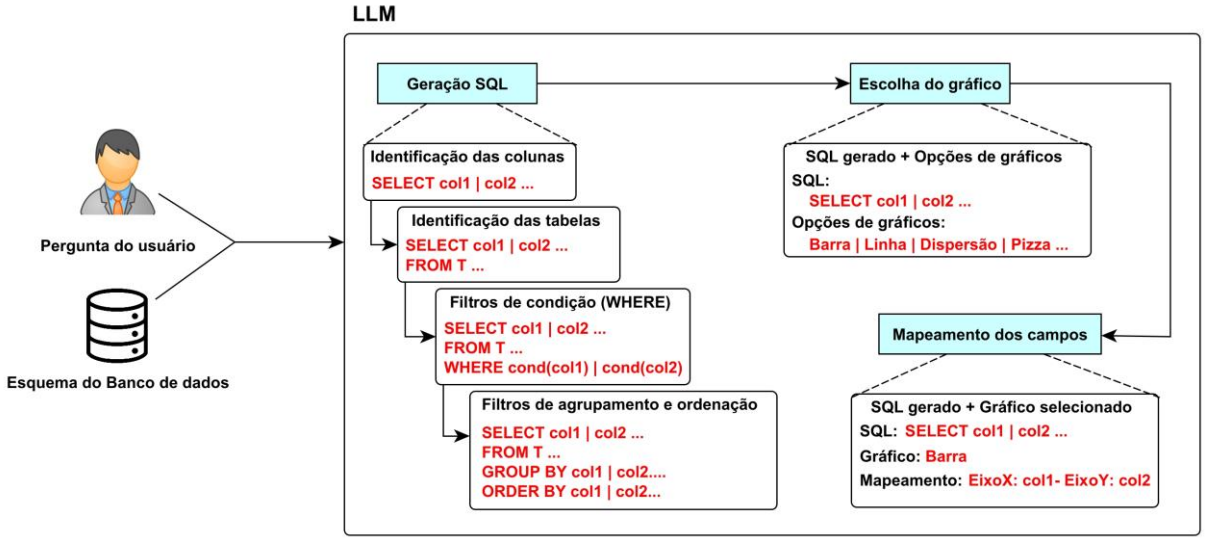
de dados do mundo real. O trabalho apresentou um desempenho superior principalmente em perguntas de maior complexidade e resultados comparáveis ao GenSQL nos demais cenários.

2.8.4 Solução para visualização automática de dados tabulares em banco de dados relacionais

O trabalho desenvolvido por Yang et al., (2024) se assemelha com artigos apresentados em relação a automatização de operações de visualização de dados tabulares e a geração de códigos SQL, ambos por meio de um LLM. A arquitetura se baseia na utilização do método CP, no qual o modelo, a partir do esquema do banco de dados e da pergunta do usuário, identifica progressivamente tabelas, colunas, filtros, agregações e ordenações para construir a consulta SQL. Em seguida, o sistema seleciona o tipo de gráfico mais adequado ao contexto da pergunta e realiza o mapeamento entre os atributos da consulta e os eixos da visualização. Nos experimentos, 141 bases oriundas de um banco de dados de pergunta-visualização foram selecionadas e técnicas de AZA e APA com e sem cadeia de pensamento foram aplicadas. Os resultados indicaram que a técnica de APA aliada à cadeia de pensamento apresentou o melhor desempenho, embora foi apontado a necessidade de métodos adicionais para aumentar a precisão do modelo.

Na Figura 11 é apresentada a arquitetura proposta por Yang et al., (2024). De início, tanto a questão do usuário quanto o esquema do banco de dados são transmitidos para o LLM, que gera a consulta SQL de forma incremental por meio da cadeia de pensamento. Em seguida, a consulta completa é enviada para a etapa de escolha do gráfico, que, com base nas opções disponíveis, determina a melhor visualização para o contexto. Por último, ocorre a etapa de mapeamento, a qual as colunas das consultas são associadas aos eixos do gráfico selecionado.

Figura 11 – Arquitetura do trabalho de Yang et al., (2024)



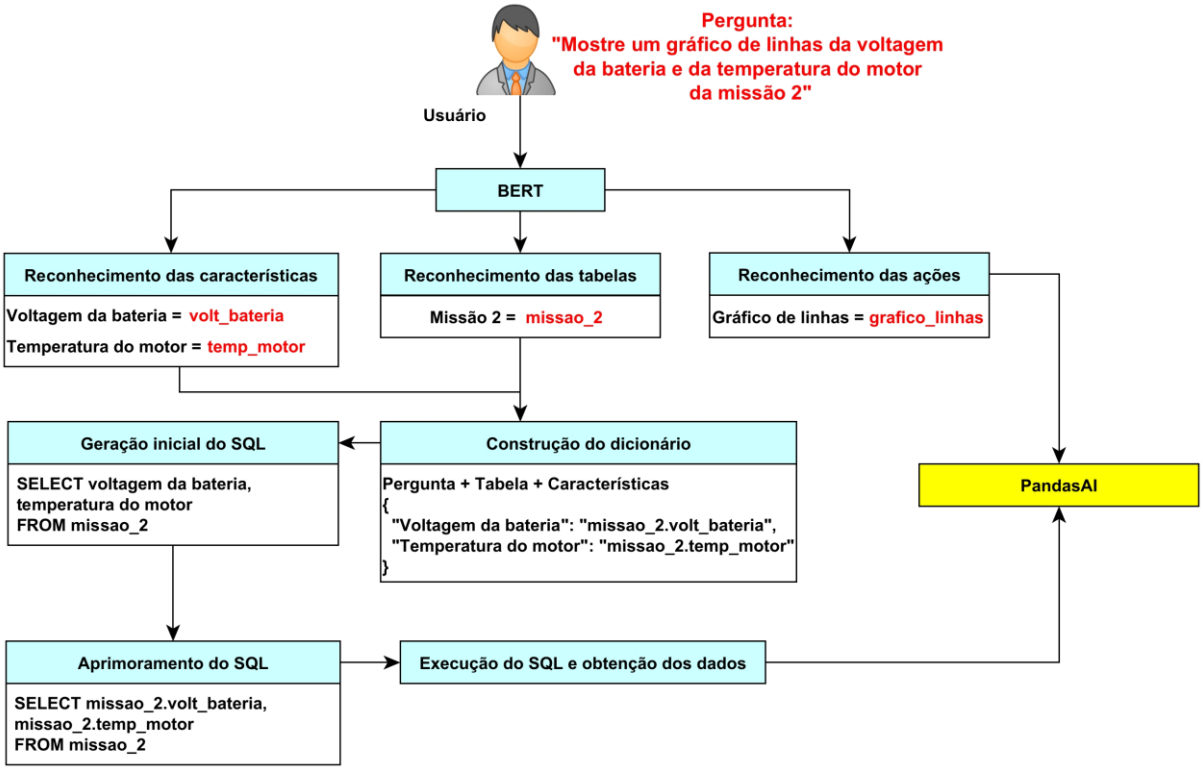
Fonte: Adaptado de Yang et al., (2024).

2.8.5 Fluxo de instruções para monitoramento de saúde de aeronaves

Ram, Ashinee e Kumar (2024) elaboraram um sistema para monitoramento em tempo real da saúde de aeronaves em missões espaciais. A metodologia baseou-se na construção de bases de dados voltadas a missões espaciais e no uso do método de Reconhecimento de Entidade Nomeada (REN) para realizar dois ajustes finos no modelo BERT: o primeiro identifica o tipo de gráfico e colunas relevantes, e o segundo detecta o esquema e as tabelas do banco de dados. A saída do BERT e a pergunta são transferidas tanto para um LLM e um algoritmo ingênuo, os quais elaboram a consulta SQL. Os dados obtidos após a execução do código são analisados pela ferramenta *PandasAI*, que constrói a visualização. Nos testes realizados, o trabalho atingiu uma acurácia de 71% em 50 consultas avaliadas por pessoas.

Na Figura 12 é ilustrado o funcionamento do trabalho de Ram, Ashinee e Kumar (2024). O BERT recebe a pergunta de entrada e identifica o tipo de visualização, as tabelas e colunas relevantes. Em seguida, um dicionário é criado que relaciona a pergunta com as tabelas e colunas identificadas. Tal dicionário serve de base para a construção de um código SQL inicial, o qual é posteriormente aprimorado com a inserção das tabelas/colunas reais do banco de dados. Por último, a consulta elaborada é executada, e os dados retornados são transmitidos para o *PandasAI*, juntamente com o tipo de visualização reconhecido pelo BERT.

Figura 12 – Fluxo do trabalho de Ram, Ashinee e Kumar (2024)



Fonte: Adaptado de Ram, Ashinee e Kumar (2024).

2.8.6 Sistema de geração de interfaces multi-visualização

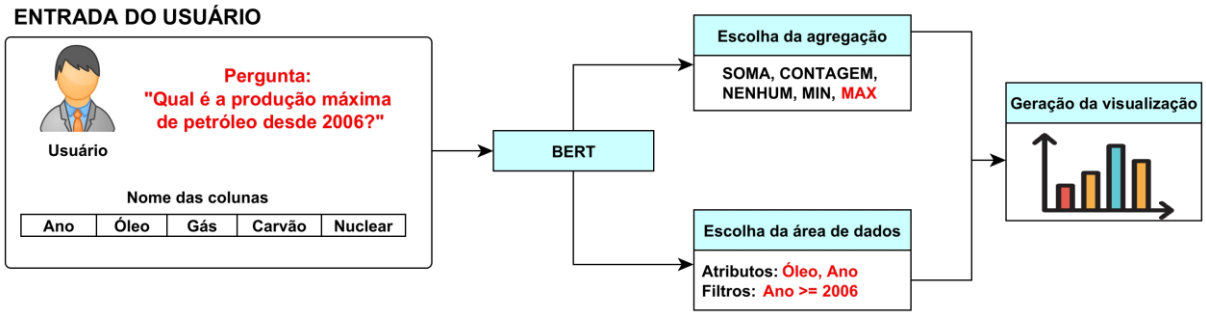
O estudo de Chen et al., (2022) criou um sistema para gerar interfaces de multi-visualização oriunda de textos em linguagem natural, os quais são transformados num formato chamado SQL parametrizado com o uso de um LLM. Por meio das técnicas de APA e AC, o modelo é preparado para criar consultas que incluem atributos de escolha, categorizados como ANY, OPT e SUBSET, e que permitem gerenciar grandes conjuntos de variações em um único código. Após a geração do SQL parametrizado, é realizado um mapeamento para um outro sistema de geração de interfaces baseado em SQL, o qual constrói uma visualização gráfica para cada consulta. Nos experimentos realizados com 12 perguntas de bases de dados distintas, a abordagem alcançou uma acurácia de 66%, o que indica a necessidade de melhorias nas instruções do APA e uma avaliação sistemática mais robusta nos trabalhos futuros.

2.8.7 Fluxo de processamento de visualização automática para perguntas em linguagem natural provenientes de tabelas

O trabalho de Liu et al., (2021) propôs um fluxo de processamento para criar visualizações baseadas na linguagem natural e dados tabulares. Por meio de um PLM, ajustado com a base de dados WikiSQL, as perguntas e os dados tabulares são passados para os módulos de classificação do modelo que reconhecem as principais colunas, os filtros de condição e os filtros de agregação. Posteriormente, algumas regras são aplicadas que relaciona as informações obtidas na escolha do melhor gráfico, que é gerado no final do processo. Nos testes de validação, os módulos atingiram acurácia superior a 80% na base WikiSQL, e comparações com o estado da arte e ferramentas comerciais demonstraram que a abordagem obteve desempenho superior tanto em precisão quanto na eficácia das visualizações.

Na Figura 13 é apresentado um exemplo do funcionamento do trabalho de Liu et al., (2021). Como entrada, são passadas para o BERT a pergunta e as colunas associadas. O modelo realiza dois processos principais: a escolha das agregações e a definição da área de dados, que inclui os atributos e filtros mais adequados ao contexto da pergunta. Por fim, algumas operações adicionais são realizadas para que, no final, a visualização correspondente seja gerada.

Figura 13 – Fluxo do trabalho de Liu et al., (2021)



Fonte: Adaptado de Liu et al., (2021).

2.9 CONSIDERAÇÕES FINAIS

Neste capítulo, foi apresentada a fundamentação teórica deste trabalho, com foco nas na relação entre o processamento de linguagem natural e a visualização de dados, além de destacar as principais abordagens utilizadas na construção de interfaces baseadas em linguagem natural. Adicionalmente, foram apresentados sete trabalhos correlatos voltados a conversão de textos em linguagem natural para

visualização de dados. Ao analisá-los, foi possível verificar que a utilização de um LLM ou PLM para a tarefa de Pergunta-Visualização ainda se encontra em fases iniciais e, portanto, há espaço para avanços nesta área. Por exemplo, existe uma limitação de visualizações a serem geradas, em que são abordados apenas gráficos mais simples, como barras ou linhas, e nenhum dos artigos abrange para representações complexas, como mapas, diagramas de Sankey e gráficos de rede. Outro fator envolve a ausência de diferentes modelos de linguagem de trabalharem em conjunto, em que cada um se especializa numa tarefa por meio de ajustes finos.

Desta forma, com base na amostragem e análise dos artigos, foi construído um quadro, apresentado no Quadro 1, que compara os estudos em relação aos principais conceitos discutidos neste capítulo.

Quadro 1 – Comparativo entre os trabalhos correlatos

	Lee et al., (2024)	Sah et al., (2024)	Zhu et al., (2024)	Yang et al., (2024)	Ram, Ashinee e Kumar (2024)	Chen et al., (2022)	Liu et al., (2021)
Geração de visualização oriundas de linguagem natural por meio de um LLM ou PLM	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Uso de técnicas de aprendizado em contexto	Sim	Sim	Sim	Sim	Não	Sim	Não
Extração dos atributos mais importantes da pergunta do usuário	Sim	Sim	Sim	Sim	Sim	Não	Sim
Uso de ajuste fino no LLM ou PLM	Não	Não	Não	Não	Sim	Não	Sim
Conversão da linguagem natural para SQL	Sim	Não	Sim	Sim	Sim	Sim	Não
Geração de visualizações complexas	Não	Não	Não	Não	Não	Não	Não
Combinação de diferentes modelos de linguagem	Não	Não	Não	Não	Não	Não	Não

Fonte: Elaborado pelo autor.

CAPÍTULO

3

3. ARQUITETURA HÍBRIDA BASEADA EM MODELOS DE LINGUAGEM PARA VISUALIZAÇÃO INTELIGENTE DE DADOS

Neste capítulo é detalhado o desenvolvimento da arquitetura proposta, a qual tem como objetivo a geração automática de visualizações de dados adequadas ao contexto de uma pergunta em linguagem natural feita em base de dados relacionais. A arquitetura se baseia na combinação de dois modelos de linguagem para o processamento de texto para SQL, em que um PLM reconhece as tabelas e colunas mais importantes com base na pergunta do usuário e no esquema do banco de dados associado, e um LLM, a partir do contexto fornecido pelo modelo anterior e a pergunta original, gera a consulta SQL correspondente.

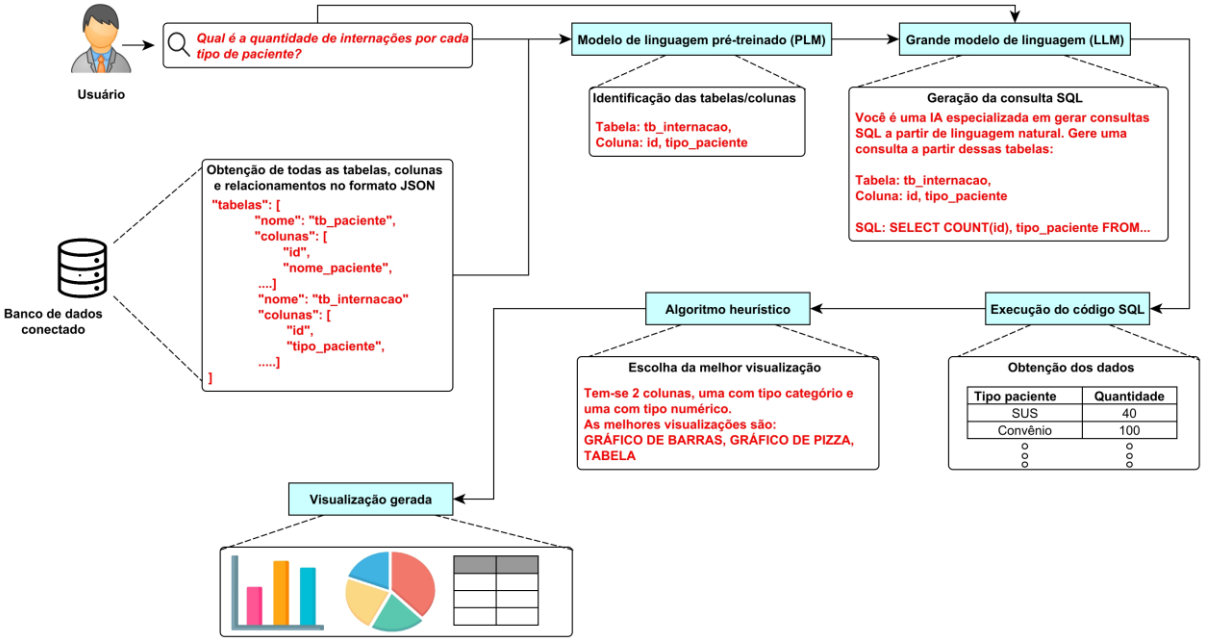
Após a construção e execução do SQL, os dados obtidos são analisados por um algoritmo heurístico baseado em uma árvore de decisão, cujo objetivo é identificar características estruturais do conjunto de dados e direcionar para as visualizações mais adequadas ao contexto da pergunta, que são então apresentadas ao usuário. Dessa forma, a arquitetura integra modelos de linguagem e uma técnica heurística em um fluxo que se inicia com a pergunta em linguagem natural e tem como resposta a visualização final dos dados.

3.1 VISÃO GERAL DA ARQUITETURA

De maneira geral, a arquitetura é composta por seis etapas principais nesta ordem: entrada da pergunta em linguagem natural do usuário; reconhecimento das tabelas e colunas importantes, realizado pelo modelo de linguagem pré-treinado; geração de consultas SQL, baseado no grande modelo de linguagem; execução do código SQL e obtenção dos dados; análise do conjunto de dados pelo algoritmo

heurístico e direcionamento para as visualizações adequadas; construção e exibição das visualizações ao usuário. Na Figura 14 é ilustrado o fluxo da arquitetura proposta com todas as etapas citadas.

Figura 14 – Fluxo da arquitetura híbrida proposta



Fonte: Elaborado pelo autor.

A etapa do PLM recebe como entrada a pergunta do usuário e uma representação estruturada do banco de dados conectado, o qual contém os nomes das tabelas, colunas e relacionamentos. O intuito dessa fase é filtrar semanticamente os elementos do esquema mais importantes relacionados a pergunta. Em seguida, o contexto obtido por essa etapa, juntamente com a pergunta do usuário, é passado para o LLM, o qual elabora a consulta SQL correspondente e executa no banco de dados. Essa filtragem semântica combinada com a consulta SQL constitui a arquitetura híbrida proposta.

Os dados retornados pela consulta são então processados pelo algoritmo heurístico que verifica a estrutura do conjunto de dados, como a quantidade de colunas e os tipos de dados presentes. Com base nessa análise, o sistema seleciona os tipos de visualização mais adequados ao contexto. No final, as visualizações selecionadas são construídas e apresentadas ao usuário por meio de uma ILN, a qual foi desenvolvida com o objetivo de validar a arquitetura.

3.2 ASPECTOS TÉCNICOS DA ARQUITETURA

A ILN foi desenvolvida com o uso da linguagem de programação *Python* para o *backend*. Para o *frontend*, utilizou-se a linguagem *TypeScript* 5.7.2 combinada ao framework *React* 19.0.0. Pelo fato de a interface suportar dois modelos de linguagem, foi adotado o framework *FastAPI*, que permite a conexão com os modelos por meio de requisições *Hypertext Transfer Protocol* (HTTP). O SGBD usado foi o *PostgreSQL* 12, então houve a necessidade da integração da biblioteca *Psycopg2* 2.9.10. Para a geração das visualizações, foram empregadas as bibliotecas *Chart.js* 4.4.8, *React-Force-Graph-2d* 1.29.0 e *Leaflet* 1.9.4, responsáveis pela construção de gráficos simples, gráficos de redes e mapas, respectivamente.

No caso do PLM, foram selecionados três modelos pré-treinados para o português brasileiro, baseados na arquitetura BERT: *BERTimbau* base, *BERTimbau* grande e *BERTugues*. Todos foram ajustados para reconhecer as tabelas e colunas mais relevantes de uma pergunta associada a um banco de dados. Essa tarefa é vista como uma classificação binária, em que o modelo avalia, para cada tabela e coluna pertencente ao banco, se o atributo é ou não relevante a questão do usuário. No final da comparação, o PLM com o melhor resultado foi integrado a ILN.

Em relação ao LLM, foi utilizado o modelo *Llama Instruct 3.1*, com 8 bilhões de parâmetros. Neste trabalho, foi realizado um ajuste fino para gerar consultas SQL com base na pergunta do usuário. Tal modelo foi comparado a outros dois modelos: a sua versão base, sem qualquer ajuste fino, e o SQLCoder, especializado na tarefa de texto para SQL. Assim como no caso do PLM, o LLM com o melhor desempenho foi integrado à interface.

Para os ajustes nos PLMs e o *Llama*, foram empregadas as bibliotecas *Transformers* 4.48.3 e *PyTorch* 2.5.1+cu124 para as operações de treinamento e validação. No caso do *Llama*, também foram usadas as bibliotecas *PEFT* 0.14.0 e *BitsAndBytes* 0.45.3, responsáveis pela aplicação das técnicas *LoRA* e *QLoRA*, respectivamente. Em relação aos dados de treinamento, foi utilizado o conjunto de dados *Spider* (Yu et al., 2018), o qual contém pares de perguntas-SQL com seus respectivos bancos de dados.

Para a parte das visualizações, um algoritmo heurístico foi desenvolvido baseado no trabalho de Healy e Holtz (2017), o qual constrói as visualizações por meio de uma árvore de decisão. Essa estrutura suporta, no total, 17 tipos de

visualizações, que vão desde gráficos mais simples até representações mais complexas, como mapas, diagrama de Sankey, gráficos de rede etc.

3.3 PREPARAÇÃO DOS DADOS PARA OS AJUSTES FINOS

Para a realização do ajuste fino dos modelos de linguagem, foi utilizado o conjunto de dados *Spider* (Yu et al., 2018), o qual possui, no total, 9991 pares de perguntas-SQL oriundos de 139 bases de dados. Originalmente, cada base de dados estava armazenada em uma pasta com dois arquivos: o código SQL do esquema, com as tabelas, colunas e relacionamentos, e uma lista de perguntas com suas respectivas consultas. Para facilitar a manipulação das informações, tais dados foram unidos em um único arquivo JSON. Nesse novo formato, cada objeto contém o código do banco de dados e uma lista de perguntas-SQL associadas.

Além disso, o conjunto *Spider* foi construído apenas no idioma inglês, assim tornou-se necessária a tradução completa para o português, ou seja, da pergunta, dos nomes das tabelas e colunas, e da consulta SQL. Como os comandos SQL não podem ser traduzidos, ferramentas tradicionais de tradução como o *Google Tradutor* não se mostraram adequada à essa tarefa. Por conta disso, foram utilizadas as interfaces do *ChatGPT* e *Gemini*, juntamente com instruções baseadas na técnica de aprendizado em contexto, com o objetivo de preservar as sintaxes do código. Ao final do processo, o *JSON* base foi montado, o qual é apresentado na Figura 15. Nessa figura, tem-se um exemplo de um banco de dados com algumas perguntas em inglês e a versão traduzida para o português, após a adaptação pelos modelos.

Figura 15 – Exemplo de um objeto do arquivo JSON em inglês e a respectiva tradução para o português

JSON - VERSÃO INGLÊS

```
"esquema": "CREATE TABLE author ( aid int, homepage text, name text, oid int, primary key(aid) );
CREATE TABLE journal ( homepage text, jid int, name text, primary key(jid) );
CREATE TABLE publication ( abstract text, cid text, citation_num int, jid int, ...
..."
"lista_perguntas_sql": [
 {
      "pergunta": "return me the homepage of PVLDB .",
      "sql": "SELECT homepage FROM journal WHERE name  =  'PVLDB';"
 },
 {
      "pergunta": "return me the homepage of ' H. V. Jagadish ' .",
      "sql": "SELECT homepage FROM author WHERE name  =  'H. V. Jagadish';"
 },
 {
      "pergunta": "return me the abstract of ' Making database systems usable ' .",
      "sql": "SELECT abstract FROM publication WHERE title  =  'Making database systems usable';"
 }
```

JSON - VERSÃO PORTUGUÊS

```
 "esquema": "CREATE TABLE autor (aid int, pagina_inicial text, nome text, oid int, primary key(aid));
CREATE TABLE jornal ( pagina_inicial text, jid int, nome text, primary key(jid) );
CREATE TABLE publicacao ( resumo text, cid text, numero_citacoes int, jid int, ...
..."
"lista_perguntas_sql": [
{
      "pergunta": "retorne-me a página inicial da PVLDB",
      "sql": "SELECT pagina_inicial FROM jornal WHERE nome  =  'PVLDB';"
},
{
      "pergunta": "retorne-me a página inicial de 'H. V. Jagadish'",
      "sql": "SELECT pagina_inicial FROM autor WHERE nome  =  'H. V. Jagadish';"
},
{
      "pergunta": "retorne-me o resumo de 'Making database systems usable'",
      "sql": "SELECT resumo FROM publicacao WHERE titulo  =  'Making database systems usable';"
}
```

Fonte: Elaborado pelo autor.

Após a montagem do JSON base, foram realizadas alterações adicionais com o intuito de criar os conjuntos de treinamento e teste para os modelos. No caso dos PLMs, tal conjunto incluiu a pergunta em linguagem natural, os nomes das tabelas e colunas de todo o banco de dados, os relacionamentos e os nomes das tabelas e colunas relevantes para pergunta, chamado de esquema relevante, extraído a partir da consulta SQL. O esquema relevante representa a saída esperada do PLM e os demais itens compõe a entrada. Na Figura 16 é apresentado um objeto do conjunto de treinamento/teste do modelo como exemplo. Percebe-se que tal objeto possui a pergunta, todos os nomes das tabelas e colunas com seus respectivos relacionamentos, e o esquema relevante.

Figura 16 – Exemplo de um objeto do conjunto de treinamento/teste do PLM

```
{
    "pergunta": "retorne-me a página inicial da PVLDB",
    "esquema": {
      "tabelas": [
        {
          "nome": "autor", "colunas": ["aid","pagina_inicial","nome","oid"]
        },
        {
          "nome": "conferencia", "colunas": ["cid","pagina_inicial","nome"]
        },
        {
          "nome": "dominio", "colunas": ["did","nome"]
        }, ...
      ]
    },
    "relacionamentos": [
      "autor_dominio.aid > autor.aid", "autor_dominio.did > dominio.did", ...
    ],
    "esquema_relevante": {
      "tabelas": [
        {
          "nome": "jornal", "colunas": ["nome","pagina_inicial"]
        }
      ]
    }
  }
```

Fonte: Elaborado pelo autor.

Em relação ao LLM, cada objeto utilizado como entrada contém a pergunta, o esquema relevante com seus relacionamentos, e a consulta SQL correspondente. Um detalhe a ser destacado é que, em cenários do mundo real, o modelo de linguagem pré-treinado dificilmente reconhecerá todas as tabelas e colunas importantes em todos os casos. Em algumas situações, poderão ser incluídas tabelas ou colunas irrelevantes ou a possibilidade de informações serem omitidas. Por conta disso, foram inseridos ruídos no conjunto de treinamento, de modo que cada objeto foi expandido para gerar 4 variações: o objeto original, uma versão com tabelas adicionais, outro com colunas adicionais e outra com colunas faltantes. Essa abordagem tem como objetivo permitir que o LLM aprenda a fazer inferências mesmo diante de um contexto não ideal, com ruídos ou incompleto. Na Figura 17 é mostrado um exemplo do conjunto de dados do LLM com as 4 variações citadas.

Figura 17 – Exemplo de um objeto do conjunto de treinamento/teste do LLM com as 4 variações

EXEMPLO ORIGINAL

```
{
    "pergunta": "retorne-me a página inicial da PVLDB",
    "relacionamentos": [
       "dominio_jornal.jid > jornal.jid", "publicacao.jid > jornal.jid"
    ],
    "esquema_relevante": {
       "tabelas": [
          { "nome": "jornal", "colunas": ["nome", "pagina_inicial"] }
       ]
    },
    "sql": "SELECT pagina_inicial FROM jornal WHERE nome  =  'PVLDB';"
 }
```

EXEMPLO COM ADIÇÃO DE TABELAS

```
{
    "pergunta": "retorne-me a página inicial da PVLDB",
    "relacionamentos": [
       "dominio_jornal.jid > jornal.jid", "dominio_palavra_chave.kid > ..."
    ],
    "esquema_relevante": {
       "tabelas": [
          { "nome": "jornal", "colunas": ["nome", "pagina_inicial"] },
          { "nome": "palavra_chave","colunas": ["palavra_chave"] }
       ]
    },
    "sql": "SELECT pagina_inicial FROM jornal WHERE nome  =  'PVLDB';"
 }
```

EXEMPLO COM RETIRADA DE COLUNAS

```
{
    "pergunta": "retorne-me a página inicial da PVLDB",
    "relacionamentos": ["dominio_jornal.jid > jornal.jid", "publicacao.jid > ....
    ],
    "esquema_relevante": {
       "tabelas": [
          {"nome": "jornal", "colunas": ["pagina_inicial"]}
       ]
    },
    "sql": "SELECT pagina_inicial FROM jornal WHERE nome  =  'PVLDB';"
 }
```

EXEMPLO COM ADIÇÃO DE COLUNAS

```
{
    "pergunta": "retorne-me a página inicial da PVLDB",
    "relacionamentos": ["dominio_jornal.jid > jornal.jid","publicacao.jid > ...],
    "esquema_relevante": {
       "tabelas": [
          { "nome": "jornal","colunas": ["nome","pagina_inicial","jid"] }
       ]
    },
    "sql": "SELECT pagina_inicial FROM jornal WHERE nome  =  'PVLDB';"
 }
```

Fonte: Elaborado pelo autor.

3.4 AJUSTE FINO NOS MODELOS DE LINGUAGEM PRÉ-TREINADOS

Após a preparação dos dados comentado na seção anterior, foi realizado o ajuste fino nos modelos de linguagem pré-treinados *BERTimbau* base*, BERTimbau* grande e *BERTugues*, para a tarefa de ligação de esquemas, cujo intuito é identificar as tabelas e colunas mais importantes baseadas em uma pergunta em linguagem natural e no esquema do banco de dados. O processo de ajuste descrito a seguir foi aplicado individualmente a cada um dos PLMs.

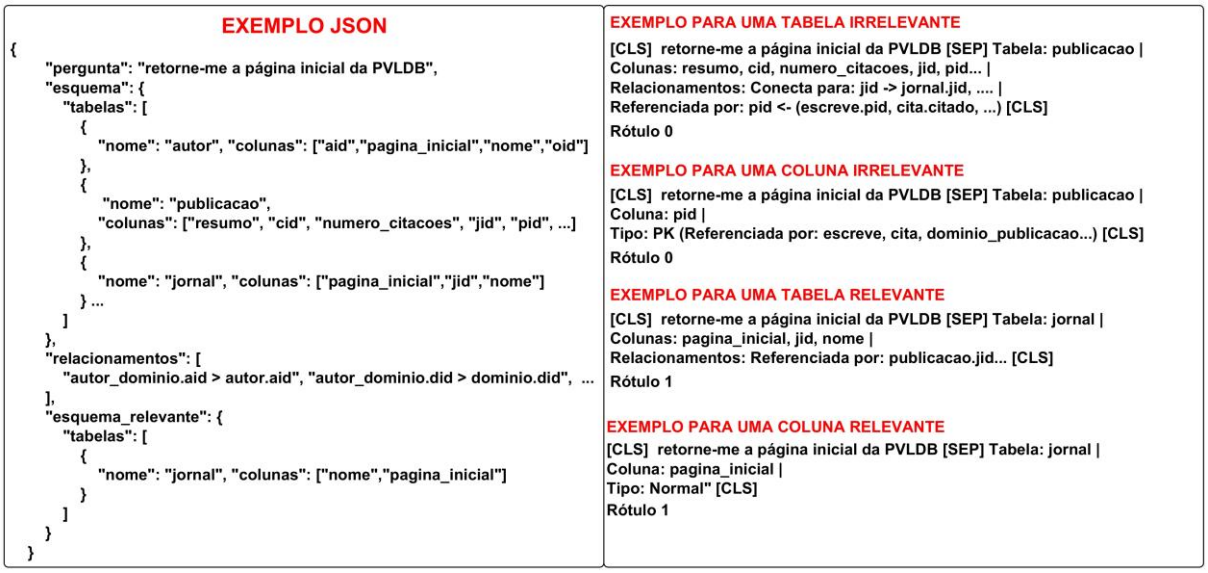
Antes do início do ajuste fino, foi necessário adaptar o conjunto de dados ao formato de entrada do modelo. Para isso, cada tabela e cada coluna, juntamente com os seus relacionamentos e a pergunta, foram concatenados por meio da seguinte estrutura: *“[CLS] [Pergunta] [SEP] [Contexto] [CLS]”.* Os elementos *“[CLS]”* e *“[SEP]”* são tokens especiais do modelo, em que o primeiro indica o começo e final da sequência textual e o segundo separa a pergunta do contexto.

A parte da *“[Pergunta]”* consiste na questão em linguagem natural fornecida pelo usuário e o *“[Contexto]”* é a descrição de informações importantes de uma tabela ou coluna. Para a tabela, são agrupados o nome da tabela, uma lista parcial de suas colunas e um resumo dos relacionamentos. Esse resumo detalha quais tabelas a tabela do contexto se conecta e por quais tabelas ela é referenciada. Em relação a

coluna, o contexto contém o nome da tabela à qual pertence, o nome da coluna em si e seu tipo: chave primária, chave estrangeira ou coluna comum. Se for uma chave primária, são listadas as tabelas as quais se conectam a coluna analisada. E no caso da chave estrangeira, é especificado qual é a chave primária que a coluna analisada referencia.

Por último, cada tabela ou coluna é rotulada com 1 se for parte do esquema relevante ou com 0 caso contrário. Na Figura 18 é demonstrado o processo de adaptação para um exemplo do conjunto do treinamento/teste. No lado esquerdo da figura tem-se o exemplo JSON completo e no canto direito o formato para uma tabela e coluna irrelevante e uma tabela e coluna relevante.

Figura 18 – Exemplo do processo de adaptação para a entrada dos PLMs



Fonte: Elaborado pelo autor.

Com o processo de adaptação finalizado, o conjunto de dados foi particionado nos conjuntos de treino e validação. Em seguida, o modelo, juntamente com o seu *tokenizador*, foram carregados. *O tokenizador* é responsável por transformar os textos que agrupam a pergunta e o contexto em sequências de *tokens*, os quais posteriormente são convertidos em um vetor numérico. Para organizar os textos concatenados e os rótulos associados em relação ao treinamento, foram utilizadas as classes *DataSet* e *DataLoader*, ambas pertencentes a biblioteca do *PyTorch*.

A classe *Dataset* recebe como entrada o conjunto de dados de treino/validação, o rótulo, o *tokenizador* e a quantidade máxima de tokens permitida por entrada. Para

cada objeto do conjunto de dados, tal classe gera como saída três itens: o vetor numérico correspondente a sequência de entrada, que representa os textos da pergunta combinada ao contexto após a passagem pelo *tokenizador*; a máscara de atenção, a qual indica os *tokens* importantes que devem ser considerados pelo modelo; e o rótulo da tabela ou coluna, denominada *label*. As saídas obtidas são enviadas ao *Dataloader*, que agrupam esses dados em lotes para serem processados de forma paralela durante o treinamento.

Posteriormente, o otimizador foi configurado com uma taxa de aprendizado inicial, os quais atualizam os pesos do modelo conforme o ajuste fino é realizado. Com o intuito de controlar a variação dessa taxa ao longo das épocas, foi usada a função *get_linear_schedule_with_warmup*, que consiste em aumentar a taxa de aprendizado de maneira gradativa nos primeiros passos do treinamento, processo denominado de aquecimento, e depois reduz linearmente até o término do processo.

Com todos os componentes configurados, o processo de ajuste fino foi executado. Na Figura 19 é apresentado uma parte do código fonte que realiza esse procedimento. A seção superior da figura contém a classe *classeDataset*, que como mencionado, é responsável por organizar os dados de entrada para o modelo. Logo abaixo, tem-se a função *funcaoTreinarModelo*, a qual executa o ajuste fino em si. Tal função recebe os parâmetros de treinamento, como a taxa de aprendizado, tamanho do lote, número de épocas, tamanho máximo de tokens para cada entrada e o conjunto de dados completo. Posteriormente, é chamada a função *funcaoAdaptarDadosFormatoModelos*, que concatena a pergunta, o contexto do banco de dados e os tokens especiais, além de atribuir os rótulos. Após essa etapa, nesta ordem, o conjunto de dados é dividido em treino e validação, o modelo e o *tokenizador* são carregados, os conjuntos de treino e validação são organizados pelas classes *funcaoDataSet* e *DataLoader*, e o otimizador da taxa de aprendizado juntamente com o controlador da variação dessa taxa (*scheduler*) são configurados. Por último, o uso da GPU é habilitado para o ajuste e o treinamento é iniciado. O objetivo geral desse processo é permitir que os modelos de linguagem pré-treinados aprendam a associar, por meio do contexto, perguntas em linguagem natural aos elementos estruturais do banco de dados. Dessa forma, dada uma tabela ou coluna nunca analisada, determina-se se tal item é ou não relevante a questão.

Figura 19 – Parte do código fonte do ajuste fino dos modelos pré-treinados

```
#Classe Dataset responsável por organizar o conjunto de dados para o treinamento
class classeDataset(Dataset):
    def __init__(self, conjunto_dados, rotulos, tokenizador, tamanho_maximo):
        self.conjunto_dados = conjunto_dados
        self.rotulos = rotulos
        self.tokenizador = tokenizador
        self.tamanho_maximo = tamanho_maximo

    def __len__(self):
        return len(self.conjunto_dados)

    def __getitem__(self, idx):
        texto_conjunto = self.conjunto_dados[idx]
        rotulo = self.rotulos[idx]

        encoding = self.tokenizador.encode_plus(
            texto_conjunto,
            add_special_tokens=True,
            max_length=self.tamanho_maximo,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt'
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(rotulo, dtype=torch.long)
        }


#Função de treinamento do modelo
def funcaoTreinarModelo(nome_caminho_conjunto_dados_json, tamanho_maximo=128, tamanho_lote=16,
numero_epocas=4, taxa_aprendizado=1e-5):

    #Função que realiza a adaptação do conjunto de dados ao formato de entrada dos modelos
    conjunto_dados_adaptado, rotulos_binarios_conjunto_dados_adaptado =
funcaoAdaptarDadosFormatoModelos(nome_caminho_conjunto_dados_json)

    # Divisão do conjunto de dados adaptado em treino e validação (80% treino // 20% validação)
    conjunto_treino, conjunto_validacao, rotulos_treino, rotulos_validacao = train_test_split(
        conjunto_dados_adaptado, rotulos_binarios_conjunto_dados_adaptado,
        test_size=0.2, random_state=42, stratify=rotulos_binarios_conjunto_dados_adaptado
    )

    #Carregamento do tokenizador e do modelo
    tokenizador = AutoTokenizer.from_pretrained('nome_modelo_pre_treinado')
    modelo = BertForSequenceClassification.from_pretrained('nome_modelo_pre_treinado', num_labels=2) #
Classificação binária para o modelo

    #Classe Dataset que organiza os conjuntos de treinamento e validação para o modelo
    dataset_conjunto_treino = classeDataset(conjunto_treino, rotulos_treino, tokenizador,
tamanho_maximo)
    dataset_conjunto_validacao = classeDataset(conjunto_validacao, rotulos_validacao, tokenizador,
tamanho_maximo)

    #Agrupamento dos conjuntos de treinamento e validação em lotes pela classe DataLoader
    dataloader_conjunto_treino = DataLoader(dataset_conjunto_treino, batch_size=tamanho_lote,
shuffle=True)
    dataloader_conjunto_validacao = DataLoader(dataset_conjunto_validacao, batch_size=tamanho_lote,
shuffle=False)

    #Configuração do otimizador com a taxa de aprendizado
    otimizador_adam = AdamW(modelo.parameters(), lr=taxa_aprendizado)

    #Configuração da função que controla a taxa de aprendizado, com aumento gradativo inicial (warmup)
e depois decaimento
    scheduler = get_linear_schedule_with_warmup(
        otimizador_adam,
        num_warmup_steps=int(0.1 * len(dataloader_conjunto_treino) * numero_epocas),
        num_training_steps=len(dataloader_conjunto_treino) * numero_epocas
    )

    #Habilitar o uso da GPU
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    modelo.to(device)
```

Fonte: Elaborado pelo autor.

3.5 AJUSTE FINO NO GRANDE MODELO DE LINGUAGEM LLAMA

O processo de ajuste fino no *LLama* consistiu na execução de um treinamento adicional direcionado a tarefa de tradução de texto para SQL, em que fornecida a pergunta em linguagem natural combinada com as tabelas, colunas e relacionamentos do banco de dados, o modelo gera à consulta SQL referente a pergunta formulada.

Para tal ajuste, inicialmente o *Llama* e o *tokenizador* associado foram carregados. Durante esse processo, foi aplicado a técnica de quantização do *QLoRA*, que possibilita a redução do uso de memória e, como consequência, viabiliza o treinamento para computadores com recursos limitados. Em seguida, assim como ocorreu para os PLMs, o conjunto de dados precisou ser adaptado ao formato de entrada no *LLama*. Cada item pertencente ao conjunto, composto pela pergunta, o esquema relevante, os relacionamentos e o código SQL, seguiu a estrutura apresentado na Figura 20. Tal estrutura contém 3 blocos: "*system*", "*user*" e "*assistant*", os quais definem, respectivamente, as instruções de como o modelo deve se comportar, a pergunta do usuário com o contexto do banco de dados e a resposta SQL correta que o modelo deve gerar. Os termos *"<|start_header_id|>"*, *"<|end_header_id|>"* e "<|*eot_id*|>" são *tags* especiais do *Llama*, em que os dois primeiros delimitam o cabeçalho de cada bloco e o último marca o final do conteúdo de cada bloco.

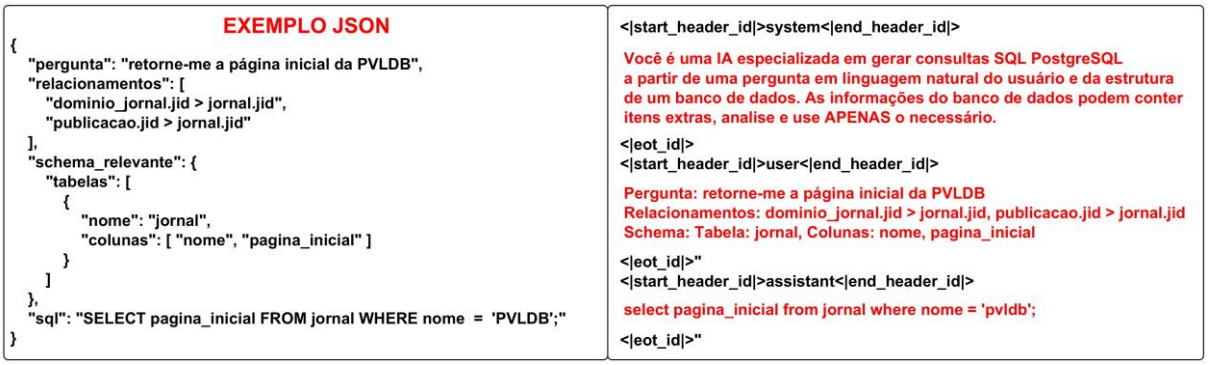
Figura 20 – Estrutura de entrada para o *Llama*

<|start_header_id|>system<|end_header_id|>
[INSTRUÇÕES PARA O MODELO]
<|eot_id|>
<|start_header_id|>user<|end_header_id|>
[PERGUNTA DO USUÁRIO + CONTEXTO]
<|eot_id|>"
<|start_header_id|>assistant<|end_header_id|>
[SQL ESPERADO]
<|eot_id|>"

Fonte: Elaborado pelo autor.

Na Figura 21 é apresentado, no lado esquerdo, um exemplo de objeto JSON do conjunto de treinamento e sua respectiva entrada para o *Llama* do lado direito.

Figura 21 – Exemplo de objeto JSON e respectiva entrada para o *Llama*



Fonte: Elaborado pelo autor.

Após a montagem da estrutura, o *tokenizador* realiza o papel de transformar o texto elaborado em um vetor numérico que representam a estrutura em si. Ao mesmo tempo, é gerada uma máscara de atenção, que orienta o modelo a focar apenas na parte correspondente à resposta esperada, ou seja, na consulta SQL.

Com a preparação dos dados concluída, foi configurada a técnica *LoRA*. Como comentado, tal paradigma consiste na adição de duas subcamadas de funções lineares, também chamada de adaptadores, nos módulos de atenção do *Transformer*. Durante o ajuste fino, apenas os pesos desses adaptadores são atualizados, enquanto os pesos do modelo original permanecem intactos. Isso possibilita uma redução no número de parâmetros treináveis, e consequentemente, o uso de memória. Para a implementação da *LoRA* no código, foi necessário o uso da classe *LoraConfig*, responsável por receber os parâmetros de configuração juntamente com os módulos do *Transformer* nos quais os adaptadores seriam inseridos. Essa configuração foi passada para a função *get_peft_model*, que tem como objetivo integrar os adaptadores ao *Llama*. Dessa forma, os parâmetros base do modelo ficam congelados para o treinamento.

Posteriormente, o conjunto de dados é separado em treino e validação. Após, foram definidos os hiperparâmetros do treinamento por meio da classe *TrainingArguments*. Com isso, o modelo, os dados de treino/validação, o *tokenizador* e as configurações foram incluídos à classe *Trainer*, que executa o processo de ajuste

fino. Esse procedimento encerra a preparação do modelo em relação a tarefa texto para SQL.

Na Figura 22 é exibida uma parte do código referente ao ajuste fino do o *LLama*. Inicialmente, a técnica *QLoRA* é configurada e tanto o *tokenizador* quanto o modelo são carregados com o uso dessas configurações. Em seguida, a função *prepare_model_for_kbit_training* é chamada com o intuito de preparar o modelo para o ajuste fino. Durante esse processo, os preenchimentos, do inglês *paddings*, também são definidos, os quais são responsáveis por garantir que todos os dados de entrada possuam o mesmo tamanho por meio da adição de tokens especiais.

Logo após, como comentado anteriormente, o conjunto de dados é carregado e cada exemplo é adaptado ao formato de entrada do *Llama* por meio da função *funcaoAdaptarDadosFormatoModelo*. Posteriormente, a técnica *LoRA* é configurada e aplicada ao modelo, o conjunto de dados é particionado em treino e validação e os hiperparâmetros do ajuste fino são definidos.

Depois, é utilizada a função *DataCollatorForLanguageModeling* com a finalidade de agrupar os dados dos conjuntos em lotes, adicionar os *paddings* necessários e gerar as máscaras de atenção. Por último, a função *Trainer* organiza os itens do ajuste fino, com a junção do modelo, parâmetros, conjuntos de treino/validação, *tokenizador* e *DataCollator*, e com isso, o treinamento é iniciado. Após a conclusão do processo, o modelo e o *tokenizador* são salvos em um diretório previamente estabelecido.

Figura 22 – Parte do código fonte para o ajuste fino do *Llama*

```
    #Configuração dos Bits and Bytes (BnB) com quantização de 4 bits (QLoRA)
    configuracoes_bits_and_bytes = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=torch.float16,
        bnb_4bit_use_double_quant=True,
    )

    #Carregamento do tokenizador
    tokenizador = AutoTokenizer.from_pretrained("nome_modelo")

    #Configuração do padding (preenchimento) dos textos de entrada.
    #O objetivo é garantir que todas as entradas tenham o mesmo tamanho.
    if tokenizador.pad_token is None:
        tokenizador.pad_token = tokenizador.eos_token

    #Os preenchimentos serão adicionados da esquerda para a direita
    tokenizador.padding_side = "left"

    #Carregamento do modelo quantizado usando as configurações do BnB
    modelo = AutoModelForCausalLM.from_pretrained(
        "nome_modelo",
        quantization_config=configuracoes_bits_and_bytes, #Configurações do BnB são passadas para o
parâmetro quantization_config
        torch_dtype=torch.float16,
        device_map="auto"
    )

    #Função que prepara o modelo para o treinamento com a quantização QLoRA
    modelo = prepare_model_for_kbit_training(modelo)

    #Carregamento do conjunto de dados JSON para o treinamento
    with open("nome_caminho_conjunto_dados_treinamento", "r", encoding="utf-8") as f:
        conjunto_dados = json.load(f)

    #Função que realiza a adaptação de cada exemplo do conjunto de dados ao formato de entrada do
modelo
    conjunto_dados_formatado = [funcaoAdaptarDadosFormatoModelo(objeto_conjunto, tokenizador, 384) for
objeto_conjunto in conjunto_dados]

    #Configuração do LoRA
    configuracoes_lora = LoraConfig(
        r=16,
        lora_alpha=32,
        lora_dropout=0.1,
        #Estes são as camadas lineares do modelo que receberão as subcamadas/adaptadores do LoRA
        target_modules=["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj"],
        bias="none",
        task_type="CAUSAL_LM"
    )

    #Aplicação das configurações do LoRA ao modelo
    modelo = get_peft_model(modelo, configuracoes_lora)

    #Print dos parâmetros do modelo que estão sendo ajustados
    modelo.print_trainable_parameters()

    #Função Dataset que organiza o conjunto de dados e divisão do conjunto em treino (80%) e validação
(20%)
    dataset = Dataset.from_list(conjunto_dados_formatado)
    dataset_dividido = dataset.train_test_split(test_size=0.2, seed=42)
    conjunto_treino = dataset_dividido['train']
    conjunto_validacao = dataset_dividido['test']

    #Configurações dos paramêtros de treinamento
    parametros_treinamento = TrainingArguments(
        output_dir="nome_diretorio_modelo_apos_treinamento",
        per_device_train_batch_size=1,
        gradient_accumulation_steps=32,
        learning_rate=1e-5,
        optim="paged_adamw_8bit",
        fp16=True,
        num_train_epochs=5,
        lr_scheduler_type="cosine",
        logging_steps=50,
        save_steps=200,
        eval_steps=200,
        evaluation_strategy="steps",
        save_total_limit=2,
        load_best_model_at_end=True,
        report_to="none",
        gradient_checkpointing=True,
        gradient_checkpointing_kwargs={'use_reentrant': False}
    )

    #Função que realiza o agrupamento dos dados em lotes, adiciona os paddings necessários
    # e cria as máscaras de atenção
    data_collator = DataCollatorForLanguageModeling(
        tokenizer=tokenizador,
        mlm=False,
    )

    #Função que organiza o ciclo de treinamento com a junção do itens necessários para o processo
    #(modelo, parâmetros, conjunto de treino/validação, tokenizador...)
    trainer = Trainer(
        model=modelo,
        args=parametros_treinamento,
        train_dataset=conjunto_treino,
        eval_dataset=conjunto_validacao,
        tokenizer=tokenizador,
        data_collator=data_collator
    )

    #Inicio do treinamento
    trainer.train(resume_from_checkpoint=True)

    #Modelo e tokenizador são salvos em um diretório pré-estabelecido após a conclusão do treinamento
    modelo.save_pretrained("nome_diretorio_modelo_apos_treinamento")
    tokenizador.save_pretrained("nome_diretorio_modelo_apos_treinamento")
```

Fonte: Elaborado pelo autor.

3.6 ALGORITMO HEURÍSTICO PARA VISUALIZAÇÕES

Para a construção das visualizações da arquitetura, foi desenvolvido um algoritmo heurístico baseado na árvore de decisão proposta por Healy e Holtz (2017). Tal trabalho classifica inicialmente os dados em 6 categorias principais: numérico, categórico, numérico e categórico, mapas, fluxo e série temporal. Em seguida, dada como entrada o conjunto de dados, o algoritmo analisa cada coluna do conjunto em relação à sua categoria. Com base na categoria obtida, a árvore de decisão se ramifica progressivamente, em que informações mais específicas são verificadas, como o número de colunas, a existência de ordenação, agrupamento entre variáveis, até se chegar nas visualizações mais adequadas.

Diante desse cenário, nas Figuras 23, 24, 25, 26, 27 e 28 são apresentados o pseudocódigo do algoritmo heurístico desenvolvido. Na Figura 23 são ilustradas as funções que efetuam a análise inicial do conjunto de dados. As funções "*funcaoCalcularNumeroColunas*" e "*funcaoCalcularNumeroLinhas*" recebem como entrada o conjunto de dados e retornam o número total de colunas e linhas pertencentes ao conjunto, respectivamente. E a função "*funcaoAnalisarCategoria*" é responsável por verificar o tipo de dado presente em cada coluna do conjunto e direcionar para uma dentre cinco categorias disponíveis: "*Apenas numérico*", se todas as colunas apresentarem valores numéricos; "*Apenas categórico*", caso todas as colunas contenham textos; "*Numérico e categórico*", quando houver pelo menos uma coluna com números e outra com textos; "*Mapa*", se o conjunto possuir dados geográficos, como coordenadas; e "*Série temporal*", para situações em que se tem pelo menos uma coluna com informações de datas ou tempo.

Figura 23 – Pseudocódigo para a análise inicial do conjunto

```
FUNÇÃO SugerirVisualizacoes(conjuntoDados){

    INTEIRO quantidadeColunas = funcaoCalcularNumeroColunas(conjuntoDados)
    INTEIRO quantidadeLinhas = funcaoCalcularNumeroLinhas(conjuntoDados)
    STRING tipoCategoria = funcaoAnalisarCategoria(conjuntoDados)
```

Fonte: Elaborado pelo autor.

Algumas observações a serem comentadas é que a categoria fluxo, presente no trabalho de Healy e Holtz (2017), foi descartada por não ser possível determinar, de forma lógica e automatizada, estruturas de dados que representem fluxo diretamente. Além disso, em relação aos mapas, o algoritmo desenvolvido realiza a verificação da presença de coordenadas geográficas, como a latitude e longitude. No entanto, tais valores também se encaixam na categoria numérica, o que pode gerar conflitos na classificação. Por conta disso, foi adotada uma restrição adicional dos mapas baseada na análise dos nomes das colunas. Com isso, se o nome da coluna conter termos relacionados a "*latitude*" ou "*longitude*", a categoria mapa é selecionada, mesmo que os dados, de maneira isolada, sejam numéricos.

Na Figura 24 é apresentado o fluxo de decisão para o caso em que foi selecionada a categoria "*Apenas numérico*". Inicialmente, é verificado o número total de colunas presentes do conjunto. Quando há apenas uma coluna, realiza-se uma análise adicional do número de linhas. Se existir somente uma linha, a visualização recomendada é o gráfico de Valor Único, que é destacado um único valor numérico ou texto. Caso contrário, com duas ou mais linhas, as visualizações sugeridas são os Gráficos de Histograma, Densidade e Violino. Para as condições em que se têm duas colunas, verifica se alguma delas está ordenada. Se confirmada, tem-se como saída os Gráficos de Linhas e Área. De maneira contrária, é retornado o Gráfico de Dispersão. Em relação a três colunas, as visualizações mais apropriadas são o Gráfico de Bolhas e Tabela. Por último, se os dados não se enquadrarem em nenhuma das condições citadas, a visualização padrão é uma Tabela.

Figura 24 – Pseudocódigo para a condição “Apenas numérico”

```
SE (tipoCategoria == "Apenas numérico")
{
       SE (quantidadeColunas == 1)
       {
            SE (quantidadeLinhas == 1)
               RETORNAR "Valor único"
            SENÃO
               RETORNAR "Histograma", "Gráfico de Densidade", "Gráfico de Violino"
        }
        SENÃO SE (quantidadeColunas == 2)
        {
            SE (coluna1EstaOrdenada OU coluna2EstaOrdenada)
                RETORNAR "Gráfico de Linhas", "Gráfico de Área"
            SENÃO
                RETORNAR "Gráfico de Dispersão"
         }
         SENÃO SE (quantidadeColunas == 3)
              RETORNAR "Gráfico de Bolhas", "Tabela"
          SENÃO
              RETORNAR "Tabela"
}
```

Fonte: Elaborado pelo autor.

As condições para a categoria “*Apenas categórico*” são apresentadas na Figura 25. Para o caso em que se tem apenas uma coluna e uma linha, é retornado o Valor único. Se o conjunto tiver duas colunas, as visualizações mais adequadas são Tabela e o Gráfico de Redes. E, nas situações em que o conjunto não satisfaça nenhuma das condições, por padrão a Tabela se torna a representação mais apropriada.

Figura 25 – Pseudocódigo para a condição “Apenas categórico”

```
SE (tipoCategoria == "Apenas categórico")
{
      SE (quantidadeColunas == 1 E quantidadeLinhas == 1)
            RETORNAR "Valor único"
       SENÃO SE (quantidadeColunas == 2)
            RETORNAR "Gráfico de Redes", "Tabela"
        SENÃO
            RETORNAR "Tabela"
}
```

Fonte: Elaborado pelo autor.

Para a categoria de mapas, o algoritmo sugere 3 tipos de visualizações: Mapa de Conexão, que exige coordenadas de latitude e longitude tanto do ponto de origem quanto do ponto destino; Mapa de Bolhas, em que há a necessidade de existir uma coluna numérica em conjunto com as coordenadas; e Mapa de Pontos, o qual contém apenas as coordenadas geográficas. Com isso, na Figura 26 é representado o pseudocódigo para tal categoria. Inicialmente, o algoritmo valida se a categoria identificada é do tipo "*Mapa*" e se o nome das colunas inclui os termos "*latitude*" e "*longitude*". Caso confirmado, duas funções auxiliares, denominadas "*funcaoCalcularNumeroColunasDeMapa*" e "*funcaoCalcularNumeroColunasNumericas*", são utilizadas. A primeira calcula quantas colunas do conjunto são de coordenadas, ou seja, é verificado se os nomes das colunas contêm termos de "latitude" e "longitude" e se seus valores estão no intervalo geográfico, entre -180 e 180. A segunda função retorna a quantidade de colunas numéricas, com a exclusão daquelas que foram classificadas como coordenadas previamente. Após esse processo, as condições são iniciadas. Primeiro, é verificado se o conjunto possui exatamente quatro colunas de mapa, que corresponde a latitude e longitude de origem e destino. Se confirmado, é selecionado o Mapa de Conexão. Se tal condição não for satisfeita, o algoritmo checa se há pelo menos uma coluna numérica disponível. Caso positivo, é retornada à visualização Mapas de Bolhas. Por fim, se nenhuma dessas condições se encaixar, a saída padrão é o Mapa de Pontos.

Figura 26 – Pseudocódigo para a condição "Mapa"

```
SE (nomeColunas CONTER "latitude" E "longitude" E tipoCategoria CONTER ("Mapa"))
{
    INTEIRO quantidadeColunasDeMapa = funcaoCalcularNumeroColunasDeMapa(ConjuntoDados)
    INTEIRO quantidadeColunasNumericas = funcaoCalcularNumeroColunasNumericas(ConjuntoDados)
    SE (quantidadeColunasDeMapa == 4)
        RETORNAR "Mapa de Conexão"
    SENÃO SE (quantidadeColunasNumericas > 0)
        RETORNAR "Mapa de Bolhas"
     SENÃO
        RETORNAR "Mapa de Pontos"
}
```

Fonte: Elaborado pelo autor.

Na Figura 27 é mostrado o pseudocódigo referente a categoria "*Série temporal*". O processo inicia com a verificação da quantidade de colunas do conjunto. Se o valor for igual a 1, retorna-se como visualização uma Tabela. Ao contrário, com duas ou mais colunas, efetua-se o cálculo do número de colunas numéricas e categóricas por meio das funções "*funcaoCalcularNumeroColunasNumericas*" e "*funcaoCalcularNumeroColunasCategoricas*", respectivamente. Em seguida, é verificado se não foi encontrado colunas categóricas. Caso confirmado, é feito mais uma checagem para a quantidade de colunas numéricas. Se o número for menor ou igual a 4, as visualizações recomendadas são os Gráficos de Linhas e de Área. Caso contrário, é retornada a Tabela. Quando o conjunto apresenta exatamente uma coluna categórica, é analisado se foi encontrado 3 ou menos colunas numéricas. Se sim, são sugeridos os Gráficos de Linhas, de Área e Tabela. Senão, é adotada a Tabela. Por último, se nenhuma das condições anteriores estarem de acordo com o conjunto, tem-se como saída a Tabela.

Figura 27 – Pseudocódigo para a condição "Série temporal"

```
SE (tipoCategoria == "Série temporal")
{
    SE (quantidadeColunas == 1)
    {
        RETORNAR "Tabela"
    }
    SENÃO
    {
        INTEIRO quantidadeColunasNumericas = funcaoCalcularNumeroColunasNumericas(ConjuntoDados)
        INTEIRO quantidadeColunasCategoricas = funcaoCalcularNumeroColunasCategoricas(ConjuntoDados)
        SE (quantidadeColunasCategoricas == 0)
        {
            SE (quantidadeColunasNumericas > 0 E quantidadeColunasNumericas <= 4)
                RETORNAR "Gráfico de Área", "Gráfico de Linhas"
            SENÃO
                RETORNAR "Tabela"
        }
        SENÃO SE (quantidadeColunasCategoricas == 1)
        {
            SE (quantidadeColunasNumericas > 0 E quantidadeColunasNumericas <= 3)
                RETORNAR "Gráfico de Área", "Gráfico de Linhas", "Tabela"
            SENÃO
                RETORNAR "Tabela"
        }
        SENÃO
            RETORNAR "Tabela"
    }
}
```

Fonte: Elaborado pelo autor.

Para a última opção, na Figura 28 é apresentada a lógica referente à categoria "*Categórico e Numérico*". No começo, efetua-se a verificação do número de colunas presentes no conjunto. Caso existam apenas duas colunas, conclui-se que há uma coluna categórica e uma numérica. Nesse cenário, aplica-se a função "*funcaoCalcularMedianaPorCategoria*", que calcula a mediana dos valores numéricos agrupados para cada valor categórico. O valor da mediana obtido serve como critério em direção para as visualizações, em que se for menor que 1,5, as visualizações recomendadas são Gráfico de Barras, Gráfico de Pizza e Tabela. Caso contrário, são selecionados os Gráficos Boxplot, de Barras e Violino. Para as situações em que o conjunto conter 3 colunas, é checado se existem exatamente duas colunas categóricas e uma numérica. Se tal condição for satisfeita, as visualizações mais adequadas são o Diagrama de Sankey, Gráfico de Redes e Tabela. No final, caso nenhuma condição seja satisfeita, retorna-se à Tabela como visualização padrão.

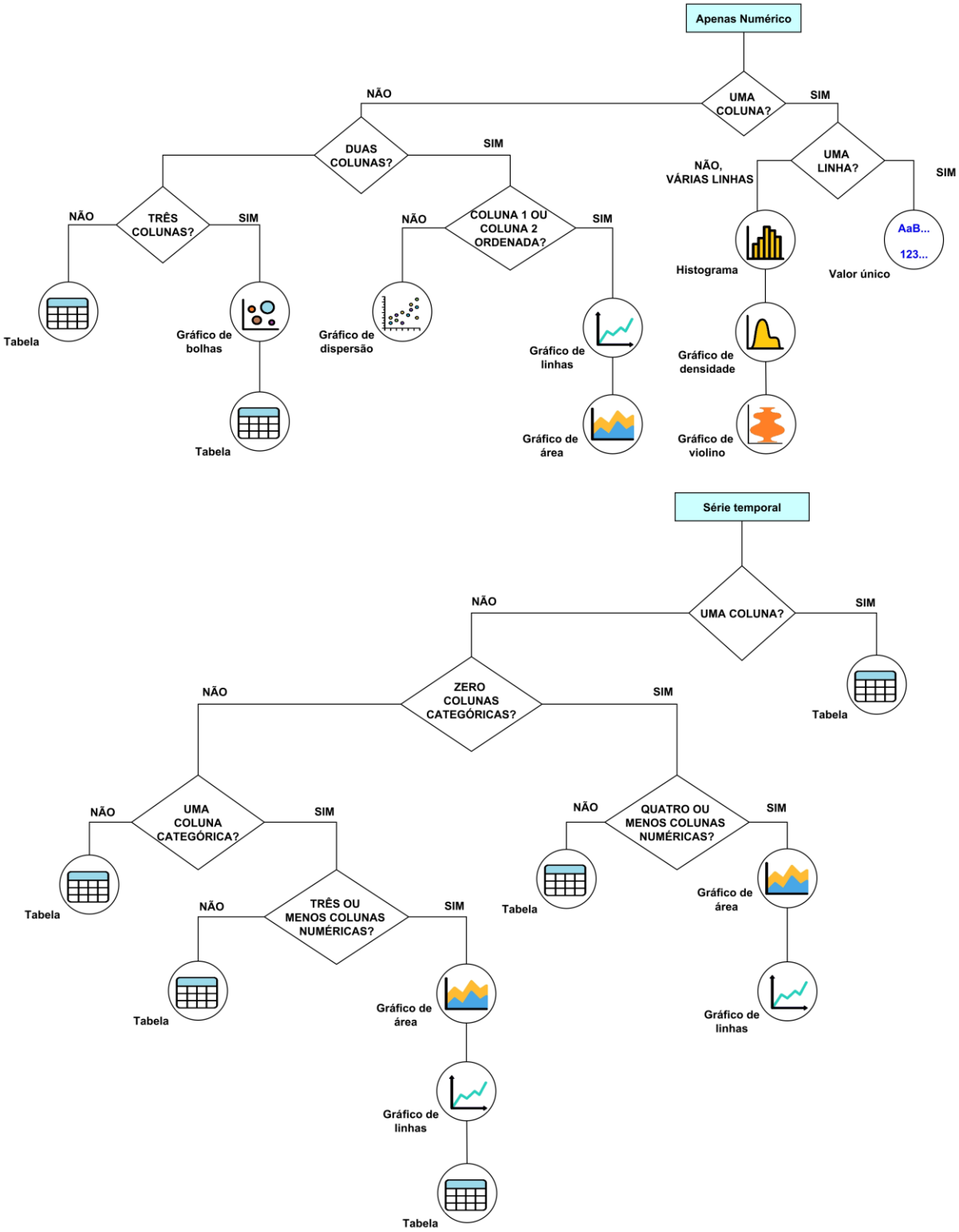
Figura 28 – Pseudocódigo para a condição "Numérico e categórico"

```
SE (tipoCategoria == "Categórico e numérico")
    {
        SE (quantidadeColunas == 2)
        {
            REAL mediana = funcaoCalcularMedianaPorCategoria(ConjuntoDados)
            SE (mediana <= 1,5)
                RETORNAR "Gráfico de Barras", "Gráfico de Pizza", "Tabela"
            SENÃO
                RETORNAR "Boxplot", "Gráfico de Barras", "Gráfico de Violino"
         }
         SENÃO SE (quantidadeColunas == 3)
         {
             INTEIRO quantidadeColunasNumericas = funcaoCalcularNumeroColunasNumericas(ConjuntoDados)
             INTEIRO quantidadeColunasCategoricas = funcaoCalcularNumeroColunasCategoricas(ConjuntoDados)
             SE (quantidadeColunasCategoricas == 2 E quantidadeColunasNumericas == 1)
                 RETORNAR "Diagrama de Sankey", "Gráfico de Redes", "Tabela"
             SENÃO
                 RETORNAR "Tabela"
         }
         SENÃO
             RETORNAR "Tabela"
}
```

Fonte: Elaborado pelo autor.

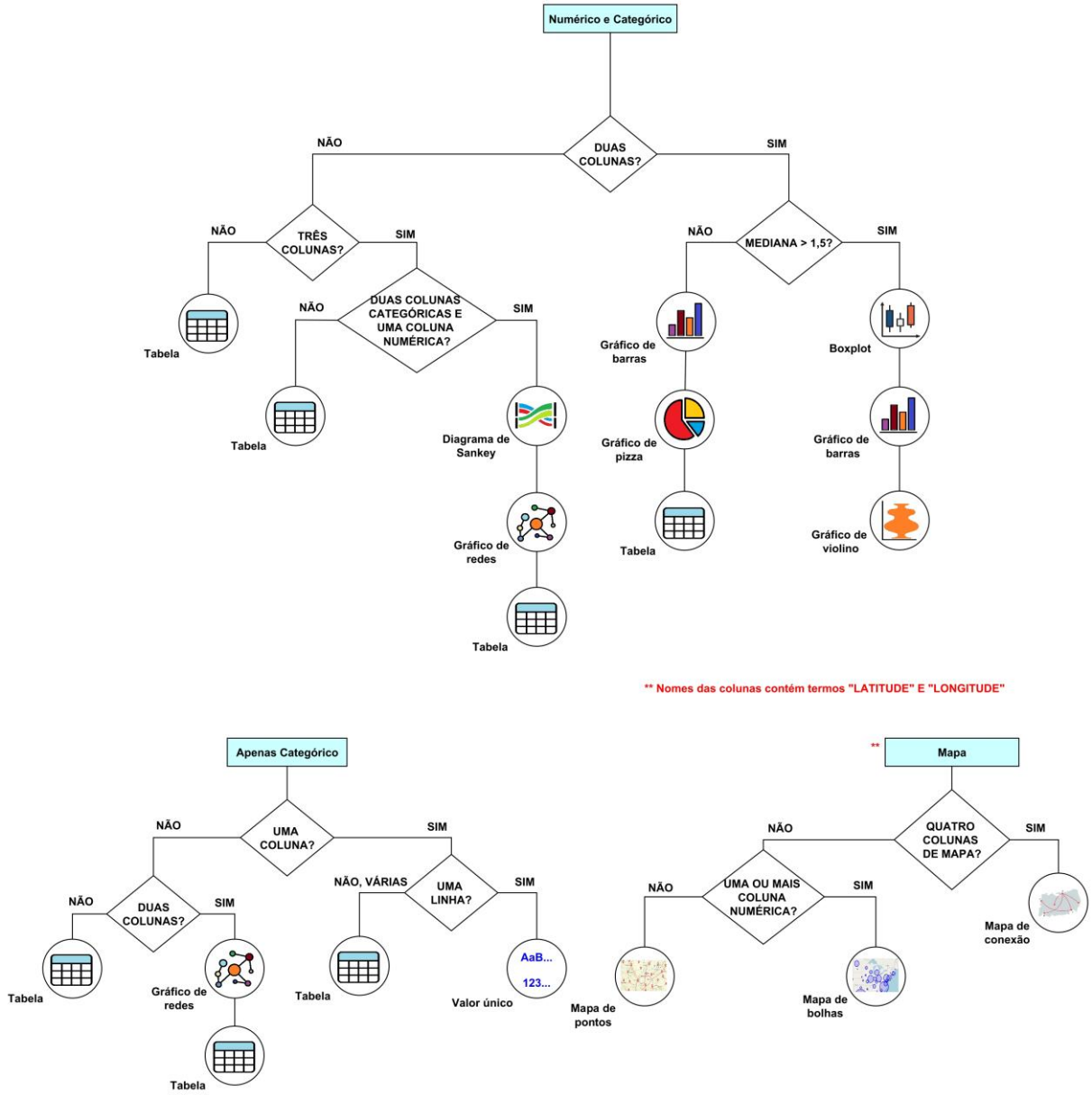
Com a explicação do pseudocódigo, nas Figura 29 e 30 são ilustradas a árvore de decisão completa do algoritmo heurístico desenvolvido. No total, a estrutura suporta 17 tipos de representações gráficas.

Figura 29 – Visualizações suportadas pelo algoritmo heurístico – parte 1



Fonte: Elaborado pelo autor.

Figura 30 – Visualizações suportadas pelo algoritmo heurístico – parte 2



Fonte: Elaborado pelo autor.

3.7 VALIDAÇÃO DA ARQUITETURA HÍBRIDA

A validação da arquitetura consistiu na elaboração de um conjunto de testes com 60 pares de pergunta-SQL relacionados a uma base de dados do mundo real, voltada a temática de dicionários multilíngues, a qual contém 25 tabelas e 236 colunas. Tais perguntas foram divididas uniformemente entre os níveis de dificuldade fácil, médio e difícil, e inseridas na ILN que contém a arquitetura.

O objetivo foi avaliar o comportamento do fluxo arquitetural completo, desde a identificação das tabelas e colunas relevantes pelo modelo de linguagem pré-treinado,

a construção da consulta SQL por meio do grande modelo de linguagem e, por fim, a recomendação de visualizações pelo algoritmo heurístico. Para os módulos baseados em modelos de linguagem, PLM e LLM, foi realizada uma avaliação manual, na qual os resultados produzidos pela arquitetura foram comparados com as perguntas e as consultas SQL correspondentes do conjunto de teste.

3.8 CONSIDERAÇÕES FINAIS

Neste capítulo, foi apresentada a arquitetura híbrida baseada em modelos de linguagem, bem como as suas etapas, que envolvem a extração das tabelas e colunas relevantes pelo PLM, o envio dessas informações para o LLM para a geração da consulta SQL e, no final, a definição das visualizações adequadas ao contexto por meio de um algoritmo baseado em árvore de decisão.

Adicionalmente, foram descritos os processos de ajuste fino aplicados tanto aos PLMs quanto ao LLM, com a preparação dos dados, configuração dos parâmetros e execução do treinamento. Por último, foi explicada a lógica do algoritmo heurístico responsável pela recomendação de visualizações e a metodologia usada na validação da arquitetura por meio de testes efetuados em um banco de dados real.

CAPÍTULO

4

4. AVALIAÇÃO EXPERIMENTAL

Neste capítulo são apresentadas as telas da interface de linguagem natural desenvolvida, assim como a metodologia e os resultados obtidos do processo de ajuste fino nos PLMs e no LLM. Além disso, é descrito os testes realizados na arquitetura com o uso de uma base de dados do mundo real.

4.1 AMBIENTE DE EXECUÇÃO

A arquitetura proposta, juntamente com a ILN e o processo de ajustes finos dos modelos PLMs e LLM, foi executada no ambiente descrito no quadro 2.

Quadro 2 – Especificações do ambiente de execução

Especificação	Valor
Processador	Ryzen 5 5600G
Memória RAM	32 GB DDR4 a 3200Mhz
Placa de vídeo	RTX 3060 12GB VRAM
Disco rígido	SSD 480GB
Sistema operacional	Windows 10

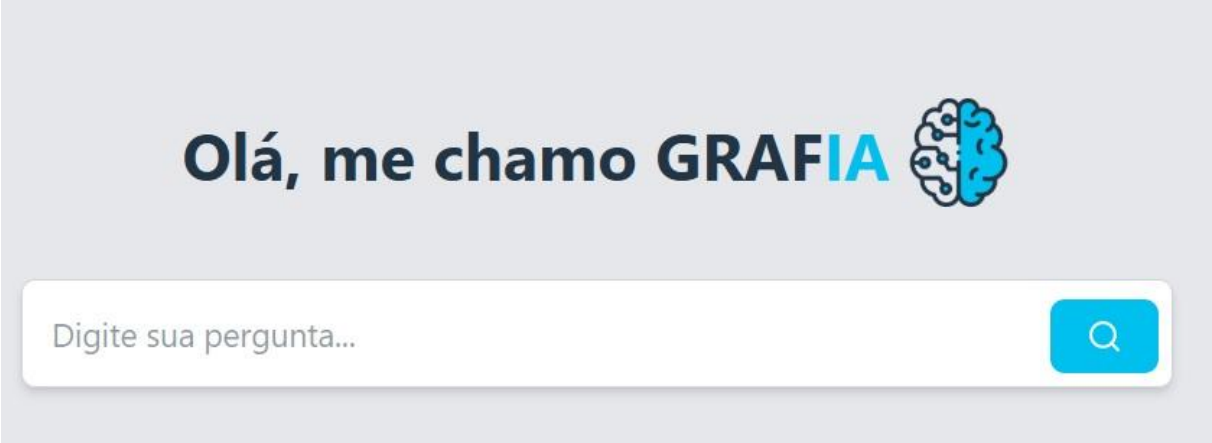
Fonte: Elaborado pelo autor.

4.2 TELAS DA INTERFACE DE LINGUAGEM NATURAL

A inicialização da interface envolve o carregamento dos modelos PLM e LLM, além da conexão com o banco de dados utilizado. Após esse processo, é exibida a

tela inicial, ilustrada na Figura 31. Nessa tela, são apresentados a logomarca da plataforma e o campo destinado à digitação da pergunta em linguagem natural.

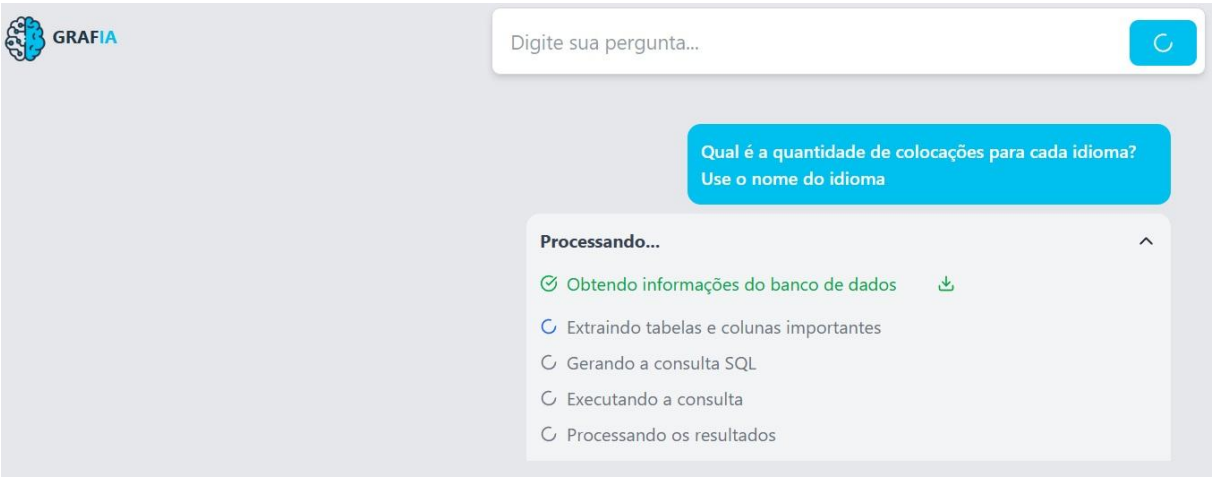
Figura 31 – Tela inicial da interface de linguagem natural



Fonte: Elaborado pelo autor.

Após a inserção da pergunta pelo usuário, a interface é atualizada e passa a exibir uma tela para troca de mensagens, semelhante a um bate-papo, ilustrado na Figura 32. A questão digitada é exposta na tela e, logo abaixo, são dispostos os passos realizados pelo sistema até a geração das visualizações mais adequadas ao contexto. Cada etapa, assim que é concluída, é destacada com a cor verde.

Figura 32 – Tela após o usuário digitar a pergunta

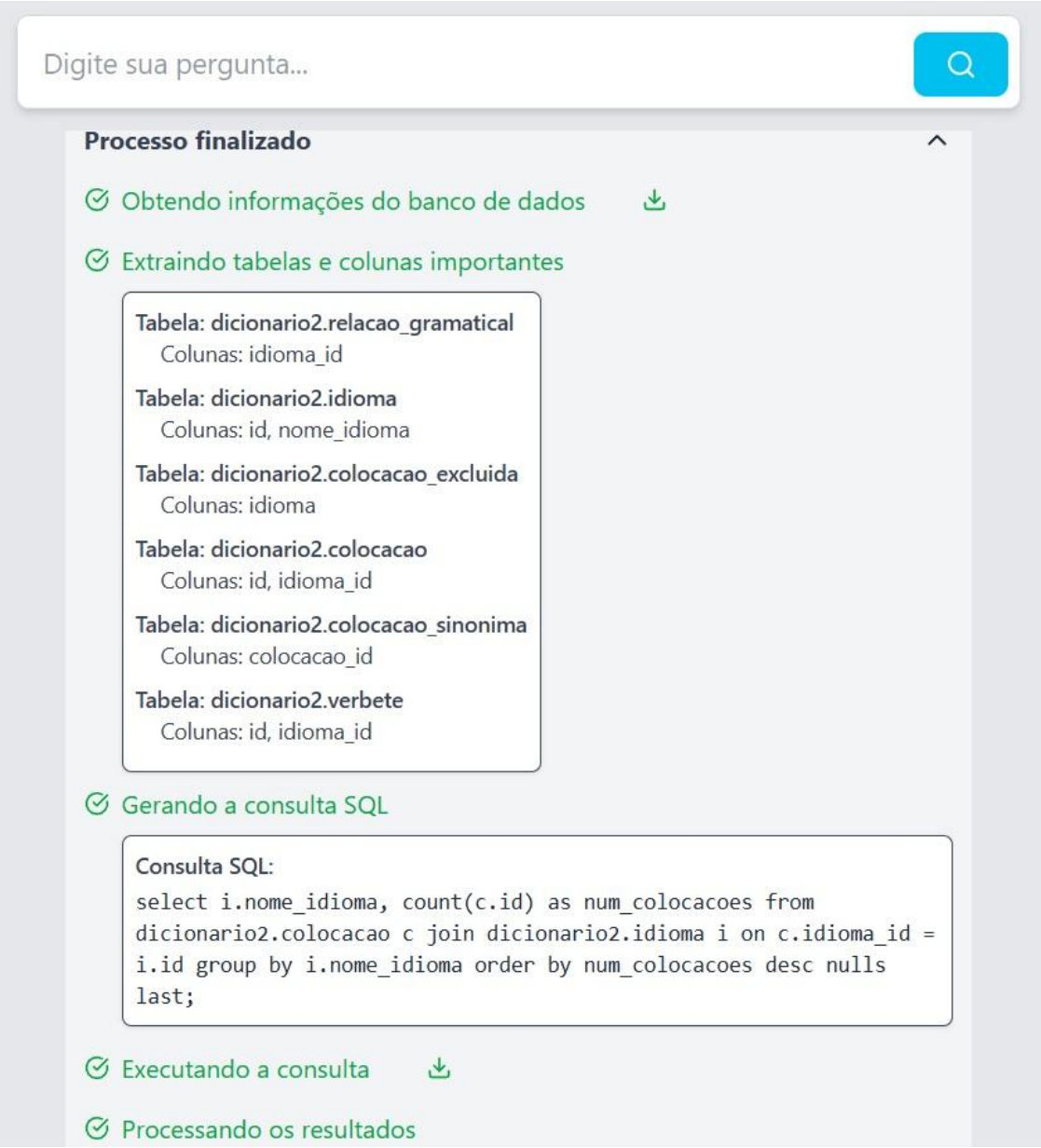


Fonte: Elaborado pelo autor.

Na Figura 33 é ilustrado o processo finalizado de geração das visualizações. Na primeira etapa, todas as tabelas, colunas e relacionamentos do banco de dados

conectado são extraídos e convertidos para o formato JSON. Tais informações, em conjunto com a pergunta inserida, são enviadas ao modelo de linguagem pré-treinado, o qual identifica as tabelas e colunas relevantes ao contexto da pergunta. O resultado dessa etapa é apresentado ao usuário e, ao mesmo tempo, enviado ao grande modelo de linguagem, que gera a consulta SQL correspondente. A consulta elaborada é exibida ao usuário e posteriormente executada no banco de dados. Os dados retornados a partir dessa execução são analisados pelo algoritmo heurístico, que determina e exibe as visualizações mais adequadas relacionada a pergunta. Um detalhe a ser destacado é que a plataforma permite o download tanto do arquivo JSON com o esquema do banco de dados quanto de um CSV que contém os dados resultantes da consulta SQL.

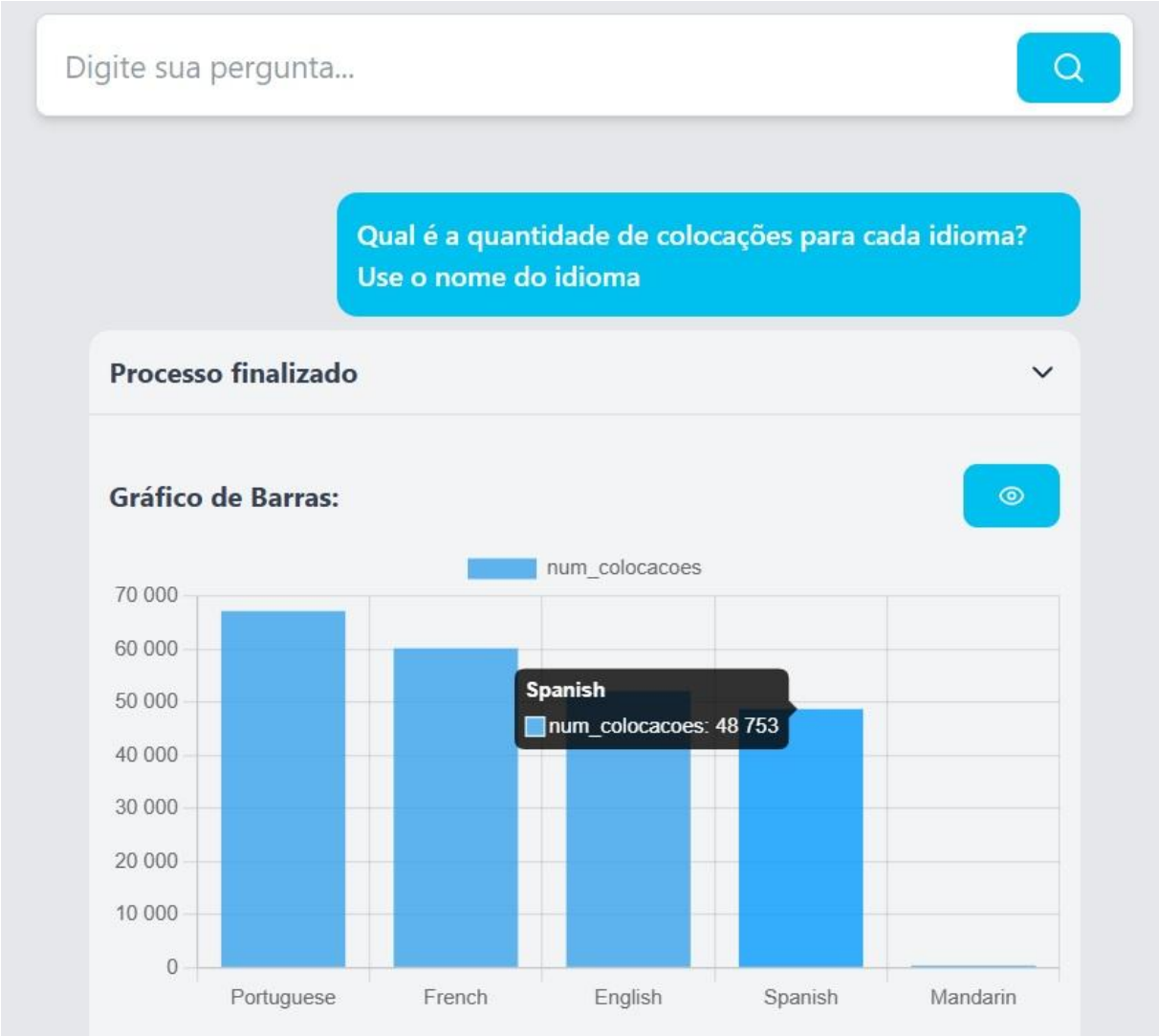
Figura 33 – Tela do processo concluído para a geração das visualizações



Fonte: Elaborado pelo autor.

Com a conclusão do processo, as etapas apresentadas na Figura 33 são minimizadas, e as visualizações geradas passam a ser exibidas na interface, as quais são mostradas nas Figuras 34, 35 e 36. Nesse exemplo, foram elaborados os gráficos de barra, pizza e tabela, pois o algoritmo heurístico identificou que os dados de retorno da consulta SQL continham duas colunas, uma categórica e uma numérica. Após a exibição das visualizações, o usuário pode inserir novas perguntas na mesma interface e as interações subsequentes serão exibidas abaixo das visualizações anteriores. **Mais exemplos podem ser vistos no Apêndice A.**

Figura 34 – Tela do gráfico de barras gerado



Fonte: Elaborado pelo autor.

Figura 35 – Tela do gráfico de pizza gerado

Fonte: Elaborado pelo autor.

Figura 36 – Tela da tabela gerada

NOME_IDIOMA	NUM_COLOCACOES
Portuguese	67228
French	60209
English	52152
Spanish	48753
Mandarin	427

Fonte: Elaborado pelo autor.

4.3 RESULTADOS DO AJUSTE FINO DOS MODELOS PRÉ-TREINADOS

Como comentado na seção 3.4, os modelos *BERTimbau* base*, BERTimbau* grande e *BERTugues* passaram por um processo de ajuste fino para a tarefa de ligação de esquemas, que consiste em identificar, a partir de uma pergunta feita em linguagem natural, as tabelas e colunas mais importantes de um banco de dados. O objetivo foi comparar os três modelos e avaliar qual deles apresentou o melhor desempenho. A versão base do *BERTimbau*, denominada "*bert-base-portuguese-cased*", foi pré-treinada para a língua portuguesa e contém aproximadamente 110 milhões de parâmetros, características presentes também no modelo *BERTugues* ("*BERTugues-base-portuguese-cased*"). Já a versão grande do BERTimbau ("*bert-large-portuguese-cased*") possui cerca de 330 milhões de parâmetros, além de ter sido pré-treinado com um volume maior de dados. Isso proporciona um compreendimento maior da linguagem e um melhor desempenho na realização de tarefas complexas.

Para o treinamento, foi usado conjunto de dados *Spider* (Yu et al., 2018), traduzido para o português e preparado para a entrada dos modelos, conforme detalhado nas seções 3.2 e 3.3, respectivamente. Dessa forma, o conjunto é composto por 9991 pares de perguntas-SQL de 139 diferentes bancos de dados, e foi dividido em 80% para treinamento (7993 exemplos) e 20% para validação (1998 exemplos).

Todos os modelos usaram os mesmos parâmetros de treinamento, mostrados no quadro 3.

Quadro 3 – Parâmetros utilizados pelos três modelos durante o treinamento

Parâmetro	Valor	Significado
Número de épocas (*num_epochs*)	4	Quantidade de vezes que o modelo percorre todo o conjunto de treinamento
Taxa de aprendizado (*learning_rate*)	1e-5	Realiza o controle da velocidade dos pesos durante o treinamento
Comprimento máximo (*max_length*)	128	Limite máximo de *tokens* por entrada após a tokenização
Tamanho do lote (*batch_size*)	16	Número de amostras processadas a cada iteração

Otimizador (*optimizer*)	AdamW	Algoritmo de otimização usado durante o treinamento para ajuste dos pesos
Número de *labels* do modelo	2	Indica a tarefa de classificação binária, classe positiva e negativa

Fonte: Elaborado pelo autor.

Durante o processo de treinamento, foram monitoradas as métricas *training loss* e *validation loss*, que indicam a capacidade do modelo em aprender a generalizar a tarefa de ligação de esquemas. A *training loss* consiste em verificar o quanto o modelo se ajusta aos dados de treinamento e a *validation loss* avalia o desempenho do modelo em dados nunca vistos, ou seja, por meio do conjunto de validação.

Tais métricas são fundamentais para identificar casos de *overfitting* ou *underfitting*. O primeiro acontece quando a *training loss* diminui progressivamente, mas a *validation loss* se estabiliza ou começa a aumentar. Isso significa que o modelo memorizou os dados de treinamento e perdeu a capacidade de generalizar. Em relação ao segundo caso, este ocorre quando ambas as métricas de perda permanecem com valores altos e constantes, e com isso, indica que o modelo não conseguiu aprender os padrões e complexidade da tarefa. Apesar de não existir um valor ideal para as métricas de *training loss* e *validation loss*, o intuito é que ambas sejam minimizadas ao longo do treinamento. Ao atingirem um ponto de estabilização, continuar o processo de ajuste pode levar ao *overffiting*.

Diante desse cenário, os valores obtidos para as métricas de *training loss* e *validation loss* durante o treinamento dos três modelos são apresentados nas tabelas 1 e 2, respectivamente.

Tabela 1 – Valores da *training loss* para os três modelos

Época	BERTimbau base	BERTimbau grande	BERTugues
1	0,1656	0,1328	0,1550
2	0,0569	0,0426	0,0576
3	0,0301	0,0193	0,0302
4	0,0157	**0,0077**	0,0165

Fonte: Elaborado pelo autor.

Tabela 2 – Valores da *validation loss* para os três modelos

Época	BERTimbau base	BERTimbau grande	BERTugues
1	0,0733	0,0596	0,0730

2	0,0444	0,0385	0,0441
3	0,0362	0,0280	0,0377
4	0,0350	**0,0273**	0,0365

Fonte: Elaborado pelo autor.

Ao analisar as tabelas 1 e 2, é possível observar que todos os modelos apresentaram quedas constantes nas métricas de *training* e *validation loss* ao longo das quatro épocas. Nesse contexto, o *BERTimbau* grande se destacou dos demais, em que foram obtidos os melhores resultados em todas as épocas e métricas, com os menores valores para a *training loss* (0,0077) e *validation loss* (0,0273) na quarta época. Adicionalmente, é válido ressaltar que não ocorreram *overfitting* ou *underfitting* pois as métricas caíram de forma progressiva sem nenhum aumento além de não apresentaram estabilização precoce nem valores elevados. Isso reforça que os modelos aprenderam os padrões da tarefa.

Para avaliar o processo de ajuste fino, como se trata de uma tarefa de classificação binária, foram aplicadas as métricas de Acurácia, Precisão, Revocação e F1-Score em cada época para o conjunto de validação. Dessa forma, os resultados obtidos por tais métricas são detalhados nas tabelas 3, 4 e 5 para os modelos *BERTimbau* base, *BERTimbau* grande e *BERTugues*, respectivamente.

Tabela 3 – Métricas para o conjunto de validação do *BERTimbau* base por época

Época	Acurácia	Precisão	Revocação	F1-Score
1	0,9740	0,9292	0,8828	0,9054
2	0,9843	0,9545	0,9328	0,9435
3	0,9882	0,9668	0,9492	0,9579
4	**0,9898**	**0,9656**	**0,9623**	**0,9639**

Fonte: Elaborado pelo autor.

Tabela 4 – Métricas para o conjunto de validação do *BERTimbau* grande por época

Época	Acurácia	Precisão	Revocação	F1-Score
1	0,9777	0,9538	0,8848	0,9180
2	0,9871	0,9614	0,9467	0,9540
3	0,9915	0,9721	0,9678	0,9699
4	**0,9933**	**0,9792**	**0,9734**	**0,9763**

Fonte: Elaborado pelo autor.

Tabela 5 – Métricas para o conjunto de validação do *BERTugues* por época

Época	Acurácia	Precisão	Revocação	F1-Score

1	0,9727	0,9096	0,8956	0,9025
2	0,9840	0,9457	0,9409	0,9433
3	0,9874	0,9517	0,9591	0,9554
4	**0,9892**	**0,9633**	**0,9597**	**0,9615**

Fonte: Elaborado pelo autor.

É perceptível, ao verificar as tabelas 3, 4 e 5, que todos os modelos apresentaram uma evolução consistente ao longo das épocas, com valores superiores a 94% em todas as métricas a partir da terceira época. O *BERTimbau* grande novamente se destacou com o melhor desempenho geral, o qual atingiu os maiores resultados na quarta época, especialmente métrica F1-Score, que relaciona Precisão e Revocação. Sobre *BERTimbau* base e *BERTugues*, ambos obtiveram resultados muito próximos ao *BERTimbau* grande, com uma diferença média de aproximadamente um ponto percentual em todas as métricas.

4.3.1 Avaliação com a utilização de um conjunto de teste

Após o processo de ajuste fino, foi elaborado um conjunto de teste com 991 pares de pergunta-SQL, que não estavam presentes no conjunto de treinamento, com o objetivo de avaliar o desempenho final dos modelos para dados nunca vistos. As mesmas métricas utilizadas no conjunto de validação, Acurácia, Precisão, Revocação e F1-Score, também foram aplicadas ao conjunto de teste. No entanto, há uma diferença importante a ser comentada: no conjunto de validação, essas métricas foram calculadas por meio da combinação das tabelas e colunas, enquanto no conjunto de teste, as tabelas e colunas foram avaliadas separadamente, de forma granular. Adicionalmente, foram aplicados limiares (*thresholds*), com variação entre 0,5 até 0,9, a fim de analisar o impacto de diferentes intervalos sobre as métricas. Dessa forma, nas tabelas de 6 a 11 são apresentados os resultados obtidos do conjunto de teste para os três modelos, em que as tabelas 6, 7 e 8 são destinadas para o desempenho no reconhecimento das tabelas e nas tabelas 9, 10 e 11 são exibidos os valores para a identificação das colunas.

Tabela 6 – Métricas para as tabelas sobre conjunto de teste do *BERTimbau* base

Limiar	Acurácia	Precisão	Revocação	F1-Score
0,5	0,8911	0,8850	0,9036	0,8942
0,6	**0,8920**	**0,8892**	**0,8998**	**0,8945**

0,7	0,8900	0,8935	0,8901	0,8918
0,8	0,8914	0,9024	0,8820	0,8921
0,9	0,8900	0,9141	0,8653	0,8890

Fonte: Elaborado pelo autor.

Tabela 7 – Métricas para as tabelas sobre o conjunto de teste do *BERTimbau* grande

Limiar	Acurácia	Precisão	Revocação	F1-Score
0,5	0,9013	0,8983	0,9089	0,9036
0,6	0,9024	0,9037	0,9046	0,9041
0,7	**0,9029**	**0,9090**	**0,8992**	**0,9041**
0,8	0,9029	0,9140	0,8933	0,9035
0,9	0,8996	0,9190	0,8804	0,8993

Fonte: Elaborado pelo autor.

Tabela 8 – Métricas para as tabelas sobre o conjunto de teste do *BERTugues*

Limiar	Acurácia	Precisão	Revocação	F1-Score
0,5	0,8999	0,8926	0,9133	0,9028
0,6	**0,9018**	**0,8993**	**0,9089**	**0,9041**
0,7	0,9021	0,9054	0,9019	0,9036
0,8	0,9018	0,9097	0,8960	0,9028
0,9	0,8988	0,9179	0,8798	0,8985

Fonte: Elaborado pelo autor.

Tabela 9 – Métricas para as colunas sobre o conjunto de teste do BERTimbau base

Limiar	Acurácia	Precisão	Revocação	F1-Score
0,5	**0,8979**	**0,8194**	**0,7788**	**0,7986**
0,6	0,8974	0,8270	0,7657	0,7952
0,7	0,8976	0,8387	0,7503	0,7921
0,8	0,8958	0,8502	0,7274	0,7840
0,9	0,8929	0,8689	0,6925	0,7708

Fonte: Elaborado pelo autor.

Tabela 10 – Métricas para as colunas sobre o conjunto de teste do *BERTimbau* grande

Limiar	Acurácia	Precisão	Revocação	F1-Score
0,5	**0,9032**	**0,8352**	**0,7819**	**0,8076**
0,6	0,9038	0,8438	0,7730	0,8068
0,7	0,9036	0,8516	0,7620	0,8043
0,8	0,9028	0,8621	0,7454	0,7995
0,9	0,9015	0,8775	0,7220	0,7922

Fonte: Elaborado pelo autor.

Tabela 11 – Métricas para as colunas sobre o conjunto de teste do *BERTugues*

Limiar	Acurácia	Precisão	Revocação	F1-Score
0,5	**0,8994**	**0,8307**	**0,7699**	**0,7992**
0,6	0,8990	0,8408	0,7545	0,7953
0,7	0,8996	0,8530	0,7419	0,7935
0,8	0,8983	0,8668	0,7194	0,7863
0,9	0,8948	0,8839	0,6853	0,7720

Fonte: Elaborado pelo autor.

De forma geral, com base na análise das tabelas 6 a 11, observa-se que o desempenho dos modelos na identificação das tabelas relevantes foi superior comparado a identificação das colunas, especialmente nas métricas de Precisão, Revocação e, consequentemente a F1-Score, que apresentaram diferenças mais significativas. Além disso, é possível notar um padrão comum para todos os modelos: conforme o limiar aumenta, a Precisão tende a subir, enquanto a Revocação diminui, o que é esperado para a tarefa de classificação binária. As demais métricas, como Acurácia e F1-score, se mantiveram estáveis conforme os valores do limiar variavam.

Em relação aos modelos, foi utilizado como principal métrica para a avaliar o modelo com melhor desempenho a F1-score, que realiza a combinação entre a Precisão e Revocação. Desse modo, no reconhecimento das tabelas, o *BERTimbau* grande e *BERTugues* apresentaram os melhores resultados, com F1-score de 0,9041 nos limiares 0,7 e 0,6, respectivamente. Ao considerar também a Acurácia, o *BERTimbau* grande (0,9029) demonstrou uma ligeira vantagem sobre o *BERTugues* (0,9018), o que o torna o modelo de melhor desempenho geral. O *BERTimbau* base ficou logo em seguida com uma diferença de aproximadamente um ponto percentual na F1-Score (0,8945) no limiar 0,6 em comparação aos demais modelos.

Na tarefa de identificação das colunas, o *BERTimbau* grande novamente obteve a melhor performance, o qual conseguiu uma F1-Score de 0,8076 no limiar 0,5 enquanto o *BERTimbau* base e o *BERTugues* registraram 0,7986 e 0,7992, respectivamente, também no limiar 0,5.

Por último, quanto a escolha do limiar, é observado que, para a identificação das tabelas, o valor 0,7 foi o mais adequado e, no reconhecimento das colunas, o limiar ideal foi de 0,5 pelo modelo *BERTimbau* grande. Esses limiares proporcionaram o melhor equilíbrio entre Precisão e Revocação e, consequentemente, garantiram a maximização da F1-score.

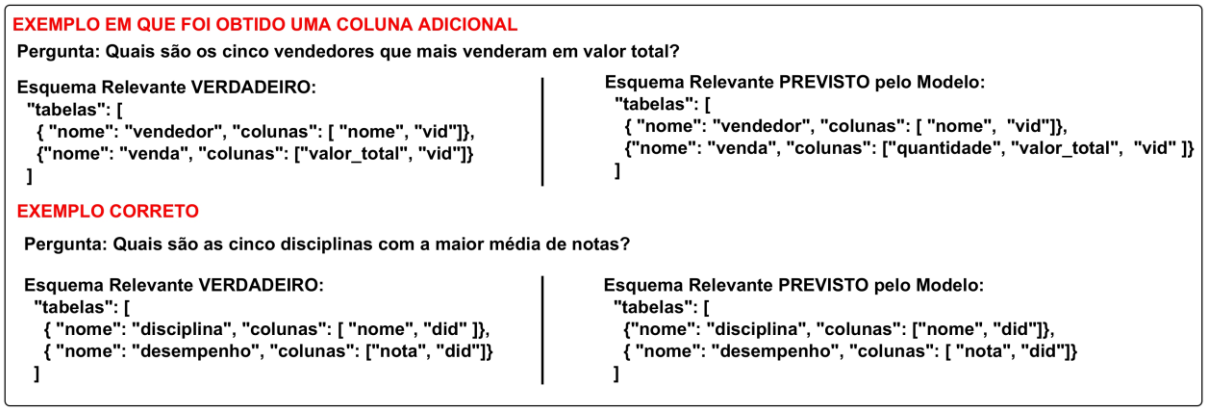
4.3.2 Análise final do ajuste fino dos modelos pré-treinados

Em todos os testes realizados, com inclusão das análises das métricas de *tranining* e *validation loss*, e os resultados nos conjuntos de validação e teste, o *BERTimbau* grande superou os outros dois modelos em todos os cenários. Isso se deve, principalmente, ao fato de ter sido pré-treinado com um volume maior de dados em língua portuguesa e possuir aproximadamente três vezes mais parâmetros comparado aos outros dois modelos. Como consequência, tem-se um compreendimento melhor da linguagem e uma maior capacidade de lidar com tarefas complexas. Ainda assim, tanto o *BERTimbau* base quanto o *BERTugues* apresentaram resultados excelentes e muito próximos ao modelo de maior porte, com destaque para a tarefa de identificação das tabelas, na qual o *BERTugues* igualou o desempenho na métrica F1-Score.

Além disso, apesar deste trabalho não ter foco na comparação do tempo de execução e consumo de recursos computacionais, é válido mencionar que o processo de ajuste fino do *BERTimbau* grande demorou cerca de 15 horas e 22 minutos para ser executado, com um consumo de 8,4 GB de VRAM. Nos casos do *BERTimbau* base e do *BERTugues*, ambos levaram cerca de 5 horas e 14 minutos para a conclusão do treinamento com uso de 3,1 GB de VRAM.

Como conclusão, o modelo selecionado para ser integrado a arquitetura foi o *BERTimbau* grande, uma vez que, apesar de possuir um custo computacional mais alto e da diferença de desempenho ser relativamente pequena comparado aos outros dois modelos, tal diferença tem um impacto significativo na identificação correta das tabelas e colunas relevantes. Essa precisão é fundamental pois a geração da consulta SQL depende de que as informações sejam precisas e compatíveis com o banco de dados. Por último, como um item complementar, na Figura 37 são apresentados dois exemplos dos resultados obtidos pelo *BERTimbau* grande, com o esquema relevante verdadeiro e o que foi previsto pelo modelo. No primeiro exemplo, observa-se a adição da coluna “*quantidade*” como um elemento extra. Para o segundo exemplo, o modelo obteve todas as tabelas e colunas corretamente.

Figura 37 – Exemplos gerados pelo *BERTimbau* grande



Fonte: Elaborado pelo autor.

4.4 RESULTADOS DO AJUSTE FINO DO GRANDE MODELO DE LINGUAGEM LLAMA

Em relação ao *LLama*, foi realizado um ajuste fino voltado à tarefa de texto para SQL, que consiste em converter uma pergunta em linguagem natural, juntamente com o seu contexto de banco de dados, em uma consulta SQL correspondente. Para isso, foi utilizada a versão "*Meta-Llama-3.1-8B-Instruct*", com 8 bilhões parâmetros, pré-treinada em múltiplos idiomas, com inclusão do português.

Para os dados do treinamento, foi usado como base o conjunto *Spider* (Yu et al., 2018), o qual realizou-se a inserção de ruídos nas tabelas/colunas e posteriormente foi adaptado para o formato de entrada do modelo, com ambos os processos detalhados na seção 3.5. Com isso, foi obtido um total de 42.450 exemplos, cada um composto pela pergunta em linguagem natural, o esquema relevante do banco de dados com ruídos e a consulta SQL correspondente. Tal conjunto foi dividido em 80% para treinamento (33.960 exemplos) e 20% para validação (8.490 exemplos).

Além disso, para viabilizar o ajuste fino de forma eficiente, foram empregadas as técnicas LoRA e *QLoRA*, responsáveis por otimizar o uso memória e o reduzir o custo computacional de carregamento do modelo. Dessa forma, no quadro 4 são descritas todas as métricas utilizadas no treinamento.

Quadro 4 – Parâmetros utilizados para o treinamento do *LLama*

Tipo	Parâmetro	Valor	Significado
Treino	Número de épocas (*num_train_epochs*)	5	Quantidade de vezes que o modelo percorre todo o conjunto de treinamento
Treino	Taxa de aprendizado (*learning_rate*)	1e-5	Taxa de aprendizado inicial usada pelo otimizador
Treino	Tamanho do lote (*per_device_train_batch_size*)	1	Tamanho do lote pela GPU
Treino	Acumulação de gradiente (*gradient_accumulation_steps*)	32	Número de lotes que os gradientes acumulam antes da atualização dos pesos
Treino	Comprimento máximo da entrada (*max_seq_length*)	384	Comprimento máximo da entrada processada pelo modelo
Treino	Otimizador (*optim*)	paged_adamw_8bit	Algoritmo de otimização usado durante o treinamento para ajuste dos pesos
Treino	Escalanador da taxa de aprendizado (*lr_scheduler_type*)	Cosine	Tipo de decaimento para a taxa de aprendizado ao longo do tempo
Treino	Ponto flutuante de 16 bits (*fp16)*	True	Uso de ponto flutuante em 16 bits para economia de memória
QLoRA	Carregamento em 4 bits (*load_in_4bit*)	True	Carrega o modelo quantizado em 4 bits
QLoRA	Dupla quantização (*bnb_4bit_use_double_quant*)	True	Aplica uma dupla quantização ao modelo para reduzir mais ainda o uso de memória
QLoRA	Tipo de quantização (*bnb_4bit_quant_type*)	Nf4	Tipo de quantização não uniforme
LoRA	Rank da decomposição (*r*)	16	Controla o tamanho dos adaptadores
LoRA	Lora Alpha (*lora_alpha*)	32	Fator de escalonamento aplicado aos adaptadores

LoRA	Taxa de abandono (*lora_dropout*)	0,1	Probabilidade de abandono aplicada aos adaptadores
LoRA	Módulos alvo do modelo(*target_modules*)	q_proj,k_proj,v_proj,o_proj, gate_proj, up_proj, down_proj	Módulos do modelo onde os adaptadores são inseridos

Fonte: Elaborado pelo autor.

Assim como foi feito no treinamento dos modelos pré-treinados, durante o ajuste fino do *Llama* foram monitoradas as métricas de *training* e *validation loss* a cada época, com o objetivo de verificar a ocorrência de *overfitting* ou *underfitting*. Os resultados obtidos estão detalhados na tabela 12.

Tabela 12 – Valores da *training* e *validation loss* para o *Llama*

Época	Training loss	Validation loss
1	0,2382	0,2374
2	0,1985	0,1990
3	0,1813	0,1926
4	0,1716	0,1891
5	0,1771	0,1874

Fonte: Elaborado pelo autor.

Com a análise da tabela 12, é possível observar que ambas as métricas tiveram quedas constantes nas quatros primeiras épocas, o que indica que o modelo teve um bom aprendizado, sem sinais de *overffiting*. Na quinta época, houve um leve aumento para a *training loss*, enquanto a *validation loss* continuou em queda, o qual indica que modelo atingiu um ponto de estabilidade.

Para a avaliação do processo de ajuste fino, o modelo *LLama* ajustado foi comparado a outros dois modelos: a versão base do *Llama*, com 8 bilhões de parâmetros, sem qualquer ajuste fino, e o modelo *SQLCoder* (“*defog/sqlcoder-7b-2*”), derivado de uma versão distinta do *LLama*, chamada *CodeLlama*, com 7 bilhões de parâmetros. Tal modelo passou por um ajuste fino completo, sem uso das técnicas de *LoRA* e *QLoRA*, voltada especificamente para as tarefas de conversão de texto para SQL. O objetivo dessa comparação foi verificar se o *Llama* ajustado obteve resultados

expressivos em relação a sua versão base e como foi o seu desempenho diante de um modelo especializado e totalmente ajustado.

Todos os três modelos foram avaliados com o mesmo conjunto de teste utilizado na etapa anterior para os PLMs, porém reduzido para 160 exemplos. A avaliação consistiu na comparação entre a consulta SQL gerada por cada modelo e a consulta esperada do conjunto de teste. Esse processo foi feito manualmente, ou seja, foi inspecionado os dois códigos SQL e verificou se estavam semanticamente corretos. Dessa maneira, na tabela 13 é detalhada a quantidade de acertos e erros por cada modelo.

Tabela 13 – Desempenho dos modelos no conjunto de teste

Modelo	Acertos	Erros	% de acertos
Llama base	61	99	38,125%
Llama ajustado	95	65	59,375%
SQLCoder	**105**	**55**	**65,625%**

Fonte: Elaborado pelo autor.

Com base nos resultados da tabela 13, o *SQLCoder* foi o modelo com o melhor desempenho no conjunto de testes, o qual obteve uma acurácia de 65,625%. Esse valor representa uma vantagem de 6,25% e 27,5% em relação ao *LLama* com o ajuste fino e ao *Llama* base, respectivamente. Isso comprova que o processo de ajuste fino, seja por meio das técnicas como *LoRA* e *QLoRA*, que mantêm os pesos originais do modelo, seja pelo ajuste completo como no caso do *SQLCoder*, trazem melhorias significativas no desempenho de tarefas específicas.

4.4.1 Análise final do ajuste fino do *Llama*

Com o processo de ajuste fino efetuado no *Llama*, foi possível analisar que as métricas de *training* e *validation loss* tiveram uma queda contínua ao longo das épocas, com um leve aumento da *training loss* na última. As reduções consistentes e os valores finais baixos de ambas as métricas demonstram que o modelo se adaptou adequadamente ao conjunto de treinamento, aprendeu os padrões da tarefa e não apresentou indícios de *overfitting* ou *underfitting*. Sobre o tempo de execução, o treinamento completo demandou aproximadamente 6 dias e 13 horas, com um

consumo de 11 GB de VRAM durante o processamento do conjunto de treino e pico de 15 GB de VRAM na etapa do conjunto de validação.

Em relação a avaliação do conjunto de testes, o *SQLCoder* superou os outros dois modelos, o que atesta que o ajuste fino completo proporciona uma precisão maior na geração de consultas SQL. Vale ressaltar que, diferente do *LLama* 3.1, o qual foi pré-treinado para responder a uma ampla variedade de tarefas, o *SQLCoder* foi derivado do *CodeLlama*, um modelo previamente ajustado com foco em múltiplas linguagens de programação. Por conta disso, o *SQLCoder* apresenta um conhecimento maior em tarefas específicas de conversão de texto para SQL. Ainda assim, o modelo *Llama* ajustado com LoRA e *QLoRA*, adotado neste trabalho devido ao menor custo computacional, superou em mais de 20 pontos percentuais o seu modelo base e ficou relativamente próximo ao *SQLCoder*, mesmo sem passar por um ajuste fino completo.

Diante desses resultados, o *SQLCoder* foi escolhido para ser integrado a arquitetura, com o objetivo de gerar os códigos SQL referente as perguntas do usuário, pelo fato de apresentar o melhor resultado entre os modelos avaliados. Por fim, na Figura 38 é apresentado um exemplo gerado pelos três modelos. Percebe-se que a versão base do *LLama* realiza um processo de raciocínio antes de gerar a consulta, ao contrário dos outros dois modelos, que iniciam diretamente com o código SQL. Além disso, destaca-se que o *SQLCoder* usou uma função específica (*cume_dist*) para responder à pergunta.

Figura 38 – Exemplo gerado pelos modelos

Pergunta: Como se distribuem os valores das tarifas pagas no metrô?
SQL Esperado: select am.valor_tarifa from acessos_metro am

MODELO LLAMA BASE

SQL Gerado:
para responder à sua pergunta, precisamos criar uma consulta sql que mostre...
aqui está uma consulta sql que pode ajudar a responder à sua pergunta:

```
select m.nome_estacao,  avg(am.valor_tarifa) as media_tarifa,
count(am.valor_tarifa) as total_acessos, sum(am.valor_tarifa) as total_recebido
from acessos_metro am
join metro m on am.estacao_id = m.estacao_id
group by m.nome_estacao
order by media_tarifa desc
```

ERRADO

MODELO LLAMA COM AJUSTE FINO

SQL Gerado: select valor_tarifa from acessos_metro
group by valor_tarifa order by valor_tarifa
CORRETO

MODELO SQLCODER

SQL Gerado:select acessos_metro.valor_tarifa, cume_dist() over
(order by acessos_metro.valor_tarifa) as cumulative_distribution
from acessos_metro order by acessos_metro.valor_tarifa nulls last
CORRETO

Fonte: Elaborado pelo autor.

4.4 RESULTADOS DA VALIDAÇÃO DA ARQUITETURA HÍBRIDA

Com o ajuste fino dos modelos PLM e LLM concluído, foram selecionados o *BERTimbau* grande e o *SQLCoder* para integração à arquitetura proposta. Para validar o desempenho da arquitetura híbrida, foi aplicada a metodologia descrita na seção 3.7, com a elaboração de um conjunto de teste composto por 60 pares de pergunta-SQL, distribuídos em 20 exemplos para cada nível de dificuldade fácil, médio e difícil. Posteriormente, cada pergunta foi inserida na ILN que implementa a arquitetura, a qual foi conectada a um banco de dados de dicionários multilíngues. A categorização dos níveis de dificuldade foi definida por meio do seguinte processo:

a) Fácil –consultas simples, com uso de uma única tabela ou um *JOIN* básico com outra tabela, além de agregações e agrupamentos simples, como *COUNT*, *AVG* e *GROUP BY*, e condições *WHERE* diretas.
b) Médio – consultas com múltiplos *JOINs*, subconsultas simples e cláusulas *WHERE* mais complexas.
c) Difícil – presença de subconsultas complexas, *auto-joins* e combinações elaboradas de *JOIN* e GROUP BY.

Como avaliação, para o *BERTimbau* grande, foi efetuada uma análise manual a fim de verificar se as tabelas e colunas importantes identificadas estavam de acordo com o contexto de cada pergunta. Foi considerado correta a saída em que todas as tabelas e colunas necessárias à elaboração da consulta foram reconhecidas, mesmo que o modelo incluísse elementos extras não diretamente associados à questão. Essa abordagem permite que o LLM filtre apenas os itens relevantes e construa o código SQL a partir deles.

Sobre o *SQLCoder*, também foi efetuado uma checagem manual, em que foi verificado se código SQL gerado pelo modelo era sintaticamente equivalente a consulta do conjunto de teste. Dessa forma, os resultados do *BERTimbau* grande e do *SQLCoder* são exibidos na tabela 14 e 15, respectivamente, organizados pelos diferentes níveis de dificuldade das perguntas.

Tabela 14 – Desempenho do *BERTimbau* grande para a arquitetura híbrida

Dificuldade da pergunta	Total de perguntas	Acertos	Acurácia (%)
Fácil	20	19	95%
Média	20	19	95%
Difícil	20	17	85%
Total	**60**	**55**	**91,67%**

Fonte: Elaborado pelo autor.

Tabela 15 – Desempenho do *SQLCoder* para a arquitetura híbrida

Dificuldade da pergunta	Total de perguntas	Acertos	Acurácia (%)
Fácil	20	17	85%
Média	20	16	80%
Difícil	20	12	60%
Total	**60**	**45**	**75%**

Fonte: Elaborado pelo autor.

Por meio da tabela 14, é possível notar que o *BERTimbau* grande obteve resultados consistentes, inclusive nas perguntas de maior complexidade, com uma acurácia geral de 91,67%. Ao realizar uma análise qualitativa, foi observado que, na maioria das perguntas, o modelo identificou corretamente o esquema relevante, além de alguma tabela ou coluna adicional não pertencente ao escopo da questão, o que era esperado. Em algumas perguntas, como "*Quantas colocações existem por tipo de status, mas apenas para o idioma português?*", "*Qual é o número total de citações para colocações que são da classe gramatical verbo e do idioma inglês?*" e "*Conte quantas colocações do idioma português são marcadas como frequente vs não frequente*", o *BERTimbau* grande retornou uma quantidade maior de tabelas e colunas extras, porém em todos os cenários o *SQLCoder* foi capaz de filtrar os elementos mais importantes e gerar a consulta SQL correta.

Além disso, foi verificado também que pequenas variações linguísticas da pergunta afetavam a saída do modelo. Alguns casos são mostrados a seguir:

a) A pergunta "*Qual é o número total de verbetes?*" deveria retornar a tabela "*verbete*" com a coluna "*id*", mas o modelo não reconheceu nenhum elemento. Porém, ao reformular a questão para "*Qual é a número total de verbetes? Conte pelo seu id*", o *BERTimbau* grande identificou corretamente os itens. Essa situação ocasionou falhas ao *SQLCoder* que inferiu elementos inexistentes ao gerar o código SQL, embora a intenção da consulta, com o comando COUNT, estivesse correta.

b) Para a questão "*Qual o nome do usuário com o id 5?*", o modelo retornou corretamente a tabela "*usuario*" e a coluna "*id*". No entanto, ao alterar o "id" por "ID", o modelo não reconheceu a coluna correspondente.

c) Uma das tabelas identificadas na saída da pergunta "*Compare o t_score da colocação para cada nome da relação gramatical, mas apenas para o idioma inglês"* é a "*relacao_gramatical*" com as colunas "*id*" e "*nome_relacao_gramatical*". Porém, se a pergunta for reformulada, com a substituição do trecho "*para cada nome da relação gramatical*," por "*para cada relação gramatical*", o modelo deixa de incluir a coluna "*nome_relacao_gramatical*" na saída. O mesmo caso ocorreu na questão "*Quantas colocações existem por tipo de status, mas apenas*

para o idioma português?", em que uma das tabelas obtidas é a "*status*" associada as colunas "*id*" e "*tipo_status*". Ao retirar o termo "*tipo de*" da questão, o modelo identifica apenas o "*id*".

d) Por último, na pergunta "*Quais usuários estão associados aos verbetes, mas não estão associados a nenhuma colocação?*", a tabela "*colocacao*", essencial para a geração da consulta, não foi reconhecida. Mas ao remodelar a pergunta para "*Quais usuários estão associados aos verbetes, mas não estão associados as colocações?*", o modelo retornou corretamente a tabela comentada. Nesse caso, o *SQLCoder* conseguiu inferir a tabela ausente e gerou o código adequado.

Essas limitações comentadas do modelo podem ser minimizadas com o uso de um conjunto de dados mais extenso. O corpus utilizado no ajuste fino contém cerca de 10.000 exemplos, o qual cobre diferentes tipos de perguntas e permite uma boa capacidade de generalização. Com a ampliação desse conjunto, por exemplo, a adição de mais 10.000 exemplos, o modelo conseguiria capturar melhor as variações linguísticas e identificar com mais precisão as tabelas e colunas relevantes.

Quanto ao *SQLCoder*, conforme mostrado na tabela 15, o desempenho foi satisfatório nos níveis fácil e médio de dificuldade, com acurácias de 85% e 80%, respectivamente. Para perguntas de complexidade difícil, foi obtido um valor inferior, de 60%, em que resultou em uma média geral de 75%. Ao analisar as consultas geradas, de uma forma qualitativa, foram identificados três tipos principais de erros, a saber:

a) **Dependência do *BERTimbau* grande:** em algumas perguntas, como "**"***Para cada idioma, qual é o número médio de colocações por verbete?***" e "*Quais usuários estão associados aos verbetes, mas não estão associados a nenhuma colocação?*", o PLM falhou em reconhecer os elementos essenciais para a construção do código SQL. Como consequência, o *SQLCoder* teve que inferir tabelas e colunas inexistentes, e consequentemente, gerou consultas incorretas. Porém, essa situação aconteceu apenas para uma pequena parcela do conjunto de testes.

b) **Alucinação do SQLCoder:** o modelo alucinou durante a elaboração da consulta SQL, com inclusão de erros de sintaxe e adição cláusulas

*JOIN*s e *WHERE* extras sem relação com a pergunta ou referentes a entidades inexistentes.

c) **Interpretação errada da pergunta:** em alguns casos, o LLM interpretou de forma equivocada o enunciado do conjunto. Por exemplo, na questão "*Conte quantos verbetes são 'Raros' (frequência < 100), 'Comuns' (frequência entre 100 e 1000) e 'Frequentes' (frequência > 1000)*", o modelo gerou uma consulta incorreta. Ao reformular a questão, com mais detalhes, para "*Conte quantos verbetes são considerados 'Raros', com frequência menor 100, os verbetes considerados 'Comuns', com frequência entre 100 e 1000 e os verbetes considerados 'Frequentes', com frequência maior que 1000*", foi obtido a saída correta. Outra situação, *"Para cada idioma, mostre o verbete de maior frequência e qual é essa frequência*", o modelo retornou todos os verbetes com suas frequências, em vez de apenas o de maior valor por idioma. Mais um caso, "*Qual é o número total de citações para colocações do idioma inglês que são da classe verbo?*", foi gerado a consulta com um JOIN inexistente. Porém, ao inverter a ordem da pergunta, para "*colocações que são da classe gramatical verbo e do idioma inglês*", a consulta foi elaborada corretamente. Por último, um caso específico, para a pergunta "*Mostre a distribuição da frequência dos verbetes por cada classe gramatical*", o modelo associou o termo "*distribuição*" com a função "*cume_dist*", que é uma função de distribuição cumulativa. O *SQLCoder* gerou uma consulta tecnicamente correta, mas inadequada ao contexto semântico da pergunta.

As limitações comentadas, como a alucinações, interpretações incorretas e o menor desempenho em perguntas de alta complexidade, poderiam ser mitigadas com o uso de modelos com maior número de parâmetros, que tendem a lidar melhor com ambiguidades e compreendimento da linguagem. No entanto, esse aprimoramento exigiria um poder computacional mais elevado, que não seria possível utilizar pela limitação de *hardware*.

Como conclusão, por meio dos testes realizados, foi possível comprovar que a arquitetura híbrida proposta é viável para banco de dados de porte pequeno e médio porte, especialmente em perguntas de baixa e média complexidade. Com isso, tem-se uma ferramenta promissora para aplicações na área da ciência de dados.

4.5 DISCUSSÃO COMPARATIVA COM OS TRABALHOS DA LITERATURA

A arquitetura proposta abrange três linhas de pesquisa exploradas na literatura: conversão de linguagem natural para SQL, geração automática de visualizações a partir de perguntas em linguagem natural e uso de modelos de linguagem na construção de interfaces inteligentes. No entanto, os trabalhos correlatos diferem em relação às arquiteturas adotadas, aos conjuntos de dados utilizados, aos modelos empregados e aos métodos de avaliação, o que inviabiliza uma comparação quantitativa baseada em métricas. Dessa forma, optou-se por realizar uma comparação de caráter qualitativo e conceitual deste trabalho com o estado da arte, com foco na discussão das estratégias arquiteturais e das finalidades das abordagens propostas.

No tema de texto para SQL, trabalhos como os de Wang et al. (2019) e Brunner e Stockinger (2020) se concentram em aprimorar a compreensão do esquema do banco de dados para gerar consultas SQL mais corretas. Wang et al. (2019) desenvolvem um codificador baseado em grafos que relaciona tabelas e colunas do banco de dados às palavras da pergunta em linguagem natural e posteriormente aplica essa estrutura à tarefa de ligação de esquemas. Já Brunner e Stockingerl (2020) propõem uma arquitetura que extrai valores da pergunta do usuário, denominados dicas, que geram possíveis candidatos a tabelas e colunas que não são mencionados explicitamente na pergunta. Em seguida, essas dicas, em conjunto com a pergunta e o esquema da base de dados, são fornecidas ao modelo para a construção da consulta SQL correspondente.

Apesar dos avanços, tais trabalhos possuem como foco exclusivo a geração de SQL e não contemplam etapas posteriores, como a seleção ou construção de visualizações. Além disso, suas arquiteturas concentram todas as operações semânticas em um único modelo, o que limita a modularidade do sistema e dificulta a adaptação a diferentes cenários.

Em relação ao tema de visualização orientada por linguagem natural, Liu et al. (2021) constroem um sistema capaz de gerar automaticamente visualizações a partir de perguntas sobre dados tabulares. A abordagem utiliza um fluxo baseado em regras heurísticas e um PLM ajustado para reconhecer agregações e atributos de tabelas e colunas, com o objetivo de mapear perguntas diretamente para tipos de gráficos. Embora os experimentos feitos em cenários controlados demonstrarem resultados

promissores, tal pesquisa apresenta limitações quanto à generalização para banco de dados mais complexos, uma vez que o conjunto de dados utilizado no ajuste fino conter em sua maioria consultas SQL simples, além de não explorar grandes modelos de linguagem.

Outros trabalhos mais recentes, como Wang e Crespo-Quinones (2023), Li et al. (2024) e Yang et al. (2024), exploram diretamente grandes modelos de linguagem na geração de visualizações. Wang e Crespo-Quinones (2023) combina diferentes módulos de PLMs na construção de um único modelo capaz de gerar uma linguagem de visualização chamada *vega-zero* e, posteriormente, compara essa abordagem com um LLM ajustado especificamente para tal linguagem. Li et al. (2024) propõem estratégias de aprendizado em contexto, em conjunto a módulos de filtragem de esquema e seleção automática de exemplos, para melhorar o desempenho das visualizações geradas. Já Yang et al. (2024) se baseia na técnica de cadeia de pensamento para decompor a tarefa de conversão de texto para SQL em subetapas, como o reconhecimento das colunas, tabelas, filtros e agregações em cada passo. Após esse processo, o LLM mapeia para o gráfico mais apropriado ao contexto da pergunta.

Nos três casos, a geração da visualização é conduzida por um único modelo de linguagem, ou seja, uma arquitetura monolítica. Nesses trabalhos, não há uma divisão de responsabilidades entre modelos, nem uma separação entre compreensão do esquema da base de dados e geração do código SQL. Em comparação aos artigos analisados, a principal contribuição deste trabalho consiste na proposição de uma arquitetura híbrida, na qual modelos de linguagem distintos desempenham tarefas definidas e que se complementam. O uso de um PLM dedicado à tarefa de ligação de esquemas permite uma filtragem semântica explícita do esquema relacional, que reduz a complexidade do contexto fornecido ao LLM responsável pela geração da consulta SQL. Essa separação não apenas melhora a interpretabilidade do processo, como também favorece a modularidade e robustez da arquitetura em bancos de dados reais.

Adicionalmente, ao integrar um algoritmo heurístico para recomendação de visualizações, a arquitetura amplia o conjunto de representações gráficas suportadas, com inclusões de representações mais complexas, recursos não contemplados ou tratados de forma limitada nos trabalhos correlatos analisados.

4.6 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentados os testes e validações efetuados a respeito da arquitetura híbrida e dos PLMs e LLM, além da exibição das principais telas da interface desenvolvida. Em relação ao PLM, três modelos foram submetidos ao processo de ajuste fino para a tarefa de ligação de esquemas e posteriormente avaliados por meio de métricas de classificação binária. Foi constatado, por meio da análise dessas métricas, que todos os modelos obtiveram desempenhos satisfatórios, com destaque para a identificação de tabelas em relação as colunas. Entre os modelos avaliados, o *BERTimbau* grande apresentou o melhor desempenho por uma ligeira margem, devido principalmente ao seu pré-treinamento com uma quantidade maior de dados e um número de parâmetros superior aos outros dois PLMs.

Sobre o LLM, foi realizado um ajuste fino no modelo *LLama* de 8 bilhões de parâmetros, com o uso das técnicas de *LoRA* e *QLoRA* voltado à tarefa de conversão de texto para SQL. Para a validação, o modelo ajustado foi comparado à sua versão base, sem qualquer ajuste fino, e ao *SQLCoder*, um LLM previamente ajustado de forma completa para essa tarefa. A comparação foi realizada por meio de um conjunto de testes composto por pares de pergunta-SQL, o qual foi executado nos três modelos e os resultados foram analisados manualmente. Foi verificado que o ajuste fino proporcionou ganhos expressivos de desempenho ao *Llama*, que obteve uma diferença significativa em relação a sua versão original. Por outro lado, o *SQLCoder* atingiu os melhores resultados, com indicação que o ajuste no modelo completo é mais eficaz para a tarefa de texto para SQL, embora com uma vantagem pequena sobre o *LLama* ajustado com *LoRA* e *QLoRA*.

Por último, os melhores modelos, *BERTimbau* grande e *SQLCoder*, foram integrados a arquitetura híbrida. Para validar o sistema como um todo, foram executadas 60 perguntas, com diferentes níveis de dificuldade (fácil, médio e difícil), cada uma acompanhada de seu respectivo código SQL, em um banco de dados do mundo real relacionado a dicionários multilíngues. A avaliação contemplou todas as etapas da arquitetura, desde a entrada da questão do usuário, identificação das tabelas e colunas importantes, geração do código SQL e construção das visualizações apropriadas ao contexto. Como resultado, foi observado que o *BERTimbau* apresentou ótimos resultados nos três níveis de dificuldade, com uma acurácia geral de 91,67%. O *SQLCoder* gerou corretamente a maioria das consultas para as

perguntas de níveis fácil (85%) e médio (80%), porém obteve um desempenho inferior para a dificuldade difícil (60%), que resultou em uma acurácia geral de 75%. Esses dados indicam que a arquitetura proposta é viável para ser implantada em ambientes com bases do mundo real de pequeno e médio porte, assim configura-se como uma ferramenta eficaz no apoio ao processo de análise e visualização de dados no contexto da ciência de dados.

CAPÍTULO

5

5. CONCLUSÃO

O avanço da era digital resultou em uma produção massiva de informações nas últimas décadas, um processo que segue em crescimento nas mais diversas áreas, como finanças, comércio, segurança, entre outras. Nesse contexto, a ciência de dados assume um papel essencial ao transformar esses grandes volumes de informações em conhecimento útil. Aliada a tal conceito, a análise e visualização de dados permitem identificar padrões e tendências ocultas ao mostrá-los por meio de representações visuais. Os três paradigmas, em conjunto, convergem para o mesmo propósito: auxiliar diretamente na tomada de decisões por parte de entidades e organizações.

Uma das abordagens para tornar o processo de ciência e análise de dados mais acessível e democrático, especialmente para usuários sem conhecimento prévio de ferramentas específicas, é por meio de interfaces de linguagem natural. Esses sistemas direcionam o foco para as visualizações, em vez da manipulação de operações complexas de dentro das ferramentas. Com o avanço de novas tecnologias associadas à inteligência artificial, os modelos de linguagem passaram a desempenhar um papel central na construção de ILNs mais eficazes.

Diante desse cenário, este trabalho propôs o desenvolvimento de uma arquitetura híbrida baseada na combinação de dois modelos de linguagem, um PLM e um LLM, em que cada um cumpriu uma tarefa específica, além do suporte a visualizações diversificadas e complexas.

Na parte do PLM, foram realizados ajustes finos em três modelos, no caso o *BERTimbau* base, *BERTimbau* grande e *BERTugues*, voltados à tarefa de ligação de esquemas. Como validação, utilizaram-se métricas de classificação binária, em que foram obtidos resultados satisfatórios para os três modelos. Além disso, observou-se também que todo os modelos demonstraram um desempenho superior na

identificação de tabelas relevantes do que de colunas. No final do procedimento, o *BERTimbau* grande apresentou os melhores resultados gerais.

Sobre o LLM, foi conduzido um ajuste fino na versão de 8 bilhões de parâmetros do *LLama*, voltada a tarefa de conversão de texto para SQL, combinado as técnicas *LoRA* e *QLoRA*. Para a avaliação, tal modelo ajustado foi comparado com a sua versão base, sem qualquer ajuste fino, e o *SQLCoder*, um modelo treinado de forma completa para a tarefa de texto para SQL. Essa comparação foi feita a partir de um conjunto de pares perguntas-SQL, em que as saídas dos modelos foram avaliadas manualmente. Os resultados indicaram que o ajuste fino foi benéfico ao *Llama*, que alcançou 59,375% de acurácia, um ganho de mais de 20 pontos percentuais em relação a sua versão base (38,125%). Contudo, o modelo ajustado não superou o *SQLCoder*, que obteve 65,625% de acurácia, uma diferença de aproximadamente 6 pontos percentuais. Isso reforça que o ajuste fino completo, apesar de um maior custo computacional, proporcionou melhores resultados.

Com base nesses resultados, os dois modelos com melhor desempenho, *BERTimbau* grande e *SQLCoder*, foram integrados a arquitetura, a qual foi avaliada por meio de um banco de dados do mundo real, voltado a dicionários multilíngues. Para os testes, foram elaborados 60 pares perguntas-SQL, distribuídas de maneira uniforme entre os níveis de dificuldade fácil, médio e difícil, e posteriormente foram inseridos na ILN que implementa a arquitetura. A avaliação das saídas envolveu uma checagem manual para ambos os módulos, PLM e o LLM. No caso do PLM, verificou-se se as tabelas/colunas retornadas estão de acordo com o contexto da pergunta. Sobre o LLM, foi analisado se o código SQL gerado pelo modelo era sintaticamente equivalente ao conjunto de teste.

Os resultados demonstraram que o *BERTimbau* grande alcançou excelente desempenho em todos os níveis de dificuldade, com acurácia geral de 91,67%. Em relação ao *SQLCoder*, foram obtidos valores satisfatórios nas categorias fácil (85%) e médio (80%), porém apresentou dificuldades em resolver perguntas difíceis (60%), o qual atingiu uma acurácia geral de 75%. Esses dados indicam que a arquitetura híbrida é capaz de resolver questões de níveis fácil e médio e, com isso, torna-se viável a sua implantação em bancos de dados reais de pequeno e médio porte, como uma ferramenta de apoio ao processo de ciência e análise de dados, até para visualizações mais complexas.

5.1 CONTRIBUIÇÕES CIENTÍFICAS

Ao comparar este trabalho com os artigos da literatura, destaca-se como principal diferencial a combinação de diferentes modelos de linguagem na construção de uma arquitetura híbrida voltada a visualização inteligente de dados, a qual foi comprovada por meio de uma ILN conectada a um banco de dados do mundo real. Apesar da dependência entre os modelos, essa combinação não comprometeu o processamento das consultas SQL e, consequentemente, a geração das visualizações.

Um destaque especial fica para o suporte a visualizações mais sofisticadas, como mapas, gráfico de redes e diagrama de Sankey, recursos que não foram encontrados nos trabalhos correlatos. Dessa forma, no quadro 5 são apresentadas as principais características dos trabalhos correlatos e deste trabalho, além dos diferenciais comentados.

Quadro 5 - Comparativo entre os trabalhos correlatos com este trabalho

	Lee et al., (2024)	Sah et al., (2024)	Zhu et al., (2024)	Yang et al., (2024)	Ram, Ashinee e Kumar (2024)	Chen et al., (2022)	Liu et al., (2021)	Este trabalho
Geração de visualização oriundas de linguagem natural por meio de um LLM ou PLM	Sim	Sim	Sim	Sim	Sim	Sim	Sim	**Sim**
Uso de técnicas de aprendizado em contexto	Sim	Sim	Sim	Sim	Não	Sim	Não	**Sim**
Extração dos atributos mais importantes da pergunta do usuário	Sim	Sim	Sim	Sim	Sim	Não	Sim	**Sim**
Uso de ajuste fino no LLM ou PLM	Não	Não	Não	Não	Sim	Não	Sim	**Sim**
Conversão da linguagem natural para SQL	Sim	Não	Sim	Sim	Sim	Sim	Não	**Sim**
Geração de visualizações complexas	Não	Não	Não	Não	Não	Não	Não	**Sim**
Combinação de diferentes modelos de linguagem	Não	Não	Não	Não	Não	Não	Não	**Sim**

Fonte: Elaborado pelo autor.

Um ponto adicional importante a ser comentado é que este trabalho foi desenvolvido para ser executado localmente, com o uso de modelos de linguagem *open-source* de baixo custo computacional. Enquanto a maioria dos trabalhos da literatura utilizaram de modelos *closed-source* acessados via API, o que oferece maior poder computacional, mas em troca exige o envio de dados para servidores externos, este trabalho priorizou a segurança e privacidade, assim evita-se o vazamento de informações sensíveis, principalmente relacionadas as bases de dados. Adicionalmente, apesar dos modelos usados de maneira local apresentarem uma menor capacidade de compreensão e entendimento da linguagem, em especial para perguntas de alta complexidade, foi demonstrada a eficácia para questões de níveis fáceis e médio, assim tem-se como garantia um maior controle e confiabilidade das informações.

Com isso, obteve-se como contribuição cientifica uma abordagem híbrida que explora o uso de diferentes modelos de linguagem na construção de interfaces de linguagem natural, a qual amplia o suporte para uma maior variedade de visualizações, com inclusão das mais complexas.

5.2 TRABALHOS FUTUROS

Por fim, alguns possíveis trabalhos futuros incluem a implantação da interface em um sistema com um banco de dados do mundo real, em que os usuários utilizem a plataforma e forneçam *feedback*s sobre a correção das tabelas/colunas obtidas e das consultas SQL geradas. Com isso, essas informações seriam armazenadas e posteriormente usadas no futuro em processos de aprendizado por reforço ou para um treinamento adicional mais robusto dos modelos.

Além disso, existe a possibilidade de explorar novos modelos de linguagens com maior capacidade computacional, caso haja suporte de hardware, com o objetivo de avaliar o impacto de modelos mais complexos em comparação com modelos menores. Adicionalmente, sugere-se adotar estratégias de refinamento mais avançadas, como o próprio aprendizado por reforço, e ampliar os testes para outros domínios de banco de dados, a fim de aumentar a generalização em diferentes contextos.

Como extensão da arquitetura desenvolvida, destaca-se também a possibilidade de integração entre modelos de linguagem e regras heurísticas, as quais poderiam ser utilizadas para validar ou refinar automaticamente as tabelas e colunas selecionadas durante o processo de SL. Por fim, sugere-se a investigação de técnicas de verificação automática de consultas SQL associadas a mecanismos de auto-correção. Tais abordagens podem incluir a validação sintática e semântica das consultas geradas, além do uso de ciclos com os LLMs, que identifica inconsistências e propõe correções antes da geração final do SQL.

REFERÊNCIAS

ABELA, Andrew V. **Advanced presentations by design: creating communication that drives action**. Second edition. San Francisco: Pfeiffer, A Wiley Imprint, 2013.

BHARADIYA, Jasmin. A Comprehensive Survey of Deep Learning Techniques Natural Language Processing. **European Journal of Technology**, v. 7, n. 1, p. 58–66, 2023.

BRUNNER, Ursin; STOCKINGER, Kurt. ValueNet: A Natural Language-to-SQL System that Learns from Database Information. 2020.

CAO, Longbing. Data Science: A Comprehensive Overview. **ACM Computing Surveys**, v. 50, n. 3, p. 1–42, 2017.

CHEN, Yiru; LI, Ryan; MAC, Austin; XIE, Tianbao; YU, Tao; WU, Eugene. NL2INTERFACE: Interactive Visualization Interface Generation from Natural Language Queries. 2022.

CHOWDHARY, K. R. Natural Language Processing. *In*: CHOWDHARY, K.R. (Ed.). **Fundamentals of Artificial Intelligence**. New Delhi: Springer India, 2020, p. 603–649.

DONOHO, David. 50 Years of Data Science. **Journal of Computational and Graphical Statistics**, v. 26, n. 4, p. 745–766, 2017.

DONG, Qingxiu; LI, Lei; DAI, Damai; ZHENG, Ce; MA, Jingyuan; LI, Rui; XIA, Heming; XU, Jingjing; WU, Zhiyong; LIU, Tianyu; CHANG, Baobao; SUN, Xu; LI, Lei; SUI, Zhifang. A Survey on In-context Learning. 2023.

DONG, Shi; WANG, Ping; ABBAS, Khushnood. A survey on deep learning and its applications. **Computer Science Review**, v. 40, p. 100379, 2021.

DUAN, Lian; XIONG, Ye. Big data analytics and business analytics. **Journal of Management Analytics**, v. 2, n. 1, p. 1–21, 2015.

EL MRABET, Mohammed Amine; EL MAKKAOUI, Khalid; FAIZE, Ahmed. Supervised Machine Learning: A Survey. *In*: **2021 4th International Conference on Advanced Communication Technologies and Networking (CommNet)**. Rabat, Morocco: IEEE, 2021, p. 1–10.

FLORATOU, Avrilia; PSALLIDAS, Fotis; ZHAO, Fuheng; DEEP, Shaleen; HAGLEITHER, Gunther; TAN, Wangda; CAHOON, Joyce; ALOTAIBI, Rana; HENKEL, Jordan; SINGLA, Abhik; GROOTEL, Alex van; CHOW, Brandon; DENG, Kai; LIN, Katherine; CAMPOS, Marcos; EMANI, Venkatesh; PANDIT, Vivek; SHNAYDER, Victor; WANG, Wenjing; CURINO, Carlo. NL2SQL is a solved problem... Not!. 2024.

GAO, Yunfan; XIONG, Yun; GAO, Xinyu; JIA, Kangxiang; PAN, Jinliu; BI, Yuxi; DAI, Yi; SUN, Jiawei; WANG, Haofen. **Retrieval-Augmented Generation for Large Language Models: A Survey**. arXiv , 27 mar. 2024. Disponível em: http://arxiv.org/abs/2312.10997. Acesso em: 17 mar. 2026

GE, Yuhang; DONG, Xuemei; ZHANG, Chao; MAO, Yuren; GAO, Yunjun; CHEN, lu; LIN, Jinshu; LOU, Dongfang . C3: Zero-shot Text-to-SQL with ChatGPT. 2023.

HE, Kelei; GAN, Chen; LI, Zhuoyuan; REKIK, Islem; YIN, Zihao; JI, Wen; GAO, Yang; WANG, Qian; ZHANG, Junfeng; SHEN, Dinggang . Transformers in medical image analysis. **Intelligent Medicine**, v. 3, n. 1, p. 59–78, 2023.

HEALY, Yan Holtz and Conor. **From data to Viz | Find the graphic you need**. Disponível em: https://www.data-to-viz.com/data-to-viz.com. Acesso em: 19 out. 2024.

HU, Zhiqiang; WANG, Lei; LAN, Yihuai; XU, Wanyu; LIM, Ee-Peng; BING, Lidong; XU, Xing; PORIA, Soujanya; LEE, Roy Ka-Wei . LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models. 2023.

ISLAM, Mohaiminul; JIN, Shangzhu. An Overview of Data Visualization. *In*: **2019 International Conference on Information Science and Communications Technologies (ICISCT)**. Tashkent, Uzbekistan: IEEE, 2019, p. 1–7.

J, Mathav Raj; VM, Kushala; WARRIER, Harikrishna; GUPTA, Yogesh . Fine Tuning LLM for Enterprise: Practical Guidelines and Recommendations. 2024.

JIANG, Yuchen; LI, Xiang; LUO, Hao; YIN, Shen; KAYNAK, Okyay . Quo vadis artificial intelligence? **Discover Artificial Intelligence**, v. 2, n. 1, p. 4, 2022.

KAVAZ, Ecem; PUIG, Anna; RODRÍGUEZ, Inmaculada. Chatbot-Based Natural Language Interfaces for Data Visualisation: A Scoping Review. **Applied Sciences**, v. 13, n. 12, p. 7025, 2023.

KUCHER, Kostiantyn; PARADIS, Carita; KERREN, Andreas. The State of the Art in Sentiment Visualization. **Computer Graphics Forum**, v. 37, n. 1, p. 71–96, 2018.

LEE, Christopher J.; TRAN, Giorgio; TABALBA, Roderick; LEIGH, Jason. Macro-Queries: An Exploration into Guided Chart Generation from High Level Prompts. 2024.

LI, Qi. Overview of Data Visualization. *In*: LI, Qi (Ed.). **Embodying Data**. Singapore: Springer Singapore, 2020, p. 17–47.

LI, Shuaimin; CHEN, Xuanang; SONG, Yuanfeng; SONG, Yunze; ZHANG, Chen. **Prompt4Vis: Prompting Large Language Models with Example Mining and Schema Filtering for Tabular Data Visualization**. arXiv, 2024. Disponível em: https://arxiv.org/abs/2402.07909. Acesso em: 7 fev. 2026

LIN, Tianyang; WANG, Yuxin; LIU, Xiangyang; QIU, Xipeng . A survey of transformers. **AI Open**, v. 3, p. 111–132, 2022.

LIU, Can; HAN, Yun; JIANG, Ruike; YUAN, Xiaoru. ADVISor: Automatic Visualization Answer for Natural-Language Question on Tabular Data. *In*: **2021 IEEE 14th Pacific Visualization Symposium (PacificVis)**. Tianjin, China: IEEE, 2021, p. 11–20.

LUO, Yuyu; TANG, Nan; LI, Guoliang; TANG, Jiawei; CHAI, Chengliang; QIN, Xuedi . Natural Language to Visualization by Neural Machine Translation. **IEEE Transactions on Visualization and Computer Graphics**, v. 28, n. 1, p. 217–226, 2022.

MAAMARI, Karime; ABUBAKER, Fadhil; JAROSLAWICZ, Daniel; MHEDHBI, Amine. The Death of Schema Linking? Text-to-SQL in the Age of Well-Reasoned Language Models. 2024.

MADDIGAN, Paula; SUSNJAK, Teo. Chat2VIS: Generating Data Visualizations via Natural Language Using ChatGPT, Codex and GPT-3 Large Language Models. **IEEE Access**, v. 11, p. 45181–45193, 2023.

MILLER, Catriona; PORTLOCK, Theo; NYAGA, Denis M.; SULLIVAN, Justin M.. A review of model evaluation metrics for machine learning in genetics and genomics. **Frontiers in Bioinformatics**, v. 4, p. 1457619, 2024. Disponível em: https://www.frontiersin.org/articles/10.3389/fbinf.2024.1457619/full.

MOHAMMED, Luay Thamer; ALHABSHY, AbdAllah A.; ELDAHSHAN, Kamal A. Big Data Visualization: A Survey. *In*: **2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)**. Ankara, Turkey: IEEE, 2022, p. 1–12.

NARECHANIA, Arpit; SRINIVASAN, Arjun; STASKO, John. NL4DV: A Toolkit for Generating Analytic Specifications for Data Visualization from Natural Language Queries. **IEEE Transactions on Visualization and Computer Graphics**, v. 27, n. 2, p. 369–379, 2021.

RAM, Gummuluri Venkata Ravi; ASHINEE, Kesanam; ANAND KUMAR, M. End-to-End Space-Efficient Pipeline for Natural Language Query based Spacecraft Health Data Analytics using Large Language Model (LLM). *In*: **2024 5th International Conference on Innovative Trends in Information Technology (ICITIIT)**. Kottayam, India: IEEE, 2024, p. 1–6.

RUNKLER, Thomas A. **Data analytics: models and algorithms for intelligent data analysis**. Third edition. Wiesbaden [Heidelberg]: Springer Vieweg, 2020. (Lehrbuch).

RUSSELL, Stuart J.; NORVIG, Peter; DAVIS, Ernest. **Artificial intelligence: a modern approach**. 3rd ed. Upper Saddle River: Prentice Hall, 2010. (Prentice Hall series in artificial intelligence).

SADIKU, Matthew; SHADARE, Adebowale; MUSA, Sarhan; AKUJUOBI, Cajetan; PERRY, Roy. **Data Visualization**. *International Journal of Engineering Research and Advanced Technology (IJERAT)*, v. 12, 2016. ISSN 2454-6135.

SARKER, Iqbal H. Data Science and Analytics: An Overview from Data-Driven Smart Computing, Decision-Making and Applications Perspective. **SN Computer Science**, v. 2, n. 5, p. 377, 2021.

SAH, Subham; MITRA, Rishab; NARECHANIA, Arpit; ENDERT, Alex; STASKO, John; DOU, Wenwen. Generating Analytic Specifications for Data Visualization from Natural Language Queries using Large Language Models. 2024.

SHEN, Leixian; SHEN, Enya; LUO, Yuyu; YANG, Xiaocong; HU, Xuming; ZHANG, Xiongshuai . Towards Natural Language Interfaces for Data Visualization: A Survey. **IEEE Transactions on Visualization and Computer Graphics**, v. 29, n. 6, p. 3121–3144, 2023.

SHI, Liang; TANG, Zhengju; ZHANG, Nan; ZHANG, Xiaotong; Yang, ZHI . A Survey on Employing Large Language Models for Text-to-SQL Tasks. 2024.

SHINDE, Pramila P.; SHAH, Seema. A Review of Machine Learning and Deep Learning Applications. *In*: **2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)**. Pune, India: IEEE, 2018, p. 1–6.

SUN, Kaili; LUO, Xudong; LUO, Michael Y. A Survey of Pretrained Language Models. *In*: MEMMI, Gerard; YANG, Baijian; KONG, Linghe; *et al* (Orgs.). **Knowledge Science, Engineering and Management**. Cham: Springer International Publishing, 2022, v. 13369, p. 442–456.

TORY, M.; MOLLER, T. Rethinking Visualization: A High-Level Taxonomy. *In*: **IEEE Symposium on Information Visualization**. Austin, TX, USA: IEEE, 2004, p. 151–158.

TUNSTALL, Lewis; WERRA, Leandro von; WOLF, Thomas. **Natural Language Processing with Transformers, Revised Edition**. [s.l.]: O'Reilly Media, Inc., 2022.

UNWIN, Anthony. Why is Data Visualization Important? What is Important in Data Visualization? **Harvard Data Science Review**, 2020.

VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, LUKASZ; POLOSUKHIN, Illia . **Attention Is All You Need**. arXiv.org.

WANG, Luping; CHEN, Sheng; JIANG, Linnan; PAN, Shu; CAI, Runze; YANG, Sen; YANG, Fei . Parameter-Efficient Fine-Tuning in Large Models: A Survey of Methodologies. 2024.

WANG, Shuo; CRESPO-QUINONES, Carlos. **Natural Language Models for Data Visualization Utilizing nvBench Dataset**. arXiv, 2023. Disponível em: https://arxiv.org/abs/2310.00832. Acesso em: 7 fev. 2026

WANG, Haifeng; LI, Jiwei; WU, Hua; HOVY, Eduardo; SUN, Yu . Pre-Trained Language Models and Their Applications. **Engineering**, v. 25, p. 51–65, 2023.

WANG, Bailin; SHIN, Richard; LIU, Xiaodong; POLOZOV, Oleksandr; RICHARDSON, Matthew. **RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers**. arXiv, 2019. Disponível em: https://arxiv.org/abs/1911.04942. Acesso em: 7 fev. 2026

WANG, Haifeng; WU, Hua; HE, Zhongjun; HUANG, Liang; CHURCH, Kenneth Ward. Progress in Machine Translation. **Engineering**, v. 18, p. 143–153, 2022.

WEST, Vivian L; BORLAND, David; HAMMOND, W Ed. Innovative information visualization of electronic health record data: a systematic review. **Journal of the American Medical Informatics Association**, v. 22, n. 2, p. 330–339, 2015.

WU, Aoyu; WANG, Yun; SHU, Xinhuan; MORITZ, Dominik; CUI, Weiwei; ZHANG, Haidong; ZHANG, Dongmei; QU, Huamin. AI4VIS: Survey on Artificial Intelligence Approaches for Data Visualization. **IEEE Transactions on Visualization and Computer Graphics**, v. 28, n. 12, p. 5049–5070, 2022.

YANG, Hao; YANG, Zhaoyong; ZHAO, Ruyang; LI, Xiaoran; RAO, Gaoqi. The implementation solution for automatic visualization of tabular data in relational databases based on large language models. *In*: **2024 International Conference on Asian Language Processing (IALP)**. Hohhot, China: IEEE, 2024, p. 175–180.

YIN, Pengcheng; NEUBIG, Graham; YIH, Wen-tau; RIEDEL, Sebastian. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. 2020.

YU, Tao; ZHANG, Rui; YANG, Kai; YASUNAGA, Michihiro; WANG, Dongxu; LI, Zifan; MA, James; LI, Irene; YAO, Qingning; ROMAN, Shanelle; ZHANG, Zilin; RADEV, Dragomir. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. 2018.

ZHANG, Hanchong; CAO, Ruisheng; CHEN, Lu; XU, Hongshen; YU, Kai . ACT-SQL: In-Context Learning for Text-to-SQL with Automatically-Generated Chain-of-Thought. 2023.

ZHAO, Wayne Xin; ZHOU, Kun; LI, Junyi; TANG, Tianyi; WANG, Xiaolei; HOU, Yupeng; MIN, Yingqian; ZHANG, Beichen; ZHANG, Junjie; DONG, Zican; DU, Yifan; YANG, Chen; CHEN, Yushuo; CHEN, Zhipeng; JIANG, Jinhao; REN, Ruiyang; LI, Yifan; TANG, Xinyu; LIU, Zikang; LIU, Peiyu; NIE, Jian-Yun; WEN, Ji-Rong. A Survey of Large Language Models. 2023.

ZHONG, Victor; XIONG, Caiming; SOCHER, Richard. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. 2017.

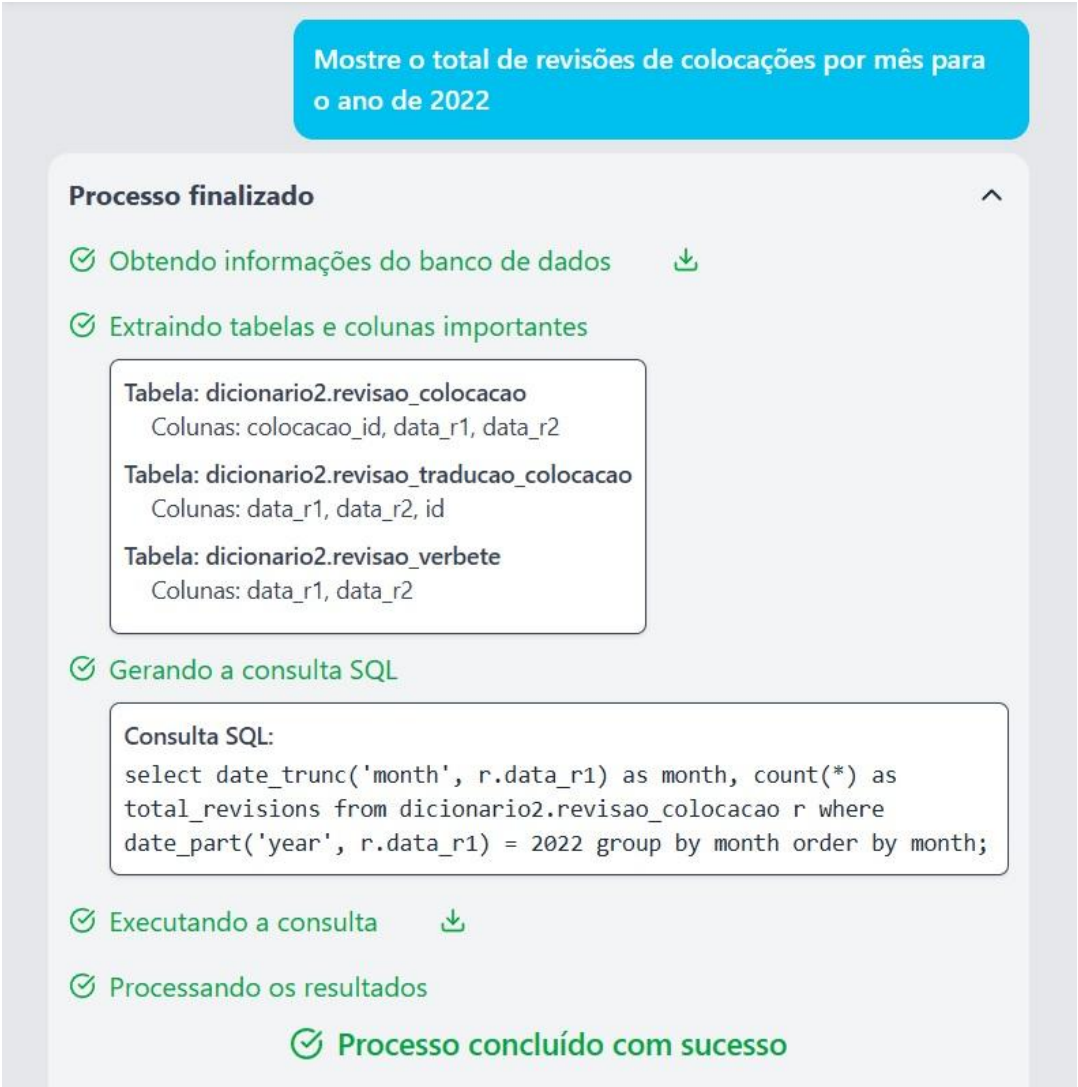
ZHU, Jun-Peng; CAI, Peng; NIU, Boyan; NI, Zheming; XU, Kai; HUANG, Jiajun; WAN, Jianwei; MA, Shengbo; WANG, Bing; ZHANG, Donghui; TANG, Liu; LIU, Qi. Chat2Query: A Zero-Shot Automatic Exploratory Data Analysis System with Large Language Models. *In*: **2024 IEEE 40th International Conference on Data Engineering (ICDE)**. Utrecht, Netherlands: IEEE, 2024, p. 5429–5432.

APÊNDICE A – EXEMPLOS DE VISUALIZAÇÕES DA INTERFACE DE LINGUAGEM NATURAL

O apêndice A foi criado com o objetivo de apresentar alguns exemplos de perguntas feitas na interface de linguagem, conectada ao banco de dados de dicionários multilíngues, com suas respectivas visualizações como saída. Nesse processo também são mostrados quais foram os esquemas relevantes obtidos pelo PLM e a geração do SQL pelo LLM.

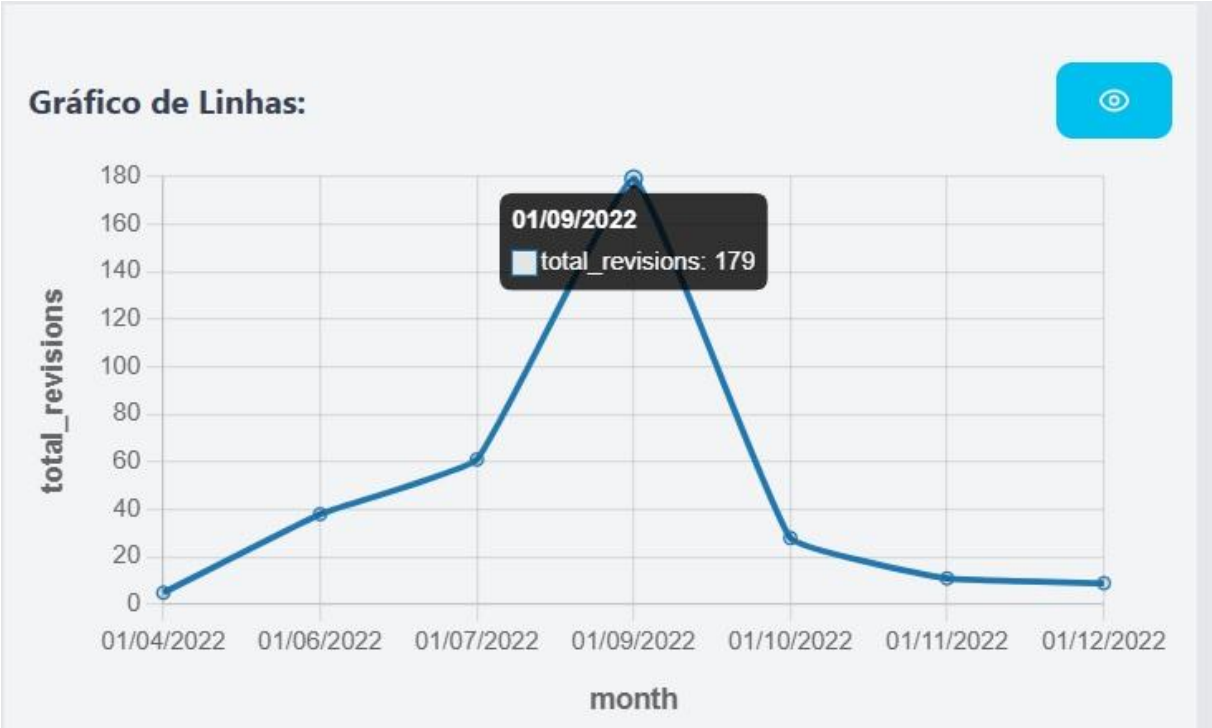
Dessa forma, na Figura A1 é exibido o primeiro exemplo para a pergunta "*Mostre o total de revisões de colocações por mês para o ano de 2022?*". Como resposta, obteve-se o gráfico de área e de linhas, exibidos nas Figuras A2 e A3, respectivamente.

Figura A 1 – Primeiro exemplo inserido na interface sobre datas



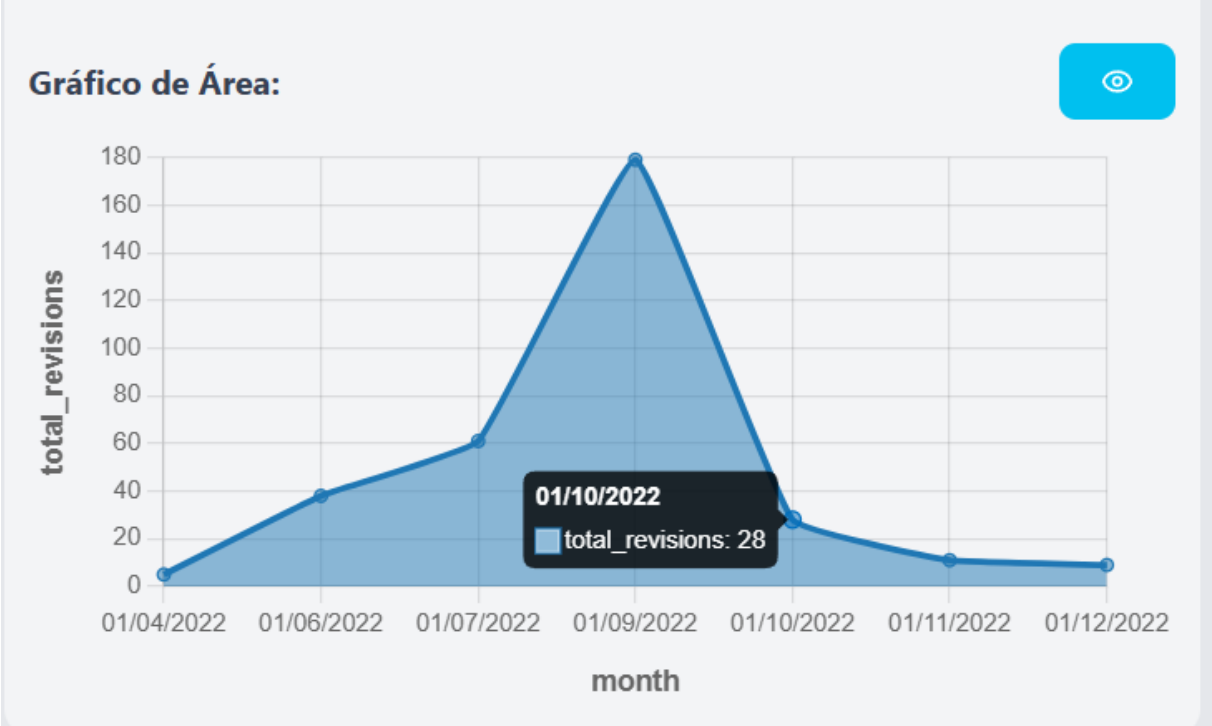
Fonte: Elaborado pelo autor.

Figura A 2 – Gráfico de linhas para o primeiro exemplo



Fonte: Elaborado pelo autor.

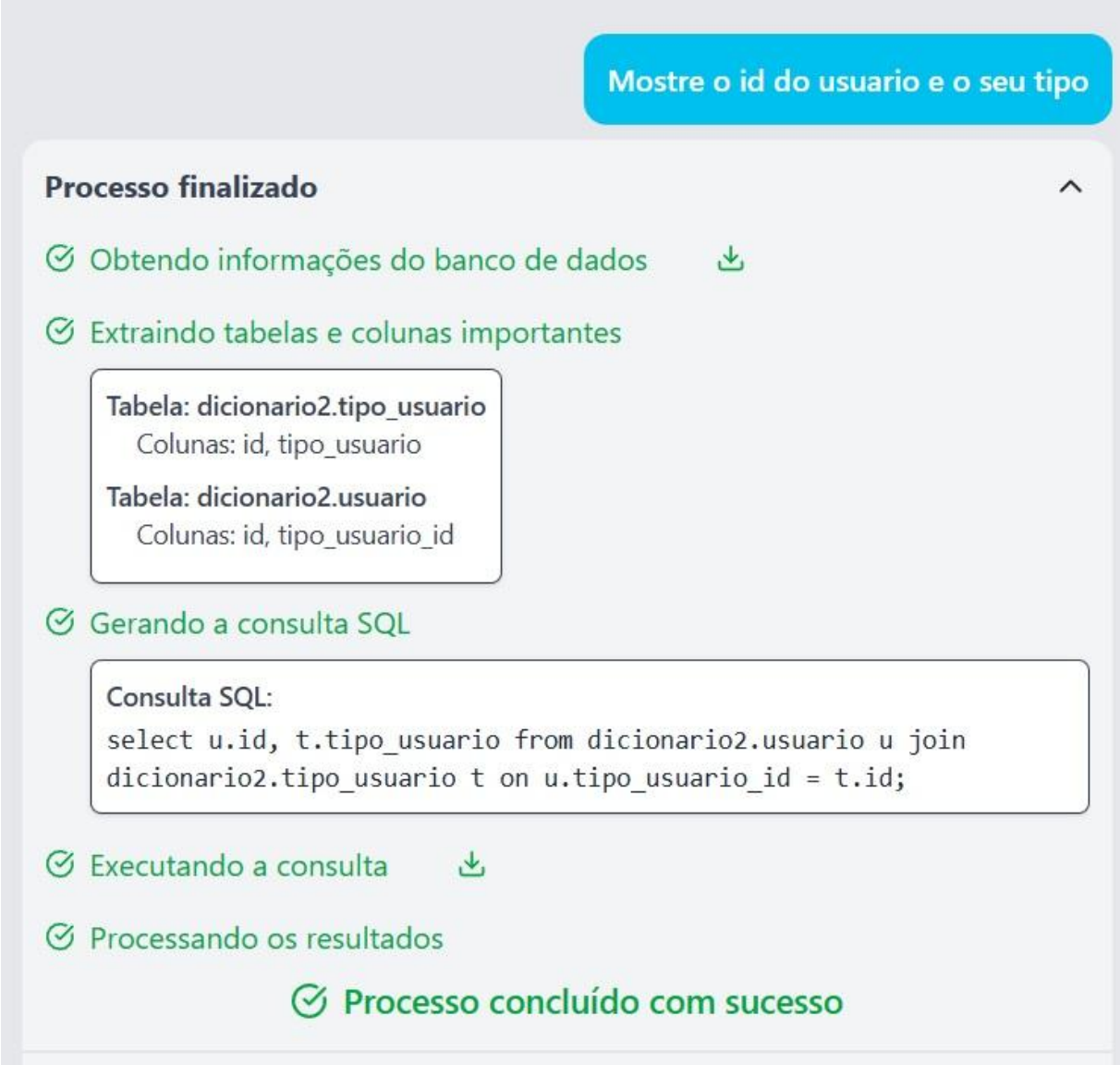
Figura A 3 – Gráfico de área para o primeiro exemplo



Fonte: Elaborado pelo autor.

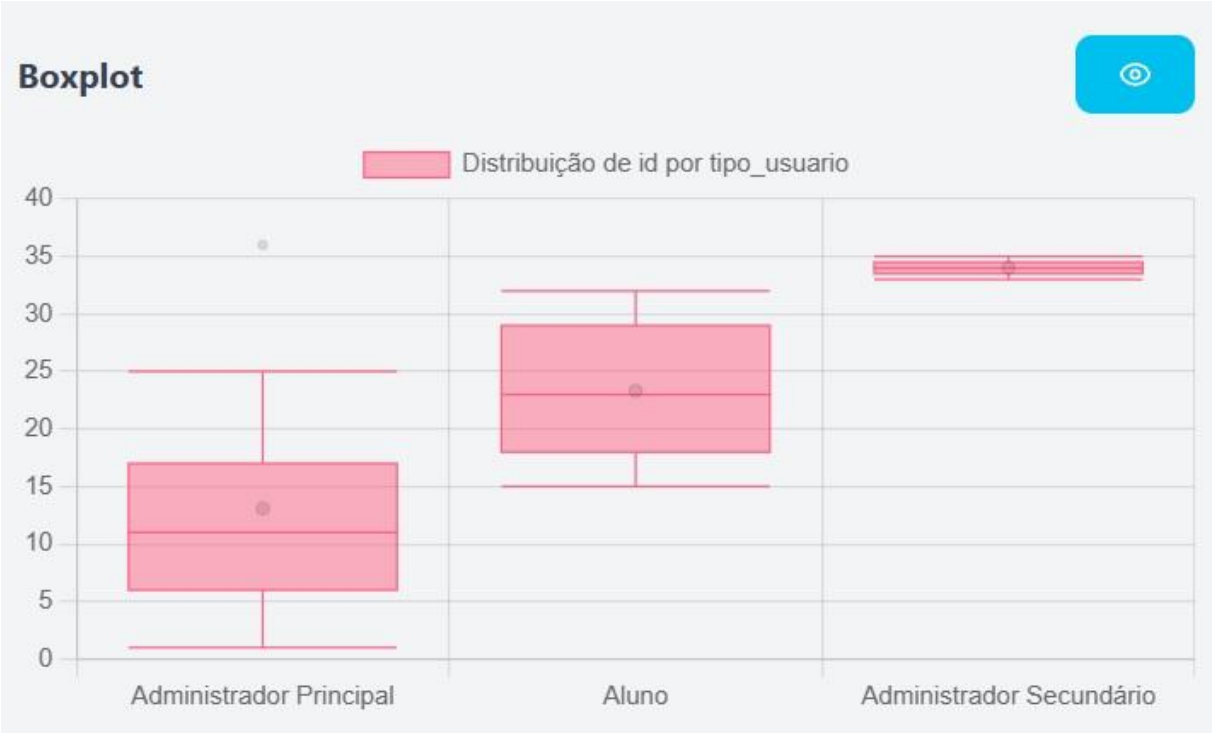
O segundo exemplo, detalhado na Figura A4, consiste na pergunta “*Mostre o id do usuário e seu tipo*”, em que foi retornado um boxplot e um gráfico de violino, ambos mostrados nas Figuras A5 e A6, respectivamente.

Figura A 4 – Segundo exemplo inserido na interface para uma coluna numérica e uma categórica



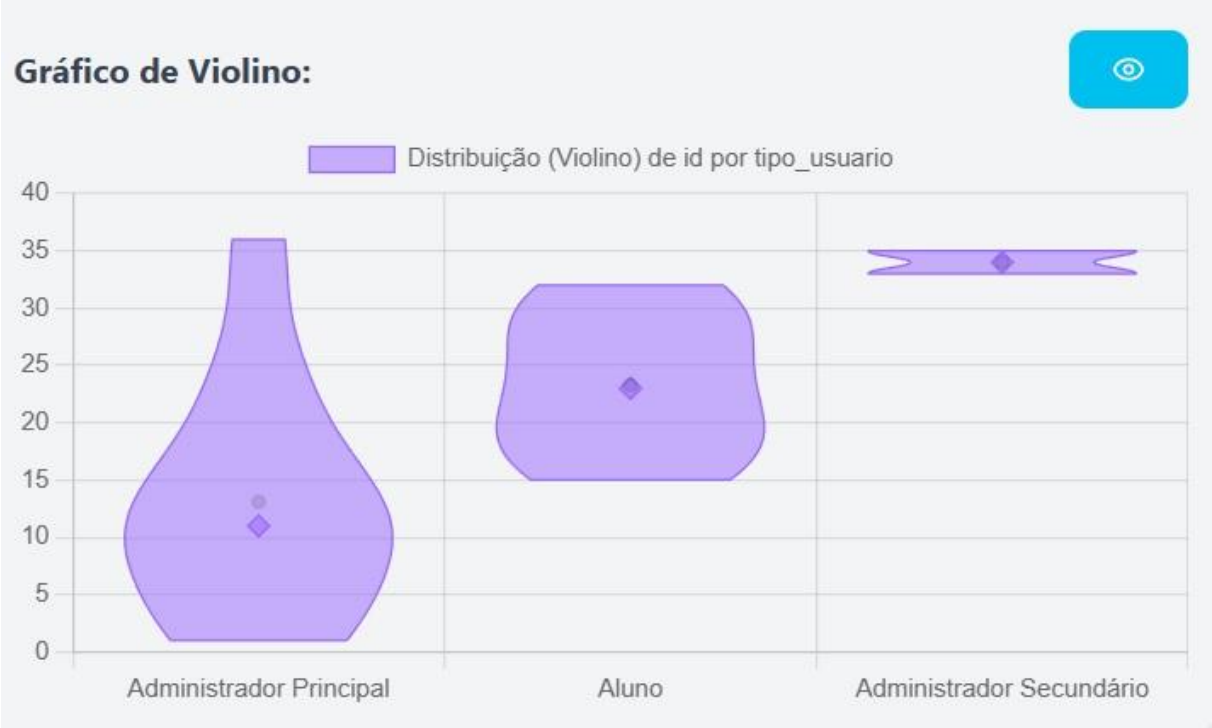
Fonte: Elaborado pelo autor.

Figura A 5 – Boxplot para o segundo exemplo



Fonte: Elaborado pelo autor.

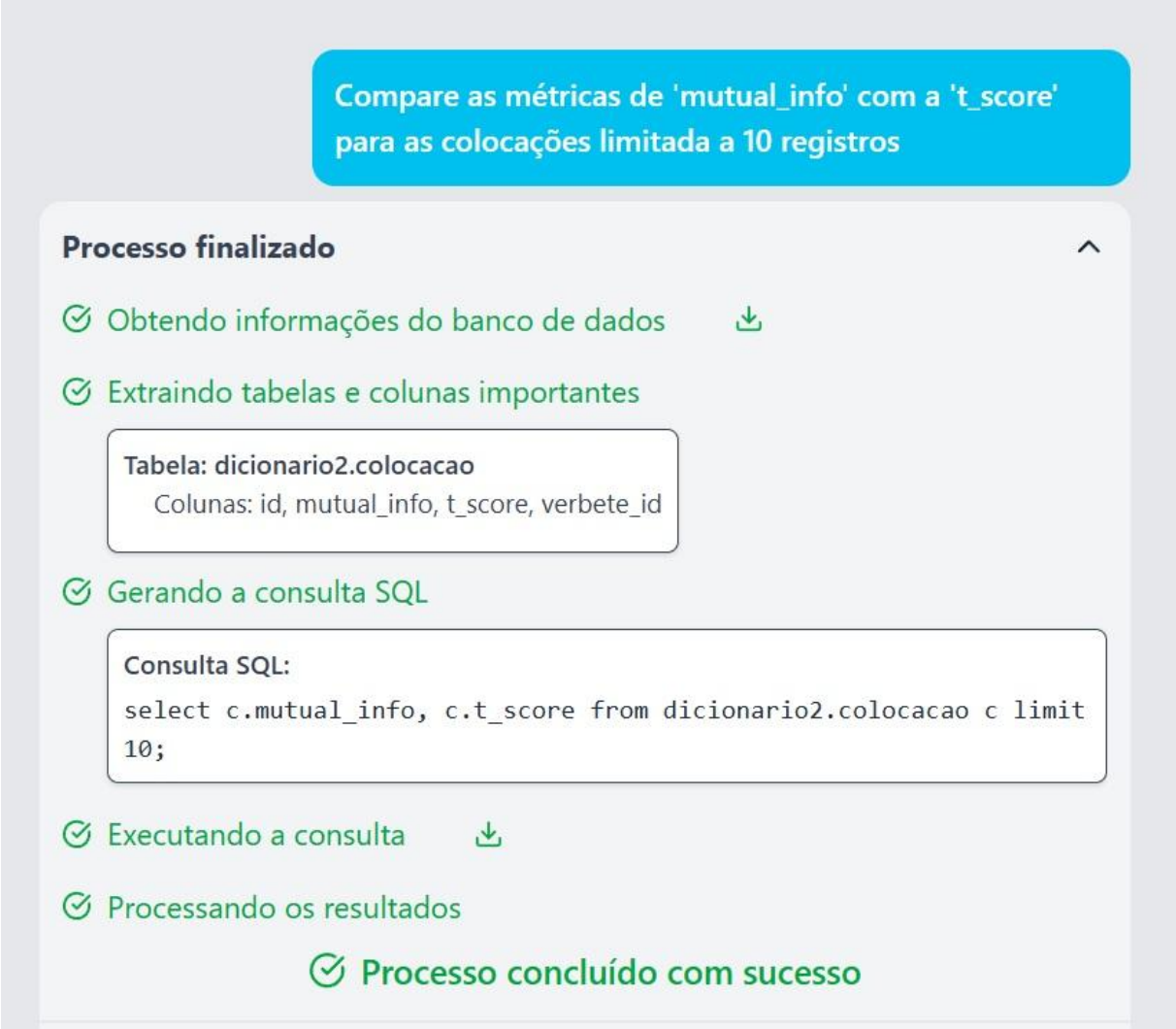
Figura A 6 – Gráfico de violino para o segundo exemplo



Fonte: Elaborado pelo autor.

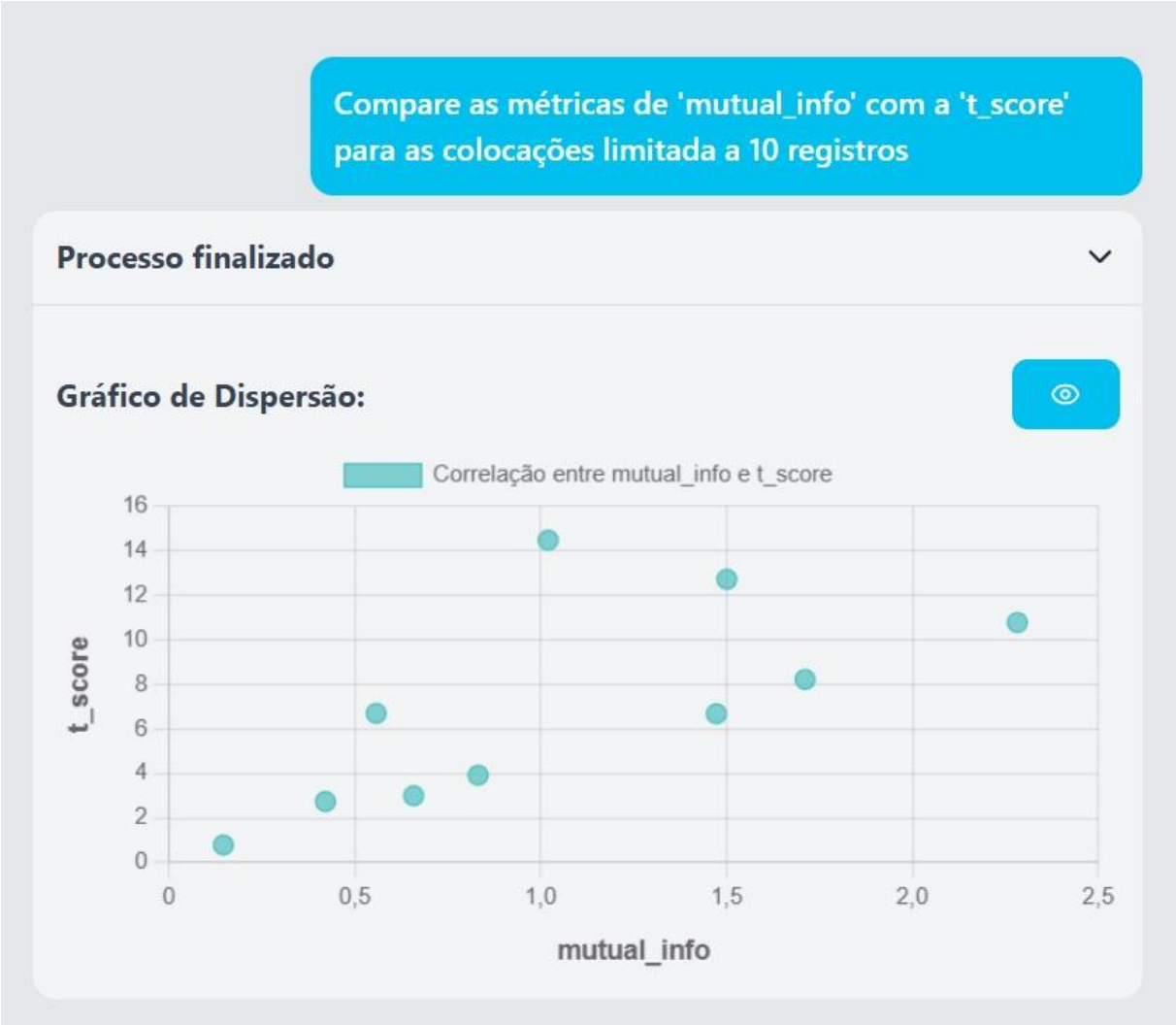
Para o terceiro exemplo, exibido na Figura A7, foi inserida a pergunta "*Compare as métricas de 'mutual_info' com a 't_score' para as colocações limitada a 10 registros*", a qual foi obtido como visualização um gráfico de dispersão, apresentado na Figura A8.

Figura A 7 – Terceiro exemplo para duas colunas numéricas sem ordenação



Fonte: Elaborado pelo autor.

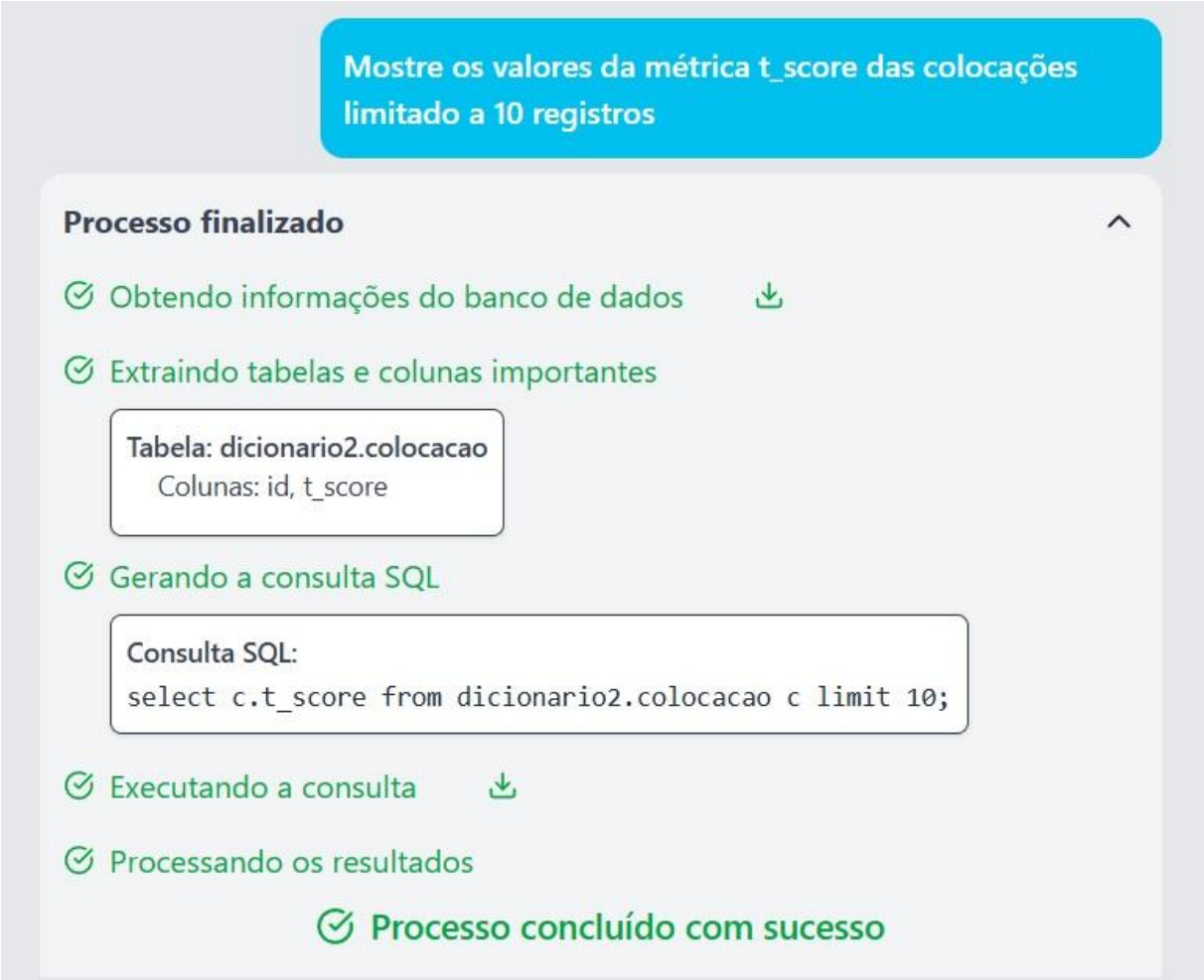
Figura A 8 – Gráfico de dispersão para o terceiro exemplo



Fonte: Elaborado pelo autor.

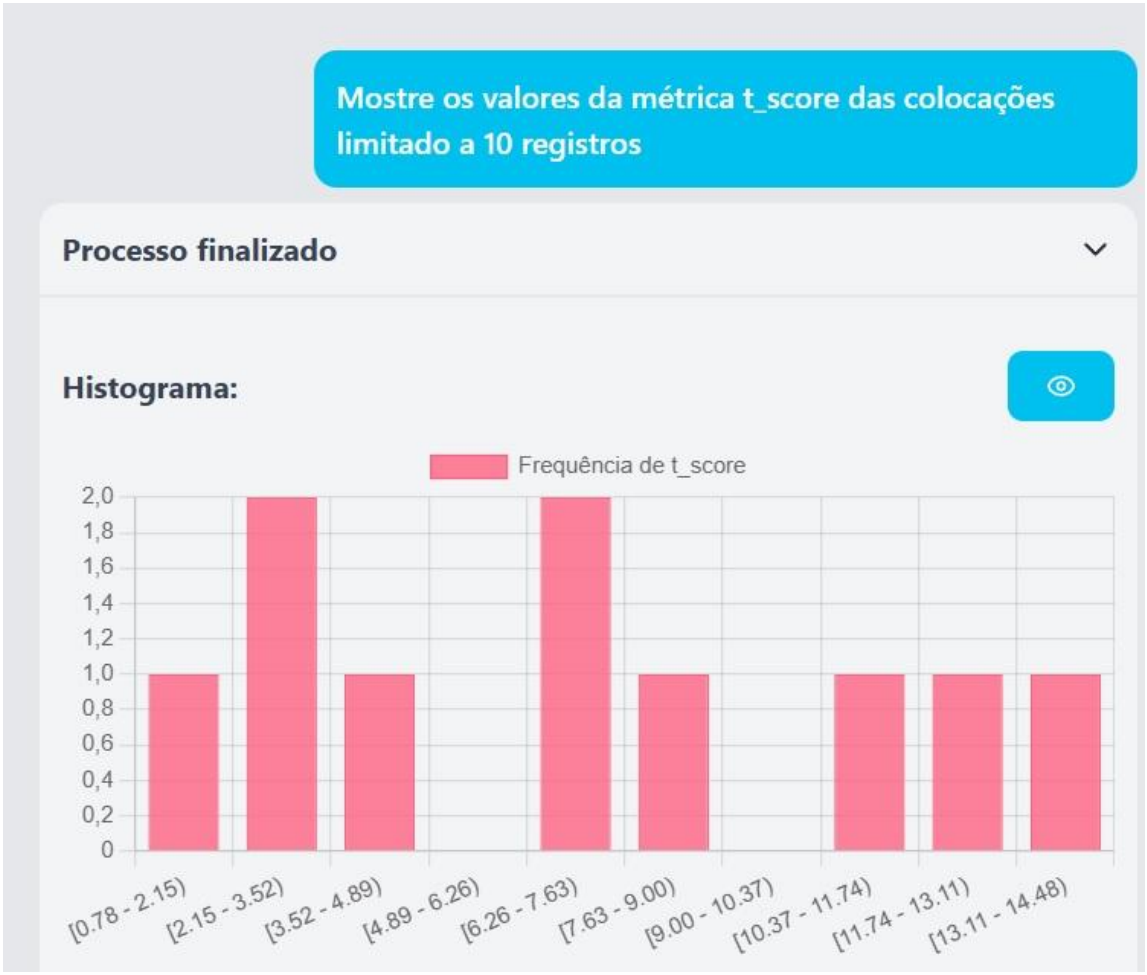
No quarto exemplo, detalhado na Figura A9, tem-se a questão "*Mostre os valores da métrica 't_score' limitado a 10 registros".* Como resposta, a interface retornou um histograma, um gráfico de densidade e um gráfico de violino, mostrados nas Figuras A10, A11 e A12, respectivamente.

Figura A 9 – Quarto exemplo para uma coluna numérica



Fonte: Elaborado pelo autor.

Figura A 10 – Histograma para o quarto exemplo



Fonte: Elaborado pelo autor.

Figura A 11 – Gráfico de densidade para o quarto exemplo

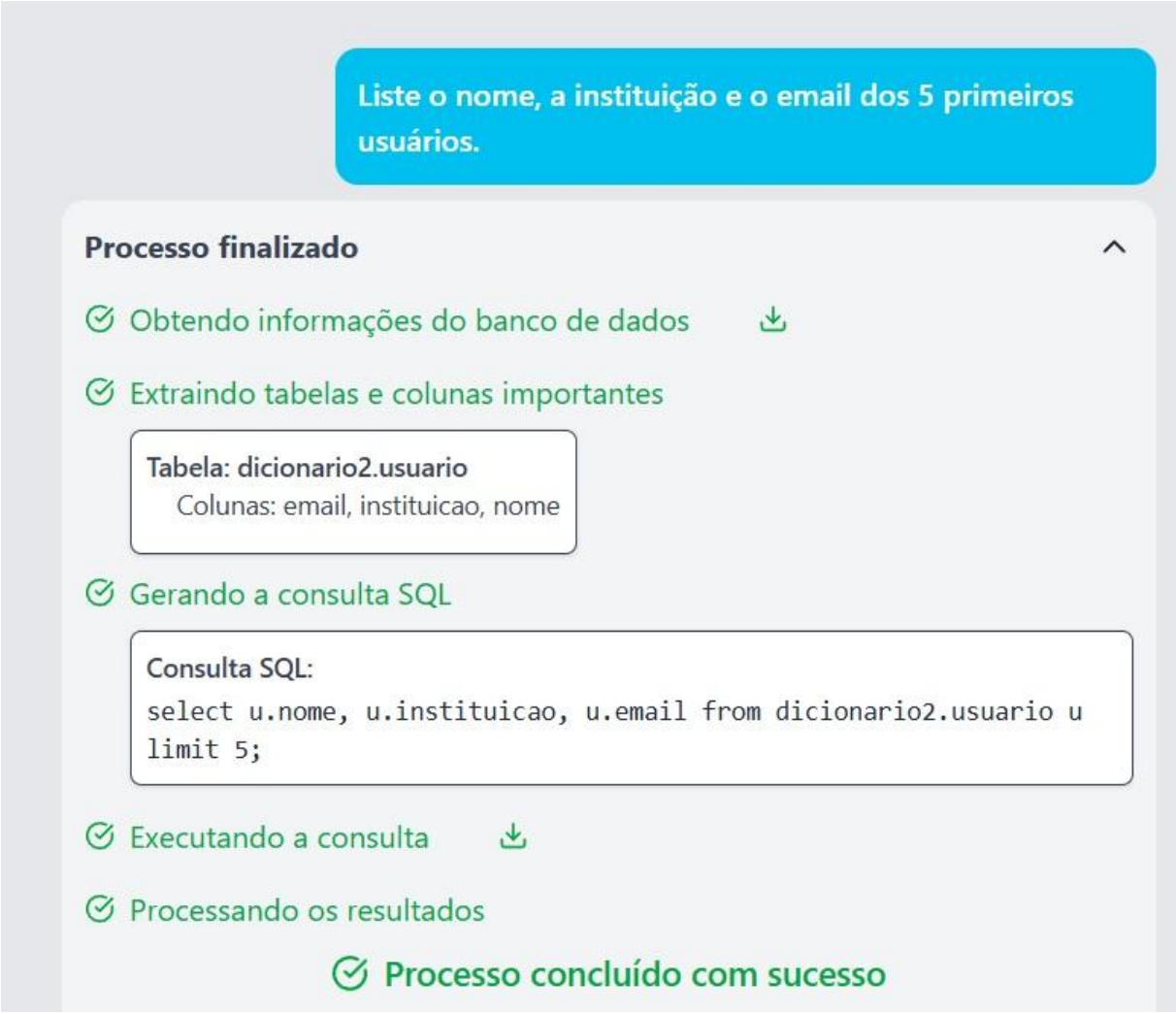
Fonte: Elaborado pelo autor.

Figura A 12 – Gráfico de violino para o quarto exemplo

Fonte: Elaborado pelo autor.

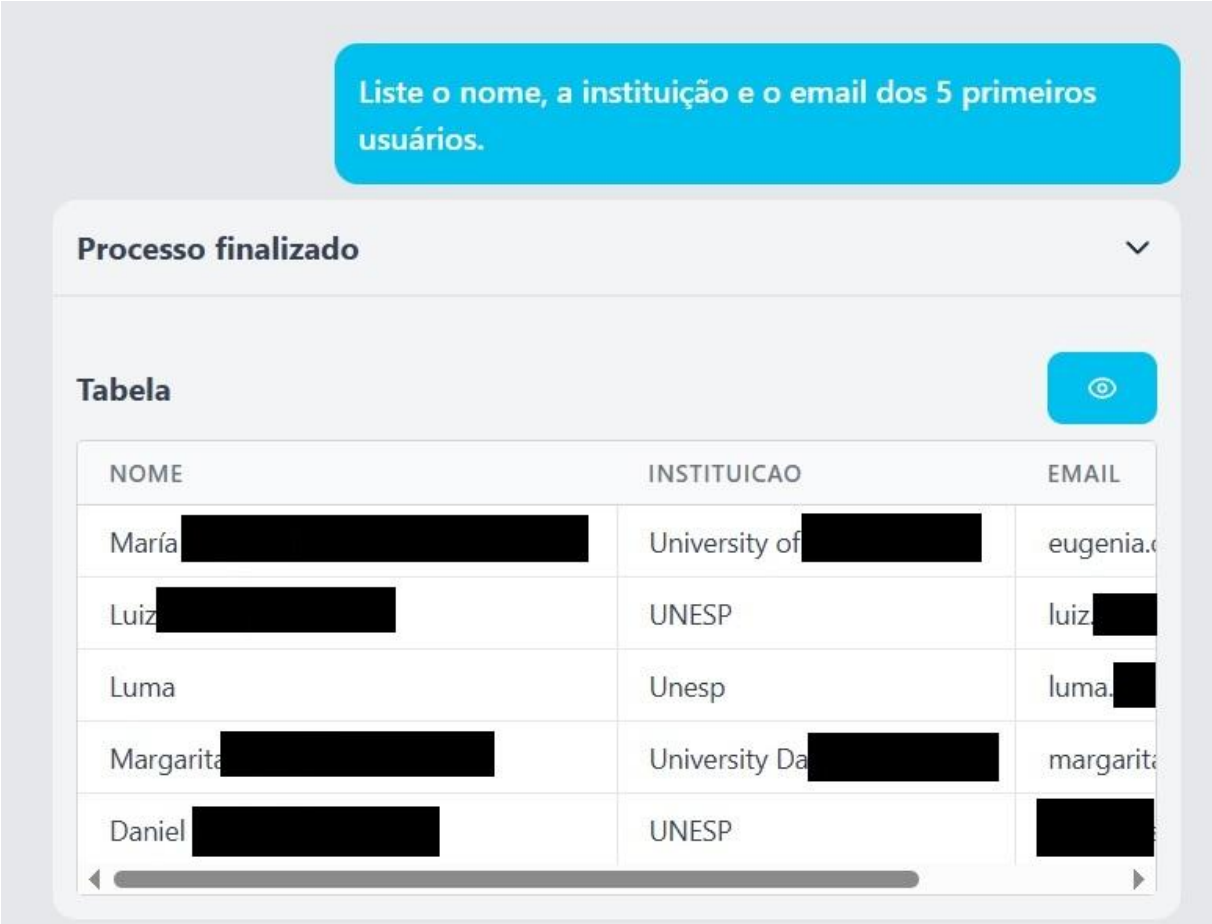
Para o quinto exemplo, a pergunta “*Liste o nome, a instituição e o email dos 5 primeiros usuários”* foi inserida na interface, ilustrada na Figura A13. Como retorno, foi construída uma tabela com 3 colunas, mostrada na Figura A14.

Figura A 13 – Quinto exemplo para três colunas categóricas



Fonte: Elaborado pelo autor.

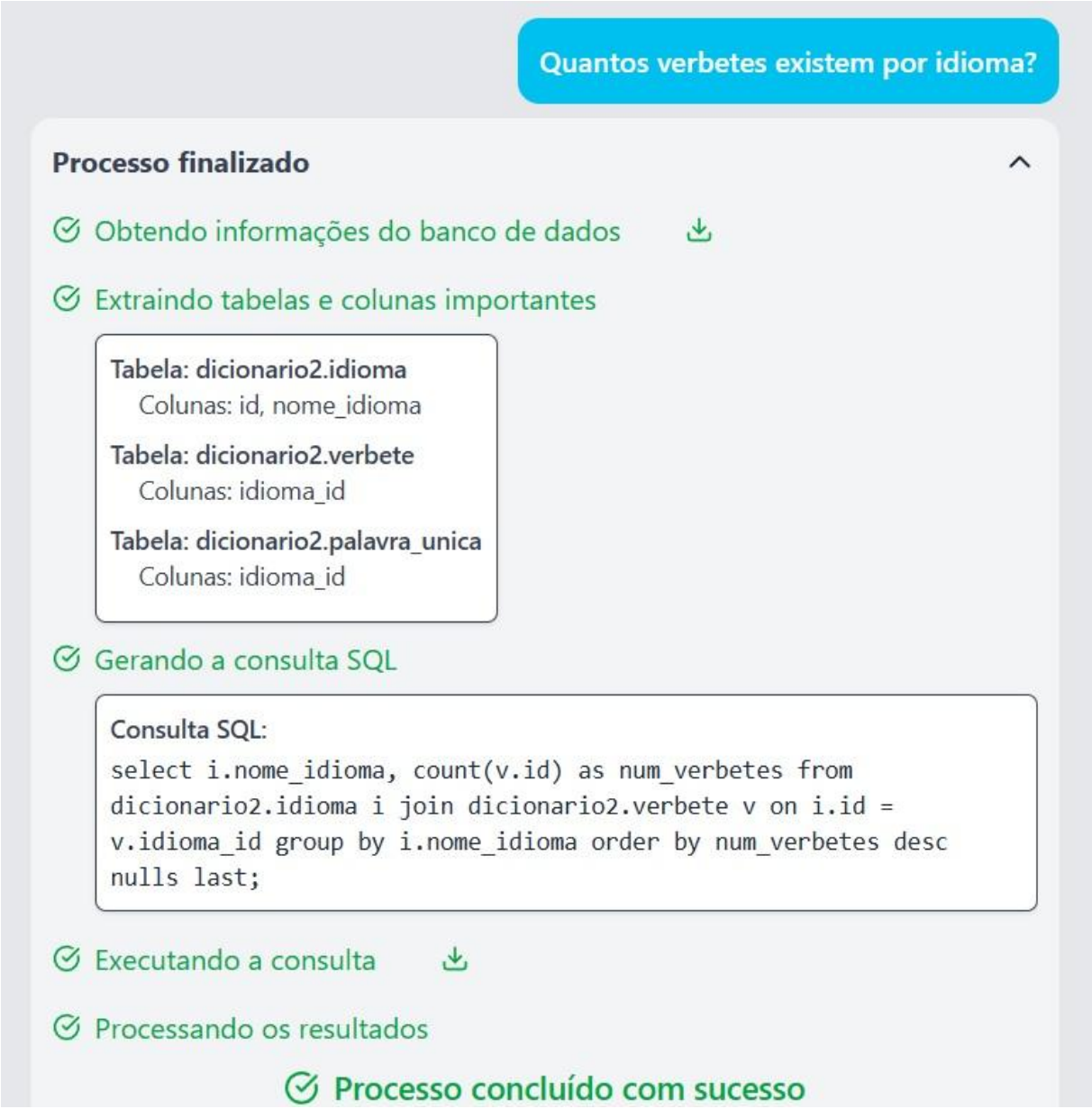
Figura A 14 – Tabela com três colunas para o quinto exemplo



Fonte: Elaborado pelo autor.

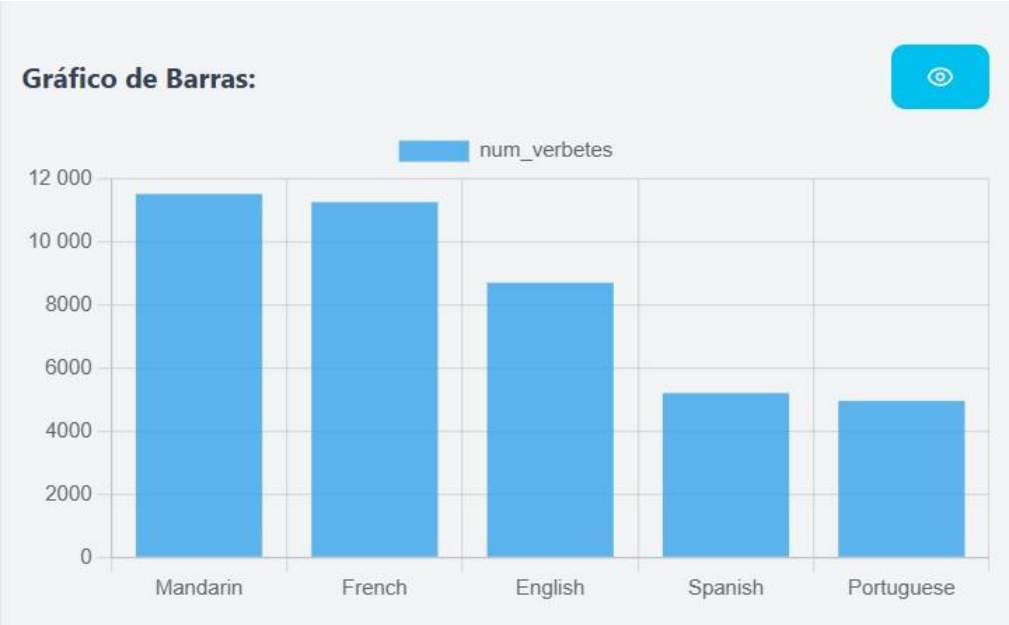
No sexto exemplo, a interface teve como entrada a questão "*Quantos verbetes existem por idioma?"*, apresentada na Figura A15. A saída obtida foi um gráfico de barras, de pizza e uma tabela, todas exibidas, respectivamente, nas Figuras A16, A17 e A18.

Figura A 15 – Sexto exemplo para uma coluna numérica e uma coluna categórica



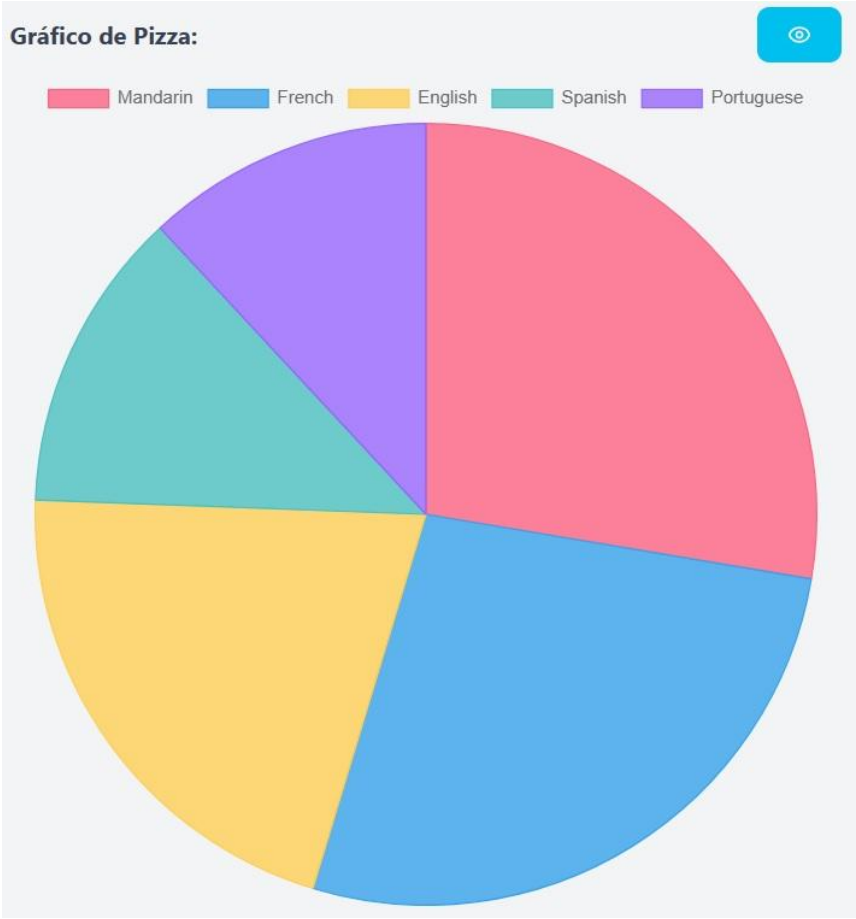
Fonte: Elaborado pelo autor.

Figura A 16 – Gráfico de barras para o sexto exemplo



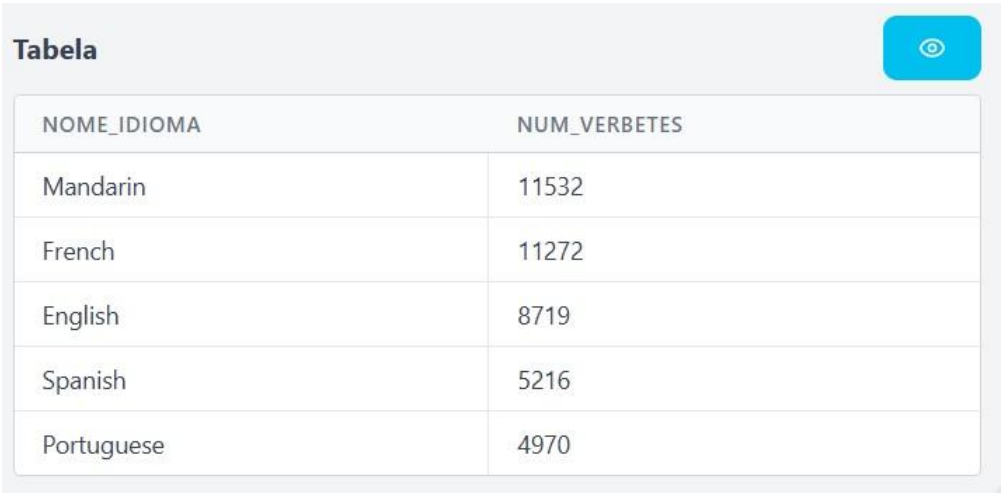
Fonte: Elaborado pelo autor.

Figura A 17 – Gráfico de pizza para o sexto exemplo



Fonte: Elaborado pelo autor.

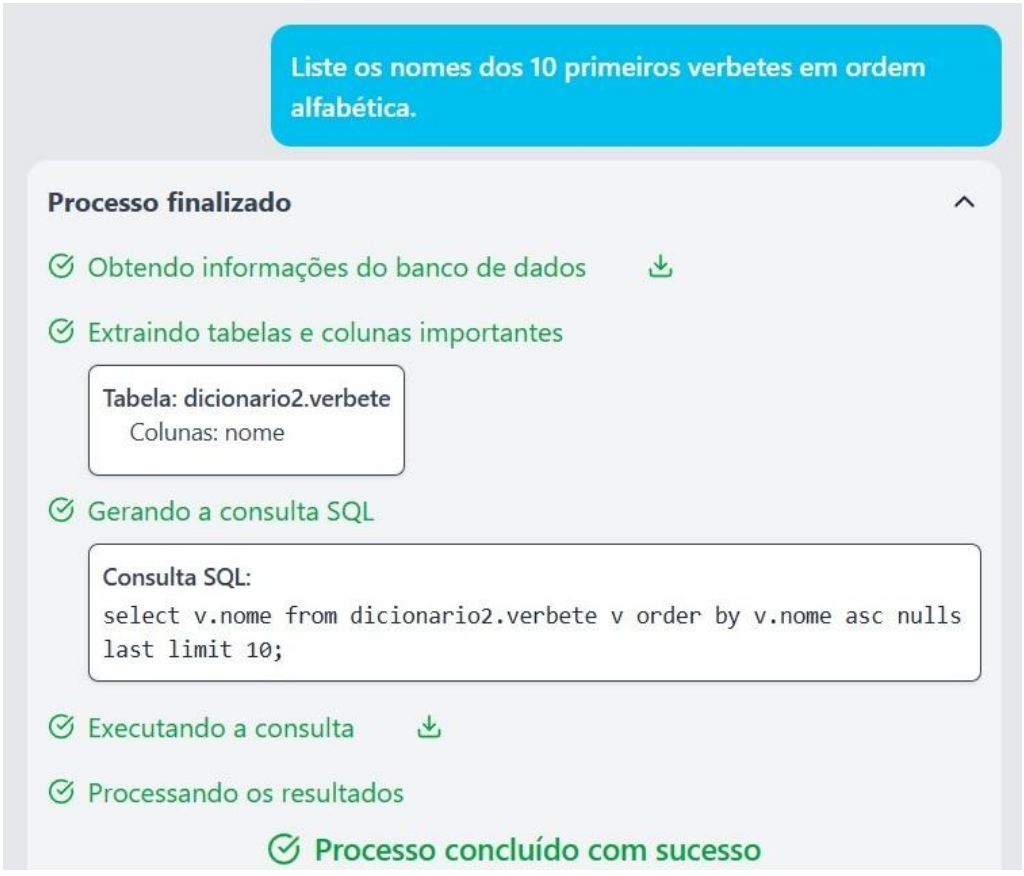
Figura A 18 – Tabela para o sexto exemplo



Fonte: Elaborado pelo autor.

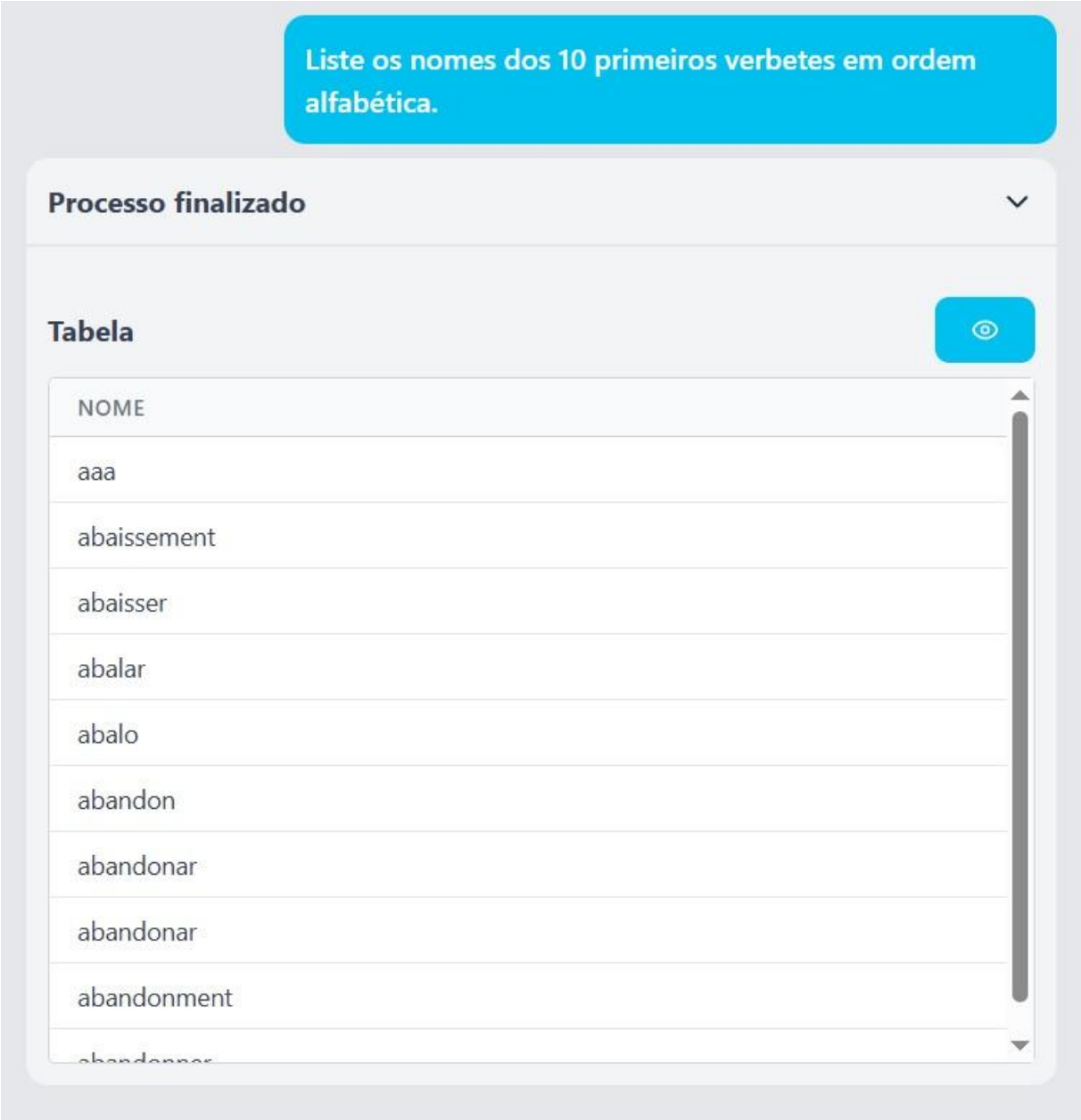
No sétimo exemplo, apresentado na Figura A19, foi elaborada a pergunta "*Liste os nomes dos 10 primeiros verbetes em ordem alfabética".* Com isso, a interface retornou uma tabela com uma única coluna, mostrada na Figura A20.

Figura A 19 – Sétimo exemplo para uma coluna categórica



Fonte: Elaborado pelo autor

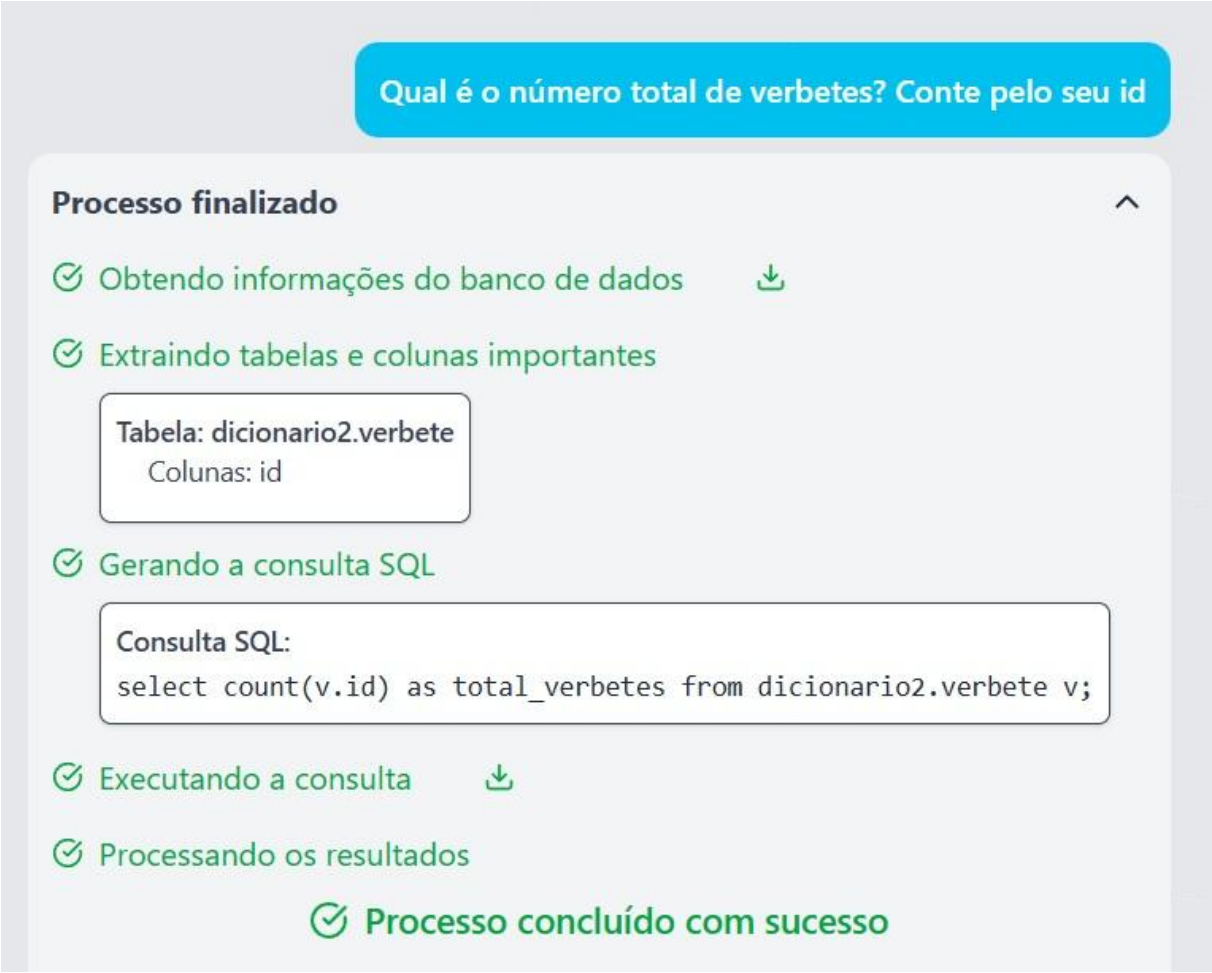
Figura A 20 – Tabela para o sétimo exemplo



Fonte: Elaborado pelo autor.

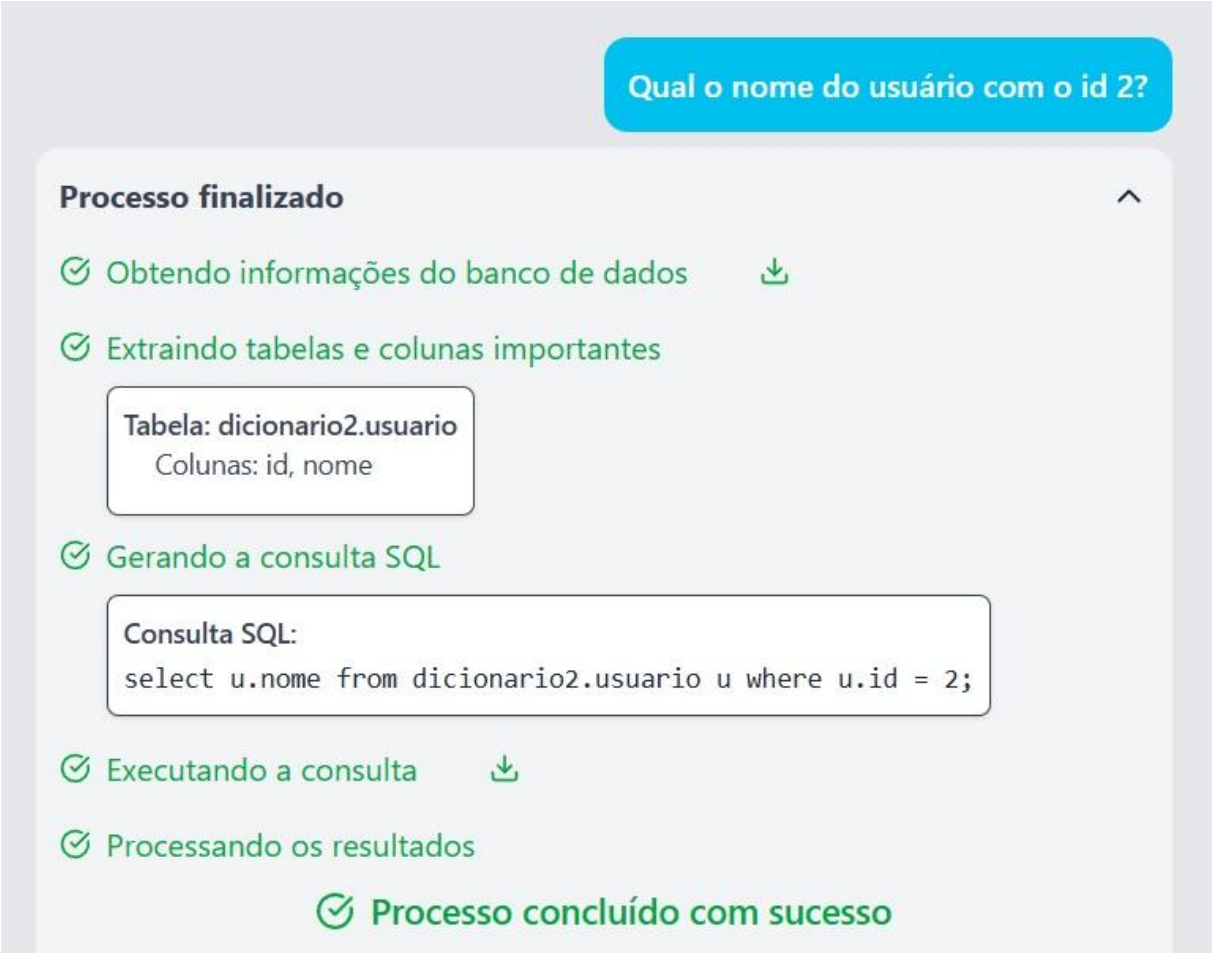
No oitavo e nono exemplo, foram inseridas as perguntas "*Qual é o número total de verbetes? Conte pelo seu id*" e "*Qual o nome do usuário com id 2?*", ambas ilustradas nas Figuras A21 e A22, respectivamente. Como retorno, foram obtidos valores únicos, apresentados na Figura A23 para o oitavo exemplo e na Figura A24 para o nono exemplo.

Figura A 21 – Oitavo exemplo para um valor único



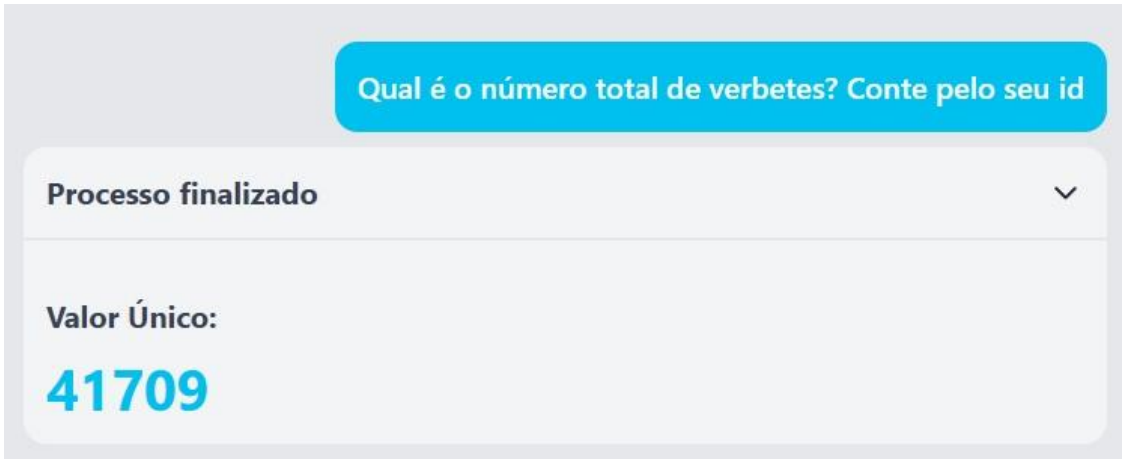
Fonte: Elaborado pelo autor.

Figura A 22 – Nono exemplo para um valor único



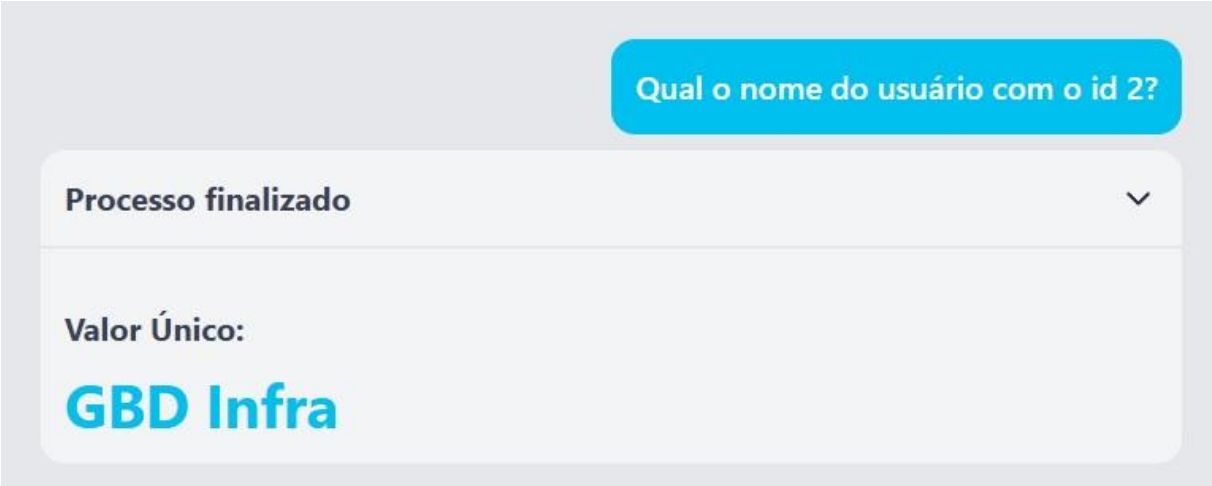
Fonte: Elaborado pelo autor.

Figura A 23 – Valor único gerado para o oitavo exemplo



Fonte: Elaborado pelo autor.

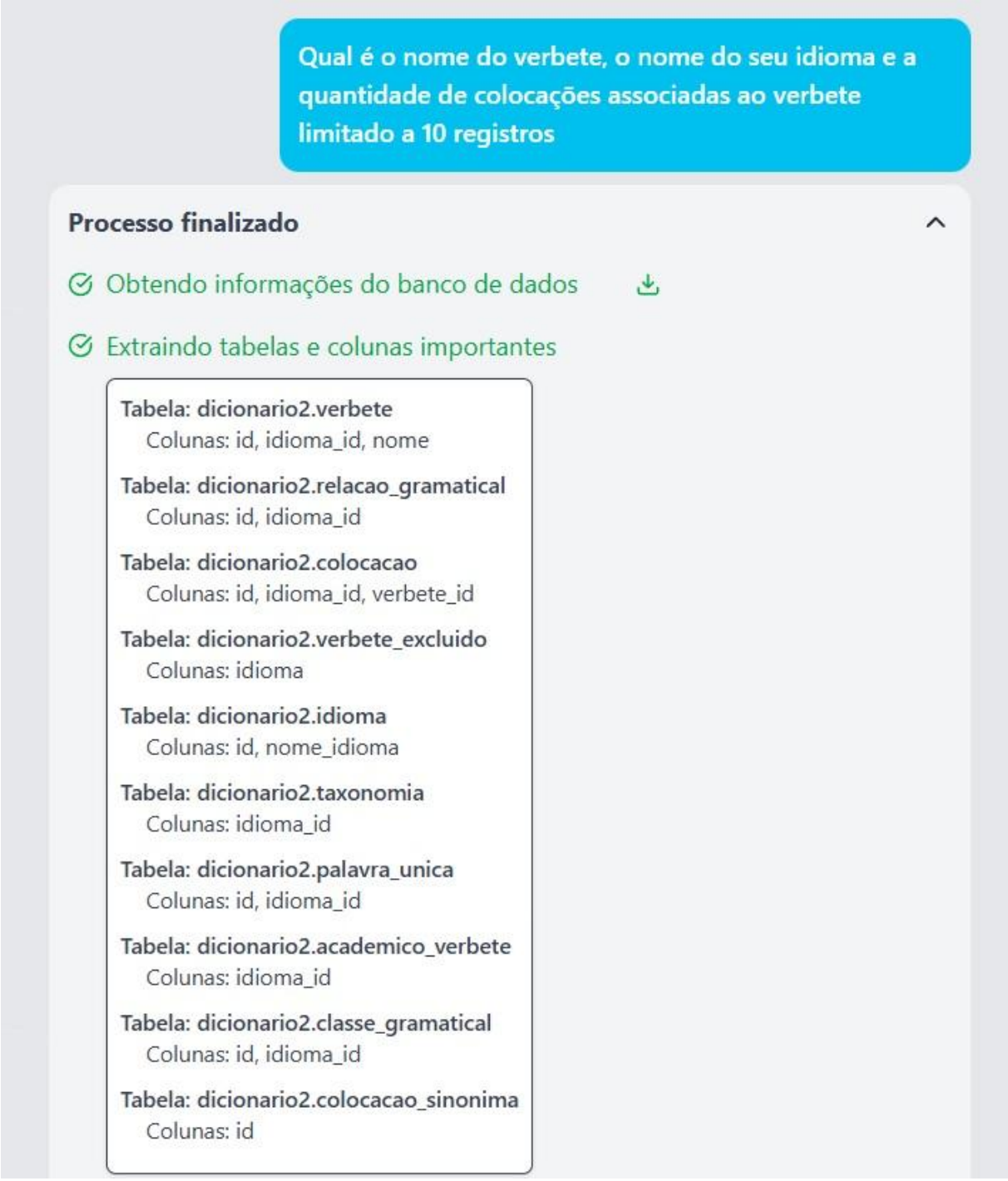
Figura A 24 – Valor único gerado para o nono exemplo



Fonte: Elaborado pelo autor.

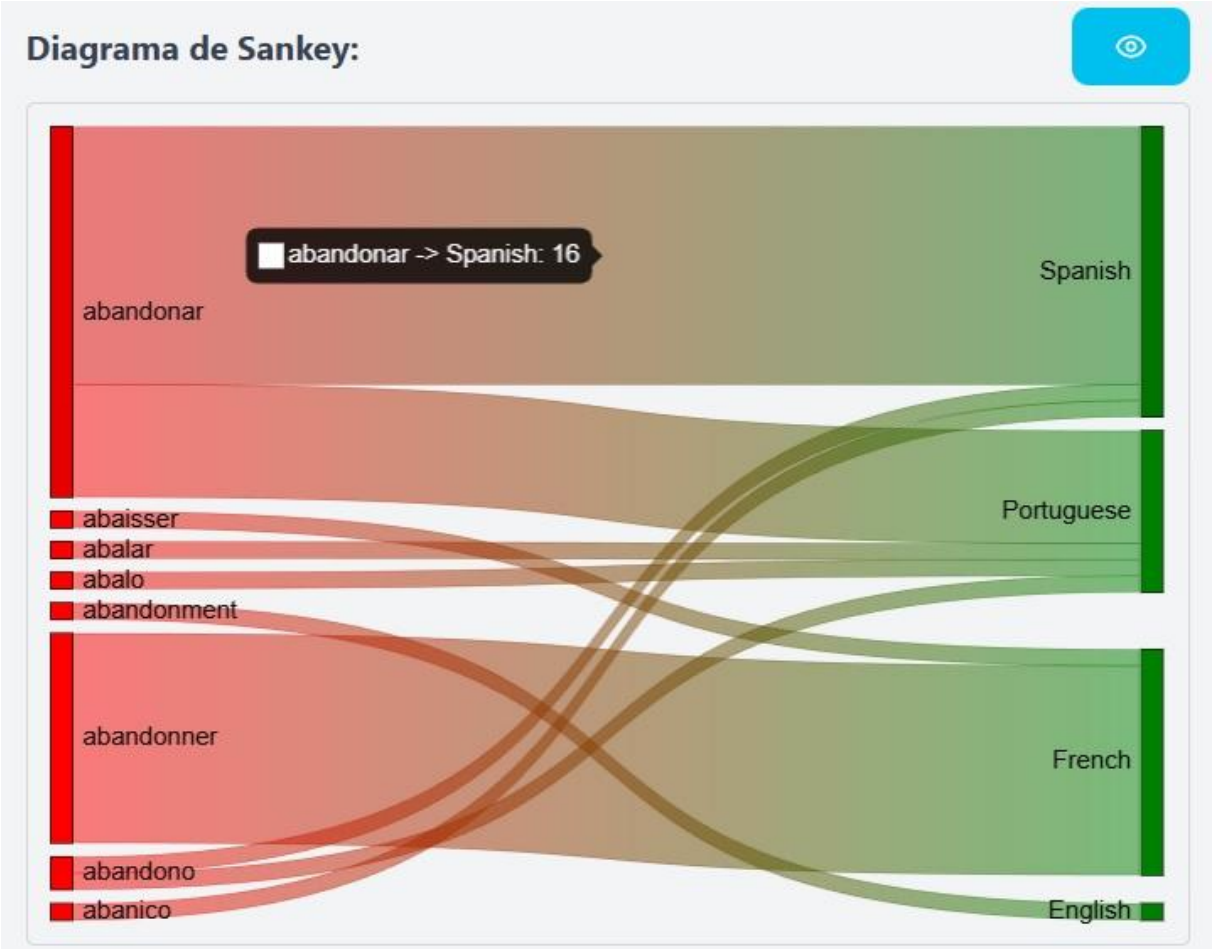
Para o décimo exemplo, com a pergunta "*Qual o nome do verbete, o nome do seu idioma e a quantidade de colocações associadas ao verbete limitado a 10 registros*", o diagrama de Sankey foi gerado como saída. Na Figura A25 é apresentado a pergunta citada juntamente com as tabelas/colunas extraídas e a consulta SQL e na Figura A26 a respectiva visualização.

Figura A 25 – Décimo exemplo para duas colunas categóricas e uma coluna numérica



Fonte: Elaborado pelo autor.

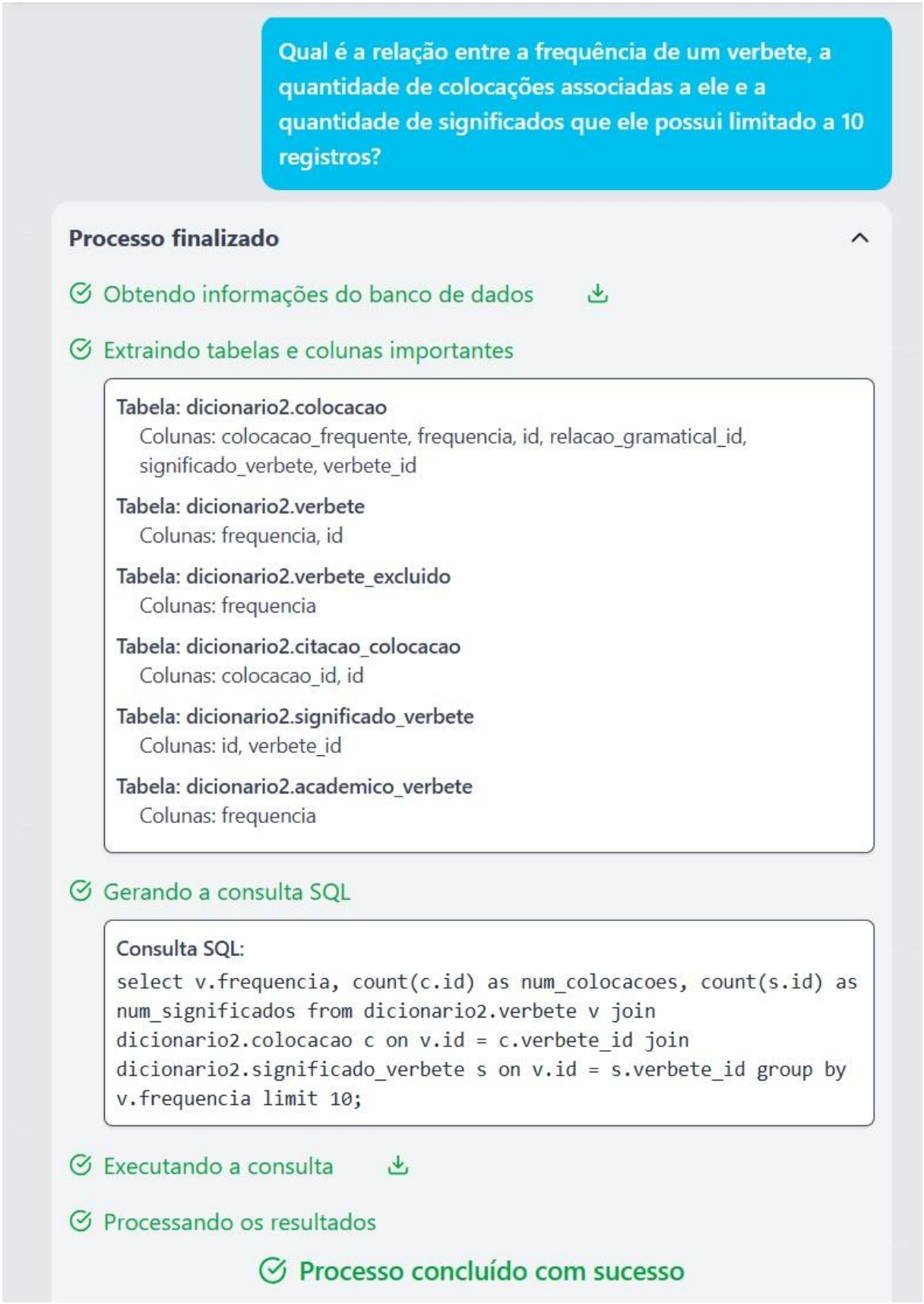
Figura A 26 – Diagrama de Sankey para o décimo exemplo



Fonte: Elaborado pelo autor.

No décimo primeiro exemplo, a inserção da pergunta "*Qual é a relação entre a frequência de um verbete, a quantidade de colocações associadas a ele e a quantidade de significados que ele possui limitado a 10 registros*", exibida na Figura A27, teve como resultado um gráfico de bolhas e uma tabela, ambos mostrados nas Figuras A28 e A29, respectivamente.

Figura A 27 – Décimo primeiro exemplo para três colunas numéricas



Fonte: Elaborado pelo autor.

Figura A 28 – Gráfico de bolhas para o décimo primeiro exemplo

Fonte: Elaborado pelo autor.

Figura A 29 – Tabela para o décimo primeiro exemplo

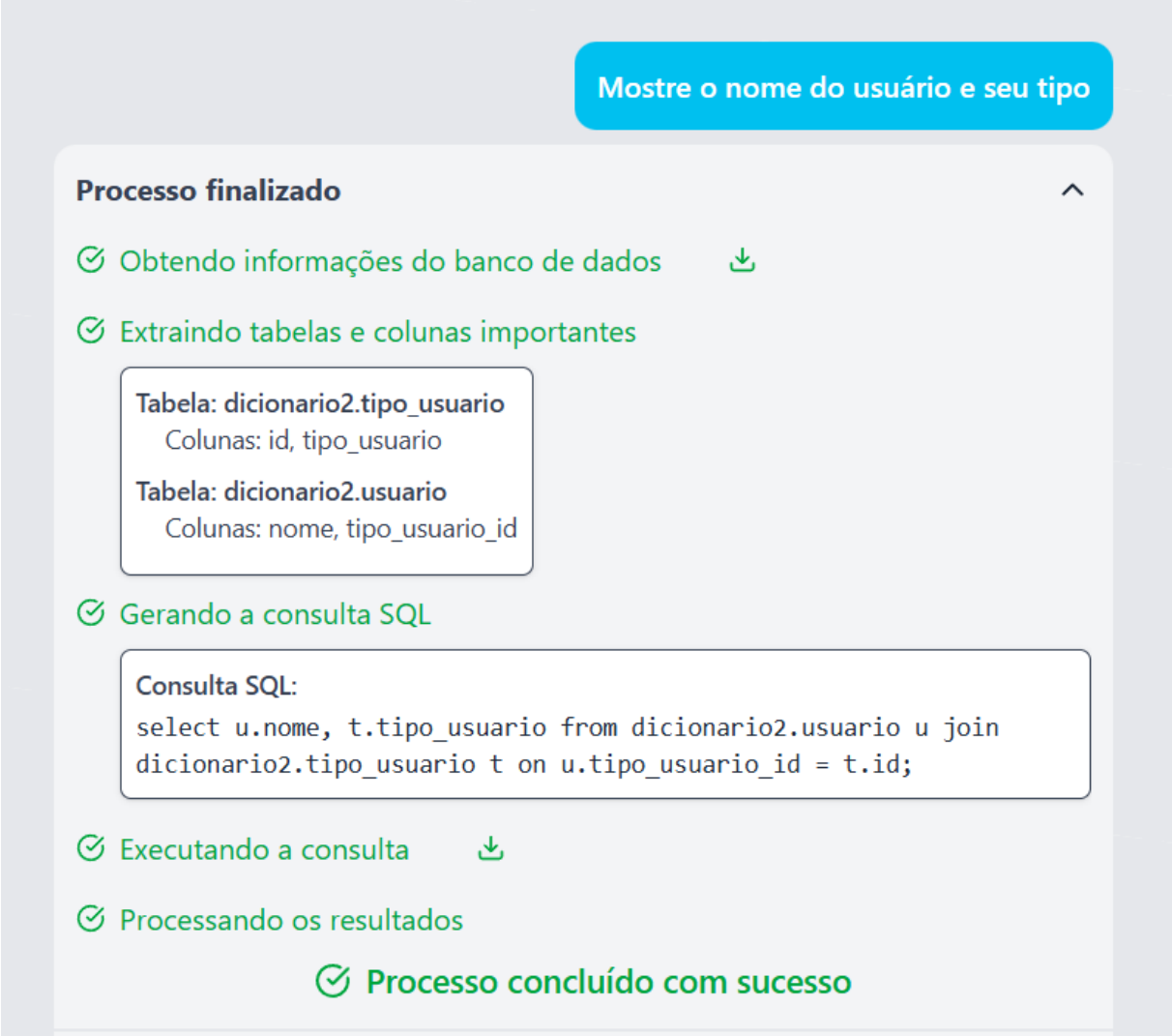
Tabela

FREQUENCIA	NUM_COLOCACOES	NUM_SIGNIFICADOS
10	604	604
100	241	241
1000	1	1
100002	12	12
10001	6	6
100016	16	16
10002	13	13
100022	19	19
100057	67	67

Fonte: Elaborado pelo autor.

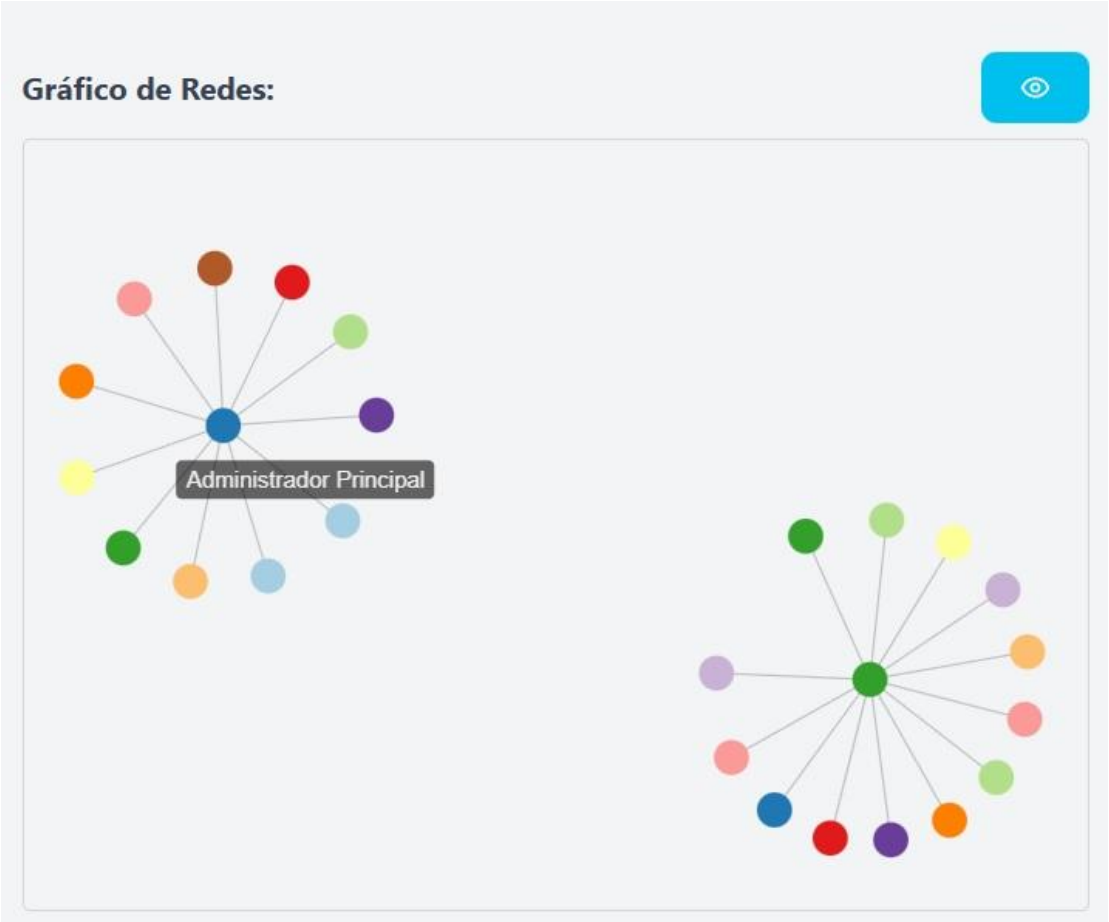
No décimo segundo exemplo, a pergunta “*Mostre o nome do usuário e seu tipo*”, apresentada na Figura A30, foi inserida na interface. Como resposta, foi gerado um gráfico de redes, mostrado na Figura A31.

Figura A 30 – Décimo segundo exemplo para duas colunas categóricas



Fonte: Elaborado pelo autor.

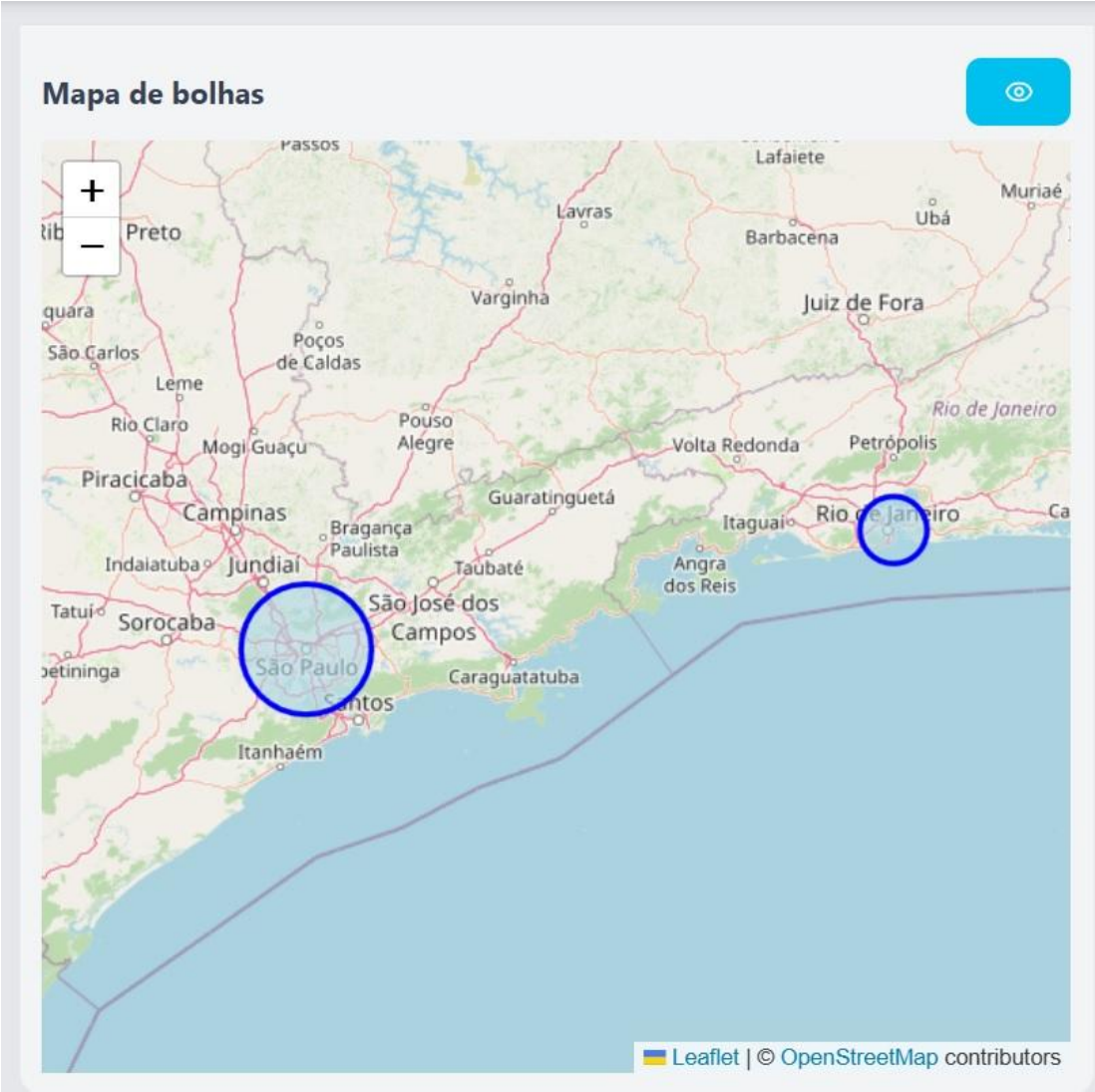
Figura A 31 – Gráfico de redes para o décimo segundo exemplo



Fonte: Elaborado pelo autor.

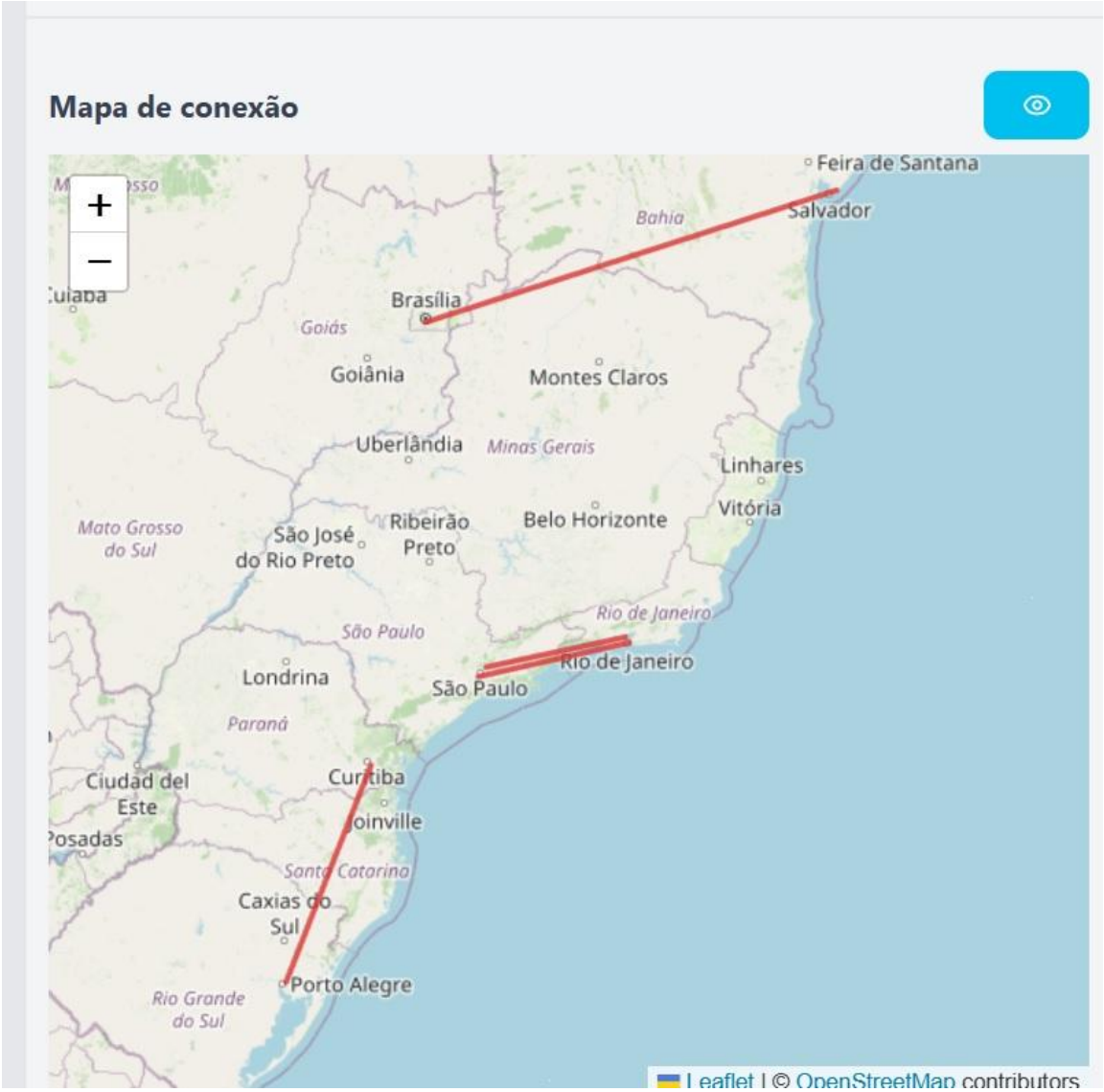
Por último, pelo fato de o banco de dados conectado a interface não possuir dados geográficos, uma simulação foi realizada, com dados fictícios, para demonstrar como ficariam os mapas no sistema. Dessa forma, nas Figuras A32, A33 e A34 são exibidos, respectivamente, o mapa de bolhas, o mapa de conexão e o mapa de pontos.

Figura A 32 – Simulação para o mapa de bolhas



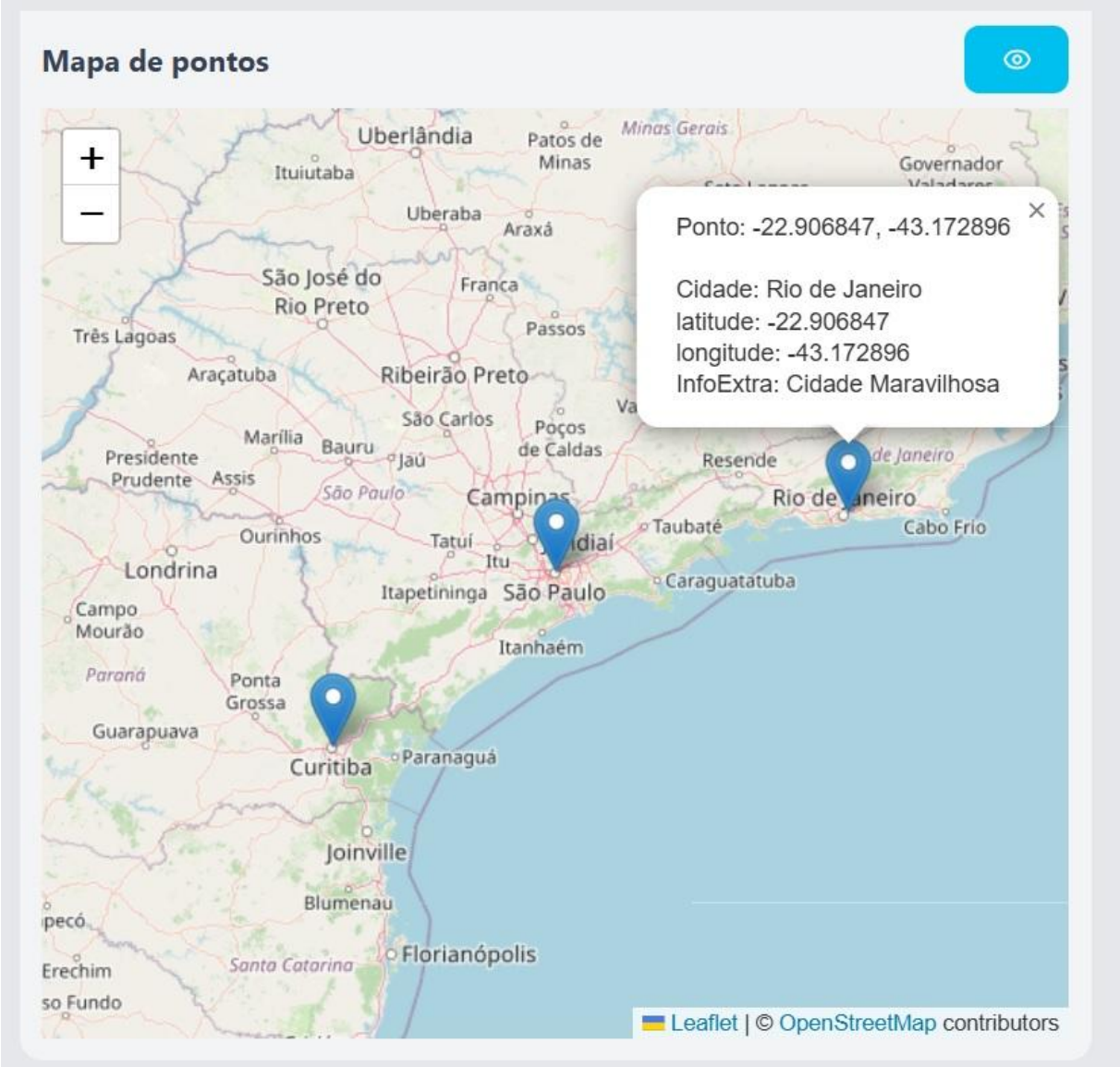
Fonte: Elaborado pelo autor.

Figura A 33 – Simulação para o mapa de conexão



Fonte: Elaborado pelo autor.

Figura A 34 – Simulação para o mapa de pontos



Fonte: Elaborado pelo autor.

DADOS CURRICULARES

IDENTIFICAÇÃO	
	João Pedro Quadrado Data de nascimento: 19/06/1999
Nacionalidade	Brasileiro
Nome em citações bibliográficas	QUADRADO, JOÃO PEDRO QUADRADO, J. P.
Currículo Lattes	http://lattes.cnpq.br/6105019807025846
ORCID	https://orcid.org/0009-0003-5186-7974
FORMAÇÃO ACADÊMICA	
2018/2022	Bacharel em Ciência da Computação Universidade Estadual Paulista “Júlio de Mesquita Filho”
2024/2026	Mestre em Ciência da Computação Universidade Estadual Paulista “Júlio de Mesquita Filho”
PRODUÇÃO BIBLIOGRÁFICA	
QUADRADO, JOÃO PEDRO; VALÊNCIO, CARLOS ROBERTO; OTTAIANO, ADRIANE ORENHA; ZAFALON, GERALDO FRANCISCO DONEGÁ; MORAES SILVA, LUÍS MARCELLO; COLOMBINI, ANGELO CESAR. Generation of New Bilingual Dictionaries Through Inference Techniques Supported by a Graph Oriented Database Management System. In: 2023 IEEE International Conference on Knowledge Graph (ICKG), 2023, Shanghai. 2023 IEEE International Conference on Knowledge Graph (ICKG), 2023. p. 19.	
VALÊNCIO, C.R. ; ORENHA OTTAIANO, ADRIANE ; QUADRADO, J. P. . Research results and outcomes of the project 'A Phraseographical Methodology and Model for an Online Corpus-Based Multilingual Collocations Dictionary Platform.. In: Electronic lexicographyin the 21st century (eLex 2023), 2023. Electronic lexicographyin the 21st century (eLex 2023). p. 143.	
VALÊNCIO, C.R. ; ORENHA OTTAIANO, ADRIANE ; QUADRADO, J. P. ; KUHN, T. Z. . Development of a methodology and enhancements of lexicographical resources for an online Platform of Academic Collocations Dictionaries in Portuguese and English.. In: Electronic lexicographyin the 21st century (eLex 2023), 2023. Electronic lexicographyin the 21st century (eLex 2023). p. 62.	
PARTICIPAÇÃO EM BANCAS E ORIENTAÇÕES	
Bancas de trabalhos de conclusão	
Orientações	
PARTICIPAÇÃO EM EVENTOS CIENTÍFICOS	