Universidade de São Paulo
Escola Politécnica
Programa de Educação Continuada em Engenharia
Especialização em Inteligência Artificial

Rogério Piazzon Teixeira Júnior

# O uso de arquiteturas híbridas na tarefa text-to-SQL

São Paulo
2024

ROGÉRIO PIAZZON TEIXEIRA JÚNIOR

# O uso de arquiteturas híbridas na tarefa text-to-SQL

— Versão Original —

Monografia apresentada ao Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de Especialização em Inteligência Artificial.

Orientador: Prof. Dr. Marcos Lopes

São Paulo
2024

Catalogação-na-publicação

Teixeira J$^{r}$, Rogério Piazzon Teixeira
O uso de arquiteturas híbridas na tarefa text-to-SQL/ R.Teixeira J$^{r}$ – São Paulo, 2024.
150p.

Monografia (Especialização em Inteligência Artificial) – Escola Politécnica da Universidade de São Paulo. PECE – Programa de Educação Continuada em Engenharia.

1. Processamento de Linguagem Natural (PLN). 2. text-to-SQL 3. Reconhecimento de Intenções 4. Transformers codificadores 5. Grandes Modelos de Linguagem (LLM) 6. Modelos gerativos 7. Modelos híbridos 8. SQL.
I. Universidade de São Paulo. Escola Politécnica. PECE – Programa de Educação Continuada em Engenharia. II.t.

Para Lucas, meu filho e meu maior e mais importante projeto. Que este trabalho te inspire e evidencie a importância dos estudos na sua vida.

# Agradecimentos

À Larissa, minha amada esposa, expresso minha sincera gratidão pela motivação e suporte incansáveis em todas as etapas desta jornada.

Ao Prof. Dr. Marcos, expresso minha profunda gratidão por ultrapassar os limites de uma simples orientação e tornar-se um mentor em cada fase deste trabalho.

# Sumário

## Resumo

TEIXEIRA J[R], R. *O uso de arquiteturas híbridas na tarefa text-to-SQL*. 2024. Monografia (Especialização em Inteligência Artificial) – Escola Politécnica da Universidade de São Paulo. PECE – Programa de Educação Continuada em Engenharia. Universidade de São Paulo, São Paulo, 2024.

A tarefa de tradução da linguagem natural para linguagem estruturada de banco de dados (SQL) continua a impor desafios significativos a pesquisadores das mais diversas áreas. Dificuldades como a adaptabilidade a novos domínios de dados e de soluções em língua portuguesa sempre estiveram presentes em discussões sobre o assunto (Androutsopoulos, Ritchie e Thanisch 1995). Isso mudou com a chegada dos grandes modelos de linguagem (LLMs), pois a facilidade de uso e sua capacidade de busca e articulação de conhecimento tornou os problemas anteriores parcialmente superados (Gao et al. 2023). Entretanto, isso trouxe novos problemas como a falta de transparência no processo de geração de informação, o alto custo para ajuste fino dos modelos e a limitação de dados sensíveis que podem ser enviados às empresas produtoras dos modelos.

A técnica proposta neste trabalho concentra-se no envio limitado de informações aos LLMs e na geração de instruções (*prompts*) direcionadas à criação da instrução SQL. O *pipeline* de processamento aqui proposto é caracterizado por ser híbrido, ao combinar modelos *Transformers* codificadores para reconhecimento intenção e grandes modelos de linguagem (LLMs) gerativos para produção de consultas SQL. Diferentes modelos Transformers de codificação (BERTimbau, alBERTina e XLM-RoBERTa) e de decodificação (Maritaca e Mixtral) foram testados e comparados na criação do pipeline de melhor desempenho. O modelo global foi ajustado e testado com dados de um dataset público desenvolvido com um banco de dados da Microsoft, AdventureWorksDW2022.

Para tanto, os modelos de reconhecimento de intenções e os de geração de texto tiveram de passar por ajustes, tais como: aplicação de taxas de *dropout* (Srivastava et al. 2014) e reajuste de estrutura nos modelos de reconhecimento de intenções, e calibração de aleatoriedade (temperatura) e engenharia de prompt para os modelos gerativos. Com isso, atingimos resultados de 92% de $F_1$ na tarefa específica de reconhecimento de intenções (com alBERTina), e 52% de acurácia ao final do pipeline (usando Mixtral como modelo gerador de consultas), isto é, na seleção correta de registros buscados. Além dos resultados numéricos propriamente ditos, deve-se levar em conta que a técnica global proposta oferece transparência, adaptabilidade, portabilidade e localização linguística dos conteúdos.

Os resultados indicam, ainda, possibilidades concretas para melhorias, seja por meio de estudos mais aprofundados nas estruturas dos LLMs para criação de prompts mais precisos, seja pela aplicação de outros LLMs gerativos.



**Palavras-chave:** Processamento de Linguagem Natural (PLN). text-to-SQL. nlp2sql. Transformers codificadores. Reconhecimento de Intenções. Grandes Modelos de Linguagem (LLM). Modelos gerativos. Modelos híbridos. SQL.

**Abstract**

TEIXEIRA J[R], R. *The use of hybrid architectures in the text-to-SQL task* 2024. Monografia (Especialização em Inteligência Artificial) – Escola Politécnica da Universidade de São Paulo. PECE – Programa de Educação Continuada em Engenharia. University of São Paulo, São Paulo, Brazil. 2024.

The task of translating natural language into structured database language (SQL) continues to pose significant challenges for researchers from various fields. Difficulties such as adaptability to new data domains and solutions in Portuguese have always been present in discussions on the subject (Androutsopoulos, Ritchie, and Thanisch 1995). This changed with the advent of large language models (LLMs), as their user-friendly nature and their ability to search and articulate knowledge partially overcame previous issues (Gao et al. 2023). However, this brought up new problems, such as lack of transparency in the information generation process, high costs for fine-tuning the models, and limitations on sensitive data that can be sent to model-producing companies.

The technique proposed in this work focuses on sending limited information to LLMs and generating prompts aimed at creating the SQL statement. The proposed processing pipeline is characterized as hybrid, combining encoder Transformers models for intention recognition and large generative language models (LLMs) for SQL query production. Different encoding Transformers models (BERTimbau, alBERTina, and XLM-RoBERTa) and decoding models (Maritaca and Mixtral) were tested and compared to create the best-performing pipeline. The overall model was fine-tuned and tested with data from a public dataset developed using a Microsoft AdventureWorksDW2022 database.

To achieve this, intention recognition models and text generation models underwent adjustments, such as applying dropout rates (Srivastava et al. 2014) and restructuring in intention recognition models, and calibrating randomness (temperature) and prompt engineering for generative models. As a result, we achieved 92% $F_1$ score in the specific task of intention recognition (with alBERTina), and 52% accuracy at the end of the pipeline (using Mixtral as the query generation model), i.e., in the correct selection of sought-after records. In addition to the numerical results, it should be noted that the proposed global technique offers transparency, adaptability, portability, and linguistic localization of contents.

The results also indicate concrete possibilities for improvement, either through more in-depth studies on LLM structures for the creation of more precise prompts, or through the application of other generative LLMs.

**Keywords:** Natural Language Processing (NLP). text-to-SQL. nlp2sql. Transformers encoders. Intent Recognition. Large Language Models (LLM). Generative models. Hybrid models. SQL.

# Lista de Figuras

# Lista de Tabelas

Capítulo 1

# Introdução

Com o crescimento exponencial da produção de dados, e dada a importância do uso destes para a tomada eficaz de decisões, tarefas como (1) extração, (2) manipulação e (3) consumo de dados vêm ganhando uma importância significativa no dia a dia das organizações. E mesmo que as duas primeiras tarefas, tradicionalmente, se restrinjam a equipes e departamentos mais técnicos, a terceira, o consumo de dados, tem extrapolado esses nichos e hoje faz parte do dia a dia da maioria dos departamentos de uma organização.

Áreas antes focadas em aspectos práticos do negócio, por vezes distantes de temáticas tecnológicas, se veem impelidas a criar estruturas para se adequar a essa nova realidade. Torna-se necessária, a partir daí, a figura de um analista de dados, que é o responsável por entender os questionamentos e traduzi-los numa linguagem de consulta estruturada, como SQL, para obtenção desses dados.

A tarefa de conversão de língua natural para SQL não é nova. Muitas vezes apresentada sob diferentes nomes, como *text-to-SQL* ou *nlp2sql* (*Natural Language Processing to SQL*), seus primeiros protótipos aparecem entre os anos 1960 e 1970, em que surgem interfaces de acesso a base de dados via linguagem natural (NLIDB – *Natural Language Interfaces to Databases*) (Androutsopoulos, Ritchie e Thanisch 1995). O tema manteve-se relevante durante décadas (Deng, Chen e Zhang (2022)), atraindo pesquisadores tanto da área de processamento de linguagem natural como banco de dados, sendo motivo de estudo até hoje.

Diante desse cenário, somado à crescente demanda de sistemas capazes de representar de forma computacional os anseios dos usuários, que diversas metodologias têm sido propostas. Grande parte delas, baseadas em parseamento semântico, tem seu alcance limitado à língua inglesa, tornando-as de difícil aplicação para nossa realidade.

Outras soluções, contudo, têm proposto o uso de grandes modelos de linguagem (LLM – *Large Language Models*) para solução do problema em questão, superando assim a limitação da língua. Entretanto, essas metodologias sugerem técnicas de refinamento

desses modelos (Gao et al. 2023), o que inviabiliza sua aplicabilidade em ambiente mais simples, pela necessidade de infraestruturas computacionais custosas para o processo de refinamento.

Outra dificuldade inerente a essa metodologia é a limitação de informações que podem ser enviadas por solicitação, o que dificulta a sua aplicação em grandes bancos de dados.

Além disso, o uso de LLMs impossibilita uma visão transparente do funcionamento do algoritmo, dificultando a identificação de problemas e o processo de melhoria continua.

Sendo assim, procurando englobar todas as demandas oriundas que a nova realidade impõe, bem como as limitações impostas pelas soluções atuais, vamos nos dedicar neste trabalho a explorar soluções que possam superar a necessidade de uma interface humana para acesso à informação estruturada, mesmo em se tratando de indivíduos sem conhecimentos de SQL.

Como ilustração, imagine-se que um usuário de um sistema de banco de dados (SGBD – Sistema de Gerenciamento de Banco de Dados) precise saber a quantidade de clientes por estado, que seria uma informação disponível nas tabelas do banco. Imagine-se, ainda, que o usuário não conheça profundamente a estrutura, nem as tabelas que compõem a fonte de informação. Certamente, parte do tempo gasto será para entender como as tabelas se relacionam, o conteúdo de cada uma delas e quais as potenciais colunas a serem usadas nessa tarefa. Após isso, ele construirá sua primeira consulta, podendo cometer erros, como referenciar incorretamente um nome de tabela ou coluna, não se atentar às regras de sintaxe da linguagem (palavras reservadas, funções, estrutura da consulta) ou até mesmo desconhecer um cálculo específico para chegar no resultado. Nessa última situação, pode ser que, para saber a quantidade de clientes, uma simples contagem não seja suficiente, e que seja necessário filtrar uma parte dos dados e aí efetuar a contagem. Nesse caso, além do tempo despendido com a análise estrutural do banco de dados (BD), será necessária uma análise do conteúdo dos dados, aumentando o tempo gasto para criação da consulta. E se, além disso, for preciso que a informação seja transmitida em formato de texto, mais uma camada de dificuldade é incluída.

Diante desse quadro ilustrativo, a solução que aqui será proposta visa abstrair todas essas tarefas dentro de um *pipeline*, no qual a única ação necessária para o usuário é fornecer uma expressão em língua natural que expresse o que ele deseja saber.

A solução é dividida em três partes: na primeira, é usado um algoritmo para identificação da intenção (IR –*Intenty Recognition*) do usuário com a expressão enviada. A intenção detectada é tida como elemento-chave para saber quais tabelas devem ser usadas para construção da resposta. A seguir, as informações obtidas são submetidas a um LLM instruído a criar uma consulta em SQL, que será aplicada ao banco de dados, retornando um resultado. Por fim, tal resultado será concatenado com a questão inicial e enviado ao

LLM que, instruído a criar uma resposta, juntará as duas partes e produzirá uma expressão em língua natural.

Assim, proporemos uma arquitetura que englobe desde a conversão da expressão em língua natural para SQL, a obtenção de dados estruturados e retorno desses dados em forma de língua natural, que procure atender às necessidades levantadas. Em seguida, colocaremos à prova, a partir de um cenário simulado, a arquitetura proposta. Compararemos de forma crítica como cada variação de componentes na solução pode afetar o resultado final, inferindo quais os motivos que expliquem a diferença de resultado. Tudo isso motivado pela necessidade de democratização do acesso aos dados nas organizações, e de ter uma solução adequada à língua portuguesa, além de contemplar a demanda crescente por sistemas computacionais que embarquem conhecimentos técnicos capazes de aplicá-los de forma transparente.

Não é o foco principal deste trabalho a entrega de um produto, mas a reflexão sobre técnicas para a solução do problema proposto. Por isso, não serão tratados de forma aprofundada conceitos relacionados a sistemas de gerenciamento de bancos de dados, como sua configuração e uso, nem metodologias de modelagem de dados. E mesmo lidando com uma interface de comunicação entre indivíduos e máquina, não serão tratados temas associados à usabilidade, layout, distribuição e acesso.

Não pretendemos tirar a necessidade de pessoas especializadas em tarefas mais complexas de recuperação de informação, mas promover uma interface computacional capaz de facilitar a geração de consultas, automatizando partes repetitivas e potencialmente produtoras de erros no trabalho dos especialistas em bancos de dados.

Para realização do trabalho nos concentramos inicialmente na escolha de um banco de dados. Optamos pelo AdventureWorksDW2022[1], por se aproximar da complexidade de estrutura e de dados das organizações. Uma vez definido o banco de dados, dedicamo-nos a entender sua estrutura e campos, por meio da criação de diagramas entidade-relacionamento e dicionários de dados. Também durante esse processo, tratamos de adequar seus campos e estruturas para uso em um SGDB portátil (SQLite).

Em seguida, tratamos de definir qual metodologia usaríamos e depois de uma avaliação definimos um modelo híbrido, que combina a tarefa de IR com a geração dos LLMs. A escolha desse modelo híbrido visou a transparência em tarefas de ajuste, a diminuição de custo computacional e a assertividade.

Para as tarefas de IR, selecionamos três versões do BERT (Devlin et al. 2019) (*Bidirectional Encoder Representations for Transformers.*) em português: BERTimbau(Souza, Nogueira e Lotufo 2020), RoBERTaXLM(Liu et al. 2019) e alBERTina(Rodrigues et al. 2023). Enquanto para a tarefa de geração com LLMs, optou-se pelo uso do Maritaca e do

[1]Esse banco de dados está publicamente disponível aqui.

Mixtral(Jiang et al. 2024). Fizemos essa seleção com base em critérios de desempenho, limitação orçamentária e de recursos computacionais disponíveis.

Ainda durante o processo de definição dos algoritmos que seriam usados, criamos uma base de dados com 350 afirmações/questões com o sua correspondente instrução SQL, que após revisão foram categorizadas em 17 intenções que, em seguida, foram usada como fonte para refinamento dos algoritmos de IR.

Após o ajuste fino dos algoritmos de IR, as soluções (já treinadas) de IR e de geração pelos LLMs foram combinadas para a criação do pipeline. E de forma randômica foram selecionados 70 registros para avaliação do geral do sistema.

Os resultados obtidos foram analisados considerando as métricas de como precisão, cobertura e Medida-F, para os algoritmos de IR. Para a tarefa de geração de SQL pelo LLM, avaliamos a fração do resultado pelo valor resultante da consulta SQL gerada versus o resultado da consulta SQL indicada na base de dados dourada.

Para a tarefa de reconhecimento de intenções, atingimos um índice de 95% de Medida-F. Já para a tarefa de aplicação das instruções SQL geradas, ou seja, os registros corretamente selecionados pelas instruções automaticamente produzidas, tivemos 52% de acurácia.

Este trabalho está organizado da seguinte maneira: O capítulo Revisão da literatura oferece uma análise critica de trabalhos relacionados ao tema de text-to-SQL. Mostramos a relevância do tema frente a artigos relevantes sobre o tema.

No próximo capítulo, Métodos, detalhamos o processo de desenvolvimento da solução, desde as adequações necessárias para uso da fonte de dados, a construção de arquivos que suportem os ajustes dos algoritmos e a validação dos resultados, a escolha e treinamento dos algoritmos de reconhecimento de intenção e de geração de texto e a construção do pipeline da solução.

No seguinte, Resultados e Discussão, descrevemos os resultados alcançados desde a aplicação dos algoritmos de reconhecimento de intenção, de geração de instruções SQL e geração de texto. Discutimos os resultados de cada uma das sub tarefas da solução, comparando como os diferentes algoritmos se saíram, inferindo os possíveis motivos das diferenças.

Por fim, no capítulo Conclusão, oferecemos uma analise dos resultados atingidos frente à literatura de referencia e aos objetivos colocados. Além disso, pontuamos possíveis pontos de melhoria na solução e refletimos sobre futuros estudos sobre o tema.

Capítulo 2

# Revisão da literatura

Androutsopoulos, Ritchie e Thanisch (1995) coloca de forma sistemática as motivações, problemas e desafios até hoje inerentes as tarefas text-to-SQL. Destaca como motivação principal o acesso facilitado a informações sem a necessidade de conhecimentos técnicos. Em relação aos problemas enfrentados na tarefa, destaca como as características linguísticas impactam a solução, ressaltando a complexidade introduzida pela natureza flexível e ambígua da linguagem natural. Coloca como desafio a adequação de soluções para outros domínios de conhecimento.

Essa análise preliminar forneceu subsídios para compreender a origem e os desafios fundamentais para a tarefa de text-to-SQL. Não nos concentraremos na superação dos desafios linguísticos impostos à tarefa, mas em responder a motivação colocada pelo autor, buscando soluções para superar o desafio proposto.

Para definir os pontos a serem abordados neste trabalho, utilizamos como referência o artigo de Deng, Chen e Zhang (2022). Nele os autores fornecem detalhes sobre bases de dados, métodos e formas de avaliação para algoritmos de text-to-SQL baseados em estruturas diversas. Ao final, destacam os desafios para o futuro, incluindo a adaptação de algoritmos a novas estruturas, a exploração de técnicas para línguas além do inglês, a ampliação do escopo de uso e o uso de LLMs para a tarefa.

Motivado pelos desafios que os autores destacaram no final de seu texto, definimos os objetivos deste trabalho. Procuraremos propor uma solução adaptável a novos contextos, com ênfase na língua portuguesa, incorporando a capacidade gerativa dos LLMs e propondo uma interface capaz de traduzir linguagem natural para SQL.

Para definição do *pipeline*, baseamo-nos no artigo de Shankar (2023), que demonstra cinco estruturas possíveis usando LLMs capazes de sustentar a tarefa de text-to-SQL. Após delinear o problema, indicar as dificuldades e demonstrar a relevância das LLMs para a tarefa, são descritas de forma prática as cinco abordagens possíveis, das quais optamos por explorar nesse trabalho a estratégia de Reconhecimento de Intenção e Reconhecimento de

Entidades Nomeadas com o uso de LLMs. Essa estratégia usa essas técnicas para refinar as informações que serão depois enviadas aos LLMs.

Essa estratégia destaca-se por ser uma solução de fácil customização, transparente e com uma precisão aceitável. Nós a aplicamos neste trabalho divergindo da solução apresentada, ao não abordar de forma destacada a sub-tarefa de NER, atribuindo essa responsabilidade à parte geracional dos LLMs.

Na definição de qual técnica seria usada no reconhecimento de intenções, baseamo-nos no artigo de Khan e Meenai (2021). Nele, os autores iniciam definindo a tarefa de reconhecimento de intenções. Após isso, eles descrevem quais são as abordagens possíveis para a tarefa. Em seguida, descrevem o funcionamento do algoritmo escolhido por eles (BERT – *Biderectional Encoders Representations from Transformers* (Devlin et al. 2019)), apresentando o processo de treinamento e avaliando a assertividade da solução a partir de métricas especificas.

Optamos por seguir com a mesma técnica apresentada pelos autores, a de ajuste finos de modelos pré-treinados na arquitetura BERT para domínios específicos. Entretanto, diferenciamo-nos pela escolha de qual algoritmo foi usado. Enquanto os autores usaram a versão original do BERT, nesse trabalho utilizamos versões pré treinadas em português ou multilíngues.

Acerca da definição do método a ser usado nas tarefas de geração de instruções SQL, baseamo-nos no artigo de Gao et al. (2023). Nele, os autores detalham os usos dos LLMs para a tarefa de geração de instruções SQL, inicialmente delimitando os modos de usos já existentes a partir de chamadas diretas aos algoritmos, propondo a realização de um ajuste fino desses modelos para melhor performance. O texto oferece, além disso, instruções de como construir instruções para os modelos de linguagem de forma eficiente e demonstra seus resultados. Nele é descrito o estado da arte, nas tarefas de text-to-SQL, que se baseia na construção eficaz de prompts, a partir de técnicas de *few-shot*[1], associadas ao ajuste fino dos grandes modelos de linguagem.

Escolhemos representar, no nosso trabalho, uma versão que não contempla a técnica de ajuste fino do modelo proposta no referido artigo. A escolha foi feita devido a limitações computacionais e orçamentárias para esse projeto. As instruções relativas a construção eficaz de prompts para os modelos de linguagem foram aplicadas em todas as etapas associadas nesse projeto.

Em relação às métricas possíveis para avaliação da geração de instrução SQL pelo LLMs, tivemos como referência o artigo de Yu et al. (2018). Nele, é apresentada uma base de dados para servir de referência e comparação para algoritmos de text-to-SQL. Além

[1]O termo se refere à técnica de envio de informações de base necessárias para resolução da tarefa ao LLM. Por exemplo, no nosso caso, envia-se a questão e quais as tabelas possíveis para construção da resposta.

disso, o texto trata da definição de métricas para avaliação da solução e categorização das instruções SQL por dificuldade de geração: correspondência de componentes, correspondência exata, acurácia de execução e divisão das instruções em quatro categorias: fácil, média, difícil e extra difícil. No nosso trabalho, inspiramo-nos no que foi proposto, mas focamos somente em métricas de execução. Além disso, aplicamos critérios diferentes a categorização das instruções.

Capítulo 3

# Métodos

Os métodos descritos a seguir visam à replicação da solução como proposta na Figura 3.1. A solução completa, bem como os arquivos para configuração, encontram-se disponíveis no repositório do GitHub.

## 3.1 Procedimentos

Nessa seção detalharemos quais procedimentos são necessários para replicação integral da solução, desde a camada de dados, configuração e pipeline.

### 3.1.1 Dados

Nesta sub-seção apresentaremos os métodos usados de adequação do banco de dados para o ambiente do SQLite. É nessa camada onde os dados ficam armazenados e disponíveis para consulta a partir de instruções SQL.

#### Replicação de estrutura de dados

Para este trabalho, as instruções SQL de construção do banco de dados, fornecidas pela Microsoft , tiveram que ser reescritas visando a adequação de tipo e formato de colunas para o SQLite.

Além de ajustes nas estruturas das colunas, comandos para inclusão de índice específicos para sistemas Microsoft foram retirados e reescritos conforme o sistema destino.

Também foram retirados da instrução original os comandos de carga em lote dos dados, novamente por não estarem disponíveis no sistema destino e reescritos usando a linguagem Python na versão 3.10.12 e a biblioteca Pandas.

### 3.1.2 Configuração

Detalharemos, a partir daqui, os procedimentos usados para geração dos arquivos necessários na camada de configuração da solução. Esses arquivos são usados em processos de ajuste fino dos algoritmos de IR e também no pipeline da solução proposta.

#### Dicionário de dados

**Listing 1** Estrutura do dicionário de dados.

```
1  <NOME DA TABELA>: {
2     "TipoTabela": <DIMENSÃO OU FATO>,
3     "Objetivo": <INFORMAÇÃO REPRESENTADA NA TABELA>,
4     "Colunas": {
5         <NOME DA COLUNA>: {
6             "Descrição": <INFORMAÇÃO REPRESENTADA NA COLUNA>,
7             "Valores Possíveis": <LISTA DE VALORES POSSÍVEIS>
8         }
9     },
10    "DDL": <INSTRUÇÃO DE CRIAÇÃO DA TABELA> ,
11    "Relacionamentos": <LISTA OS RELACIONAMENTOS DA TABELA>
12 }
```

O dicionário de dados foi desenvolvido a partir de métodos empíricos para servir como base de informação de características relacionadas as estruturas do banco de dados. Esse método se baseia na analise dos nomes e do conteúdo das tabelas e suas respectivas colunas para obter informações relacionadas ao objetivo da tabela ou a que dado a coluna se refere.

Unificamos todas essas informações em arquivos delimitados por vírgulas (CSV – *comma-separated-values*) e, em seguida, usando Python e as bibliotecas Pandas e JSON criamos um único arquivo JSON (*JavaScript Object Notation*) com todas as informações.

A estrutura desse aquivo é indicado na Listing 1 e seu preenchimento é feito da seguinte maneira:

- Linha 2, a partir do prefixo do nome da tabela
- Linhas 3 e 6, a partir dos arquivos preenchidos pelo método empírico
- Linha 7, a partir de dos dados disponíveis na coluna
- Linhas 10 e 11, a partir dos arquivos de criação do BD

Essa estrutura é um elemento-chave dentro da solução proposta por ser parte necessária no processo de construção do *prompt*, que é enviado ao algoritmos gerativo responsável pela construção das instruções SQL.

### AdventureQI

Da mesma forma que o anterior, o arquivo AdventureQI foi desenvolvido empiricamente, a partir da escrita de 350 registros com o paralelo entre questões/afirmações de usuários, as intenções, a instrução SQL correspondente e as tabelas que serão usadas em arquivo CSV.

As questões e afirmações contidas no arquivo foram escritas para admitirem uma única resposta possível a partir da instrução SQL correta, que também é indicada no arquivo.

Como forma de subsidiar nosso trabalho, seus dados são utilizados como insumo e critério de avaliação no processo de ajuste fino do modelos de reconhecimento de intenção, e também como avaliação da resposta resultante da instrução SQL gerada via pipeline versus o esperado.

Seu processo de conversão para arquivo json é feito usando Python e as bibliotecas Pandas e JSON.

## 3.1.3 Pipeline

O pipeline global da solução é o que se apresenta na Figura 3.1.

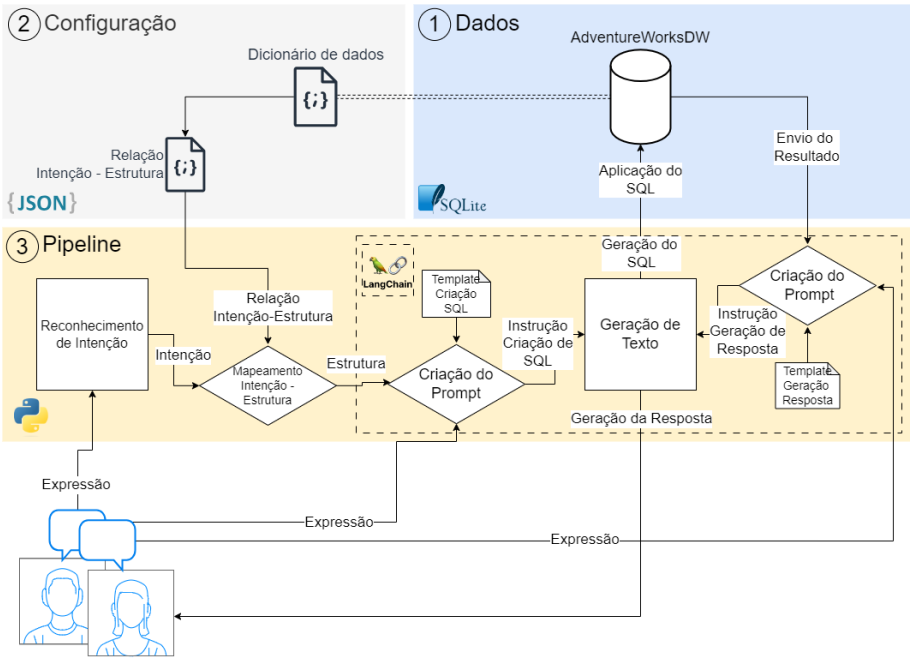


**Figura 3.1:** Diagrama da solução proposta.

Na primeira camada, que é a camada de dados, está o banco de dados que será usado como exemplo na solução. Nessa camada estão também os métodos para acesso e consulta aos dados.

Na camada 2, estão os arquivos de configuração: o dicionário de dados que contém uma descrição das tabelas e das colunas e o arquivo de Mapeamento Intenção-Estrutura, que contém o relacionamento entre a intenção identificada e as tabelas relacionadas.

Na última camada, a de número 3, ficam os algoritmos de identificação de intenções e os algoritmos de geração de texto, que faz a comunicação entre as demais camadas da aplicação.

### Reconhecimento de intenções

Para a sub-tarefa de reconhecimento de intenções, são utilizados algoritmos baseados em BERT (Devlin et al. 2019) pré-treinados em língua portuguesa, (Souza, Nogueira e Lotufo (2020), Conneau et al. (2020) e Rodrigues et al. (2023)) que passaram por um processo de ajuste fino. Esse processo, chamado também de *fine-tuning*, propõe que, a partir de uma base de dados nova, os pesos internos do modelo sejam alterados. Essa alteração ocorre em todos as camadas da rede, na grandeza dos valores obtidos nas camadas de atenção do modelo.

Neste trabalho, o ajuste fino dos modelos foi feito utilizando a biblioteca Transformers da plataforma HuggingFace. Essa biblioteca oferece uma API[1] (*Application Programming Interface*) para acesso aos modelos pré-treinados hospedados na plataforma como, também, ferramentas para treinamento e análise desses.

Como preparação para essa etapa, foi feito o tratamento dos dados do AdventureQI com o uso bibliotecas Pandas e Numpy. Esse tratamento inicial consistiu na seleção dos campos que serão usados (questões/interações e intenção), no ajuste de nomenclatura e na conversão do campo de intenções em representações numéricas.

Ainda na fase de tratamento de dados, as informações de questões/interações são submetidas ao processo de *tokenização* por meio das funções assistentes disponíveis na biblioteca Transformers. Essa ação consiste em quebrar uma sentença em pedaços (palavras ou parte delas), chamados de *tokens*, e em seguida atribuir a cada um deles uma representação numérica. Conforme Mielke et al. (2021), essa tarefa não é propriamente nova, mas nas arquiteturas BERT ela é feita de forma bidirecional e dentro de uma janela de contexto (Devlin et al. 2019).

O resultado da função de tokenização são duas novas estruturas: uma com a representação numérica de cada token nas respectivas sentenças e, outra, com a máscara de atenção. Essa segunda estrutura é necessária para indicar quais valores, da representação

[1] Interface de comunicação entre diferentes sistemas.

numérica das sentenças, devem ser levados em conta em processamentos posteriores. Isso acontece pelo fato de as sentenças tokenizadas não terem o mesmo tamanho e, por isso, deve-se completar a quantidade de tokens minímas configurada, conforme documentação. Sendo assim, usamos essa técnica para que esses tokens adicionais não influenciem no treinamento que utiliza a mascara de atenção para indicar quais deles devem ser levados em conta.

Os dados tratados são separados em estruturas de treino e teste, sob a divisão de 80/20. Essa divisão é realizada usando a biblioteca sklearn e é feita na representações numéricas das intenções, nos tokens das sentenças e nas máscaras de atenção de forma estratificada.

Para o ajuste fino dos modelos escolhidos, foi utilizada a biblioteca tensorflow (Abadi et al. 2016) para configuração das métricas de performance, função de perda e otimização, e a biblioteca transformers para construção e treinamento do modelo. A integração entre as duas é feita de forma natural pelo fato de a segunda se basear em muitas das suas funções na primeira.

As opções disponibilizadas pela biblioteca transformers para construção e treinamento do modelo basicamente implantam uma rede neural composta por três camadas: a primeira com o algoritmo pré-treinado, a segunda por uma camada de *dropout* e, a última, por uma camada de saída com uma função *softmax*.

Na configuração do processo de treinamento usamos como métricas de perda (*loss*) a entropia cruzada esparsa, e, como medida de performance, a acurácia. Para ajuste dos pesos da rede escolhemos empregou-se a abordagem de otimização em gradientes descentes baseados em estimativas adaptadas de primeira e segunda ordem, também conhecido como ADAM (Kingma e Ba 2017).

Todos os modelos foram treinados por até 600 épocas, com os mesmos dados de treino e de teste.

Nesse ponto, um dos desafios encontrados no processo de ajuste fino dos modelos de reconhecimento de intenção foi a quantidade limitada de dados. Isso fazia com que os algoritmos com poucas iterações de treinamento ficassem sobre-ajustados aos dados (também conhecido como *overfitting*).

A metodologia usada para solução desse problema foi a aplicação de *dropout* (Srivastava et al. 2014), que consiste no desligar de neurônios aleatórios naquele momento (época) do treinamento, conforme exemplificado na Figura 3.2.

Outra técnica usada para melhoria do resultado desses algoritmos foi a alteração da estrutura de camadas ocultas. Para alguns deles, a complexidade padrão indicada pela biblioteca impedia que houvesse melhora no processo de aprendizagem.

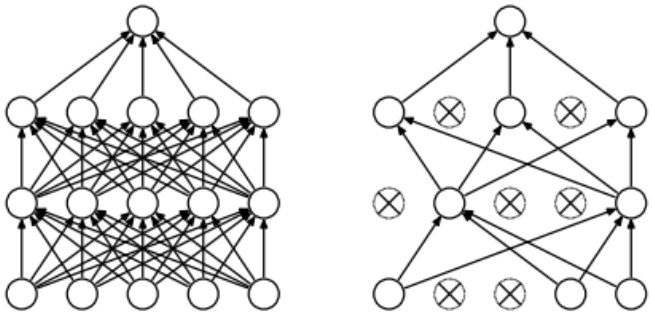


**Figura 3.2:** Remoção temporária de neurônios de uma rede neural, *dropout*. Fonte: Srivastava et al. (2014).

Ao final de cada processo de treinamento os modelos foram avaliados com base nas métricas de precisão, cobertura e Medida- F. Os pesos que fossem correspondentes às melhores versões foram salvos num repositório persistente para uso posterior.

Os resultados obtidos a partir das técnicas de dropout, alteração de estrutura da rede neural e a avaliação dos modelos pós treinamento serão discutidos no capítulo Resultados e Discussão.

Acerca das características dos modelos pré-treinados usados nesse trabalho, todos são baseados em arquiteturas BERT (Devlin et al. 2019). São modelos transformers de redes neurais com camadas totalmente conectadas (*fully connected*), e diferem entre si seja pela quantidade de parâmetros treináveis, pela arquitetura da rede ou pelas base de dados em sua construção.

| | **BERTimbau** | **alBERTina** | **XLM-RoBERTa** |
|---|---|---|---|
| Idioma | Português | Português | Multilíngue |
| Parâmetros | 335 milhões | 900 milhões | 560 milhões |
| Camada Transformers | 12 | 24 | 24 |
| Dimensões Ocultas | 768 | 1536 | 1024 |
| Camadas de Atenção | 12 | 24 | 16 |
| Base Pré-Treino | brWaC | brWaC | CommonCrawl |

**Tabela 3.1:** Comparativo entre características dos modelos.

No caso do BERTimbau (Souza, Nogueira e Lotufo 2020) e do alBERTina (Rodrigues et al. 2023), eles foram treinados com o corpus brWaC (Wagner Filho et al. 2018), mas diferem na arquitetura da rede neural. O primeiro tem 12 camadas, tanto de transformadores como de atenção, enquanto o segundo, tem 24 de cada. Isso muda tanto o nível de complexidade que a rede pode representar como seus parâmetros treináveis: 335M e 900M.

Já o XLMRoBERTa (Conneau et al. 2020) se distingue principalmente por ser um modelo multilíngue. Ele foi treinado com uma base de cerca de 2.5TB de dados coletados de paginas Web pelo CommonCrawl, contendo mais de 100 línguas. Sua estrutura se

baseia numa arquitetura com 24 camadas de transformadores e 16 de atenção, com 560M de parâmetros treináveis.

Para uso dos modelos de reconhecimento de intenções após o treinamento, foram criadas três classes em linguagem Python que abstraíssem as etapas necessárias para uso.

Na primeira, *UtilsData*, é feita a comunicação com as estruturas de dados responsáveis por mapear o resultado da rede neural, necessariamente em representação numérica, para a intenção textual.

A seguinte, *UtilsBert*, trata de implementar funções comuns a todos os modelos treinados, como, por exemplo, o processo de tokenização de uma nova sentença.Nessa etapa, ainda, foram implementados métodos para tratamento da informação de saída da rede, como a transformação das probabilidades das intenções em valor textual.

Por fim, a última classe procura encapsular as informações modelo treinado com todas as variáveis e funções que possam ser usadas. Nesse caso, cada modelo tem uma classe especifica (*FineTuningBERTimbau, FineTuningAlBERTina e FineTuningRoBERTaXLM*) que recebe através de herança as variáveis e métodos das classes anteriores.

### Geração de texto

Usamos para a tarefa de geração de texto dois modelos grandes de linguagem (LLMs): o Maritaca (Maritaca 2024) e o Mixtral (Jiang et al. 2024), ambos implementados com a biblioteca LangChain.

No Maritaca (Maritaca 2024), o uso do modelo foi feito via API oficial da solução, o Maritalk, e sua interface foi construída a partir de códigos disponibilizados (que podem ser consultados nesse link) pelos próprios autores e pela o uso da biblioteca LangChain.

Já no Mixtral (Jiang et al. 2024), o uso do modelo foi feito de forma local e operacionalizado com métodos de quantização mista e uso por demanda de frações do modelo por meio de *experts* (Eliseev e Mazur 2023) e a partir de códigos disponibilizados (que podem ser consultados nesse link) pelos autores.

O LangChain foi usado como facilitador nas tarefas de criação modelos de prompts (*templates*) e uso encadeado dos modelos (*chain*). Nos templates definimos previamente a estrutura da instrução que será enviada ao modelo, podendo ser reusada com novas informações. Questões relativas a tokens e estruturas especificas que o modelo necessita no input da informação são abstraídas nesse processo, pois a própria biblioteca cuidará, quando necessário, da inclusão dessas estruturas.

Dois templates específicos foram desenvolvidos nesse trabalho: um com instruções para geração da instrução SQL, como regras para criação da instrução, questão a ser respondida e estrutura do DB, e, outro, para criação de uma resposta ao usuário, contendo a questão a ser respondida e o resultado da instrução SQL gerada.

**Listing 2** Estrutura básica dos templates criados.

```
[
    ("system", <INSTRUÇOES DA TAREFA> ),
    ("human",  <INSUMOS PARA REALIZACAO DA TAREFA> )
]
```

Naturalmente, o processo de geração de informação pelos modelos de linguagem passa por etapas de inserção de dados, processamento e resposta. A técnica de *chain*, do LangChain, simplifica a construção desses mini-pipelines, encadeando todas as etapas do processo num único objeto. Além disso, ela permite o reuso de diversos componentes, como templates, chamadas a modelos de linguagem e analisadores de saída (*OutputParses*).

Para cada modelo, criamos duas chains especificas: uma para geração da instrução SQL e outra para geração do texto resposta. Cada uma com um template distinto mas os componentes de chamadas aos modelos de linguagens e saídas iguais.

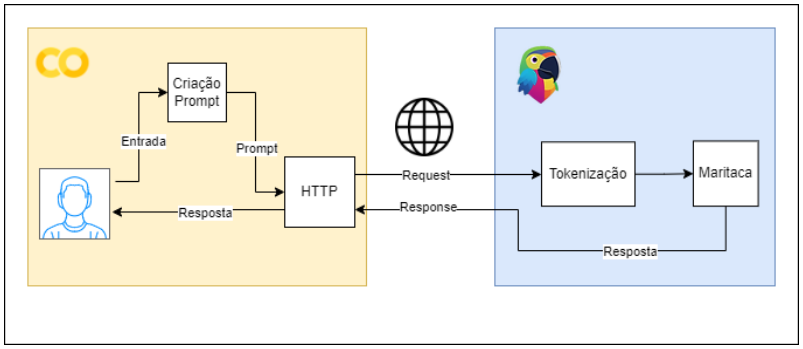


**Figura 3.3:** Diagrama do funcionamento do Maritaca na solução.

De forma prática, o resultado que cada modelo traz ao usuário tende a ser parecido em termos de formato e conteúdo. Entretanto, a forma interna como isso acontece difere bastante. Em se tratando de Maritaca, o funcionamento é simples: o prompt é construído a partir dos templates e enviado via uma requisição HTTP para um servidor externo que processa a informação e retorna com o resultado. No entanto, esse processo obscurece etapas importantes dos algoritmos gerativos, como a tokenização. De maneira não explícita, ao enviar uma requisição HTTP para o servidor do Maritalk, ocorre a tokenização da instrução fornecida, assim como a criação das máscaras de atenção, como exemplificado na Figura 3.3.

Por se tratar de um processamento local, isso não ocorre com o Mixtral. As etapas de tokenização e criação de máscaras de atenção são explícitas e devem ser realizadas antes do uso efetivo do algoritmo. Essa dinâmica é exemplificada na Figura 3.4.

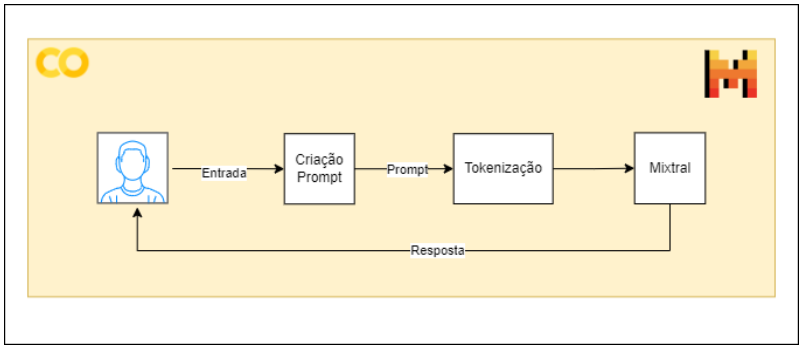


**Figura 3.4:** Diagrama do funcionamento do Mixtral na solução.

Em se tratando da integração desses modelos com o LangChain, no caso do Maritaca, os autores já tinham disponível um código com a integração. Utilizamos este código como referência e o refatoramos para atendimento às necessidades específicas do trabalho. Por outro lado, para o Mixtral, essa integração pré-existente não existia, e, por isso, desenvolvemos a solução com base na documentação de referência do LangChain.

Testamos a funcionalidade de geração de instruções SQL por 70 registros anotados com um exemplo de instrução SQL que atenda à solicitação e à complexidade de tal instrução. As instruções cuja dificuldade foi classificada como "Muito Baixa" ou "Baixa" exigiam somente filtros simples e ordenações. Já as classificadas como "Moderada" ou "Alta" exigiam a construção de relacionamentos entre as tabelas e a aplicação de funções específicas da linguagem.

Partindo da premissa que as instruções SQL geradas estão corretas do ponto de vista sintático, simulamos suas possíveis respostas para identificar quais as características cada modelo emprega na forma de geração da resposta. Para isso, testamos como os algoritmos gerariam a resposta, com as informações solicitadas disponíveis ou não, visando a simular uma situação real de aplicação da solução. Os achados e resultados dessas últimas tarefas serão descritos na Seção Resultados e Discussão.

## 3.2 Materiais

### 3.2.1 AdventureWorksDW2022

O banco de dados utilizado na solução foi o AdverntureWorksDW2022, que é um banco de dados fictícios, desenvolvido pela Microsoft com o intuito de servir como fonte de dados de teste para suas soluções.

Criado sob as características de um *DataWarehouse* [2] simula os dados de um empresa multinacional de manufatura chamada AdventureWorks. Os dados são públicos (sob licença MIT), mantidos pela Microsoft e podem ser obtidos aqui.

### 3.2.2 AdventureQI

O AdventureQI é uma base de dados construída a partir do AdventureWorksDW2022 usada como recurso para treinamento dos modelos de reconhecimento de intenção desse trabalho e avaliação de resultados.

Os dados são públicos e trazem em paralelo expressões em língua portuguesa, a intenção associada à expressão e uma consulta SQL para obtenção do resultado. Ela foi desenvolvida pelo autor deste trabalho e pode ser acessada por aqui.

## 3.3 Instrumentos

Como instrumento de hardware para esse trabalho utilizamos a plataforma Google Colaboratory, usando a versão PRO+.

Para treinamento de cada modelo de reconhecimento de intenções, configuramos a plataforma com as seguintes especificações:

- **BERTimbau**: Processador: Intel(R) Xeon(R) CPU @ 2.00GHz, Armazenamento: 166.8 GB, Memória RAM: 83.5 GB de RAM, GPU: NVIDIA V100 com 15GB
- **alBERTina**: Processador: Intel(R) Xeon(R) CPU @ 2.20GHz, Armazenamento: 166.8 GB, Memória RAM: 83.5 GB de RAM, GPU: NVIDIA A100 com 40GB
- **XLM RoBERTa**: Processador: Intel(R) Xeon(R) CPU @ 2.00GHz, Armazenamento: 166.8 GB, Memória RAM: 51 GB de RAM, GPU: NVIDIA T4 com 15GB

Para a execução da solução completa, usamos as seguintes configurações: Processador: Intel(R) Xeon(R) CPU @ 2.20GHz, Armazenamento: 166.8 GB, Memória RAM: 83.5 GB de RAM, GPU: NVIDIA A100 com 40GB.

Como banco de dados utilizamos o SQLite que "é uma biblioteca em linguagem C que implementa um mecanismo de banco de dados SQL pequeno, rápido, independente, de alta confiabilidade e completo´´ (conforme página oficial). Como SGDB, usamos o SQLiteExpert.

Os softwares usados foram a linguagem Python na versão 3.10 e suas seguintes bibliotecas:

[2] [DataWarehouse] "É um sistema de armazenamento digital que conecta e harmoniza grandes volumes de dados de várias fontes diferentes" Fonte: SAP

- The Python Standard Library: biblioteca que armazena uma lista extensiva de procedimentos e funções utilitárias do Python
- Pandas: oferece ferramentas para analise e tratamento de dados tabulares
- Numpy: biblioteca numérica para cálculos em matrizes.
- TensorFlow: biblioteca para aprendizado de máquina focada em redes neurais
- Transformers: API para integração com modelos de inteligencia artificial
- Scikit-learn: biblioteca para aprendizado de máquina
- LangChain: biblioteca dedicada para desenvolvimento de modelos de linguagem

Capítulo 4

# Resultados e Discussão

Neste capítulo, descreveremos os resultados obtidos em relação aos modelos de reconhecimento de intenções e de geração de instruções SQL e, posteriormente, a geração dos textos. Após isso, discutiremos os resultados alcançados decorrentes em cada etapa citada.

## 4.1 Descrição dos achados

Nesta seção, apresentaremos os dados do processo de treinamento dos modelos de reconhecimento de intenção bem como seus resultados. Na geração de instruções SQL, avaliaremos a assertividade da informação gerada a partir da consulta SQL criada. Por fim, analisaremos a diferença entre as respostas geradas pelos diferentes algoritmos de geração de texto.

### 4.1.1 Reconhecimento de Intenções

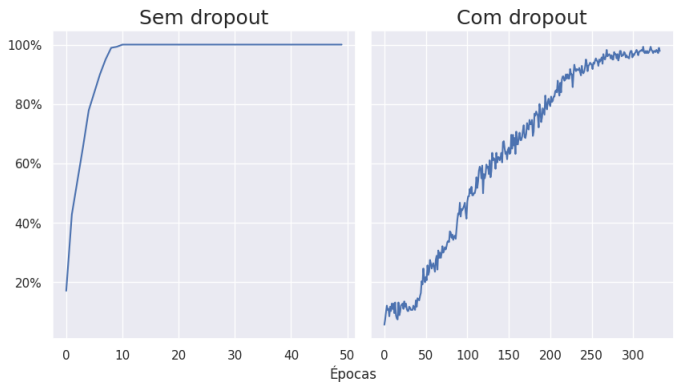


**Figura 4.1:** Comparativo da acurácia entre modelos treinados com e sem a técnica de *dropout* (BERTimbau).

Devido à quantidade limitada de dados, as primeiras tentativas de ajuste fino dos modelos não atingiram o resultado esperado, gerando um sobre ajuste nos dados (*overfitting*). Já nas primeiras épocas, os modelos atingiam a 100% de acurácia nos dados de treino, o que inviabilizava o uso do modelo para previsão de novos resultados.

Aplicamos então taxas de 40% até 60% de *dropout*, nos algoritmos de reconhecimento de intenção, e com isso tivemos uma melhora expressiva na capacidade de generalização do modelo, que evitou que ocorresse o sobre ajuste nos dados, conforme Figura 4.1.

Esse ajuste foi o suficiente para atingirmos uma performance aceitável com o BERTimbau (Souza, Nogueira e Lotufo 2020). Entretanto, para os demais, foi necessário alterarmos a estrutura das camadas ocultas. O padrão das arquiteturas dos modelos alBERTina(Rodrigues et al. 2023) e XLM-RoBERTa(Conneau et al. 2020) estava orientada para tarefas muito mais complexas.

Como estratégia para contornar esse problema, alteramos a quantidade de camadas de transformadores em cada um dos modelos de 24 para 12. Isso produziu o resultado esperado: os modelos foram capazes de assimilar as características dos exemplos e assim reduzir seu valor de perda, como pode ser visto na Figura 4.2.

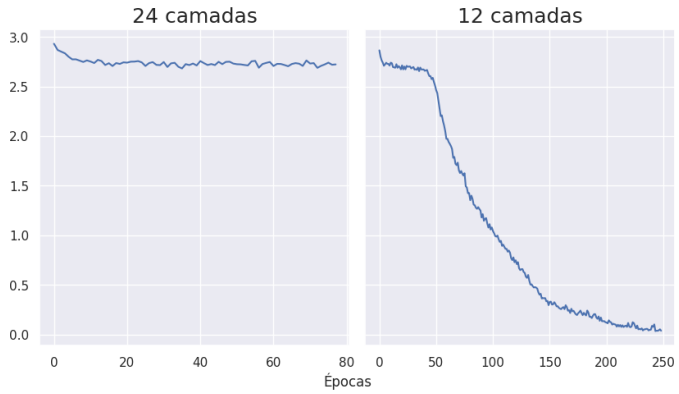


**Figura 4.2:** Comparativo da perda entre modelos treinados com quantidade de camadas diferente (XLMRoBERTa).

Passando para o processo de treinamento, identificamos comportamentos diferentes para cada modelo. O momento em que cada modelo atingiu o estado estacionário em relação ao valores de perda diferiu consideravelmente. Para o BERTimbau , ocorreu perto da época 333. Já para o alBERTina e o XLM- RoBERTa , ocorreu bem antes, próximo a época 279 e 249 consecutivamente.

Entretanto, o fato do processo de otimização para um determinado modelo ser mais rápido do que outro não necessariamente o torna melhor. Como demonstrado na Figura 4.3, os modelos que convergiram a um estado estacionário com menos épocas não diminuíram os valores de perda de forma paralela nos dados de treino e teste. De forma prática, eles aprendem mais rápido mas não tão bem quanto o primeiro.

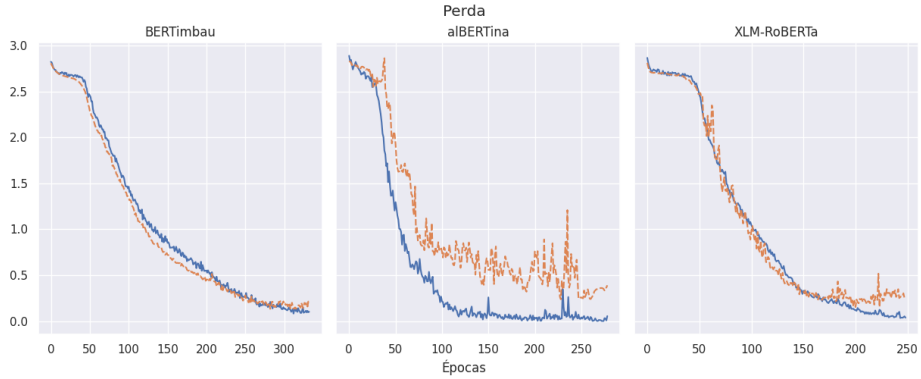


**Figura 4.3:** Evolução do valor de perda no processo de treinamento dos modelos. As linhas em azul se referem aos dados de treino e, as em laranja, aos de teste.

| BERTimbau | | | alBERTina | | | XLM-RoBERTa | | |
|---|---|---|---|---|---|---|---|---|
| Precisão | Cobertura | $F_1$ | Precisão | Cobertura | $F_1$ | Precisão | Cobertura | $F_1$ |
| 89.95% | 93.27% | 91.25% | **95.72%** | **95.51%** | **95.56%** | 93.38% | 91.31% | 91.39% |

**Tabela 4.1:** Avaliação comparativa de modelos de IR.

Com o término dos treinamentos, avaliamos os modelos através das métricas de precisão, cobertura e Medida-F. Dos três modelos treinados, somente o alBERTina ficou acima dos 95% em todos os indicadores, conforme exposto na Tabela 4.1. Mesmo assim, os valores acima de 90% na métrica $F_1$ indicam que os modelos atingiram o resultado esperado: são capazes de prever na maioria dos casos as intenções corretas.

### 4.1.2 Geração de instruções SQL

| | Maritaca | | | | | | Mixtral | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acertos | | Erros | | | | Acertos | | Erros | | | |
| T | # | % | S | % | C | % | # | % | S | % | C | % |
| 0.1 | **27** | **38.57%** | 13 | 18.57% | **30** | **42.85%** | 33 | 47.14% | 13 | 18.57% | **27** | **38.57%** |
| 0.4 | 23 | 32.85% | **20** | **28.57%** | 27 | 38.57% | 34 | 48.57% | **14** | **20.00%** | 22 | 31.42% |
| 0.7 | 23 | 32.85% | **20** | **28.57%** | 27 | 38.57% | **37** | **52.85%** | 10 | 14.28% | 23 | 32.85% |

**Tabela 4.2:** Avaliação comparativa da métricas de geração de instruções SQL. A coluna T se refere ao parâmetro de temperatura do modelo. A S, aos erros de sintaxe e C aos erros de conteúdo. Os valores destacados se referem ao maior valor dentro daquele modelo.

A capacidade de aleatoriedade de um modelo de geração de texto pode determinar se ele será eficaz ou não numa determinada tarefa. Nos modelos que usamos no trabalho, a partir de um estudo comparado com nossas bases de dados, identificamos que, para

o Maritaca (Maritaca 2024), valores mais baixos de temperatura, entre 0.1 e 0.4, são melhores, enquanto pro Mixtral (Jiang et al. 2024) isso não ocorre, os valores começam a melhorar a partir de 0.4. Esse comportamento pode ser visto na Tabela 4.2.

De forma prática, o aumento na permissividade de geração de valores aleatórios permitiu que os algoritmos construíssem instruções mais elaboradas. Isso foi benéfico em tarefas mais complexas [1] para o Mixtral, mas nem tanto para o Maritaca (Figura 4.4).

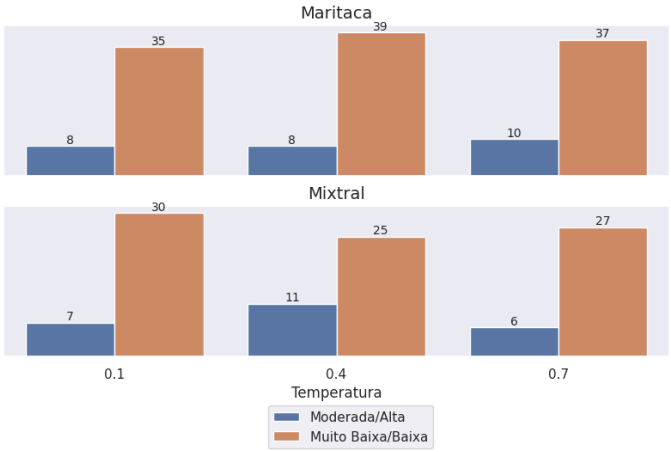


**Figura 4.4:** Quantidade de erros pela complexidade da instrução SQL.

Conforme o valor de temperatura aumentava, erros de sintaxe ocorriam mais no Maritaca, diferentemente do Mixtral, conforme explicitado na Figura 4.5.

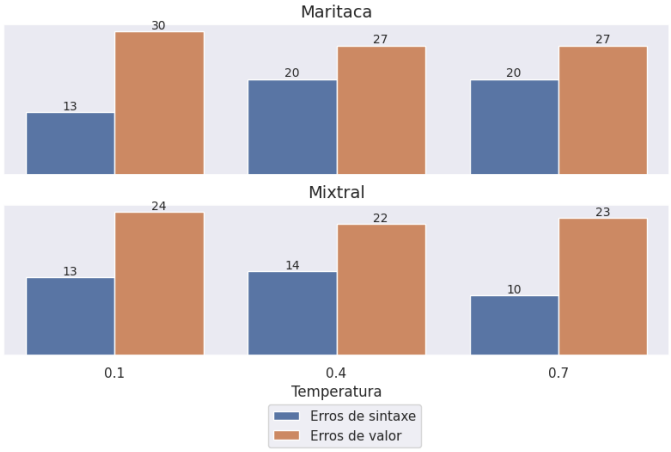


**Figura 4.5:** Quantidade de erros pelo tipo de erro.

Um outro fator para se levar em conta quando falamos de performance dos algoritmos da geração de texto é o *prompt*. Durante o trabalho, alguns modelos de prompts

[1] As instruções classificadas como Muito Baixa/Baixa exigiam somente filtros simples e ordenações, as classificadas como Moderada/Alta exigiam a construção de relacionamento entre as tabelas e aplicação de funções especificas da linguagem.

foram testados. Identificamos que prompts mais completos (no sentido de restrição da tarefa)eram mais precisos, principalmente no uso do Mixtral.

De forma prática, essa restrição é feita pela inclusão no prompt para qual sistema a o LLM deve gerar a instrução SQL (no caso o SQLite) e pela indicação direta que o resultado deve levar em conta estritamente as estruturas do banco de dados enviadas ao prompt.

O exemplo apresentado na Figura 4.6 demonstra como a construção do prompt influencia na geração pelo algoritmo. Enquanto no primeiro modelo é passada a tarefa e as informações necessárias para a conclusão, no segundo é passado explicitamente que a instrução deveria ser formulada para o SQLite e que os campos a serem usados nela deveriam estar obrigatoriamente contidos no prompt.

| | |
|---|---|
| Questão | Qual o cargo de Susan Eaton? |
| Instrução esperada | SELECT Title FROM DimEmployee WHERE FirstName = 'Susan' AND LastName = 'Eaton' |
| Instrução gerada pelo prompt geral | SELECT TOP 1 `Title` FROM `DimEmployee` WHERE `EnglishName` = 'Susan Eaton' |
| Problema | Uso da instrução TOP não está disponível no SQLite e coluna EnglishName não existe na tabela. |
| Instrução gerada pelo prompt restrito | SELECT Title FROM DimEmployee WHERE FirstName = 'Susan' AND LastName = 'Eaton' |

**Figura 4.6:** Exemplo da geração com diferentes prompts. O modelo usado foi o Mixtral sob uma temperatura de 0.7.

### 4.1.3 Geração de respostas

Como rotina em aplicações desse escopo, é plausível que em determinados momentos a solução se depare com informações faltantes para a construção da resposta. Buscando avaliar o comportamento diante dessas situações, realizamos testes em uma base de dados com 20 registros de questões/interações separados em proporção igual entre prompts com informações e sem informações. Os pontos importantes estão representados na Figura 4.7 e serão detalhados a seguir.

A análise do comportamento dos algoritmos gerativos em casos onde há ou não registro indicou características importantes para o pleno funcionamento da solução. Para os casos aonde a informação era fornecida identificamos dois comportamentos: (1) a resposta ser fornecida em língua inglesa e (2) a resposta ser omitida de forma proposital.

Na primeira situação, mesmo que a questão fosse fornecida em língua portuguesa, sua resposta era dada em língua inglesa. A análise que fizemos indicou que, a partir da informação passada ao prompt com a resposta da instrução SQL (que é em língua inglesa), o algoritmo entendia que a resposta também deveria ser em inglês. Sendo assim, identificamos a necessidade de inclusão de forma explícita no prompt que a resposta seja

sempre em português. A alteração surtiu o efeito esperado: as respostas, mesmo que a informação seja passada na língua inglesa, foram geradas em português.

Na segunda, o algoritmo, ao ser perguntado sobre uma informação que poderia ser classificada como pessoal, negava a informação na forma de resposta. Isso aconteceu somente no Maritaca, mas foi o suficiente para ajustarmos o prompt para os dois algoritmos. Dado que a solução lida com informações de dados abertos, essa camada de segurança não se torna necessária.

Em se tratando das questões fornecidas sem a informação necessária para resposta, tivemos comportamentos distintos para cada algoritmo. O Maritaca conseguiu interpretar esses prompts e de fato afirmou que não seria possível responder a pergunta por não ter os dados necessários. Já o Mixtral alternou entre a geração de textos sem sentido e a produção de informações não verídicas em algumas situações. Na tentativa de solucionar o problema, incluímos mais detalhes no prompt, mas sem um efeito consistente: em alguns momentos funcionava, em outros, não.

| | | |
|---|---|---|
| Maritaca | Questão | Em que cargo Suchitra Mohan esta? |
| | Informação recebida pelo prompt | [{'Title': 'Production Technician - WC60'}] |
| | Resposta | According to the information I have, Suchitra Mohan is currently a Production Technician - WC60 |
| Maritaca | Questão | De quanto é a renda anual do cliente Arthur Sanchez |
| | Informação recebida pelo prompt | [{'YearlyIncome': 20000.0}] |
| | Resposta | Não tenho informações específicas sobre o cliente Arthur Sanchez, pois não tenho acesso a informações pessoais. |
| Mixtral | Questão | Renda Anual cliente Carina Ramos |
| | Informação recebida pelo prompt | [] |
| | Resposta | Entendi |
| Mixtral | Questão | Qual o salário de Susan Eaton? |
| | Informação recebida pelo prompt | [] |
| | Resposta | O salário de Susan Eaton é $20. |

**Figura 4.7:** Exemplo da geração com e sem informação. Os modelos usados foram configurados sob uma temperatura de 0.7.

## 4.2 Discussão

Todos os modelos treinados neste trabalho para a tarefa de reconhecimento de intenção apresentaram uma boa performance dentro dos dados que tínhamos (Tabela 4.1). A diferença entre eles, como já apresentada anteriormente, baseia-se na complexidade e na base de dados com o qual forma treinados (Tabela 3.1).

Neste contexto, é relevante destacar que o BERTimbau, mesmo apresentando a arquitetura mais simples em comparação aos demais, obteve um $F_1$ de 91%, demonstrando-se como eficaz para a tarefa de reconhecimento de intenção. É justamente por causa de

sua arquitetura enxuta que o BERTimbau pode ser treinado em contextos mais limitados computacionalmente, tornando-o altamente viável para personalização em diversas bases de dados.

O alBERTina, por outro lado, é o algoritmo entre os usados neste trabalho que se destaca pela sua arquitetura mais complexa e que tem resultados melhores se comparado aos demais. Contudo, é importante notar que esse benefício vem acompanhado da necessidade de máquinas mais robustas para o processo de treinamento e uso, o que pode ser uma limitação para soluções de menor porte.

O XLM-RoBERTa, como os outros, é efetivo na tarefa e, diferentemente do alBERTina, não precisa de maquinas robustas para processamento, mas também não é tão portátil quanto o BERTimbau. Seu diferencial, entretanto, está em ser um modelo multilíngue (Tabela 3.1). Isso torna-o significativamente relevante em contextos onde a capacidade de lidar com múltiplos idiomas é necessária.

Para critério de comparação, demonstramos na Tabela 4.3 os tempos de execução médio para treinamento de cada modelo, bem como as épocas necessárias para convergirem para um estado estacionário. Como é possível observar, mesmo que o alBERTina apresente menos épocas para o estado estacionário, no total, ele demora mais para ser treinado.

Além de custo computacional, questões relacionadas a limitações orçamentarias devem ser levadas em conta. Os modelos mais complexos necessitaram infraestruturas mais robustas para serem processadas, e para isso no Google Colaboratory foram usadas máquinas disponíveis somente nos planos pagos.

| | Máquina utilizada | Tempo médio por época | Épocas necessárias | Tempo total |
|---|---|---|---|---|
| BERTimbau | V100 | 3s | **333** | 16m30s |
| alBERTina | A100 | **21s** | 279 | **1h37m30s** |
| XLM-RoBERTa | T4 | 4s | 249 | 16m29s |

**Tabela 4.3:** Avaliação comparativa da performance no treinado dos modelos de IR.

Em se tratando dos modelos gerativos e com base nos dados disponíveis e nas técnicas empregadas, a performance no geral do Mixtral foi superior quando comparada à do Maritaca nas tarefas de geração das instruções SQL (Tabela 4.2). A superioridade, nesse cenário, fica evidente pelo índice de acertos do modelo a questão solicitada, mas também é demonstrado pela capacidade de inferência para conhecimentos não fornecidos pelo prompt, nas instruções de complexidade moderada e alta.

O Maritaca, diante das mesmas questões e com os mesmos ajustes, não desempenhou um papel satisfatório. Na realidade, com o aumento da permissividade de inferência/aleatoriedade, era esperado que o algoritmo pudesse gerar consultas que articulassem as informações passadas pelo prompt e seu conhecimento prévio, o que não ocorreu.

Por outro lado, enquanto a dificuldade do Maritaca apareceu na geração da instrução SQL, a do Mixtral foi seguir as instruções contidas no prompt. O algoritmo diversas vezes apresentou informações erradas com base em dados que não foram fornecidas 4.5, mesmo sendo tendo sido apontado de forma explícita o uso exclusivo de informações do prompt. O mesmo problema não ocorre com o Maritaca. Em situações onde as informações fornecidas são insuficientes ou não existem, o algoritmo indica essa limitação em sua resposta.

Em resumo, no contexto deste trabalho, o Mixtral se apresenta melhor nas tarefas onde é exigido algum tipo de inferência e associação de relações previas aprendidas pelo algoritmo. Já o Maritaca se sai melhor na parte conversacional e de seguir as regras de prompt, oferecendo respostas mais precisas quando tem de fato as informações requeridas.

A discussão sobre o custo relativo ao uso dos algoritmos também deve ser considerada. Do ponto de vista computacional, o Maritaca, ao ser utilizado via API, apresenta um tempo menor de resposta ($\approx$ 20 s por chamada), enquanto o Mixtral, por ter sido executado em um ambiente com limitações computacionais, teve um tempo de resposta bem mais elevado ($\approx$ 3 min por chamada). Entretanto, do ponto de vista financeiro, o custo relativo ao uso da API do Maritaca se evidencia, já que a cobrança é feita por números de tokens, o que pode encarecer potenciais projetos que utilizam o algoritmo.

Capítulo 5

# Conclusão

Diante dos problemas inerentes às tarefas de text-to-SQL, buscamos propor uma solução viável, adaptativa e que atenda o propósito de ser uma interface entre língua natural e linguagem SQL. Entretanto, durante o processo de criação, algumas descobertas foram feitas e, a partir delas, ajustes forma necessários a fim de superar os desafios que a proposta impunha.

A superação da barreira linguística foi alcançada mediante o uso de LLMs multilíngues para tarefa de geração de instruções SQL. A questão da transparência foi atendida pelo uso de métodos de reconhecimento de intenção que restringem o escopo e permite ajustes quando necessários. Por fim, a necessidade de customização foi atendida ao empregar técnicas de engenharia de prompt, incorporando informações das tabelas diretamente às instruções enviadas ao modelo de linguagem, eliminando a necessidade de treinamentos prévios.

O sistema resultante se situa, âmbito das soluções text-to-SQL, como uma solução transparente, customizável e econômica, habilitada para trabalhar com consultas simples. Aponta direções para estudos aprofundados nas técnicas de construção de prompt e apresenta oportunidades dentro de sua estrutura para a automatização de tarefas que ainda demandam intervenção manual.

Sua aplicação em outras bases de dados, contudo, só será possível a partir da criação de tabelas com o dicionário de dados do banco de dados e uma amostra das questões, com intenções e tabelas associadas para o treinamento do algoritmo de reconhecimento de intenções.

Dentro dessa temática, só podemos afirmar que os resultados serão satisfatórios quando aplicados dentro de contextos onde a informação solicitada possa ser extraída a partir de instruções SQL de consulta de dados (DQL – *Data Query Language*) simples, de preferência que utilizem uma única tabela.

Essa, inclusive, é uma das limitações deste trabalho: somente consultas de recuperação de dados foram avaliadas, isto é, nenhum dos outros tipos[1] da linguagem SQL foram avaliados. Além disso, as consultas avaliadas têm como característica retornarem apenas um registro, o que impossibilitou a avaliação do comportamento da solução em situações em que seriam obtidos múltiplos registros.

Outra limitação diz respeito à complexidade das consultas avaliadas. Devido à quantidade limitada de dados classificadas como consultas de complexidade alta e muito alta, não foi possível avaliar com precisão o desempenho da solução dentro desses escopos mais complexos.

O tempo para execução completa do pipeline também é uma restrição da solução. Na combinação com melhor desempenho (utilizando alBERTina como modelo de reconhecimento de intenções e Mixtral para geração de instruções SQL), o tempo médio para execução completa do pipeline chega a 4 minutos nas infraestruturas testadas (com GPUs NVIDIA A100).

O uso de uma quantidade restrita de LLMs igualmente se colocou como obstáculo. Neste contexto, restrições relacionadas a custos orçamentários e computacionais impediram a exploração de outros algoritmos.

A pesquisa desenvolvida estabelece fundamentos para investigações futuras. Nossa intenção é progredir no estudo do tema a partir da experimentação de técnicas para maior refinamento dos dados enviados ao prompt.

Além disso, o estudo aprofundado das estruturas dos algoritmos de LLM visando à criação de prompts mais eficientes também se coloca como etapa para obtenção de melhores resultados. Nesse ponto, devemos incluir também prompts específicos para atender aos demais tipos de linguagem SQL, aumentando o escopo de atuação da solução.

O resultado esperado desses estudos futuros é o desenvolvimento de uma interface de código aberto que será capaz de responder a solicitações que demandem ações de consulta (DQL) bem como ações de criação de estrutura (DDL) em banco de dados, utilizando linguagem natural. Idealmente, será transparente, plenamente adaptável e em língua portuguesa, capaz de atender a todas as ações aplicáveis a um banco de dados, isto é, não só consultas mas, também, execuções, e em qualquer banco de dados relacional, de qualquer complexidade.

[1] DDL (Data Definition Language), para definição de dados; DML (Data Manipulation Langague), para manipulação de dados; DTL (Data Transaction Language), para transação de dados e DCL (Data Control Language), para controle de acesso aos dados

# Referências


Abadi, Martín et al. (2016). *TensorFlow: A system for large-scale machine learning*. arXiv: 1605.08695 [cs.DC].

Androutsopoulos, I., G. D. Ritchie e P. Thanisch (1995). *Natural Language Interfaces to Databases - An Introduction*. arXiv: cmp-lg/9503016 [cmp-lg].

Conneau, Alexis et al. (2020). *Unsupervised Cross-lingual Representation Learning at Scale*. arXiv: 1911.02116 [cs.CL].

Deng, Naihao, Yulong Chen e Yue Zhang (out. de 2022). "Recent Advances in Text-to-SQL: A Survey of What We Have and What We Expect". Em: *Proceedings of the 29th International Conference on Computational Linguistics*. Ed. por Nicoletta Calzolari et al. Gyeongju, Republic of Korea: International Committee on Computational Linguistics, pp. 2166–2187. URL: https://aclanthology.org/2022.coling-1.190.

Devlin, Jacob et al. (jun. de 2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". Em: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. por Jill Burstein, Christy Doran e Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

Eliseev, Artyom e Denis Mazur (2023). *Fast Inference of Mixture-of-Experts Language Models with Offloading*. arXiv: 2312.17238 [cs.LG].

Gao, Dawei et al. (2023). *Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation*. arXiv: 2308.15363 [cs.DB].

Jiang, Albert Q. et al. (2024). *Mixtral of Experts*. arXiv: 2401.04088 [cs.LG].

Khan, Vasima e Tariq Azfar Meenai (2021). "Pretrained Natural Language Processing Model for Intent Recognition (BERT-IR)". Em: *Human-Centric Intelligent Systems* 1 (3-4), pp. 66–74. ISSN: 2667-1336. DOI: 10.2991/hcis.k.211109.001. URL: https://doi.org/10.2991/hcis.k.211109.001.

Kingma, Diederik P. e Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].

Liu, Yinhan et al. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv: 1907.11692 [cs.CL].

Maritaca (jan. de 2024). URL: https://www.maritaca.ai/.

Mielke, Sabrina J. et al. (2021). *Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP*. arXiv: 2112.10508 [cs.CL].

Rodrigues, João et al. (2023). *Advancing Neural Encoding of Portuguese with Transformer Albertina PT-**. arXiv: 2305.06721 [cs.CL].

Shankar, Arun (nov. de 2023). *Architectural patterns for text-to-SQL: Leveraging LLMS for enhanced BigQuery Interactions*. URL: https://medium.com/google-cloud/architectural-patterns-for-text-to-sql-leveraging-llms-for-enhanced-bigquery-interactions-59756a749e15.

Souza, Fábio, Rodrigo Nogueira e Roberto Lotufo (2020). "BERTimbau: Pretrained BERT Models for Brazilian Portuguese". Em: *Intelligent Systems*. Ed. por Ricardo Cerri e Ronaldo C. Prati. Cham: Springer International Publishing, pp. 403–417. ISBN: 978-3-030-61377-8.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". Em: *Journal of Machine Learning Research* 15.56, pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Wagner Filho, Jorge A. et al. (mai. de 2018). "The brWaC Corpus: A New Open Resource for Brazilian Portuguese". Em: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Ed. por Nicoletta Calzolari et al. Miyazaki, Japan: European Language Resources Association (ELRA). URL: https://aclanthology.org/L18-1686.

Yu, Tao et al. (out. de 2018). "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task". Em: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Ed. por Ellen

Riloff et al. Brussels, Belgium: Association for Computational Linguistics, pp. 3911–3921. DOI: 10.18653/v1/D18-1425. URL: https://aclanthology.org/D18-1425.