Gabriel de Melo Evangelista

USO DE LLM OPEN SOURCE NA TRADUÇÃO DE LINGUAGEM NATURAL PARA SQL

Trabalho de Graduação

Universidade Federal de Pernambuco
graduacao@cin.ufpe.br
portal.cin.ufpe.br/graduacao/

RECIFE
2023

Universidade Federal de Pernambuco
Centro de Informática
Graduação em Engenharia da Computação

Gabriel de Melo Evangelista

USO DE LLM OPEN SOURCE NA TRADUÇÃO DE LINGUAGEM NATURAL PARA SQL

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: *Luciano de Andrade Barbosa*

RECIFE
2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Evangelista, Gabriel de Melo.
Uso de LLM Open Source na tradução de linguagem natural para SQL / Gabriel de Melo Evangelista. - Recife, 2023.
56 p. : il., tab.

Orientador(a): Luciano de Andrade Barbosa
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2023.
9.5.

1. Linguagem Natural. 2. Modelo de Linguagem. 3. Geração de Texto. 4. Ajuste Fino Supervisionado. 5. Geração de Consulta SQL. I. Barbosa, Luciano de Andrade. (Orientação). II. Título.

000 CDD (22.ed.)

Trabalho de Graduação apresentado por **Gabriel de Melo Evangelista** ao programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **Uso de LLM Open Source na tradução de linguagem natural para SQL**, orientada pelo **Prof. Luciano de Andrade Barbosa** e aprovada pela banca examinadora formada pelos professores:

Prof. Tsang Ing Ren
Centro de Informática/UFPE

RECIFE
2023

Eu dedico esse trabalho a minha família, amigos e professores que me deram todo o apoio durante a minha jornada.

Agradecimentos

Neste momento significativo de minha jornada acadêmica, gostaria de expressar minha sincera gratidão às pessoas e instituições que contribuíram para a realização deste projeto e para minha formação como um todo.

Em primeiro lugar, gostaria de prestar uma homenagem especial à minha amada mãe, cujo apoio e amor inabaláveis foram uma fonte constante de inspiração. Mesmo que ela não esteja mais entre nós para testemunhar a conclusão deste curso, sei que sua influência positiva e seu incentivo contínuo sempre estarão comigo. Ela foi minha maior incentivadora e dedicarei minhas conquistas a ela.

Aos meus colegas de faculdade, muitos dos quais se tornaram amigos próximos, agradeço por compartilharem comigo essa jornada de aprendizado e crescimento. Nossa camaradagem e apoio mútuo tornaram esses anos de estudo inesquecíveis, e espero levar essas amizades para toda a vida.

Ao meu estimado professor orientador, Luciano de Andrade, expresso minha profunda gratidão. Sua orientação, experiência e apoio foram cruciais para o desenvolvimento deste projeto de pesquisa. Sua dedicação em me auxiliar a compreender os desafios e alcançar os objetivos desta dissertação foi inestimável, e sou grato por sua orientação ao longo desta jornada acadêmica.

Por fim, gostaria de agradecer ao Centro de Informática e à Universidade Federal de Pernambuco por fornecerem a estrutura necessária durante a minha graduação. Agradeço pelo acesso a recursos, bibliotecas e laboratórios que enriqueceram minha formação acadêmica.

Cada um desses agradecimentos é uma manifestação sincera de gratidão àqueles que fizeram parte desta jornada. Suas contribuições e apoio foram essenciais para a conclusão bem-sucedida deste trabalho.

Waste no more time arguing what a good man should be. Be one.

—MARCUS AURELIUS

Resumo

Modelos de linguagem recentes têm demonstrado uma grande capacidade de entender e gerar texto em linguagem natural. Além disso, com a florescente quantidade de dados armazenados em bases relacionais, é crescente o número de consultas e acessos a esses dados. Porém, muitos usuários não estão aptos a consultar os dados na linguagem apropriada, o *Structured Query Language* (SQL). Portanto, neste Trabalho de Graduação, de posse do conjunto de dados *Spider* ([29]), o primeiro conjunto de dados de análise semântica complexa e interdomínio de *text-to-SQL*, propomos explorar o potencial dos *large language models* de código aberto na tradução de linguagem natural para SQL (*text-to-SQL*) a fim de democratizar o acesso aos dados.

Palavras-chave: Linguagem Natural, Modelo de Linguagem, Geração de Texto, Ajuste Fino Supervisionado, Geração de Consulta SQL.

Abstract

Recent advances in language models have demonstrated a significant ability to understand and generate text in natural language. Furthermore, with the burgeoning amount of data stored in relational databases, the number of queries and accesses to this data is on the rise. However, many users are not proficient in querying data using the appropriate language, Structured Query Language (SQL). Therefore, in this undergraduate thesis, utilizing the Spider dataset ([29]), the first dataset for complex cross-domain semantic analysis of text-to-SQL, we propose to explore the potential of open-source large language models in translating natural language into SQL (text-to-SQL) with the aim of democratizing data access.

Keywords: Natural Language, Large Language Models, Text Generation, Supervised Fine-tuning, SQL query generation.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

Sumário

1

Introdução

Nos últimos anos, o interesse pelas aplicações de modelos de linguagem apresenta crescimento significativo resultado do desenvolvimento acelerado de técnicas no campo de Processamento de Linguagem Natural (PLN). Essas aplicações, sendo a principal delas o ChatGPT, surpreendem pela alta capacidade de geração de textos semelhantes à textos humanos. O ChatGPT inovou pela sua habilidade notável em compreender e gerar texto em linguagem natural de maneira coesa e convincente. Sua capacidade de resposta contextualmente relevante e a natureza fluida de suas interações com os usuários impressionaram, aproximando-se cada vez mais da comunicação humana. Além disso, sua versatilidade é notável, pois pode ser aplicado em uma ampla gama de tarefas, desde responder perguntas até auxiliar na criação de conteúdo e até mesmo na programação, tornando-se uma ferramenta poderosa para diversas aplicações.

Porém, o ChatGPT é um modelo de código fechado, que possui diversas limitações. Essas incluem uma compreensão de contexto limitada, tendência a respostas incorretas ou inseguras, sensibilidade ao viés presente nos dados de treinamento, falta de conhecimento atualizado, geração de conteúdo inadequado, potencial para uso inadequado, necessidade de ajuste fino personalizado e custos associados ao seu uso.

Além do ChatGPT, há um notável crescimento no desenvolvimento de modelos de linguagem de código aberto [32]. Esses modelos, muitas vezes treinados em grandes conjuntos de dados, têm se destacado por sua acessibilidade e flexibilidade. Ao contrário de modelos proprietários, eles estão disponíveis para a comunidade de desenvolvedores e pesquisadores, promovendo uma maior democratização do conhecimento e inovação.

Os modelos de linguagem de código aberto, como BERT [10], *Generative Pre-trained Transformer* (GPT)-2 e seus sucessores, têm sido aplicados em uma ampla variedade de domínios, desde tradução automática até análise de sentimentos, respondendo a perguntas e até mesmo gerando código de programação. Eles servem como uma base sólida para a criação de soluções personalizadas, permitindo o refinamento (*fine-tuning*) para tarefas específicas.

No contexto atual, a utilização de bancos de dados relacionais desempenha um papel central em inúmeras aplicações, desde sistemas de gerenciamento de dados empresariais até aplicativos de consumo diário. O *Structured Query Language* (SQL) é a linguagem padrão para interagir com esses bancos de dados e, consequentemente, o conhecimento proficiente em SQL tornou-se um ativo crucial para desenvolvedores, analistas de dados e profissionais de TI. À medida que a complexidade das bases de dados e a demanda por análises de dados avançadas aumentam, a necessidade de traduzir com eficiência linguagem natural em consultas SQL é cada vez mais premente. Nesse cenário, a capacidade de um *Large Language Model* (LLM) para tradução *text-to-SQL* oferece uma solução promissora, potencialmente democratizando o acesso a bancos de dados e simplificando a interação com eles, tornando-a mais intuitiva e acessível a um público mais amplo.

Com isso, este trabalho tem como objetivo a pesquisa e desenvolvimento da aplicação de modelos de linguagem para gerar consultas *SQL* de usuários, utilizando modelos de linguagem de código aberto e técnicas de refinamento. Além disso, esses modelos serão comparados com o ChatGPT e o modelo de estado da arte *SQLCoder*.

2

Fundamentos

Com o objetivo de tornar a compreensão deste estudo mais acessível, os termos e conceitos empregados ao longo do projeto serão introduzidos e brevemente explicados ao longo deste capítulo.

2.1 *Large Language Models*

Com a criação dos *transformers*, surgiram os LLM, modelos baseados em *transformers* e pré treinados utilizando conjuntos de dados textuais gigantes de *sites* como *Wikipedia*, *GitHub*, entre outros. Com isso, esses modelos são capazes de entender as complexidades da linguagem humana e de resolver as mais diversas tarefas, como Q&A, análise de sentimentos, extração de informações, geração de texto, entre outros.

Porém, para resolverem tarefas específicas, como geração de código, os modelos devem ser refinados utilizando conjuntos de dados específicos. O objetivo do refinamento é adaptar a compreensão linguística geral do modelo pré-treinado à tarefa específica em questão. Neste projeto, serão utilizados os LLMs Falcon e Llama 2 para refinamento descritos nas subseções abaixo.

2.1.1 Tokenização

A tokenização, no contexto da linguística computacional e processamento de linguagem natural, é um processo essencial que envolve a divisão de um texto em unidades linguísticas mais básicas, chamadas *tokens*. Esses *tokens* podem ser palavras individuais, pontuações, caracteres ou até mesmo frases, dependendo do nível de granularidade desejado. A tokenização desempenha um papel fundamental em tarefas de processamento de linguagem, como análise morfológica, análise sintática e modelagem de linguagem, uma vez que prepara o texto de entrada para

ser processado de maneira mais eficaz por algoritmos e modelos. Uma abordagem precisa de tokenização é crucial para garantir a qualidade e a precisão das análises linguísticas e das aplicações que dependem delas.

No contexto dos LLM, a técnica de tokenização o *Byte-Pair Encoding* (BPE)[13] é amplamente adotada, como por exemplo pelos modelos Falcon e Llama que serão descritos nas subseções 2.1.2.1 e 2.1.2.2. O BPE é um método de tokenização de "subpalavra"que divide o texto em unidades menores, frequentemente chamadas de "subpalavras"ou "*tokens* subpalavra", ao invés de dividir o texto em palavras completas. Ele funciona identificando sequências de caracteres frequentes e fundindo-as em novos *tokens* subpalavra, o que permite que o modelo lide eficazmente com palavras raras e compreenda subpalavras e morfologia. O BPE é particularmente útil para idiomas com morfologia complexa e para lidar com palavras que não estão presentes no vocabulário pré-definido do modelo, tornando-o uma escolha valiosa para tarefas de geração de texto e tradução automática em ambientes multilíngues.

2.1.2 *Transformers*

Os modelos *Transformers* representam um avanço significativo na capacidade de processamento de linguagem natural (NLP), revolucionando a maneira como as máquinas entendem e geram texto. Os modelos *Transformers* [25] se destacam por sua habilidade em capturar relacionamentos entre palavras distantes entre si e nuances semânticas em textos extensos. A arquitetura do *transformer*, observada na Figura 2.1, é dividida em *encoder*, esquerda da figura, e *decoder*, direita da figura. O *encoder* processa a entrada e captura informações contextuais em representações intermediárias, enquanto o *decoder* utiliza essas representações para gerar saídas, como traduções ou respostas. O *encoder* realiza múltiplas camadas de *self-attention* para calcular as relações entre todas as palavras do texto de entrada. Cada camada de *self-attention* permite que o modelo atribua diferentes níveis de importância às palavras, considerando a relação contextual entre elas. As representações resultantes, uma para cada palavra no texto, contêm informações abrangentes sobre o contexto local e global da sentença original.

Por outro lado, o *decoder* utiliza as representações geradas pelo *encoder* para produzir saídas sequenciais. Similar ao *encoder*, o *decoder* empilha camadas de *self-attention* e camadas totalmente conectadas. No entanto, o *decoder* também incorpora uma máscara de atenção direcional, que garante que, durante a geração, o modelo considere apenas as palavras anteriores na sequência de saída. Isso é crucial para evitar vazamento de informações futuras durante a geração sequencial.

Em conjunto, o *encoder* e *decoder* nos modelos Transformers permitem que as informações contextuais sejam capturadas e utilizadas tanto na compreensão de entrada quanto na

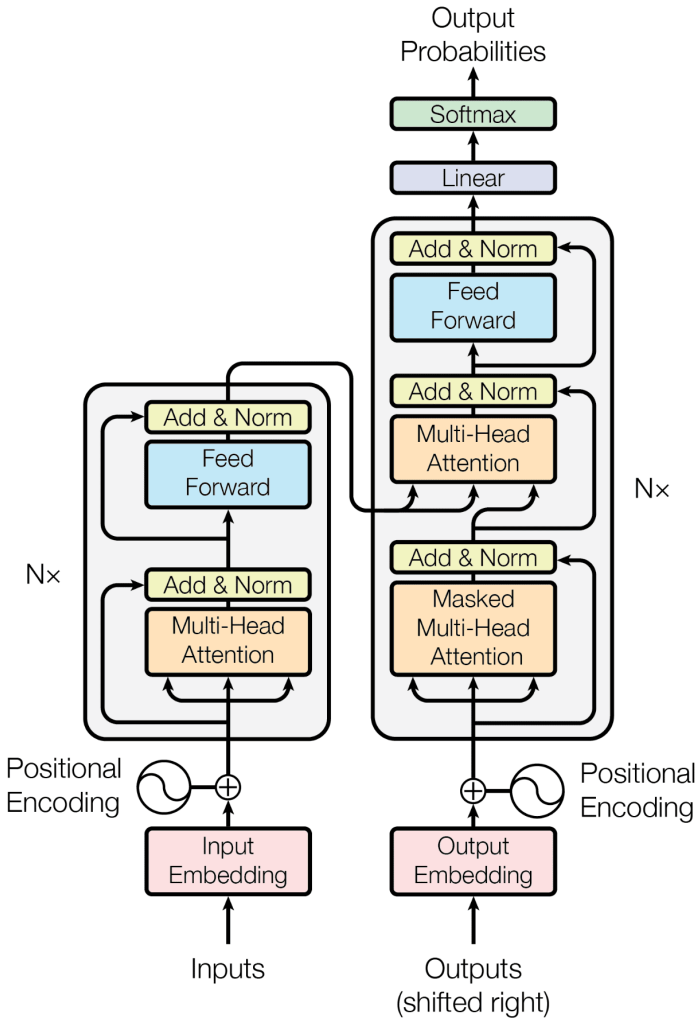

Figura 2.1: Arquitetura do *Transformers*

geração de saída. Um dos exemplos mais proeminentes dos modelos *Transformers* é o GPT [21], que utiliza pré-treinamento em grandes conjuntos de dados para aprender representações linguísticas abrangentes, e pode ser ajustado para tarefas específicas com relativa facilidade. A aplicação dos modelos *Transformers* na tradução de linguagem natural para *SQL* apresenta um cenário promissor, onde a capacidade de capturar nuances semânticas complexas pode levar a uma geração mais precisa e contextualmente informada de consultas *SQL* a partir de instruções em linguagem humana.

2.1.2.1 *Falcon*

O *Falcon* [2] é um LLM de código aberto com arquitetura *decoder-only*, modelos que utilizam apenas o *decoder* do *transformer* atingindo performance de generalização superior

[26], construído pela *Technology Innovation Institute* (TII) em 2023. Foram disponibilizados 2 modelos com 40 e 7 bilhões de parâmetros, chamados de *Falcon*-40B e *Falcon*-7B, respectivamente. No lançamento, o *Falcon*-40B se tornou o melhor modelo de código aberto no *ranking* do *HuggingFace*(Figura 2.2), já o *Falcon*-7B superou diversos modelos de código aberto(Figura 2.3), como o MPT-7B [22], LLaMA [24], Dolly-v2-7B [6], entre outros.

Model	Revision	Average	ARC (25-shot)	HellaSwag (10-shot)	MMLU (5-s
tiiuae/falcon-40b	main	60.4	61.9	85.3	52.7
ausboss/llama-30b-supercot	main	59.8	58.5	82.9	44.3
llama-65b	main	58.3	57.8	84.2	48.8
MetaIX/GPT4-X-Alpasta-30b	main	57.9	56.7	81.4	43.6
digitous/Alpacino30b	main	57.4	57.1	82.6	46.1
Aeala/GPT4-x-AlpacaDente2-30b	main	57.2	56.1	79.8	44
TheBloke/dromedary-65b-lora-HF	main	57	57.8	80.8	50.8
TheBloke/Wizard-Vicuna-13B-Uncensored-HF	main	57	53.6	79.6	42.7
llama-30b	main	56.9	57.1	82.6	45.7

Figura 2.2: *Falcon*-40B *Ranking* em Maio 2023

Model	Revision	Average	ARC (25-shot)	HellaSwag (10
tiiuae/falcon-7b	main	48.8	47.9	78.1
mosaicml/mpt-7b	main	48.6	47.7	77.7
tiiuae/falcon-7b-instruct	main	48.4	45.9	70.8
chainyo/alpaca-lora-7b	main	48.4	45.5	75.2
dvruette/oasst-gpt-neox-20b-1000-steps	main	47.8	48.2	74.6
llama-7b	main	47.6	46.6	75.6
databricks/dolly-v2-7b	main	44.4	43.7	69.3
EleutherAI/gpt-j-6b	main	44.3	41.4	67.6
stabilityai/stablelm-tuned-alpha-7b	main	38.3	31.9	53.6

Figura 2.3: *Falcon*-7B *Ranking* em Maio 2023

Por motivos computacionais, neste projeto será utilizado apenas o *Falcon*-7B, pois necessita de menos memória para ser carregado. Este modelo foi treinado com 1500B de *tokens* do *Refined-Web*[20], um conjunto de dados filtrado e sem duplicidade de alta qualidade. A Tabela 2.1 mostra a divisão dos dados utilizados.

2.1.2.2 Llama 2

O Llama 2 [24] é uma coleção de LLMs pré-treinadas e refinadas variando de 7 a 70B de parâmetros desenvolvidas pela *Meta*. O modelo anterior, chamado Llama [23], mostrou que é possível a criação de um modelo de estado da arte utilizando apenas dados públicos, obtendo performance superior ao GPT, que utiliza dados privados. Em relação ao Llama, o Llama 2

Tabela 2.1: Fontes de Dados de Treinamento para o *Falcon*-7B

Fonte de Dados	Fração	Número de *Tokens*	Fontes
RefinedWeb-English	79%	1185B	Extensa Coleta de Dados na *Web*
Livros	7%	110B	
Conversações	6%	85B	*Reddit, StackOverflow, HackerNews*
Códigos	3%	45B	
RefinedWeb-French	3%	45B	Extensa Coleta de Dados na *Web*
Conteúdo Técnico	2%	30B	*arXiv, PubMed, USPTO, etc.*

apresentou diversas melhorias como um novo conjunto de dados públicos 40% maior, o tamanho do contexto do modelo foi dobrado e foi adotado o *grouped-query attention* [1]. Com isso, no lançamento o Llama 2 70B superou o Falcon e se tornou o melhor modelo no *ranking* do *HuggingFace* (Figura 2.4).

Model	Average
meta-llama/Llama-2-70b-hf	67.35
huggyllama/llama-65b	64.23
llama-65b	64.23
circulus/Llama-2-13b-orca-v1	62.91
lmsys/vicuna-13b-v1.5	61.69
llama-30b	61.68
lmsys/vicuna-13b-v1.5	61.63
tiiuae/falcon-40b	61.48

Figura 2.4: *Llama 2 Ranking* em Agosto 2023

Assim como para o modelo Falcon, por questões computacionais, neste projeto será utilizado apenas o Llama 2 com 7B de parâmetros. Este modelo foi treinado com 2T de tokens, os dados são até setembro de 2022 e incluem conjuntos de dados de instruções e mais de 1M de exemplos anotados por humanos. Além disso, nenhum desses dados incluem dados de usuário do *Meta*.

2.2 Técnicas de Refinamento de *Large Language Models*

Nesta seção, as técnicas de refinamento *QLoRA* [9] e *Low Rank Adapters* (LoRA) [15] serão descritas.

2.2.1 LoRA

Com o decorrer do tempo, os LLM estão se tornando cada vez maior. Por exemplo, o modelo GPT-3 possui 176B de parâmetros, já o *PaLM* [3] possui cerca de 540B. Além disso, observamos uma tendência em direção a modelos ainda maiores.

Consequentemente, esses modelos são difíceis de serem executados em dispositivos de fácil acesso. Por exemplo, para realizar inferências no *BLOOM*-176B, são necessárias 8 *Graphics Processing Unit* (GPU)s A100 de 80GB cada, já para realizar o refinamento são necessárias 72 dessas GPUs.

Devido a necessidade de várias GPUs desses modelos gigantes, diversas tecnologias foram desenvolvidas para diminuir o tamanho do modelo enquanto preserva sua performance. Foi proposta então a técnica LoRA [15], que congela os pesos pré-treinados do modelo e adiciona pares de matrizes de decomposição de posto (*rank decomposition matrices*) aos pesos já existentes, treinando apenas nos novos pesos e diminuindo consideravelmente o número de parâmetros treináveis.

2.2.2 *QLoRA*

A técnica *QLoRA* [9] reduz ainda mais o uso de memória suficientemente para permitir o treinamento de LLMs em GPUs mais acessíveis preservando a performance do modelo. O modelo *Guanaco*, por exemplo, alcançou 99.3% da performance do ChatGPT no *Vinuca benchmark* após apenas 24 horas de refinamento em uma única GPU.

O *QLoRA* utiliza quantização de 4 bits para comprimir o modelo pré-treinado. Os parâmetros são então congelados e uma porção relativamente pequena dos parâmetros treináveis são adicionados ao modelo na forma de LoRA, previamente descrita.

3

Dados e Método de Avaliação

3.1 Conjunto de Dados

A análise semântica é uma das tarefas mais importantes no PLN, pois requer ambos o entendimento das sentenças de linguagem natural e o mapeamento delas para consultas executáveis como códigos *Python*, consultas SQL, entre outros. Para resolução dessa tarefa neste projeto, o conjunto de dados utilizado foi o *Spider* [29], que é um conjunto de dados de análise semântica e *text-to-SQL* em larga escala, complexo e de múltiplos domínios. Consistindo de 10181 perguntas em inglês e 5693 consultas SQL complexas únicas em 200 base de dados com múltiplas tabelas, cobrindo 138 domínios diferentes.

Existem diversos conjuntos de dados semelhantes como o Geo [31], ATIS [8], Academic [16] e WikiSQL [34]. Porém, como é possível observar na Figura 3.1:

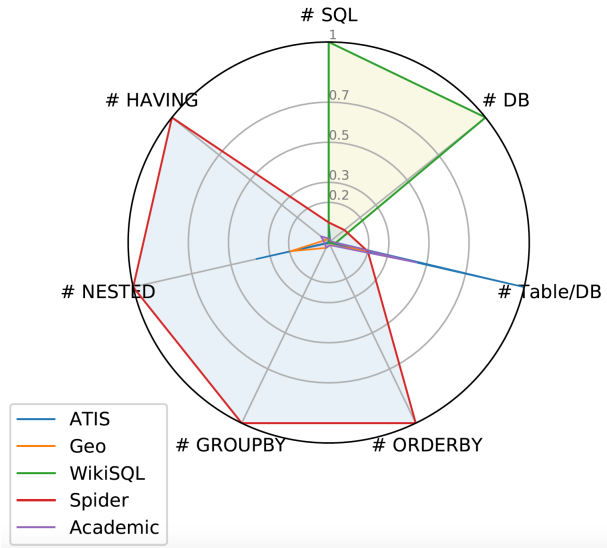

Figura 3.1: Comparação de Conjuntos de Dados

- ATIS, Geo, Academic: Cada um dos conjuntos de dados possuem apenas uma única base de dados, muitos deles com menos de 500 consultas SQL únicas. Com isso, os modelos treinados nestes conjuntos de dados não generalizam para novos domínios.

- WikiSQL: O número de consultas e tabelas SQL é substancialmente maior, porém as consultas são simples, compostas apenas de cláusulas *SELECT* e *WHERE*. Além disso, cada base de dados possui apenas uma tabela sem chaves estrangeiras. Dessa forma, os modelos treinados no WikiSQL apesar de generalizarem a novos domínios, não conseguem lidar com consultas mais complexas (e.g. consultas aninhadas, *ORDER BY*, ou *GROUP BY*) e bases de dados com múltiplas tabelas e chaves estrangeiras.

- Spider: Ocupa a maior área no gráfico, pois o conjunto de dados é:

    - Em larga escala: Apresenta mais de 10000 perguntas e cerca de 6000 consultas SQL únicas.
    - Complexo: Todas as bases de dados apresentam múltiplas tabelas conectadas por chaves estrangeiras. Além disso, a maior parte das consultas cobre os principais componentes SQL incluindo consultas aninhadas, *GROUP BY*, *ORDER BY* e *HAVING*.
    - De Múltiplos Domínios: Consiste de 200 bases de dados complexas que abrangem 138 domínios diferentes.

O conjunto de dados final é dividido em treino, desenvolvimento e teste com 7000, 1034 e 2147 consultas cada, respectivamente. Além disso, para fins de avaliação, cada consulta possui uma classificação em 4 níveis de complexidade: fácil, médio, difícil e muito difícil. O nível é definido a partir do número de componentes SQL, condições e uniões, dessa forma as consultas com mais palavras-chave SQL (*SELECT*, *JOIN*, *HAVING*, consultas aninhadas, etc) são consideradas mais difíceis. Nas Figuras 3.2, 3.3, 3.4, 3.5 é possível observar exemplos dos níveis de complexidade.

What is the number of cars with more than 4 cylinders?

```
SELECT COUNT(*)
FROM cars_data
WHERE cylinders > 4
```

Figura 3.2: Consulta SQL nível fácil

For each stadium, how many concerts are there?

```
SELECT T2.name, COUNT(*)
FROM concert AS T1 JOIN stadium AS T2
ON T1.stadium_id = T2.stadium_id
GROUP BY T1.stadium_id
```

Figura 3.3: Consulta SQL nível médio

Which countries in Europe have at least 3 car manufacturers?

```
SELECT T1.country_name
FROM countries AS T1 JOIN continents
AS T2 ON T1.continent = T2.cont_id
JOIN car_makers AS T3 ON
T1.country_id = T3.country
WHERE T2.continent = 'Europe'
GROUP BY T1.country_name
HAVING COUNT(*) >= 3
```

Figura 3.4: Consulta SQL nível difícil

What is the average life expectancy in the countries where English is not the official language?

```
SELECT AVG(life_expectancy)
FROM country
WHERE name NOT IN
   (SELECT T1.name
    FROM country AS T1 JOIN
    country_language AS T2
    ON T1.code = T2.country_code
    WHERE T2.language = "English"
      AND T2.is official = "T")
```

Figura 3.5: Consulta SQL nível muito difícil

3.2 Método de Avaliação

A avaliação de modelos *text-to-SQL* é um problema complexo: é necessário entender se a consulta SQL prevista possui o mesmo resultado que a consulta correta para todos os bancos de dados possíveis.

Diversos métodos formais foram desenvolvidos para avaliar com confiança a equivalência de consultas SQL. Para isso, eles buscam provas de equivalência após transformar as consultas em outras representações como UniNomial [5], U-semiring [4] e K-relations [14]. Porém, essas representações não abrangem todas as operações que os modelos de *text-to-SQL* podem usar como comparações de números *float* e ordenações.

Para solucionar esses problemas, o *Test Suite SQL Eval* [33] apresenta uma abordagem inovadora para avaliar a capacidade semântica desses modelos de maneira mais eficiente e precisa. Os autores utilizam a técnica de *fuzzing* para criar a suíte de testes utilizada na avaliação dos modelos. O *fuzzing* envolve a geração de um grande número de bancos de dados aleatórios que satisfaçam as restrições de tipo de entrada do programa de referência. Esses bancos de dados aleatórios servem como base para a suíte de testes.

A partir dos bancos de dados aleatórios gerados, os autores selecionam uma pequena fração que consegue distinguir as consultas vizinhas da consulta de referência. Isso significa que esses bancos de dados selecionados produzem resultados diferentes ao executar as consultas vizinhas em comparação com a consulta de referência. O conjunto desses bancos de dados selecionados é chamado de "suíte de testes destilada".

A suíte de testes é projetada para refletir uma cobertura abrangente de código e alta qualidade nos testes. Ela visa abranger uma ampla gama de cenários e tipos de consultas para garantir que os modelos sejam avaliados de forma completa.

Para avaliar os modelos, os autores verificam as denotações das consultas previstas pelo modelo na suíte de testes destilada. Isso permite a eles aproximar eficientemente a precisão semântica dos modelos. Dessa forma, a métrica de Acurácia de Correspondência Exata é

calculada a partir da execução das consultas reais e previstas na suíte de testes destilada. Para isso, o valor 1 é atribuído para cada par de consultas real e predita em que ambas geram a mesma saída em toda a suíte de testes. Em seguida, soma-se todos os 1s e divide-se pelo número total de previsões.

Ao utilizar essa abordagem, os autores conseguem criar uma suíte de testes compacta e de alta qualidade para avaliar o desempenho dos modelos na compreensão e geração de consultas SQL a partir de entradas em linguagem natural. Essa abordagem foi utilizada para avaliar 21 modelos submetidos ao *ranking* do *Spider* e comprovou-se que ela está sempre correta. Dessa forma, ela tornou-se a métrica oficial dos conjuntos de dados *Spider*, *SParC* [30] e *CoSQL* [28]. Com isso, neste projeto os modelos serão avaliados utilizando esta abordagem.

4

Trabalhos Relacionados

Neste capítulo, são apresentados e discutidos os trabalhos relevantes na área de tradução de linguagem natural para *SQL*, com ênfase na utilização de LLM. A revisão da literatura abrange abordagens tradicionais, técnicas baseadas em exemplos e o estado da arte com LLMs.

4.1 Abordagens Tradicionais

A tarefa de gerar consultas *SQL* a partir de perguntas tem sido estudado pelas comunidades de PLN e bancos de dados desde a década de 70 [7].

Abordagens tradicionais para a tradução de linguagem natural para *SQL* frequentemente utilizam correspondência de padrões, técnicas baseadas em gramática ou representações intermediárias para mapear a estrutura da frase para uma consulta *SQL*. Essas abordagens iniciais tendem a ser limitadas pela complexidade da linguagem natural e muitas vezes falham em capturar nuances semânticas. Trabalhos como *Natural Language Interfaces to Databases* [19] exploraram essas abordagens e destacaram a necessidade de maior flexibilidade.

4.2 Abordagens baseadas em Exemplos

Abordagens baseadas em exemplos buscam traduzir frases em linguagem natural para *SQL* usando um conjunto de exemplos de pares de frases e consultas *SQL* correspondentes. Embora eficazes em algumas situações, essas abordagens sofrem com a necessidade de um conjunto de dados abrangente e representativo. Fariha et al. [12] apresentaram um método baseado em exemplos que enfrentou desafios ao lidar com variações de estrutura e contexto.

4.3 Abordagens baseadas em *Large Language Models*

A ascensão dos LLM revolucionou a tradução de linguagem natural para *SQL*. Modelos como o GPT-3 têm demonstrado habilidades impressionantes na geração de consultas *SQL* a partir de descrições em linguagem natural. Esses modelos podem capturar nuances semânticas, lidar com diferentes estruturas de frases e adaptar-se a contextos variados. Para o conjunto de validação do *Spider*, os três modelos com maior Acurácia de Correspondência Exata são RESDSQL [17], G³R [27], Graphix-t5 [18] com acurácias 80.5, 78.1 e 77.1, respectivamente. Além disso, em Agosto de 2023, foi lançado o SQLCoder[1], um LLM estado da arte para *text-to-sql* de 15B de parâmetros, que significativamente supera todos os grandes modelos de código aberto e supera um pouco o GPT-3.5-turbo e text-davinci-003 em seu método de avaliação. Abaixo serão descritos os *frameworks* e melhorias para os 3 melhores modelos no conjunto de validação do *Spider*.

4.3.1 RESDSQL

Os principais componentes do framework RESDSQL são o *encoder* aprimorado para classificação e o *skeleton-aware decoder*. Esses componentes contribuem para a eficácia do RESDSQL ao separar as tarefas de vinculação de esquema e análise de estrutura, reduzindo assim a dificuldade da análise de Texto para SQL.

O *encoder* aprimorado para classificação é responsável por inserir os itens de esquema mais relevantes na sequência de entrada do *encoder*. Isso é feito usando um *encoder* cruzado para classificar os itens de esquema com base na pergunta fornecida e, em seguida, classificá-los e filtrá-los com base nas probabilidades de classificação. Ao considerar apenas os itens de esquema mais relevantes, o *encoder* pode reduzir o esforço necessário para a vinculação de esquema durante a análise de SQL.

Por outro lado, o *skeleton-aware decoder* primeiro gera o esqueleto SQL e, em seguida, a consulta SQL real. Essa abordagem orienta implicitamente a geração subsequente de SQL ao restringi-la com o esqueleto previamente analisado. Ao separar a vinculação de esquema e a análise de estrutura, a dificuldade de analisar consultas SQL corretas, especialmente aquelas que envolvem muitos itens de esquema e operadores lógicos, é reduzida.

[1] Disponível em https://github.com/defog-ai/sqlcoder

4.3.2 G³R

O G³R melhora o desempenho da análise de texto para SQL de duas maneiras principais. Primeiramente, o G³R aborda a limitação das abordagens atuais explorando completamente a estrutura da *Abstract Syntax Tree* (AST) durante o processo de decodificação. Isso é crucial para a geração de consultas SQL complexas. O gerador SQL orientado por gráfico proposto no G³R captura as mudanças estruturais da AST construindo um grafo bipartido AST-*Grammar*. Isso permite que o modelo capture dinamicamente as informações estruturais e funda informações heterogêneas do codificador para prever a sequência de ações.

Em segundo lugar, o G³R incorpora conhecimento de domínio para melhorar sua capacidade de generalização para domínios não vistos anteriormente. Um *knowledge-enhanced re-ranking mechanism* é introduzido para escolher a melhor consulta SQL a partir da saída do feixe. Esse mecanismo utiliza um Modelo de Linguagem Pré-treinado (PLM) e aprendizado contrastivo com ajuste de *prompt* híbrido para estimular o conhecimento do PLM e torná-lo mais discriminativo. Ao aproveitar o conhecimento abundante dos PLMs, o G³R mitiga a degradação de desempenho em domínios não vistos.

4.3.3 Graphix-T5

O Graphix-T5 melhora o desempenho da tradução *text-to-SQL* aprimorando a capacidade de codificação estrutural do T5, ao mesmo tempo em que mantém sua poderosa capacidade de codificação contextual. Isso é alcançado incorporando uma arquitetura PLM semi-pré-treinada de *text-to-text* chamada Graphix-T5.

O Graphix-T5 introduz uma camada Graphix no codificador do T5, permitindo uma interação intensiva entre informações semânticas e estruturais desde as camadas iniciais. Isso permite que o modelo capture e utilize eficazmente informações estruturais, o que é crucial para casos complexos de *text-to-SQL*.

Os resultados de experimentos extensivos demonstram a eficácia do Graphix-T5. Ele supera modelos de linha de base competitivos, incluindo o T5 padrão, em vários *benchmarks* de *text-to-SQL* em diferentes domínios. O Graphix-T5 alcança um desempenho de ponta no desafiador *benchmark Spider*, alcançando 77.1% no conjunto de validação. Ele também mostra melhorias significativas em cenários de generalização com recursos limitados e composicionais.

Conclui-se então que a revisão dos trabalhos relacionados destaca o progresso contínuo na tradução de linguagem natural para *SQL*. Enquanto abordagens tradicionais e baseadas em exemplos forneceram *insights* valiosos, a abordagem com LLM emerge como uma solução promissora. No entanto, adaptação a domínios específicos ainda merecem atenção.

5

Metodologia e Desenvolvimento

Neste capítulo, serão descritos cada uma das etapas do desenvolvimento do projeto. Na Figura 5.1, as etapas do projeto são divididas em:

1. Desenvolvimento do *Prompt*
2. Refinamento Supervisionado dos modelos pré-treinados
3. Inferência
    3.1. *ChatGPT*
    3.2. *SQLCoder*
    3.3. Modelos pré-treinados
    3.4. Modelos refinados

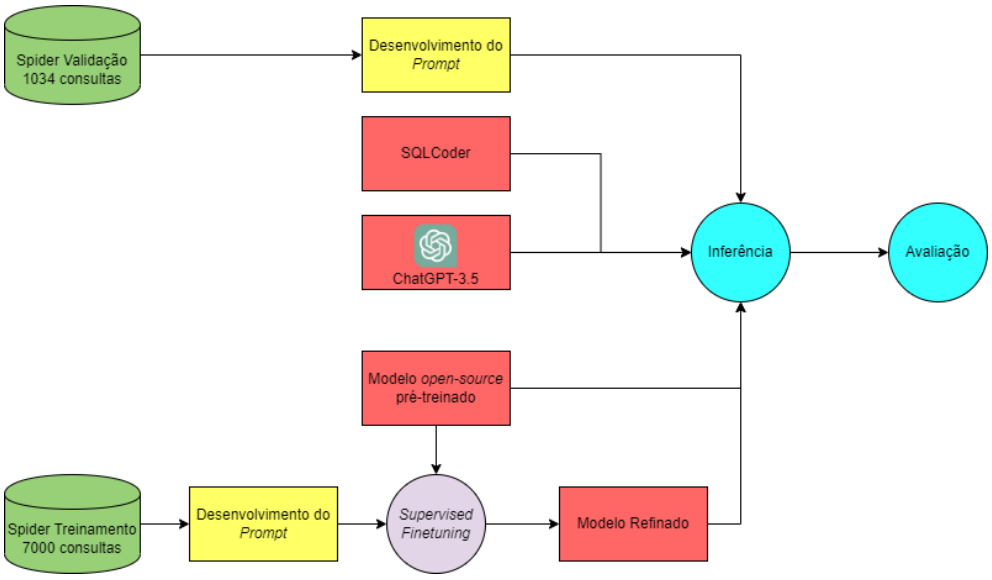

Figura 5.1: Fluxograma do Projeto

5.1 Desenvolvimento do *Prompt*

O processo inicial envolve a elaboração do *prompt* a ser utilizado pelos modelos. Essa abordagem permite a geração direta e precisa de consultas SQL em resposta às perguntas dos usuários. Para a sua construção, foram utilizadas as seguintes técnicas de *Prompt Engineering*:

- **Fornecer Instruções**: Para auxiliar na obtenção da consulta SQL, foi especificado claramente o que o modelo deve fazer e o que deve ser retornado.

- **Contexto**: Além da pergunta a ser respondido pela consulta SQL, foi adicionado ao *prompt* o *schema* da base de dados, contendo todas as tabelas, colunas, chaves estrangeiras e primárias. Dessa forma, o modelo possui o contexto necessário para gerar a consulta SQL correta.

- **Formatação**: Para delimitar cada uma das partes do *prompt*, foram inseridos separadores para o *schema* e a pergunta. Cada uma das partes é iniciada por <|*schema*|> ou <|*query*|> e finalizada por <|*endoftext*|>. Além disso, ao fim da entrada existe o separador <|SQL|> para indicar o início da resposta do modelo. Dessa forma, o modelo é capaz de compreender o contexto, o *schema* e a ação necessária.

Abaixo é possível observar um *prompt* construído após a aplicação das técnicas acima:

Convert text into SQL statements by providing a database schema and a query, and generate only the corresponding SQL statement.
<|schema|>
table people, columns = [*,people_id,nationality,name,birth_date,height]
table poker_player, columns = [*,poker_player_id,people_id,final_table_made, best_finish,money_rank,earnings]
foreign_keys = [poker_player.poker_player_id,people.people_id]
primary_keys = [poker_player.people_id = people.people_id]
<|endoftext|>
<|query|>
what is the average earnings of poker players?
<|endoftext|>
<|sql|>

5.2 Refinamento Supervisionado dos modelos pré-treinados

Ambos os modelos, Falcon 7B e Llama 2 7B foram submetidos ao processo de *Supervised Finetuning*(Refinamento Supervisionado), que permite refinar as capacidades dos modelos através do aprendizado supervisionado. Os dados de entrada consistiram das instruções, do *schema* da base de dados e da pergunta do usuário e os dados de saída foram definidos como a consulta SQL correspondente.

Nesta etapa, o refinamento dos modelos ocorreu utilizando apenas uma GPU NVIDIA TITAN Xp com 12GB de VRAM graças a utilização do QLoRA que quantiza o modelo em 4 bits, reduzindo o tamanho e mantendo a performance do modelo. Para o refinamento dos modelos, foi utilizado uma taxa de aprendizagem $1x10^{-4}$, para o Falcon 7B, e $1x10^{-5}$, para o Llama 2 7B, o otimizador AdamW de 8 bits e a função de perda de entropia cruzada. O treinamento foi conduzido por seis e duas épocas, para os modelos Falcon 7B e Llama 2 7B, respectivamente.

As perdas ao longo do treinamento dos modelos podem ser observados nas Figuras 5.2 e 5.3. Além disso, nas Figuras 5.4 e 5.5 é possível observar as perdas no conjunto de validação.

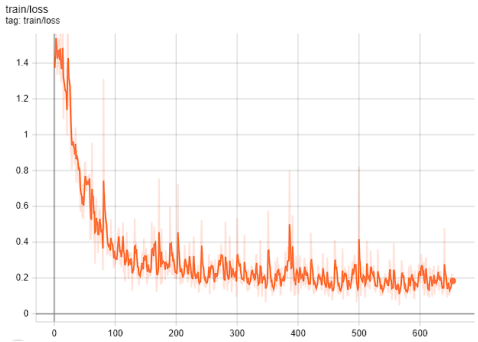

Figura 5.2: Perda de treinamento do Falcon 7B

Figura 5.3: Perda de treinamento do Llama 2 7B

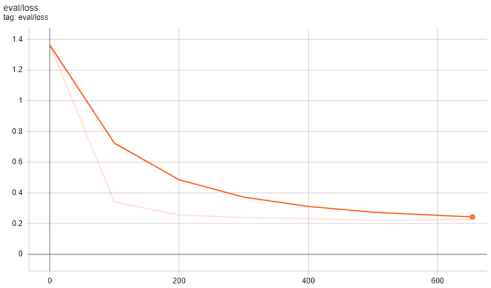

Figura 5.4: Perda de validação do Falcon 7B

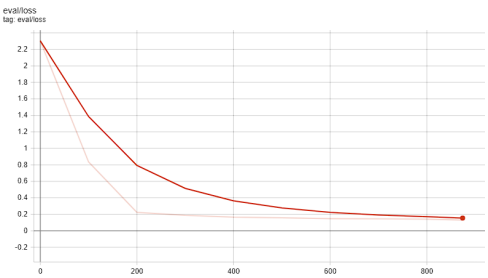

Figura 5.5: Perda de validação do Llama 2 7B

Apesar do modelo Llama 2 7B apresentar menor perda tanto no treinamento quanto na validação, não é possível concluir que o refinamento foi melhor que o do modelo Falcon 7B pois

a métrica de perda avalia apenas a capacidade do modelo em antecipar o próximo elemento na sequência, no entanto, esse aspecto por si só pode não abarcar toda a complexidade inerente à tarefa de produção de consultas SQL.

5.3 Inferência

5.3.1 Modelos Pré-treinados

Como descrito anteriormente, os dois melhores modelos *open-source* pré-treinados do *ranking* do *HuggingFace* durante a realização do projeto foram selecionados para refinamento. Porém, antes do refinamento, os modelos pré-treinados Falcon 7B e Llama 2 7B serão utilizados para gerar as consultas SQL do conjunto de validação. Para isso, foram definidos os parâmetros da geração de texto, que permitem ajustar o comportamento do modelo. Os parâmetros utilizados neste projeto para a inferência nos modelos pré-treinados e refinados com as suas explicações são:

- **Temperatura:** 0.2. Este parâmetro afeta a aleatoriedade da amostragem. Valores mais altos tornam a geração mais aleatória, enquanto valores mais baixos tornam a geração mais determinística;
- **Top K:** 0.3. Junto com o Top P, esses parâmetros controlam quais *tokens* são considerados durante a geração. Top K mantém os *tokens* mais prováveis, enquanto Top P mantém *tokens* até que a soma das probabilidades acumuladas seja maior que um determinado limite;
- **Top P:** 0.9;
- **Penalidade de Repetição:** 1.2. Este parâmetro ajuda a controlar a tendência do modelo de repetir palavras ou frases. Valores mais altos reduzem a repetição;

Na Tabela 5.1, é possível observar uma das consultas geradas, em que os modelos apesar de captarem alguns elementos da entrada, por não serem refinados, não conseguiram gerar a consulta SQL de forma correta. Além disso, as respostas são inconstantes a depender da entrada, como é observado na Tabela 5.2. Dessa forma, é necessário o refinamento desses modelos para obtenção de consultas SQL corretas.

Tabela 5.1: Consultas geradas por modelos pré-treinados

Pergunta	**Show the name and the release year of the song by the youngest singer.**
Resposta Correta	SELECT song_name , song_release_year FROM singer ORDER BY age LIMIT 1
Resposta Llama 2 7B	the oldest singer.l\end{/}# 10.2.3### Exercise:Write an SQL query to find all songs that were released in 1985 or later.# 10.3.1### Ex
Resposta Falcon 7B	country,singer_id,singer_name,song_name ,song_release_year,agel l:—:l:—:l:—:l:—:l:—:l l|USA,1,John,John,2000,20l l|USA,2

Tabela 5.2: Exemplo de resposta inconstante dos modelos pré-treinados

Pergunta	**Show name, country, age for all singers ordered by age from the oldest to the youngest.**
Resposta Correta	SELECT name , country , age FROM singer ORDER BY age DESC
Resposta Llama 2 7B	select concerts where concert_id in $select concert\_id from concert$ order by year desc limit 10;\end{code}
Resposta Falcon 7B	show name, country, age for all singers ordered by age from the youngest to the oldest.

5.3.2 *ChatGPT*

A partir da base construída, a geração de consultas SQL foi realizada pelo modelo ChatGPT-3.5 através da *API* da OpenAI. Abaixo, é possível observar um exemplo de consulta do conjunto de validação SQL gerado.

Convert text into SQL statements by providing a database schema and a query, and generate only the corresponding SQL statement.
<|schema|>
table people, columns = [*,people_id,nationality,name,birth_date,height]
table poker_player, columns = [*,poker_player_id,people_id,final_table_made, best_finish,money_rank,earnings]
foreign_keys = [poker_player.poker_player_id,people.people_id]
primary_keys = [poker_player.people_id = people.people_id]
<|endoftext|>
<|query|>
what is the average earnings of poker players?
<|endoftext|>
<|sql|>
SELECT AVG(earnings) FROM poker_player;

5.3.3 *SQLCoder*

Além do ChatGPT, o modelo *SQLCoder* foi utilizado para inferência no conjunto de validação. Porém, foi necessário a adaptação do *prompt* ao formato em que o modelo foi refinado. Abaixo, é possível observar a formatação utilizada:

> ### Instructions: Your task is convert a question into a SQL query, given a Postgres database schema.
> Adhere to these rules:
> - **Deliberately go through the question and database schema word by word** to appropriately answer the question
> - **Use Table Aliases** to prevent ambiguity. For example, ‘SELECT table1.col1, table2.col1 FROM table1 JOIN table2 ON table1.id = table2.id‘.
> - When creating a ratio, always cast the numerator as float
> ### Input:
> Generate a SQL query that answers the question ‘user_question‘.
> This query will run on a database whose schema is represented in this string:
> table_metadata_string
> ### Response:
> Based on your instructions, here is the SQL query I have generated to answer the question ‘user_question‘:
> “‘sql

Em *table_metadata_string*, é inserido o *script SQL* responsável pela criação das tabelas do banco de dados, já em *user_question* é inserido a pergunta do usuário. Após a formação do *prompt*, a geração de consultas SQL foi executada na versão do *HuggingFace* do *SQLCoder*.

5.3.4 Modelos Refinados

Ao fim dos refinamentos, inferências foram realizadas em ambos os modelos para verificar a melhoria na geração de consultas SQL, as respostas geradas com as perguntas mostradas nas Tabelas 5.1 e 5.2 podem ser vistos na Tabela 5.3. Além disso, como descrito na Seção 5.3, foram definidos os parâmetros da geração de texto na inferência.

Tabela 5.3: Consultas geradas após refinamento dos modelos

Pergunta	**Show the name and the release year of the song by the youngest singer.**
Resposta Correta	SELECT song_name , song_release_year FROM singer ORDER BY age LIMIT 1
Resposta Llama 2 7B	select t1.song_name, t2.song_release_year from singer as t1 join song as t2 on t1.song_name = t2.title where age <(select min(t3.age) from singer as t3);
Resposta Falcon 7B	select t2.song_name, t2.song_release_year from singer_in_concert as t1 join singer as t2 on t1.singer_id = t2.singer_id order by t1.singer_id asc limit 1;
Pergunta	**Show name, country, age for all singers ordered by age from the oldest to the youngest.**
Resposta Correta	SELECT name , country , age FROM singer ORDER BY age DESC
Resposta Llama 2 7B	select t1.name, t2.country, t3.age from singer as t1 join singer_in_concert as t2 on t1.singer_id = t2.singer_id order by t3.age;
Resposta Falcon 7B	select name, country, age from singer order by age desc;

É notável que após o refinamento, ambos modelos foram capazes de gerar respostas condizentes com as perguntas do usuário.

Assim como na inferência dos modelos pré-treinados, os modelos refinados serão utilizados para inferência em todas as 1034 consultas do conjunto de validação. Com isso, no Capítulo 5 serão comparados os resultados dos modelos pré-treinados, refinados, ChatGPT e *SQLCoder* utilizando a métrica de avaliação oficial do *Spider*.

6

Avaliação e Discussão de Resultados

Após o refinamento e inferência dos modelos, é necessário avaliar qual dos modelos possui a maior capacidade de geração de consultas *SQL* corretas. Para isso, será utilizada a avaliação oficial do *Spider* descrito na Seção 3.2. A métrica selecionada é a Acurácia de Correspondência Exata que além de verificar a saída, decompõe cada consulta *SQL* predita e real em diversas cláusulas, e conduz comparação entre cada cláusula *SQL*.

Para realizar o cálculo dessa métrica, será utilizada a base gerada a partir da inferência dos modelos no conjunto de validação. Na Tabela 6.1, é possível observar os valores das métricas, separados por dificuldade e em porcentagem, para os modelos refinados, pré-treinados, ChatGPT e *SQLCoder*. Além disso, na Tabela 6.2 é possível observar um exemplo de inferência dos modelos refinados, ChatGPT e SQLCoder em uma consulta do nível Muito Difícil onde nenhum dos modelos gerou a consulta correta.

Tabela 6.1: Métricas dos Modelos no Conjunto de Validação

Modelo	**Acurácia de Correspondência Exata**				
	Fácil	**Médio**	**Difícil**	**Muito Difícil**	**Total**
ChatGPT	**73**	**37.9**	**24.7**	12	**39.9**
SQLCoder	60.1	22	12.1	2.4	26.3
Llama 2 7B pré-treinado	0	0	0	0	0
Llama 2 7B refinado	50	17.5	9.8	**15.7**	23.7
Falcon 7B pré-treinado	0	0	0	0	0
Falcon 7B refinado	60.1	37.2	22.4	13.3	36.4

Na Tabela 6.2, percebe-se que apesar de não terem acertado a consulta, os modelos foram capazes de capturar diversos elementos SQL, como as colunas, tabelas, junções e filtros utilizados. Porém, o modelo Llama 2 apresentou um comportamento mais errático, ao executar um *join* extra desnecessário e não finalizar a consulta. Este exemplo ilustra bem as consultas incorretas destes modelos, em que os modelos são capazes de capturar boa parte dos elementos

Tabela 6.2: Exemplo de Consulta Muito Difícil

Pergunta	**Show the stadium name and capacity with most number of concerts in year 2014 or after.**
Resposta Correta	SELECT T2.name , T2.capacity FROM concert AS T1 JOIN stadium AS T2 ON T1.stadium_id = T2.stadium_id WHERE T1.year >= 2014 GROUP BY T2.stadium_id ORDER BY count(*) DESC LIMIT 1
Resposta Llama 2 7B refinado	select t3.name, t5.capacity from concert as t1 join stadium as t2 on t1.stadium_id = t2.stadium_id join singer_in_concert as t3 on t1.concert_id = t3.concert_
Resposta Falcon 7B refinado	select t2.name, t1.capacity from stadium as t1 join concert as t2 on t1.stadium_id = t2.stadium_id where t2.year >2014;
Resposta ChatGPT	SELECT stadium.name, stadium.capacity FROM stadium JOIN concert ON concert.stadium_id = stadium.stadium_id WHERE concert.year >= 2014 GROUP BY stadium.name, stadium.capacity ORDER BY COUNT(concert.concert_id) DESC LIMIT 1
Resposta SQLCoder	SELECT name, capacity FROM stadium JOIN (SELECT stadium_id, count(*) FROM concert WHERE year >= '2014' GROUP BY stadium_id;

SQL, porém erram em pequenos detalhes.

E na Tabela 6.1 observa-se que, o ChatGPT apresentou a melhor performance dentre os modelos avaliados, registrando as maiores pontuações nas categorias Fácil e Médio, refletindo a capacidade do modelo de traduzir com sucesso consultas menos complexas e ainda nas categorias Difícil e Muito Difícil, o ChatGPT manteve um desempenho respeitável, indicando sua adaptabilidade a contextos mais desafiadores. É importante destacar ainda, que a performance do ChatGPT pode ser melhorada a partir da aplicação de mais técnicas de *Prompt Engineering*, como demonstrado no modelo C3 [11], que utilizou diversas técnicas e obteve acurácia total de 82.3% no conjunto de teste do *Spider*. Porém, o modelo Falcon 7B refinado apresentou acurácia total próxima ao ChatGPT, superando-o na categoria Muito Difícil, além de superar o SQLCoder e o Llama 2. Esse resultado é promissor quando consideramos as limitações inerentes ao ChatGPT em relação à tarefa de tradução de linguagem natural para SQL.

O ChatGPT, embora tenha demonstrado um desempenho sólido e consistente em nossa

avaliação, não está isento de desafios. Além do custo associado ao seu uso, suas capacidades podem ser afetadas por contextos complexos ou requisitos específicos do domínio, como observado na menor acurácia na categoria Muito Difícil, 12%, quando comparados ao Llama 2 7B Refinado, 15.7%, e Falcon 7B refinado, 13.3%.

Já o modelo SQLCoder obteve a terceira melhor acurácia total, superando o modelo Llama 2 refinado e mesma acurácia no nível Fácil que o Falcon. Esse resultado contrasta com o resultado apresentado pelo modelo no seu lançamento, onde no seu método de avaliação, superou em 4% o ChatGPT, 64.6% contra 60.6%. Esse contraste pode ser associado a diferença no método de avaliação utilizado.

Por fim, os dois modelos pré-treinados apresentaram a pior performance, confirmando que sem o refinamento, os modelos pré-treinados não são capazes de gerar consultas *SQL* corretamente.

Os resultados demonstram que, apesar do ChatGPT-3.5 apresentar acurácia total superior, os modelos refinados nesse projeto, com destaque ao Falcon, são capazes de gerar consultas SQL com acurácia próxima ou até melhor em certas consultas.

7

Conclusão

Neste projeto, foram investigados a aplicação de modelos de linguagem de código aberto para tradução de linguagem natural para consultas SQL. Para isso, foi necessário o estudo e revisão da literatura, a seleção de modelos pré-treinados e especializados de código aberto, e o refinamento e avaliação destes no problema proposto. A revisão da literatura representou uma etapa bastante importante pois, os modelos Falcon, Llama 2 e SQLCoder foram lançados durante o desenvolvimento do projeto, demonstrando a rápida taxa de inovação dos modelos de linguagem de código aberto.

O projeto envolveu o desenvolvimento do *prompt* a partir do conjunto de dados *Spider*, o refinamento de modelos pré-treinados para gerar as consultas SQL e a avaliação dos modelos pré-treinados, refinados e o ChatGPT e SQLCoder. Para avaliação, foi utilizado o método de avaliação oficial do *Spider*.

Essa avaliação revelou que nem os modelos refinados nem o SQLCoder foram capazes de superar o ChatGPT-3.5 na métrica utilizada. Porém, por questões computacionais, os modelos Falcon e Llama 2 utilizados foram os menores disponibilizados, cerca de 7B de parâmetros, permitindo o refinamento e inferência em apenas uma GPU comercial. Já o ChatGPT, além de ser um modelo proprietário, possui cerca de 175B de parâmetros. Após o refinamento em apenas 7000 exemplos de treinamento, o modelo Falcon superou o SQLCoder e obteve resultado comparável ao ChatGPT. Demonstrando o potencial de refinamento de modelos pré-treinados de código aberto para tarefas específicas.

Para sugestão de próximos passos, destacam-se o refinamento das versões maiores dos modelos, como o Falcon de 40B de parâmetros e o Llama 2 de 13 e 70B de parâmetros e o aprimoramento do *prompt*, como a inserção de mais contexto e instruções. Além disso, a realização do refinamento em um conjunto maior de treinamento e em diversas etapas, como realizado no SQLCoder.

Referências

[1] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023.

[2] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, E. Goffinet, D. Heslow, J. Launay, Q. Malartic, B. Noune, B. Pannier, and G. Penedo. Falcon-40B: an open large language model with state-of-the-art performance. 2023.

[3] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. Palm: Scaling language modeling with pathways, 2022.

[4] S. Chu, B. Murphy, J. Roesch, A. Cheung, and D. Suciu. Axiomatic foundations and algorithms for deciding semantic equivalences of sql queries. *Proc. VLDB Endow.*, 11(11):1482–1495, jul 2018.

[5] S. Chu, C. Wang, K. Weitz, and A. Cheung. Cosette: An automated prover for SQL. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org, 2017.

[6] M. Conover, M. Hayes, A. Mathur, J. Xie, J. Wan, S. Shah, A. Ghodsi, P. Wendell, M. Zaharia, and R. Xin. Free dolly: Introducing the world's first truly open instruction-tuned llm, 2023.

[7] A. Copestake and K. S. Jones. Natural language interfaces to databases. *The Knowledge Engineering Review*, 5(4):225–249, 1990.

[8] D. A. Dahl, M. Bates, M. Brown, W. Fisher, K. Hunicke-Smith, D. Pallett, C. Pao, A. Rudnicky, and E. Shriber. Expanding the scope of the atis task: The atis-3 corpus. *Proceedings of the workshop on Human Language Technology*, pages 43–48, 1994.

[9] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[11] X. Dong, C. Zhang, Y. Ge, Y. Mao, Y. Gao, lu Chen, J. Lin, and D. Lou. C3: Zero-shot text-to-sql with chatgpt, 2023.

[12] A. Fariha, L. Cousins, N. Mahyar, and A. Meliou. Example-driven user intent discovery: Empowering users to cross the sql barrier through query by example, 2020.

[13] P. Gage. A new algorithm for data compression. *C Users J.*, 12(2):23–38, feb 1994.

[14] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '07, page 31–40, New York, NY, USA, 2007. Association for Computing Machinery.

[15] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models, 2021.

[16] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, September 2014.

[17] H. Li, J. Zhang, C. Li, and H. Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql, 2023.

[18] J. Li, B. Hui, R. Cheng, B. Qin, C. Ma, N. Huo, F. Huang, W. Du, L. Si, and Y. Li. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing, 2023.

[19] R. A. Pazos R., J. J. González B., M. A. Aguirre L., J. A. Martínez F., and H. J. Fraire H. *Natural Language Interfaces to Databases: An Analysis of the State of the Art*, pages 463–480. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[20] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, A. Cappelli, H. Alobeidli, B. Pannier, E. Almazrouei, and J. Launay. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only, 2023.

[21] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[22] M. N. Team. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023. Accessed: 2023-05-05.

[23] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023.

[24] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

[26] T. Wang, A. Roberts, D. Hesslow, T. L. Scao, H. W. Chung, I. Beltagy, J. Launay, and C. Raffel. What language model architecture and pretraining objective work best for zero-shot generalization?, 2022.

[27] Y. Xiang, Q.-W. Zhang, X. Zhang, Z. Liu, Y. Cao, and D. Zhou. g^3r: A graph-guided generate-and-rerank framework for complex and cross-domain text-to-SQL generation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 338–352, Toronto, Canada, July 2023. Association for Computational Linguistics.

[28] T. Yu, R. Zhang, H. Y. Er, S. Li, E. Xue, B. Pang, X. V. Lin, Y. C. Tan, T. Shi, Z. Li, Y. Jiang, M. Yasunaga, S. Shim, T. Chen, A. Fabbri, Z. Li, L. Chen, Y. Zhang, S. Dixit, V. Zhang, C. Xiong, R. Socher, W. S. Lasecki, and D. Radev. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases, 2019.

[29] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, 2019.

[30] T. Yu, R. Zhang, M. Yasunaga, Y. C. Tan, X. V. Lin, S. Li, H. Er, I. Li, B. Pang, T. Chen, E. Ji, S. Dixit, D. Proctor, S. Shim, J. Kraft, V. Zhang, C. Xiong, R. Socher, and D. Radev. Sparc: Cross-domain semantic parsing in context, 2019.

[31] J. M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, pages 1050–1055, 1996.

[32] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen. A survey of large language models, 2023.

[33] R. Zhong, T. Yu, and D. Klein. Semantic evaluation for text-to-sql with distilled test suites, 2020.

[34] V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.