UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

NICOLAS BORTOLOTTO WILLI

CONSULTAS INTERATIVAS EM BANCO DE DADOS USANDO TEXT2SQL BASEADO EM LLM

SANTA HELENA

2025

NICOLAS BORTOLOTTO WILLI

CONSULTAS INTERATIVAS EM BANCO DE DADOS USANDO TEXT2SQL BASEADO EM LLM

Interactive Database Queries Using RAG and LLM

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (**IntI@UTFPR**).

Orientador: Prof. Doutor Franck Carlos Vélez Benito

SANTA HELENA

2025

NICOLAS BORTOLOTTO WILLI

CONSULTAS INTERATIVAS EM BANCO DE DADOS USANDO TEXT2SQL BASEADO EM LLM

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (**IntI@UTFPR**).

Data de aprovação: 26 de Junho de 2025

Franck Carlos Vélez Benito
Doutorado em Informática
Universidade Tecnológica Federal do Paraná

Leiliane Pereira de Rezende
Doutorado em Ciência da Computação
Universidade Tecnológica Federal do Paraná

Thiago França Naves
Doutorado em Ciência da Computação
Universidade Tecnológica Federal do Paraná

SANTA HELENA

2025

RESUMO

Este trabalho apresenta uma análise sistemática do impacto de diferentes técnicas de *prompting* e métricas de distância vetorial na conversão de consultas em linguagem natural para SQL com o auxílio de modelos de linguagem de larga escala (LLMs). Foram avaliadas as estratégias de *prompt* padrão, Few-Shot e Chain of Thought combinadas com métricas de similaridade vetorial como cosseno, euclidiana, produto escalar e Manhattan, com o objetivo de identificar as combinações mais eficazes para a recuperação semântica e geração de consultas SQL. Os experimentos indicaram que a técnica Few-Shot associada à métrica de distância cosseno apresentou os melhores resultados em termos de similaridade semântica, execução correta das queries e qualidade textual. Para viabilizar as análises, foi desenvolvida uma aplicação baseada em uma arquitetura RAG, integrando um banco vetorial (Qdrant), a biblioteca VannaAI e o modelo LLaMA 3.1 8B. Essa aplicação serviu como ferramenta de apoio para os testes e demonstra a aplicabilidade prática das combinações analisadas.

Palavras-chave: Texto-para-SQL; Geração Aumentada de Recuperação; Engenharia de Prompt; Banco de Dados Vetorial.

ABSTRACT

This work presents a systematic analysis of the impact of different prompting techniques and vector distance metrics on the conversion of natural language queries into SQL using large language models (LLMs). The study evaluated the standard, Few-Shot, and Chain of Thought prompting strategies combined with vector similarity metrics such as cosine, Euclidean, dot product, and Manhattan, aiming to identify the most effective combinations for semantic retrieval and SQL query generation. The experiments showed that the Few-Shot technique combined with the cosine distance metric achieved the best results in terms of semantic similarity, correct query execution, and textual quality. To support the analyses, an application was developed based on a RAG architecture, integrating a vector database (Qdrant), the VannaAI library, and the LLaMA 3.1 8B model. This application served as a support tool for the experiments and demonstrates the practical applicability of the analyzed combinations.

Keywords: Text-to-SQL; Retrieval Augmented Generation; Prompt Engineering; Vector Database.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BLEU	Substituto Bilíngue para Avaliação
CoT	Cadeia de Raciocínio
CSV	Valores Separados por Vírgula
DDL	Linguagem de Definição de Dados
EM	Correspondência Exata
EX	Precisão de Execução
GPU	Unidade de Processamento Gráfico
HNSW	Pequeno Mundo Navegável Hierárquico
JSON	Notação de Objetos JavaScript
LLM	Modelo de Linguagem de Grande Escala
NLP	Processamento de Linguagem Natural
RAG	Geração Aumentada de Recuperação
SQL	Linguagem de Consulta Estruturada
VRAM	Memória de Acesso Aleatório de Vídeo

SUMÁRIO

1 INTRODUÇÃO

A tarefa de Texto-para-SQL emerge como um desafio no campo do Processamento de Linguagem Natural (NLP, do inglês Natural Language Processing), focando na tradução de consultas em linguagem natural para instruções de linguagens de consulta estruturada (SQL, do inglês Structured Query Language) que possam ser executadas em bases de dados relacionais. Esta abordagem não apenas facilita o acesso a dados estruturados, mas também democratiza a interação com bancos de dados, tornando-a acessível a usuários sem conhecimentos técnicos em SQL (LI *et al.*, 2023).

O desenvolvimento de sistemas Texto-para-SQL é impulsionado por avanços significativos em aprendizado profundo e modelos de linguagem de grande escala (LLM, do inglês Large Language Model). Desde os métodos baseados em regras e modelos, que exigiam alta dependência de engenharia manual, até a introdução de redes neurais sequenciais (*Seq2Seq*) e estratégias mais recentes como engenharia de *prompts* e *fine-tuning*, o campo tem evoluído consideravelmente (SHI *et al.*, 2024). Essas inovações permitiram lidar com consultas SQL de complexidade crescente, abrangendo desde simples seleções até operações aninhadas em cenários de alta demanda.

Além disso, a avaliação de sistemas Texto-para-SQL também é um aspecto crucial, envolvendo métricas que vão desde a correspondência literal até a execução semântica e a análise de componentes específicos das consultas. Métricas como Correspondência exata (EM, do inglês Exact Match), Precisão de execução (EX, do inglês Execution Accuracy) e a Pontuação *BLEU* têm sido amplamente empregadas para medir a eficácia e eficiência desses sistemas (SONG *et al.*, 2024).

Este trabalho tem como objetivo realizar uma comparação entre as técnicas de *prompts*, com o intuito de identificar o método mais eficaz para a conversão de texto-para-SQL. Além disso, serão avaliados o uso de um banco de dados vetorial, de um modelo LLM e da geração aumentada de recuperação (RAG, do inglês Retrieval-augmented generation), bem como técnicas de Texto-para-SQL, visando analisar suas contribuições para a tarefa. Por fim, será desenvolvido um programa capaz de realizar consultas interativas em um banco de dados relacional.

1.1 Objetivo

Expõem-se a seguir os objetivos geral e específicos que se pretende atingir com o trabalho.

1.1.1 Geral

Este trabalho tem como objetivo avaliar diferentes técnicas de *prompt* e métodos de cálculos de distância em bancos de dados vetoriais para o processo de conversão de texto-para-SQL baseado em LLM. Por fim, desenvolvido um algoritmo para realizar consultas em banco de dados relacionais utilizando a técnica de *prompt* e método de cálculo melhor avaliado.

1.1.2 Específicos

1. Realizar uma análise entre as principais técnicas de *prompts* utilizadas no processo de Texto-para-SQL.
2. Realizar uma análise entre os principais algoritmos de cálculo de distância em bancos de dados vetoriais para recuperação de contexto.
3. Construir um algoritmo para realizar pesquisa a partir de linguagem natural em banco de dados relacional.

1.2 Contribuições do Trabalho

A principal contribuição deste trabalho é a construção de um algoritmo de consultas em bancos de dados relacionais utilizando linguagem natural, demonstrando os seguintes detalhes sobre a construção do algoritmo:

1. Classificação baseada em análise dos diferentes formatos de *prompts* utilizados para realizar o processo de Texto-para-sql;
2. Classificação baseada em análise e análise dos algoritmos de cálculo de distância em bancos de dados vetoriais para a construção de *prompts*;
3. Utilização de banco de dados vetoriais para a construção de *prompt* no processo de Texto-para-SQL.

1.3 Justificativa

A conversão de linguagem natural para SQL (Texto-para-SQL) representa um avanço significativo na acessibilidade e usabilidade de bancos de dados relacionais. Esse processo permite que usuários sem conhecimento técnico em SQL possam formular consultas complexas de maneira intuitiva, reduzindo a necessidade de aprendizado da sintaxe da linguagem e facilitando a obtenção de informações de maneira rápida e eficiente (SONG *et al.*, 2024). Com o auxílio de Modelos de Linguagem de Grande Escala (LLMs), a precisão na conversão dessas consultas tem aumentado, tornando a interação com bases de dados mais fluida e acessível para diversos públicos.

Além da facilidade de uso, o Texto-para-SQL tem sido cada vez mais aplicado em processos de consultas automatizadas em diversas áreas, como análise de negócios, suporte ao cliente e recuperação de informações em tempo real. Empresas e organizações utilizam essa tecnologia para otimizar fluxos de trabalho, permitindo que sistemas interpretem comandos em linguagem natural e executem buscas estruturadas em bancos de dados (SONG *et al.*, 2024). Isso reduz a necessidade de intervenção humana em tarefas repetitivas e melhora a eficiência operacional, proporcionando respostas rápidas e precisas a partir de grandes volumes de dados.

A escolha de bancos de dados relacionais como base para a aplicação do Texto-para-SQL se deve à sua ampla adoção e estrutura bem definida, que permite a modelagem de dados de maneira eficiente e consistente. Bancos relacionais são amplamente utilizados em setores como finanças, saúde e *e-commerce*, onde a integridade e a organização dos dados são cruciais (SONG *et al.*, 2024). Ao integrar a conversão de texto-para-SQL nessas bases, permite-se extrair informações de forma dinâmica e precisa, melhorando a tomada de decisões e a acessibilidade das informações para usuários não técnicos.

1.4 Delimitações do Trabalho

Como o foco do trabalho é realizar uma análise das técnicas que envolvem o processo de conversão de texto-para-SQL, esse trabalho se limita aos seguintes pontos:

1. Não serão abordados assuntos de complexidade de algoritmo.

2. Não será desenvolvida uma interface para interação com o algoritmo.
3. Não serão explicados os passos de instalação de *softwares* utilizados.
4. Não será explicado o processo de ETL dos dados que serão utilizados no desenvolvimento do trabalho.
5. Não será realizada uma análise comparativa dos SQLs criados pela aplicação.
6. Não serão feitas modificações nos parâmetros das ferramentas utilizadas.

2 REVISÃO DA LITERATURA

A revisão da literatura foi desenvolvida com o objetivo de abordar conceitos fundamentais das áreas correlatas, essenciais para a integração de técnicas relacionadas à conversão de texto-para-SQL. Os conceitos frequentemente encontrados na literatura são explorados em detalhes, acompanhados de exemplos didáticos. Além disso, grande parte do capítulo apresenta sínteses elaboradas pelos autores, agregando novos conhecimentos aos conteúdos abordados. O objetivo é fornecer as bases necessárias para o uso das tecnologias envolvidas, bem como os conceitos específicos aplicados na integração dessas tecnologias e no controle das etapas do fluxo de treinamento dos algoritmos.

Desse modo, este capítulo desempenha um papel central na construção do conhecimento gerado por este trabalho.

2.1 Processamento de Linguagem Natural

O processamento de linguagem natural (NLP, do inglês *Natural Language Processing*) é um subcampo da Inteligência Artificial (IA) e da Ciência da Computação que estuda a interação entre computadores e a linguagem humana. Seu objetivo principal é possibilitar que máquinas compreendam, interpretem e gerem linguagem natural, permitindo aplicações diversas em áreas como ciência da informação, arquivologia e análise de sentimentos. O NLP tem sido utilizado há décadas, expandindo-se para além da linguística à medida que os avanços tecnológicos permitem a análise de grandes volumes de dados textuais e de fala (RODRIGUES, 2023).

2.1.1 Fundamentos

O NLP pode ser compreendido a partir de três aspectos principais: som (fonologia), estrutura (morfologia e sintaxe) e significado (semântica e pragmática). Essas áreas fornecem a base para o desenvolvimento de ferramentas que lidam com textos e discursos humanos de maneira automatizada (ALTAIR, 2023).

Historicamente, o NLP começou com tentativas de tradução automática na década de 1950, evoluindo para sistemas mais complexos capazes de realizar tarefas

como análise sintática, extração de informações e geração de textos. Entre os desafios do NLP, destacam-se a ambiguidade da linguagem humana, a necessidade de grande volume de dados anotados para treinamento de modelos e a dificuldade na interpretação do contexto (LUIZ VILLELA M. MIONI; RENATA S. C. DE BARBOSA; FANTINELI C. DE OLIVEIRA, 2024).

Os avanços recentes em NLP são impulsionados por técnicas de aprendizado de máquina, especialmente redes neurais profundas e modelos pré-treinados, como os baseados em *Transformers*, que permitem que sistemas de IA compreendam melhor o contexto e a semântica dos textos (RODRIGUES, 2023).

2.1.2 Aplicações

O NLP tem uma ampla gama de aplicações práticas que impactam diferentes setores. Algumas das mais relevantes são:

2.1.2.1 Extração e Mineração de Texto

O NLP possibilita a extração de informações relevantes de grandes volumes de dados textuais. No campo da arquivologia, por exemplo, ele pode ser utilizado para indexação automática de documentos e recuperação de informações, otimizando processos de gestão documental. Ferramentas de extração de informações podem identificar entidades nomeadas, classificar documentos e até mesmo sugerir categorias para arquivamento (ALTAIR, 2023).

2.1.2.2 Análise de Sentimentos

A análise de sentimentos utiliza NLP para interpretar emoções e opiniões expressas em textos. Essa aplicação é amplamente utilizada em redes sociais e no monitoramento da reputação de marcas e empresas. Além disso, estudos indicam que a análise de sentimentos pode auxiliar na detecção de discursos de ódio e *cyberbullying*, promovendo um ambiente digital mais seguro (LUIZ VILLELA M. MIONI; RENATA S. C. DE BARBOSA; FANTINELI C. DE OLIVEIRA, 2024).

2.1.2.3 Sumarização Automática de Textos

A capacidade de sintetizar grandes volumes de informação em resumos concisos é uma aplicação importante do NLP. Esse recurso é utilizado em diversas áreas, incluindo medicina, onde sistemas de NLP podem gerar sumários clínicos a partir de prontuários médicos, facilitando a tomada de decisões por profissionais da saúde (RODRIGUES, 2023).

2.1.2.4 *Chatbots* e Assistentes Virtuais

Chatbots e assistentes virtuais são algumas das aplicações mais populares do NLP. Eles são projetados para interagir com os usuários de maneira natural, respondendo perguntas, fornecendo informações e até auxiliando em diagnósticos médicos ou jurídicos. Esses sistemas evoluíram significativamente nos últimos anos, tornando-se mais precisos e eficientes na compreensão da linguagem humana (RODRIGUES, 2023).

2.1.2.5 Indexação e Recuperação de Informação

No campo da Ciência da Informação e Arquivologia, o NLP é utilizado para organizar e facilitar o acesso a grandes volumes de documentos e dados. Técnicas como a indexação automatizada permitem que documentos sejam classificados e pesquisados de maneira eficiente, melhorando a acessibilidade da informação para profissionais e pesquisadores (ALTAIR, 2023).

2.2 Modelos de Linguagem de Grande Escala

Modelos de Linguagem de Grande Escala (LLMs, do inglês *Large Language Models*) são sistemas de inteligência artificial baseados em redes neurais profundas que processam linguagem natural. Eles são projetados para prever e gerar texto coerente, responder a perguntas, traduzir línguas e realizar outras tarefas linguísticas (NAVEED *et al.*, 2024).

Os LLMs são essencialmente modelos matemáticos gerativos que predizem a próxima sequência de palavras com base na probabilidade estatística, utilizando como entrada uma sequência prévia de texto. Eles fundamentam-se em três elementos centrais: tokenização, treinamento e a arquitetura *Transformer*.

A tokenização é o processo inicial que converte o texto em unidades menores, como palavras, subpalavras ou caracteres, permitindo que o modelo lide eficientemente com dados linguísticos. Durante o treinamento, esses modelos são expostos a grandes volumes de texto para ajustar bilhões de parâmetros, aprendendo padrões complexos de linguagem e associações semânticas.

Por fim, a arquitetura *Transformer*, baseada em mecanismos de atenção, é responsável pela capacidade dos LLMs de capturar dependências contextuais em longo alcance e processar texto em paralelo, o que possibilita a geração de sequências coerentes e tarefas linguísticas avançadas.

2.2.1 Tokenização

A tokenização é uma etapa fundamental para processos de NLP, o que acaba tornando-a fundamental para os LLMs também. O seu funcionamento é basicamente codificar um texto em partes menores conhecidas como *tokens*, para o processamento computacional (MIELKE *et al.*, 2021).

Para a realização da tokenização, é necessário definir como serão feitos os *tokens* e qual será o algoritmo que irá identificar estes *tokens*. Abaixo será explicado melhor cada uma destas etapas.

2.2.1.1 *Tokens*

Os *tokens* podem ser definidos de diferentes maneiras, desde palavras espaçadas ou separadas por pontuação até unidades menores, como subpalavras ou caracteres. A granularidade é um aspecto crucial, pois afeta tanto o tamanho do vocabulário quanto a capacidade do modelo de lidar com palavras raras ou desconhecidas (MIELKE *et al.*, 2021).

- *Tokens* Inteiros: aproximações simplistas baseadas em espaços e pontuação.
- Subpalavras: utilizadas para balancear entre granularidade e eficiência.

- Caracteres ou *Bytes*: adotados em configurações de tokenização livre para reduzir dependências de regras específicas de idioma.

2.2.1.2 Algoritmos de Tokenização

Existem diversos algoritmos para realizar a tokenização, como:

- Codificação de pares de bytes (BPE, do inglês *Byte-Pair Encoding*): introduzido como um algoritmo de compressão, o BPE substitui iterativamente pares de símbolos adjacentes mais frequentes por um novo símbolo. Na tokenização, isso resulta em um vocabulário eficiente que representa subpalavras mais comuns (MIELKE *et al.*, 2021).
- Peça de palavra (*WordPiece*): semelhante ao BPE, mas otimiza a probabilidade de um modelo de linguagem baseado em n-gramas. Utiliza um método de esquerda para direita para segmentar texto em tempo real (MIELKE *et al.*, 2021).
- Modelo de linguagem Unigram (UnigramLM, do inglês *Unigram Language Model*): começa com um vocabulário amplo e reduz gradualmente os elementos com menor probabilidade, com base em um modelo de linguagem unigram. Permite regularização de subpalavras, introduzindo diversidade nas segmentações (MIELKE *et al.*, 2021).
- Peça de frase (*SentencePiece*): uma ferramenta genérica que suporta BPE e UnigramLM, além de lidar com idiomas sem espaços, como chinês e japonês, sem exigir uma etapa de pré-tokenização (MIELKE *et al.*, 2021).

2.2.2 Treinamento

Para o treinamento de uma LLM são necessárias algumas coisas, como uma grande base de dados composta por livros, artigos, páginas da *web* e outros repositórios, acumulando até centenas de *terabytes* de texto. Esses dados passam por limpeza para remover duplicatas, erros e informações irrelevantes, garantindo maior qualidade no treinamento. Além disso, o texto é tokenizado usando métodos como o BPE, transformando palavras em unidades menores que facilitam o aprendizado e a generalização em diferentes contextos linguísticos (SHANAHAN, 2024).

Também é necessária uma infraestrutura específica, como GPUs e TPUs, devido ao seu alto consumo de recursos computacionais. Técnicas de paralelismo, incluindo divisão de dados, fragmentação de cálculos e distribuição de etapas do processo, são usadas para lidar com o tamanho dos modelos e dos dados (SHANAHAN, 2024).

2.2.3 Arquitetura *Transformer*

A tecnologia *Transformer* revolucionou o campo do aprendizado de máquina, especialmente em tarefas de NLP. Proposta por (VASWANI *et al.*, 2023), essa arquitetura introduz uma abordagem inovadora que dispensa as redes neurais recorrentes (RNNs) e convolucionais (CNNs), focando exclusivamente no mecanismo de atenção.

O modelo *Transformer* é caracterizado por sua capacidade de processar entradas e saídas sequenciais por meio de um mecanismo de atenção denominado "atenção por produto escalar escalado". Esse método avalia as relações entre diferentes elementos em uma sequência ao calcular pesos baseados na similaridade entre vetores de consulta, chave e valor. Assim, o *Transformer* supera limitações de modelos anteriores que dependiam de cálculos sequenciais e eram mais suscetíveis a problemas de desempenho em sequências longas.

A arquitetura do *Transformer* é composta por duas principais pilhas de camadas, como mostrado na Figura 6: o codificador (*encoder*) e o decodificador (*decoder*). O codificador transforma a entrada em representações contínuas, enquanto o decodificador utiliza essas representações para gerar a saída. Cada camada do codificador e do decodificador incorpora mecanismos de atenção múltipla (*multi-head attention*) e redes totalmente conectadas (*feed-forward*), aprimoradas por conexões residuais e normalização de camadas. Além disso, para capturar informações posicionais na sequência, o *Transformer* utiliza codificações posicionais baseadas em funções senoidais e cossenoidais.

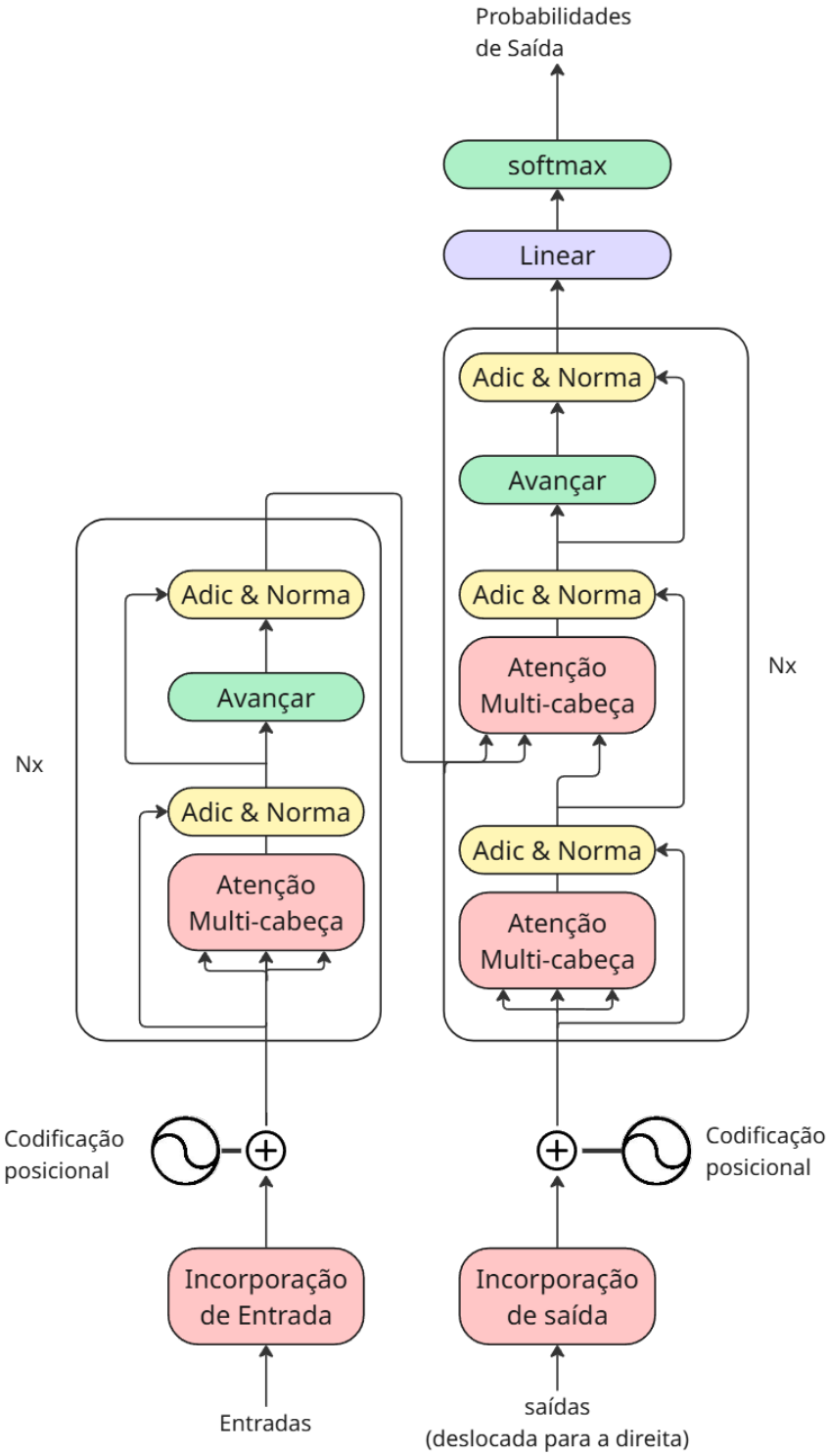


Figura 1 – Modelo da Arquitetura *Transformer*
Fonte: (VASWANI *et al.*, 2023). Tradução própria.

2.3 Geração Aumentada de Recuperação

A geração aumentada de recuperação (RAG, do inglês Retrieval-Augmented Generation) adiciona informações a mais do que o que foi previamente treinado na LLM utilizada para gerar a resposta para alguma pergunta. Existem três formas gerais de

pesquisa: o RAG ingênuo, avançado e Modular. O desenvolvimento dos dois últimos se deu pela necessidade de superar as desvantagens do RAG ingênuo.

2.3.1 RAG Ingênuo

O RAG Ingênuo foi o primeiro a ser desenvolvido. Sua arquitetura, mostrada na Figura 2, é composta por quatro etapas, as quais são descritas a seguir:

- Inexação: é a etapa do RAG onde ele irá salvar os textos separados em segmentos, e irá utilizar um LLM para gerar uma incorporação (Embedding) para os segmentos, e após gerar a incorporação, irá salvar os segmentos em um índice vetorial (GAO, Y. *et al.*, 2024).
- Consulta: é a etapa em que um usuário realiza uma consulta que pode ser respondida com base no conteúdo dos documentos, e essa consulta passa pelo mesmo processo de incorporação da etapa de indexação (GAO, Y. *et al.*, 2024).
- Recuperação: nesta etapa é feita uma comparação com a consulta transformada e os segmentos que foram feitos a indexação, e é dada uma pontuação para cada comparação, assim retornando somente os segmentos que tiveram a melhor pontuação (GAO, Y. *et al.*, 2024).
- Geração: é nesta etapa final onde os segmentos encontrados na etapa anterior são agrupados e colocados em um modelo de prompt para enviar para uma LLM responder à pergunta (GAO, Y. *et al.*, 2024).

Entretanto, este RAG ingênuo apresenta algumas deficiências, como:

- A recuperação de segmentos irrelevantes, desalinhados ou sem as informações necessárias que acontecem pelo conflito da precisão e recall.
- A geração do modelo que pode acabar gerando uma resposta imprecisa ou com um conteúdo ofensivo.
- A integração da informação recuperada, pois, dependendo da forma que for feito, pode gerar resultados desconexos, também pode gerar resultados redundantes pelo retorno da mesma informação de fontes diferentes.

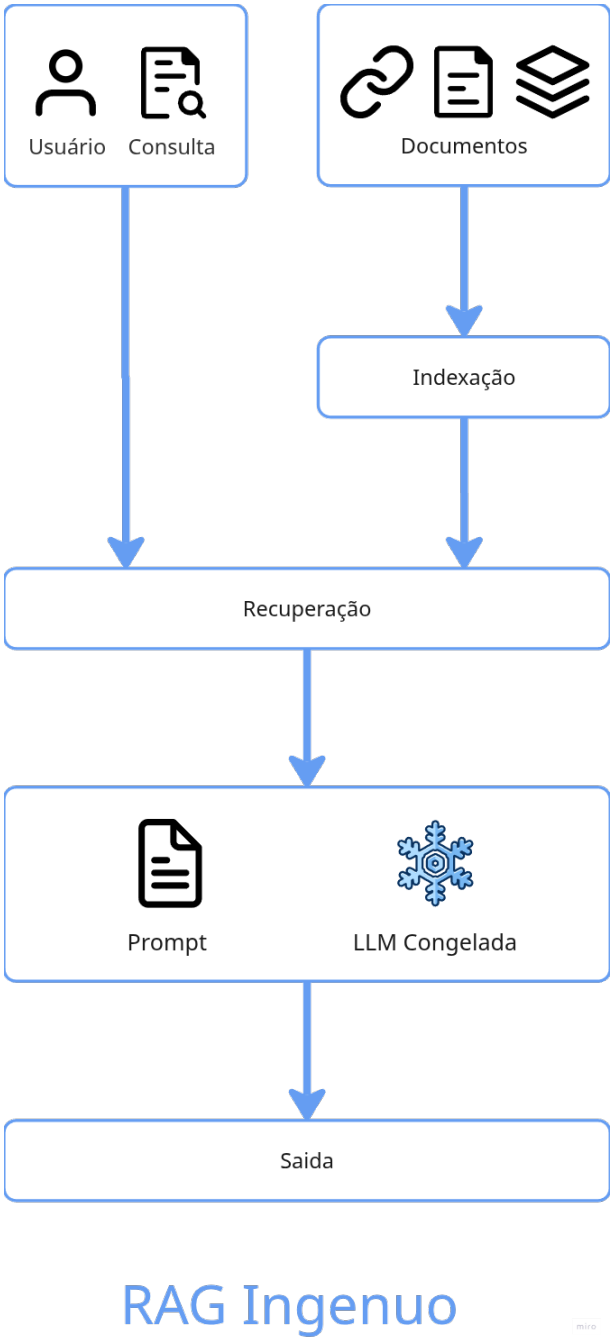


Figura 2 – Arquitetura do RAG Ingenuo.
Fonte: (GAO, Y. *et al.*, 2024). Tradução própria.

2.3.2 RAG Avançado

O RAG Avançado surge como uma solução para as deficiências do modelo Ingênuo, trazendo otimizações significativas ao longo do *pipeline* de recuperação. Primeiramente, implementa estratégias de pré-recuperação e pós-recuperação, como reescrita e expansão de consultas para otimizar as entradas no sistema de recuperação, além de técnicas de reorganização (*re-ranking*) e compressão de contexto para eliminar

informações redundantes e focar nos trechos mais relevantes. Estas estratégias podem ser observadas na Figura 3, que mostra a arquitetura do RAG Avançado.

Na etapa de indexação, o RAG Avançado introduz melhorias como o uso de janelas deslizantes para segmentação de textos e o acoplamento de metadados adicionais, o que permite filtros mais eficientes durante a recuperação. Essas melhorias reduzem significativamente os problemas de imprecisão e aumentam a eficácia da integração do contexto com o modelo de linguagem, garantindo maior fidelidade e relevância nas respostas geradas (KAMATH *et al.*, 2024).

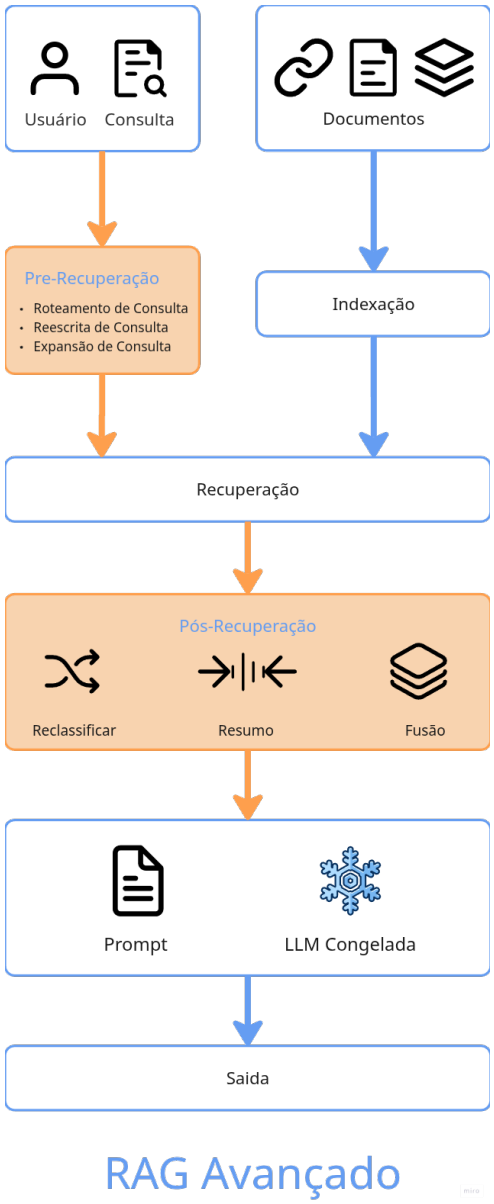


Figura 3 – Arquitetura do RAG Avançado.
Fonte: (GAO, Y. *et al.*, 2024). Tradução própria.

2.3.3 RAG Modular

O RAG Modular representa um avanço significativo ao introduzir uma arquitetura mais flexível e adaptável. Diferentemente dos modelos anteriores, o RAG Modular não se limita a um fluxo linear fixo e incorpora módulos especializados que podem ser configurados, substituídos ou ajustados conforme a demanda específica de cada tarefa (KAMATH *et al.*, 2024). A Figura 4 demonstra um pouco sobre a arquitetura do RAG modular.

Entre os novos módulos introduzidos, destacam-se os módulos a seguir:

- Módulo de memória, responsável por armazenar informações contextuais relevantes ao longo de múltiplas iterações.
- Módulo de roteamento, responsável por direcionar as consultas para diferentes fontes de dados ou pipelines de processamento.
- Módulo de predição, que reduz ruídos e redundâncias, gerando contextos mais precisos (KAMATH *et al.*, 2024).

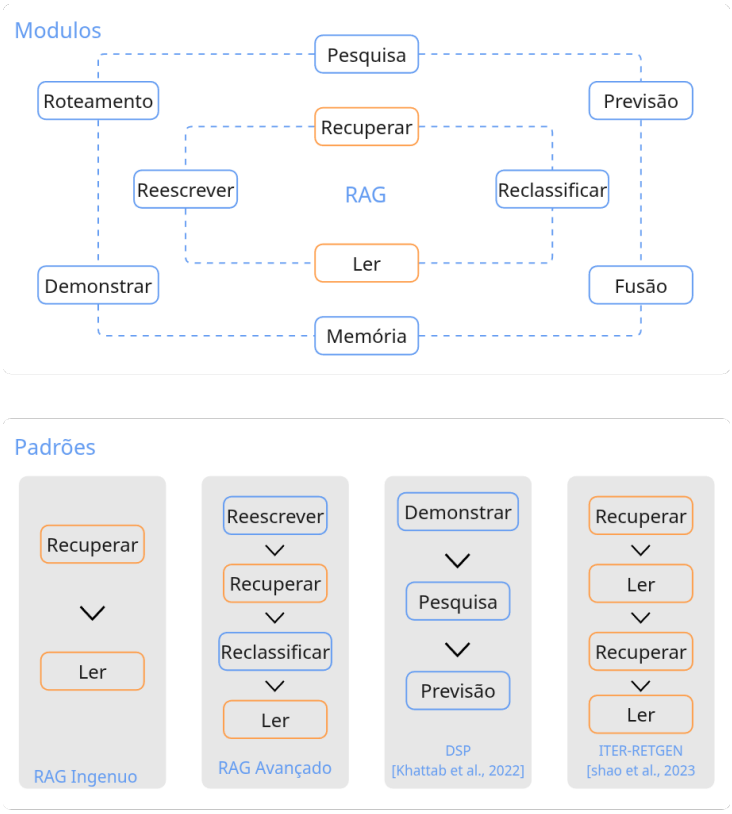


Figura 4 – Arquitetura do RAG Modular.
Fonte: (GAO, Y. *et al.*, 2024). Tradução própria.

2.4 Banco de Dados Vetorial

Os Bancos de dados Vetoriais, de forma direta, são vetores que contêm as principais características dos dados que serão armazenados, podendo ser grandes textos, vídeos, imagens. Porém, seu maior benefício pode ser a sua busca por similaridade, o que possibilita realizar pesquisas completamente diferentes das dos bancos de dados comuns, pois nos vetoriais é possível obter resultados por similaridade entre os dados (HAN; LIU; WANG, 2023a). Para entender melhor como um banco de dados vetorial funciona, vamos abordar seus principais conceitos.

2.4.1 Armazenamento

O armazenamento em um Banco de Dados Vetorial pode ser feito de algumas formas e elas são por:

2.4.1.1 Fragmentação

A fragmentação distribui os dados vetoriais em múltiplas máquinas ou *clusters*, chamados *shards*. O *sharding* pode ser implementado através do *hash-based sharding*, onde os dados são distribuídos uniformemente utilizando uma função *hash* aplicada a uma chave específica, como IDs. Outra abordagem é o *range-based sharding*, que divide os dados em *shards* com base em intervalos específicos de valores, facilitando consultas que utilizam faixas pré-definidas (HAN; LIU; WANG, 2023a).

2.4.1.2 Particionamento

Organiza os dados em partições menores e mais gerenciáveis. No particionamento de intervalo, os dados são divididos com base em intervalos específicos, como datas (ex.: mensais ou trimestrais). Já o particionamento de lista aloca os dados a partições específicas com base em valores pertencentes a uma lista pré-definida, como cores ou categorias. Essa abordagem facilita a execução de consultas mais eficientes ao permitir que apenas partições relevantes sejam acessadas (HAN; LIU; WANG, 2023a).

2.4.1.3 *Cache*

O *cache* é outra técnica importante, utilizada para armazenar dados frequentemente acessados ou recentemente utilizados em uma memória de rápido acesso, como RAM. A estratégia LRU (Least Recently Used) é uma das mais comuns, onde os dados menos utilizados são removidos do *cache* quando ele atinge sua capacidade máxima. Além disso, o *cache* particionado organiza os dados em diferentes partições e aplica políticas específicas de *cache* para cada uma delas, garantindo melhor aproveitamento dos recursos (HAN; LIU; WANG, 2023a).

2.4.1.4 Replicação

A replicação é fundamental para aumentar a disponibilidade e durabilidade dos dados, criando cópias em diferentes nós ou agrupamentos (*clusters*). No modelo de replicação sem líder (*leaderless replication*), qualquer nó pode aceitar operações de leitura e escrita, o que elimina pontos únicos de falha, mas pode introduzir desafios de consistência. Por outro lado, a replicação líder-seguidor (*leader-follower replication*) designa um nó principal (líder) para gravações, que propaga as atualizações para os nós seguidores, garantindo forte consistência, mas exigindo mecanismos de *failover* para lidar com falhas no líder (HAN; LIU; WANG, 2023a).

2.4.2 Busca

As formas de busca que os bancos de dados vetoriais realizam são formas de solucionar o problema do vizinho mais próximo, que é basicamente encontrar o dado mais semelhante ao pesquisado. Para encontrar o mais semelhante, a maioria dos algoritmos que calculam a semelhança o faz por meio das diferenças entre os dados, logo quanto menor o retorno da função, mais semelhantes os dados são entre si (HAN; LIU; WANG, 2023a). Existem duas formas principais de realizar essas pesquisas, onde cada uma possui diversos métodos.

2.4.2.1 Pesquisa de vizinho mais próximo

A pesquisa do vizinho mais próximo possui diversos algoritmos para realizá-la. Alguns deles são:

- Força Bruta (*Brute Force*): este método irá calcular a distância entre o dado a ser pesquisado e todos os salvos. Isso garante que ele irá encontrar sempre o vizinho mais próximo, porém isso gera um alto custo computacional se for um conjunto de dados grande, sua complexidade é O(n) (HAN; LIU; WANG, 2023a).
- Árvore KD (*KD-tree*): a árvore KD organiza pontos em espaços de alta dimensionalidade, particionando-os em hiperplanos perpendiculares com base nas dimensões dos dados. É eficiente para baixa dimensionalidade e fácil de implementar, mas seu desempenho degrada significativamente em dimensões altas devido à "maldição da dimensionalidade"(BENTLEY, 1975).
- Árvore-bola (*Ball-tree*): estruturas Árvore-bola particionam dados em hiperesferas, agrupando pontos semelhantes e permitindo buscas eficientes em alta dimensionalidade. Elas lidam melhor com a maldição da dimensionalidade em comparação à Árvore KD, mas exigem cálculos adicionais para manter as hiperesferas (DOLATSHAH; HADIAN; MINAEI-BIDGOLI, 2015).
- Árvore-R (*R-tree*): a árvore-R utiliza hiper-retângulos para agrupar pontos em subespaços, permitindo consultas espaciais como interseções e proximidades. É amplamente usado em aplicações geográficas e espaciais, mas seu desempenho pode ser prejudicado pela sobreposição de retângulos em altas dimensões (GUTTMAN, 1984).
- Árvore-M (*M-tree*): projetada para buscas em espaços métricos, a Árvore-M utiliza hiperesferas e oferece suporte a operações dinâmicas, como inserções e exclusões. Ele equilibra eficiência e flexibilidade, mas possui maior complexidade de implementação (CIACCIA; PATELLA; ZEZULA, 1997).

2.4.2.2 Pesquisa aproximada do vizinho mais próximo

A pesquisa aproximada do vizinho mais próximo também possui diversos algoritmos para a sua realização. Alguns deles são:

- Hashing sensível à localidade (LSH, do inglês *Locality-Sensitive Hashing*): o LSH transforma vetores de alta dimensionalidade em códigos binários que preservam proximidade e permite consultas eficientes por meio de tabelas *hash*. É ideal para reduzir memória e tempo de busca, mas pode sacrificar precisão (HAN; LIU; WANG, 2023a).
- Hashing espectral (*Spectral Hashing*): baseado na teoria espectral, este método gera funções *hash* que minimizam o erro de quantização e maximizam a variância dos códigos. É eficaz para dados que residem em variedades de baixa dimensionalidade, mas sensível a ruídos (HAN; LIU; WANG, 2023a).
- Hashing profundo (*Deep Hashing*): utilizando redes neurais profundas. Este método aprende funções *hash* que preservam características semânticas dos dados. É altamente eficaz para imagens, textos e outros dados complexos, embora exija treinamento intensivo (LIU *et al.*, 2019).
- Árvore K-means (K-means Tree): baseada em *K-means*, organiza dados hierarquicamente em agrupamentos, facilitando buscas eficientes. Funciona bem para dados naturalmente agrupados, mas depende da qualidade da inicialização do algoritmo *K-means* (HAN; LIU; WANG, 2023a).
- Pequeno mundo navegável (*Navigable Small World* - NSW): este método utiliza grafos com conexões locais e de longo alcance, permitindo buscas rápidas e eficientes em alta dimensionalidade. Ele equilibra precisão e escalabilidade, sendo ideal para diversos cenários (HAN; LIU; WANG, 2023b).
- Mundo Pequeno Navegável Hierárquico (*Hierarchical Navigable Small World* - HNSW): uma extensão do NSW, o HNSW organiza grafos em camadas hierárquicas, onde camadas superiores contêm menos pontos e conexões de maior alcance. Este método oferece alta precisão e eficiência para dados de grande escala (MALKOV; YASHUNIN, 2016).

2.4.3 Distâncias Vetoriais em Espaços Multidimensionais

A escolha do algoritmo de distância é um fator determinante para o desempenho de sistemas de busca vetorial, especialmente em aplicações de recuperação de informação semântica e sistemas baseados em *embeddings*. Cada métrica traz características matemáticas distintas que influenciam a maneira como a similaridade

entre vetores é medida. No contexto de bancos de dados vetoriais e algoritmos de busca aproximada, métricas como Euclidiana, Manhattan, Produto Escalar e Cosseno são amplamente utilizadas, cada uma com vantagens e limitações específicas.

Esta seção apresenta a definição formal de cada uma dessas métricas, discute suas propriedades e destaca os contextos nos quais elas são mais apropriadas. As análises aqui desenvolvidas têm por base tanto fundamentos teóricos clássicos quanto estudos recentes que investigam o comportamento dessas distâncias em espaços de alta dimensão.

2.4.3.1 Distância Euclidiana

A distância Euclidiana entre dois vetores $x, y \in \mathbb{R}^n$ é definida por:

$$d_{\text{Euclidiana}}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Este tipo de distância representa a menor linha reta entre dois pontos e é amplamente utilizada em problemas geométricos e de localização. Segundo (LIBERTI *et al.*, 2012), essa métrica é central em diversas aplicações, como reconstrução de formas moleculares, redes de sensores e análise de dados espaciais em geral, sendo particularmente relevante em contextos onde a magnitude das diferenças é significativa. Apesar disso, sua eficácia pode diminuir em espaços de alta dimensão devido à concentração das distâncias.

2.4.3.2 Produto Escalar

O produto escalar entre dois vetores x e y é dado por:

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$$

Embora não constitua uma métrica no sentido matemático (não satisfaz a desigualdade triangular), o produto escalar é amplamente utilizado como medida de similaridade em sistemas de recomendação e bancos de dados vetoriais. Conforme discutido por (PAN; WANG; LI, 2023), essa operação é a base para diversos mecanis-

mos de ranqueamento por similaridade e é frequentemente empregada em modelos de linguagem e sistemas baseados em *embeddings*.

2.4.3.3 Distância de Cosseno

A distância de cosseno é definida como:

$$d_{\text{Cosseno}}(x, y) = 1 - \frac{\langle x, y \rangle}{\|x\| \cdot \|y\|}$$

Essa métrica é especialmente útil em contextos onde a direção dos vetores importa mais que sua magnitude, como em sistemas de recuperação de informação semântica. (PAN; WANG; LI, 2023) apontam que, embora amplamente utilizada, a distância de cosseno pode ter limitações em tarefas de vizinhança aproximada de alta dimensão, já que vetores muito distantes ainda podem ter ângulos pequenos, levando a falsas similaridades.

2.4.3.4 Distância Manhattan

A distância Manhattan (ou L_1) é dada por:

$$d_{\text{Manhattan}}(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Diferentemente da Euclidiana, essa métrica mede a soma absoluta das diferenças entre componentes. O trabalho de (MORAES SILVA, 2024) analisa o comportamento assintótico dessa métrica em espaços de alta dimensão, demonstrando que sua distribuição tende a uma normal com expectativa $n/3$ e variância $n/18$, quando os vetores são distribuídos uniformemente. Essa propriedade a torna útil para análises estatísticas e clustering em grandes volumes de dados.

2.5 Conversão de Texto para SQL

O Texto-para-SQL é uma tarefa no campo do NLP que visa traduzir consultas em linguagem natural em instruções SQL para execução em bases de dados relacionais. Este processo é essencial para democratizar o acesso a dados estruturados, permitindo

que usuários sem conhecimento técnico em SQL interajam com bancos de dados complexos (LI *et al.*, 2023).

A conversão de Texto para SQL possui abordagens relacionadas a LLMs, como a engenharia de *prompts* e, por ela ser uma área um tanto quanto única, ela possui as suas próprias métricas para a sua avaliação.

2.5.1 Abordagens

Inicialmente, o Texto-para-SQL utilizava abordagens baseadas em regras e *templates*, que dependiam de engenharia manual significativa. Com o avanço do aprendizado profundo, métodos baseados em redes neurais sequenciais (*Seq2Seq*) tornaram-se o padrão, proporcionando mapeamento direto entre entrada e saída sem necessidade de etapas intermediárias (SHI *et al.*, 2024).

Mais recentemente, os LLMs revolucionaram a área, utilizando estratégias como:

2.5.1.1 Engenharia de *Prompts*

A engenharia de *prompts* é uma técnica que utiliza as habilidades de seguir instruções dos LLMs para gerar consultas SQL corretas com base em descrições em linguagem natural. As principais estratégias incluem:

- *Few-Shot Learning:* fornecer ao modelo exemplos de perguntas e consultas SQL correspondentes como parte do *prompt*. Isso ajuda o modelo a identificar padrões e traduzir novas perguntas (SHI *et al.*, 2024).
- *Chain-of-Thought (CoT): prompts* que incentivam o modelo a realizar etapas de raciocínio intermediárias antes de gerar a saída final. Isso é útil para consultas complexas que exigem múltiplas junções ou operações aninhadas (SHI *et al.*, 2024).

2.5.1.2 Fine-Tuning

O *fine-tuning* adapta um modelo LLM pré-treinado em dados de linguagem geral para tarefas específicas de Text-to-SQL. Essa técnica envolve:

- Treinamento Supervisionado: Ajustar o modelo com pares de entrada e saída (pergunta natural e SQL correspondente) usando *datasets* como SPIDER, WikiSQL e BIRD (LI *et al.*, 2023).
- Customização por Domínio: Para setores específicos, como finanças, onde os dados têm características únicas, é possível ajustar o modelo para incorporar valores especializados, otimizando o desempenho em bancos de dados financeiros ou médicos (SONG *et al.*, 2024).

2.5.2 Métricas de Avaliação

As métricas utilizadas em Texto-para-SQL avaliam diferentes aspectos da tarefa, desde a correspondência literal das consultas até a semântica e a eficiência computacional. Elas podem ser divididas em três categorias principais:

2.5.2.1 Correspondência

As métricas por correspondência avaliam a similaridade entre a consulta SQL gerada e a consulta de referência, independentemente da execução. Entre elas, as que se destacam são as seguintes:

- Correspondência exata (*Exact Match* - EM): mede se a consulta gerada é idêntica à referência, incluindo a ordem das cláusulas. Apesar de simples, penaliza variações semanticamente válidas com estruturas diferentes. Seu retorno é binário, indicando 1 quando a consulta gerada é idêntica à referência e 0 caso contrário (QIN *et al.*, 2022).
- Semelhança de árvore de distância de edição (*Tree Similarity of Editing Distance* - TSED): compara consultas usando árvores de sintaxe abstrata (ASTs), calculando a distância de edição entre elas. Essa métrica reconhece equivalências estruturais, mas requer ferramentas avançadas para análise. Seu retorno é uma pontuação contínua, onde valores menores indicam maior similaridade estrutural entre as árvores sintáticas das consultas (SONG *et al.*, 2024).

2.5.2.2 Execução

As métricas de execução têm como foco a semântica da consulta ao comparar os resultados da execução. Entre as existentes, as principais são:

- Precisão de execução (*Execution Accuracy* - EX): compara os resultados das consultas geradas e de referência no banco de dados. Considera consultas semanticamente equivalentes, mas depende de acesso ao banco de dados. Seu retorno é binário, com valor 1 quando os resultados das consultas gerada e de referência são idênticos, e 0 quando divergem (LI *et al.*, 2023).
- Pontuação de eficiência válida (*Valid Efficiency Score* - VES): avalia tanto a precisão quanto a eficiência computacional, medindo o tempo e os recursos necessários para executar a consulta. Útil para bancos de dados grandes e cenários de alta demanda. Seu retorno é uma pontuação composta que combina acurácia de execução com métricas de desempenho, como tempo e uso de recursos computacionais (BISWAL *et al.*, 2024).

2.5.2.3 Componentes

As métricas de componentes analisam partes específicas das consultas SQL para uma avaliação mais detalhada. Entre as existentes, as principais são as seguintes:

- Métrica de análise de consulta SQL (SQAM, do inglês *SQL Query Analysis Metric*): divide a consulta em componentes, como SELECT e WHERE, comparando-os individualmente com a referência. Seu retorno é detalhado por componente da consulta, fornecendo uma pontuação específica para partes como `SELECT`, `WHERE` e `GROUP BY` (SONG *et al.*, 2024).
- Pontuação BLEU (*BLEU Score*): adaptado da tradução automática, mede a similaridade textual entre a consulta gerada e a referência. Seu retorno é uma pontuação contínua entre 0 e 1, refletindo o grau de similaridade textual entre a consulta gerada e a referência (SONG *et al.*, 2024).

2.6 Estado da Arte

O estudo (GAO, D. *et al.*, 2023) apresenta uma análise abrangente do uso de LLMs no contexto de Text-to-SQL, com ênfase na engenharia de *prompts*. As principais contribuições incluem a proposta do DAIL-SQL, uma solução integrada que melhora a acurácia de execução para 86,6% no *benchmark* Spider. O estudo explora diversas representações de perguntas, como *prompts* baseados em SQL, e investiga estratégias de seleção e organização de exemplos para otimizar o aprendizado contextual. Além disso, destaca o impacto do ajuste fino supervisionado para modelos *open-source*, ressaltando a importância da eficiência no uso de *tokens* e sua correlação com o custo computacional.

O artigo (ZHANG *et al.*, 2023) introduz o ReFSQL, um *framework* que combina recuperação de exemplos com aprendizado contrastivo para melhorar a precisão da geração de SQL. O método utiliza um *retriever* aprimorado para identificar amostras com conhecimento específico relevante, incorporando essas informações no treinamento de LLMs. A utilização de aprendizado contrastivo com métricas como a distância de Mahalanobis possibilita uma melhor adaptação às distribuições específicas do problema. Resultados em *benchmarks* como Spider mostram ganhos consistentes de precisão e robustez, destacando a relevância do *framework* para cenários reais de consultas SQL.

O artigo (BISWAL *et al.*, 2024) propõe o paradigma Table-Augmented Generation (TAG) como solução unificada para consultas que combinam raciocínio semântico e conhecimento global. O TAG abrange três etapas principais: síntese de consulta, execução de consulta e geração de resposta. Diferente de abordagens como Text2SQL ou RAG, o TAG permite interações mais sofisticadas entre modelos de linguagem e sistemas de banco de dados, incluindo análises agregadas e razões semânticas. *Benchmarks* indicam que pipelines manuais baseados no TAG alcançam até 65% mais acurácia em consultas complexas, enfatizando seu potencial para aplicações empresariais.

O artigo (POURREZA; RAFIEI, 2023) apresenta o DIN-SQL, um método inovador que aprimora a conversão de linguagem natural para SQL por meio da decomposição da tarefa e do uso de autocorreção com LLMs. A abordagem divide a geração de SQL em quatro módulos principais: vinculação de esquema, decomposição da consulta, geração incremental do SQL e autocorreção. Essa estrutura melhora a precisão da

execução em 10% em relação a métodos convencionais *few-shot*, alcançando 85,3% de acurácia no *benchmark Spider*, superando modelos fortemente ajustados. Além disso, o DIN-SQL demonstrou resultados promissores no *benchmark* BIRD, atingindo 55,9% de precisão de execução, estabelecendo um novo estado da arte. A pesquisa destaca a importância de técnicas de decomposição e aprendizado em contexto para otimizar a precisão e eficiência de Texto-para-SQL em cenários reais.

O artigo (LEE; POLOZOV; RICHARDSON, 2021) introduz o KaggleDBQA, um novo conjunto de dados e metodologia de avaliação para sistemas Texto-para-SQL, com foco na generalização para bancos de dados reais. A pesquisa destaca desafios enfrentados por modelos de ponta ao serem aplicados em ambientes reais, como a dificuldade de vinculação de esquemas (*schema linking*) devido a nomes de colunas abreviados e terminologia específica do domínio. Além disso, o estudo propõe uma abordagem baseada em *few-shot learning*, onde modelos têm acesso a exemplos de consultas SQL previamente anotadas, resultando em um aumento de 13,2% na precisão da geração de SQL. Os experimentos demonstram que a inclusão de documentação do banco melhora significativamente o desempenho, sugerindo que estratégias híbridas de aprendizado supervisionado e contextual são mais eficazes para aplicações práticas.

O artigo (MAJ *et al.*, 2024) foca na aplicação prática de Text-to-SQL no suporte ao cliente. Este estudo avalia a integração de ferramentas como o Vanna.AI. Durante o artigo, é apresentada a estrutura de funcionamento do Vanna.AI conforme a Figura 5. A abordagem utiliza a técnica de RAG para melhorar a qualidade das consultas geradas, otimizando o acesso a informações críticas sem exigir conhecimento de SQL dos usuários. Experimentos com modelos como Llama3:70b-instruct e Gemma2:27b mostram que simplificar as informações do esquema do banco de dados aumenta a precisão das respostas. O estudo também ressalta que o excesso de contexto pode sobrecarregar os modelos, destacando a importância de balancear contexto e eficiência.

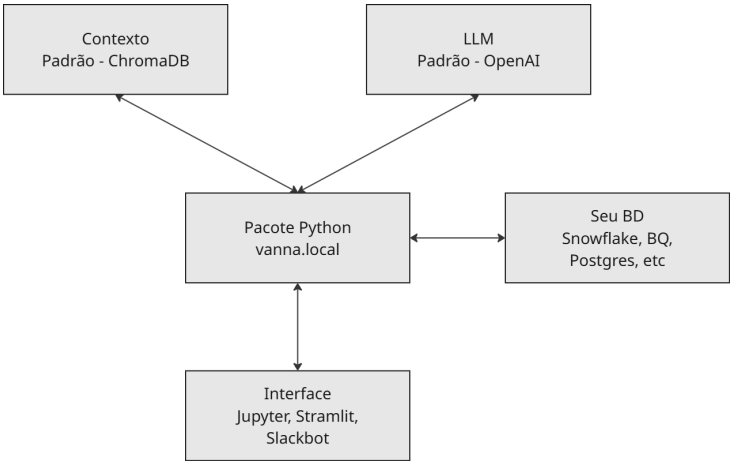


Figura 5 – Arquitetura do Vanna.AI.
Fonte: (MAJ *et al.*, 2024). Tradução própria.

Com base nos artigos apresentados, observa-se uma evolução significativa na engenharia de *prompts* e no uso de algoritmos de recuperação para otimizar a conversão de linguagem natural para SQL. No entanto, a comparação direta entre diferentes estratégias de *prompting* e algoritmos de similaridade ainda é um tema pouco explorado. Neste contexto, o presente projeto busca preencher essa lacuna ao avaliar sistematicamente o impacto de diferentes técnicas de *prompts* e métodos de recuperação de similaridade na precisão e eficiência de geração de consultas SQL.

3 MATERIAIS E MÉTODOS

A presente seção tem como objetivo detalhar os materiais e os métodos adotados para a condução do experimento proposto, conforme representado na Figura 6. Cada subseção descreve, de forma sistemática, uma etapa fundamental do processo, abrangendo desde a preparação e organização dos dados até os procedimentos de avaliação dos resultados obtidos.

A primeira etapa aborda a construção do banco de dados vetorial (ver Seção 3.1). Nessa fase, foram coletadas as estruturas de definição de dados (DDL) e realizados os procedimentos de inserção das informações necessárias para viabilizar a indexação e a busca semântica.

Em seguida, é apresentada a escolha do modelo de linguagem (LLM) utilizado no experimento (ver Seção 3.2). Essa etapa inclui a análise comparativa entre modelos disponíveis, bem como os critérios adotados para a seleção final do modelo mais adequado ao problema proposto.

A terceira etapa concentra-se na implementação da arquitetura de conversão de texto para SQL (ver Seção 3.3). Nela, descreve-se a estrutura do sistema desenvolvido, incluindo os componentes responsáveis pela interpretação das perguntas em linguagem natural e pela geração das consultas SQL correspondentes.

Na quarta etapa discute-se a aplicação das técnicas de *prompt* (ver Seção 3.4), detalhando-se o processo de agrupamento das respostas geradas, a aplicação das métricas de avaliação (como Exact Match, Execution Accuracy e BLEU Score) e a comparação dos resultados entre diferentes estratégias.

A quinta etapa consiste na análise dos algoritmos de distância vetorial utilizados na recuperação dos contextos (ver Seção 3.5). Foram comparadas diferentes métricas, como a distância de Cosseno e Euclidiana, para verificar seu impacto na relevância dos trechos recuperados e, consequentemente, na qualidade das consultas geradas.

Por fim, a sexta etapa trata da implementação da interface (ver Seção 3.6), que visa disponibilizar de forma prática e acessível todo o sistema proposto, permitindo a interação com o modelo e a visualização dos resultados.

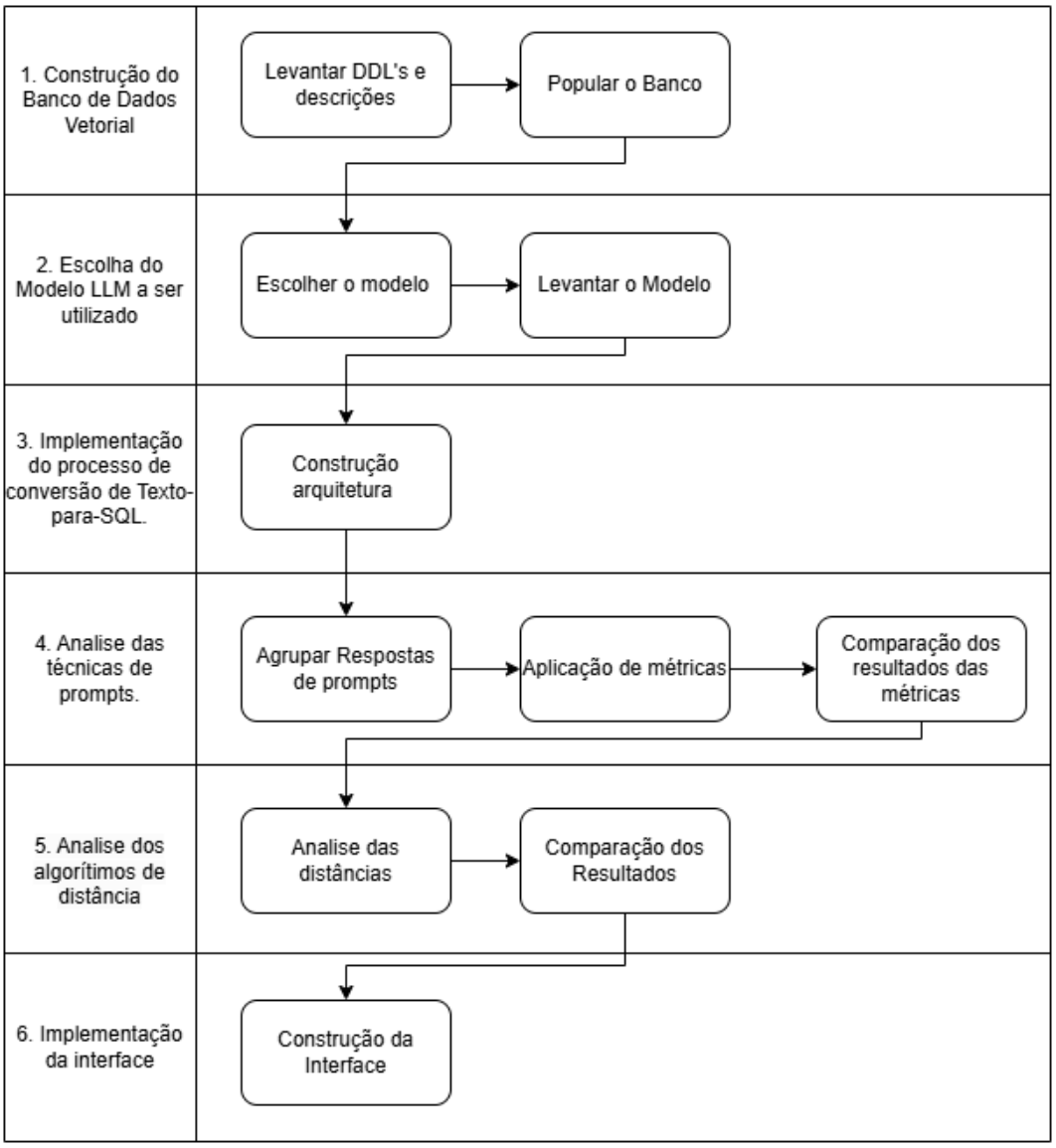


Figura 6 – Metodologia pretendida para o trabalho.
Fonte: Autoria Própria (2025).

3.1 Construção do Banco de Dados Vetorial

Os DDLs disponibilizados no artigo (LEE; POLOZOV; RICHARDSON, 2021) continham, além da definição das tabelas, comandos de inserção de dados (*inserts*). No entanto, para a construção inicial do banco vetorial, foram utilizados apenas os comandos de criação das tabelas, desconsiderando a população inicial de dados.

Essa escolha permitiu focar na estrutura das entidades e relações do banco, preparando o ambiente para inserções futuras conforme a lógica do experimento. O artigo citado oferecia diversos DDLs para escolha, conforme ilustrado na Figura 7. Para este trabalho, foi selecionado o DDL referente ao domínio de usinas nucleares.

	Nuclear	Crime	Pesticide	MathScore	Baseball	Fires	WhatCD	Soccer
#Tables	1	1	2	3	5	1	2	2
#Columns	15	6	34	15	44	19	10	37
#Fine-tuning Examples	10	9	16	9	12	12	13	6
#Test Examples	22	18	34	19	27	25	28	12

Figura 7 – Estatísticas por database do KaggleDBQA
Fonte: (LEE; POLOZOV; RICHARDSON, 2021).

As tabelas do banco de dados escolhidas para o experimento foram:

- `countries`: contém os nomes dos países com seus respectivos códigos ISO-3166;
- `nuclear_power_plant_status_type`: define os status de operação das usinas;
- `nuclear_reactor_type`: traz os tipos e descrições dos reatores;
- `nuclear_power_plants`: reúne informações sobre as usinas nucleares, incluindo localização, tipo de reator, status operacional, país, capacidade e datas de funcionamento, conectando-se às demais tabelas por meio de chaves estrangeiras.

Além da estrutura relacional, tornou-se necessário incorporar uma solução para recuperação semântica baseada em vetores, com o intuito de complementar os *prompts* utilizados. Nesse contexto, o banco vetorial escolhido foi o Qdrant[1]. A decisão se deu, principalmente, pela sua compatibilidade com o algoritmo de busca HNSW (*Hierarchical Navigable Small World*), reconhecido na literatura por seu alto desempenho e precisão na recuperação de vetores similares, conforme indicado por (MALKOV; YASHUNIN, 2016). Esse algoritmo se destaca especialmente em tarefas de busca aproximada em grandes volumes de dados.

O Qdrant é um banco de dados vetorial *open source* voltado para aplicações de *machine learning* e sistemas de recomendação. Ele permite armazenar vetores de alta dimensão com grande eficiência e fornece mecanismos de busca aproximada baseados em métricas como distância Euclidiana, cosseno e Manhattan. Sua arquitetura foi projetada para facilitar integrações com APIs modernas e frameworks de IA, sendo uma escolha robusta para sistemas que dependem de recuperação vetorial rápida e precisa.

[1] Qdrant.

Na Figura 8, temos uma representação gráfica de como é a estrutura de armazenamento do Qdrant, que organiza os vetores em coleções e permite que programadores, via clientes em Python, Rust, Go ou TypeScript, realizem buscas eficientes usando métricas como distância euclidiana, produto escalar ou similaridade cosseno, retornando os vetores mais próximos junto com suas informações adicionais (*payload*).

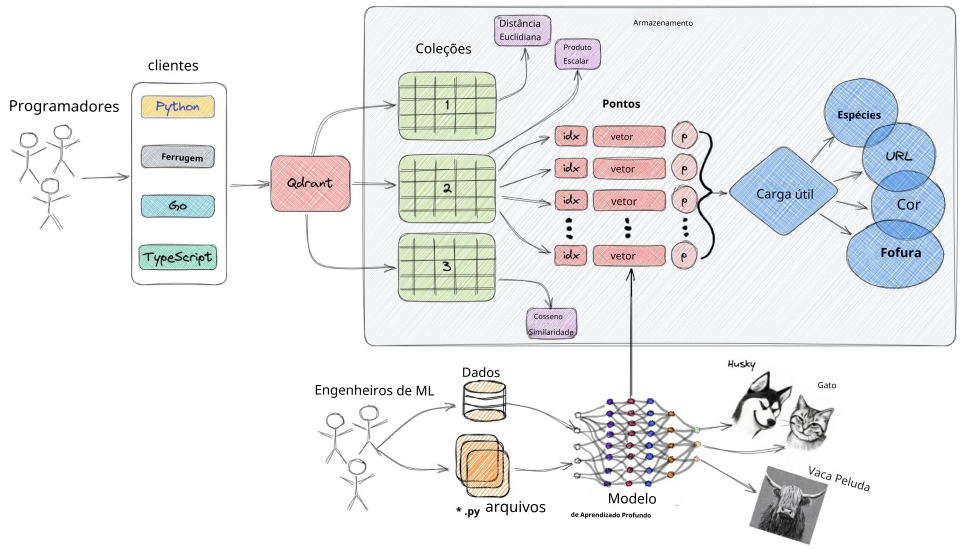


Figura 8 – Visão geral do Qdrant
Fonte: (QDRANT, s.d.) Tradução própria.

Para viabilizar o uso do Qdrant, foi empregada uma imagem Docker oficial, o que facilitou sua implantação no ambiente experimental. A integração entre os DDLs e o Qdrant foi realizada por meio da biblioteca VannaAI, que atuou como intermediária no acesso e manipulação dos dados, permitindo traduzir a estrutura relacional para o formato vetorial exigido pelo sistema de busca.

O VannaAI[2] é uma biblioteca de código aberto especializada em auxiliar o processo de geração automática de consultas SQL com o uso de inteligência artificial. Seu funcionamento se baseia em três pilares: indexação semântica das estruturas do banco de dados, recuperação de contexto por similaridade vetorial e construção dinâmica de prompts personalizados. Ele abstrai diversas etapas técnicas, como o pré-processamento de esquemas relacionais, a criação de embeddings e o roteamento da entrada para o modelo adequado. Podemos observar na Figura 9, a estrutura de processamento que o vannaAI realiza.

[2] VannaAI.

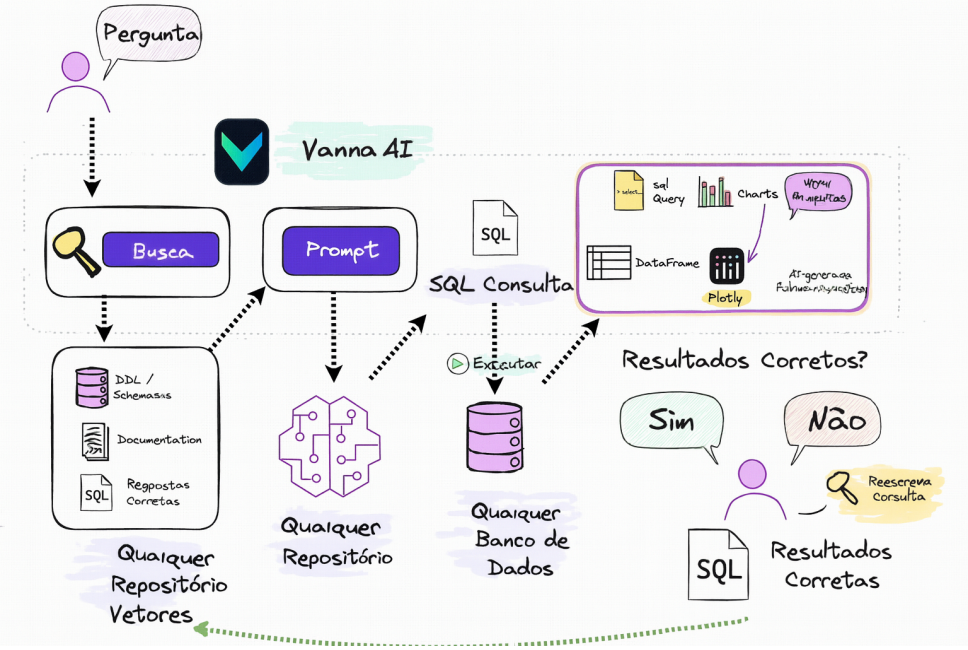


Figura 9 – Visão geral do VannaAI
Fonte: (VANNA.AI, 2025) Tradução própria.

Paralelamente, foi construído um banco relacional utilizando o mesmo DDL dentro de um container Docker com MySQL. Neste caso, os dados também foram populados com os comandos `INSERT` disponibilizados pelo artigo, permitindo realizar testes com dados reais.

3.2 Escolha do Modelo LLM a ser utilizado

O modelo de linguagem de grande escala selecionado para este experimento foi o LLaMA 3.1[3], desenvolvido pela equipe da Meta AI. A escolha foi fundamentada em uma análise comparativa apresentada por (YANG *et al.*, 2024), que avaliou diferentes modelos open source, conforme podemos ver na Figura 10. Em seu estudo, as LLMs foram avaliadas utilizando os benchmarks spider e bird.

O modelo adotado possui 8 bilhões de parâmetros, representando uma versão intermediária entre modelos menores e grandes modelos como o LLaMA 3.1 de 70 bilhões. A opção por essa variante foi motivada pelo bom equilíbrio entre custo computacional e desempenho, além da possibilidade de ser executada em ambientes com hardware acessível. Na Figura 11 temos um comparativo que influenciou na escolha do modelo, o comparativo foi feito pela (META, 2024).

[3] LLaMA 3.1.

Open-Source LLMs						
LLaMA2-7B (Touvron et al., 2023)	28.0	23.8	-	7.1	-	
LLaMA2-7B-Chat (Touvron et al., 2023)	36.9	34.9	-	11.3	-	
Qwen-1.8B (Bai et al., 2023)	54.8	48.6	-	13.2	-	
LLaMA2-13B-Chat (Touvron et al., 2023)	49.6	45.5	-	14.2	-	
LLaMA2-13B (Touvron et al., 2023)	47.4	39.4	-	15.3	-	
StarCoder-3B (Li et al., 2023d)	52.7	47.0	-	15.3	-	
StarCoder-7B (Li et al., 2023d)	60.7	55.1	-	17.2	-	
DeepSeek-Coder-1.3B (Guo et al., 2024)	59.3	53.2	-	22.0	-	
♣CodeLLaMA-7B (Roziere et al., 2023)	61.1	52.3	-	22.5	-	
♡CodeLLaMA-13B (Roziere et al., 2023)	61.7	53.5	-	22.9	-	
CodeLLaMA-7B-Instruct (Roziere et al., 2023)	63.4	54.2	-	23.0	-	
DeepSeek-Coder-1.3B-Instruct (Guo et al., 2024)	53.2	48.7	-	24.1	-	
StarCoder-15B (Li et al., 2023d)	63.9	57.9	-	24.4	-	
CodeLLaMA-13B-Instruct (Roziere et al., 2023)	62.3	52.5	-	24.7	-	
Qwen-7B (Bai et al., 2023)	63.6	54.5	-	26.1	-	

Figura 10 – LLMs open source analisadas pelo artigo de (YANG *et al.*, 2024)
Fonte: (YANG *et al.*, 2024).

Category / Benchmark	Llama 3.1 8B	Gemma 2 9B IT	Mistral 7B Instruct	Llama 3.1 70B	Mixtral 8x22B Instruct	GPT 3.5 Turbo
General MMLU (0-shot, CoT)	73.0	72.3 (5-shot, non-CoT)	60.5	86.0	79.9	69.8
MMLU PRO (5-shot, CoT)	48.3	-	36.9	66.4	56.3	49.2
IFEval	80.4	73.6	57.6	87.5	72.7	69.9
Code HumanEval (0-shot)	72.6	54.3	40.2	80.5	75.6	68.0
MBPP EvalPlus (base) (0-shot)	72.8	71.7	49.5	86.0	78.6	82.0
Math GSM8K (8-shot, CoT)	84.5	76.7	53.2	95.1	88.2	81.6
MATH (0-shot, CoT)	51.9	44.3	13.0	68.0	54.1	43.1
Reasoning ARC Challenge (0-shot)	83.4	87.6	74.2	94.8	88.7	83.7
GPQA (0-shot, CoT)	32.8	-	28.8	46.7	33.3	30.8
Tool use BFCL	76.1	-	60.4	84.8	-	85.9
Nexus	38.5	30.0	24.7	56.7	48.5	37.2
Long context ZeroSCROLLS/QuALITY	81.0	-	-	90.5	-	-
InfiniteBench/En.MC	65.1	-	-	78.2	-	-
NIH/Multi-needle	98.8	-	-	97.5	-	-
Multilingual Multilingual MGSM (0-shot)	68.9	53.2	29.9	86.9	71.1	51.4

Figura 11 – Avaliação feita pela MetaAI (YANG *et al.*, 2024)
Fonte: (META, 2024).

Um dos critérios mais relevantes na escolha foi a necessidade de recursos gráficos. O modelo requer, no mínimo, 12 GB de VRAM para funcionar de forma fluida em GPU. Modelos maiores exigem mais de 16 GB de memória de vídeo, o que inviabilizaria a execução local em estações de trabalho comuns.

A execução foi realizada por meio da ferramenta Ollama, que oferece uma

interface padronizada via Docker para rodar LLMs de código aberto localmente. Embora facilite a inicialização do ambiente, foi necessário configurar manualmente a utilização da GPU dentro do contêiner Docker para garantir o uso da aceleração por hardware, o que foi essencial para reduzir o tempo de inferência e melhorar o desempenho nos testes.

A flexibilidade do Ollama foi um diferencial. A ferramenta permite alternar entre modelos e versões de forma prática, com apenas um comando. Essa característica contribuiu para uma execução simplificada e padronizada durante os experimentos.

Além disso, foi avaliada a compatibilidade com a biblioteca VannaAI, utilizada como camada de integração entre o LLM e o banco vetorial. Como o LLaMA 3 possui amplo suporte para sistemas baseados em RAG, sua adoção favoreceu o uso de técnicas modernas de conversão de linguagem natural para SQL.

3.3 Implementação do processo de conversão de Texto-para-SQL

A Figura 12 ilustra o pipeline implementado para conversão de linguagem natural em SQL. O fluxo inicia com uma pergunta do usuário, que é utilizada para buscar contexto no banco vetorial. Essa busca retorna definições (DDLs) e documentação relevantes.

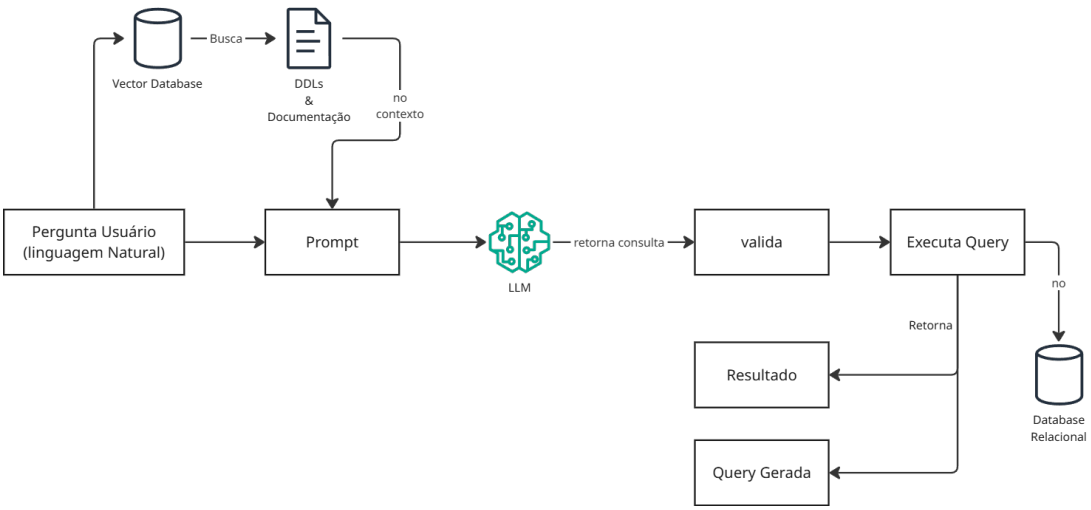


Figura 12 – Pipeline da conversão de texto-para-SQL.
Fonte: Autoria Própria.

O conteúdo recuperado é incorporado ao prompt junto à pergunta original. Esse prompt estruturado é então enviado ao modelo de linguagem (LLM), que retorna uma consulta SQL gerada com base na compreensão do contexto.

Após a geração, a consulta é validada e enviada para execução no banco de dados relacional. O sistema retorna o resultado da execução, bem como a query gerada, possibilitando análises posteriores.

A infraestrutura de conversão de Texto-para-SQL foi projetada para transformar perguntas em linguagem natural em consultas SQL válidas e executáveis. Esse processo inicia-se com a extração dos DDLs do banco vetorial, os quais são utilizados para fornecer o esquema de referência ao modelo de linguagem.

O sistema, então, estrutura um prompt contendo tanto a pergunta do usuário quanto o contexto extraído das definições do banco, permitindo que o modelo LLM compreenda as entidades envolvidas e gere uma resposta apropriada. Esse pipeline é responsável por unificar os componentes semânticos e estruturais da base de dados em um único fluxo de geração.

A integração entre os componentes foi realizada utilizando o framework VannaAI, que atua como camada intermediária entre o banco vetorial e o modelo LLM. O VannaAI oferece suporte nativo para comunicação com o Qdrant, facilitando a busca de informações relevantes no espaço vetorial a partir de embeddings da pergunta do usuário. Além disso, o framework possui compatibilidade com diferentes modelos de linguagem de grande escala, como os que estão na Figura 13, o que permite ao sistema alternar entre LLMs sem a necessidade de grandes reconfigurações.

As funcionalidades do VannaAI tornam o desenvolvimento de sistemas Texto-para-SQL mais acessível e padronizado. No contexto deste trabalho, o uso do VannaAI foi fundamental para implementar uma arquitetura baseada em RAG, na qual o conhecimento extraído do banco vetorial é incorporado à entrada do LLM, aumentando a precisão da geração das queries.

Diferentes tipos de *prompts* foram adicionados ao sistema, os *prompts* adicionados foram os mais bem citados pelo (SHI *et al.*, 2024), como o raciocínio em cadeia (chain-of-thought) e o few-shot. A escolha entre eles ocorre de forma programática conforme os parâmetros definidos para cada execução. Esses formatos auxiliam o modelo a estruturar melhor seu processo de geração ao fornecer instruções mais detalhadas e exemplos anteriores.

Também foi implementada a avaliação da geração dos SQLs feitos, essa avaliação foi feita utilizando três métricas, sendo cada uma delas de uma área de avaliação, na área de correspondência foi escolhido o EX, em execução foi escolhido o

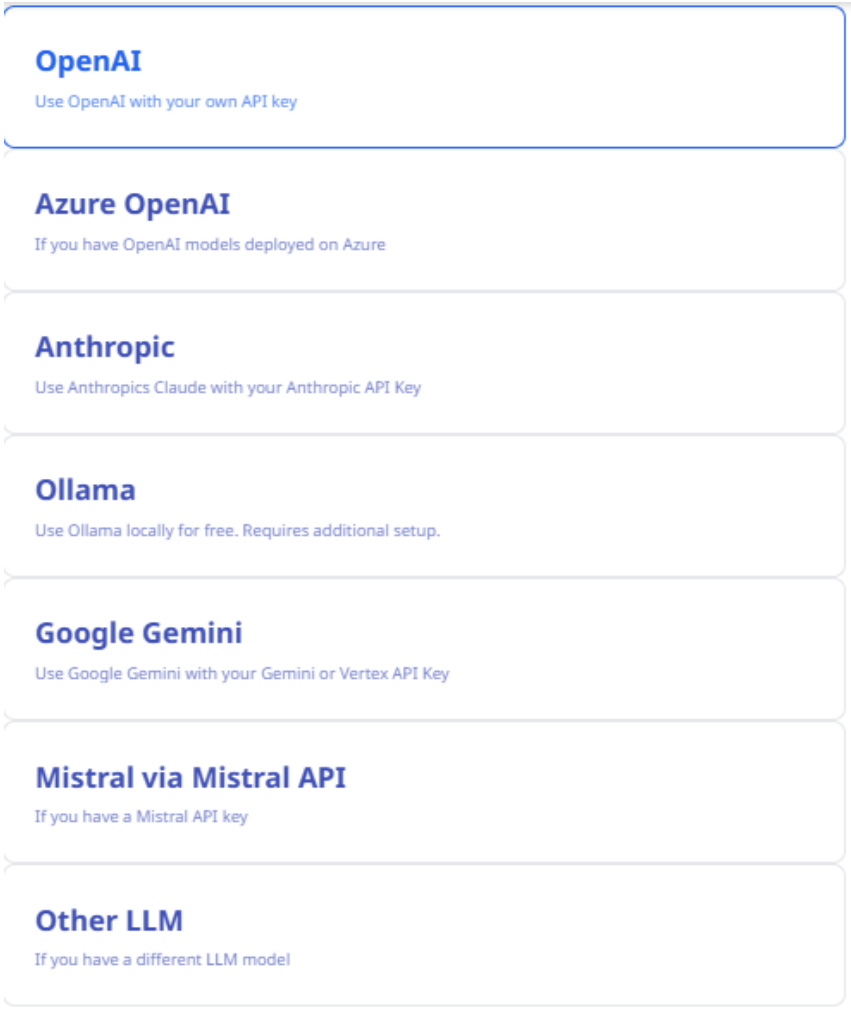


Figura 13 – Fontes de modelos com suporte no VannaAI.
Fonte: (VANNA.AI, 2025).

EX, e para componentes do escolhido a pontuação BLEU.

A estrutura dos *prompts* foi organizada por meio de uma classe Enum chamada Tags conforme a Figura 14, que define todas as marcações utilizadas nas respostas do modelo. Essas tags, como `<THINKING>`, `<REFLECTION>` e `<FINAL_OUTPUT>`, padronizam a entrada e facilitam a extração posterior da consulta SQL de forma confiável e estruturada.

Para extrair corretamente a consulta SQL das respostas do modelo, foram utilizadas expressões regulares que priorizam o conteúdo entre as tags `<FINAL_OUTPUT>...</FINAL_OUTPUT>`. Caso a resposta não possua essas tags, o sistema tenta identificar blocos SQL por meio de outras estratégias, como blocos markdown ou instruções diretas iniciadas por palavras-chave SQL.

Também foi implementado um sistema de registro com a biblioteca logging, responsável por monitorar etapas relevantes do pipeline. Entre os pontos registrados

```python
class Tags(Enum):
    def __str__(self):
        return str(self.value)

    DDL_LIST_OPEN = "<DDL_LIST>"
    DDL_LIST_CLOSE = "</DDL_LIST>"
    TABLE_OPEN = "<TABLE>"
    TABLE_CLOSE = "</TABLE>"
    CONTEXT_DOCS_OPEN = "<CONTEXT_DOCS>"
    CONTEXT_DOCS_CLOSE = "</CONTEXT_DOCS>"
    RESP_GUIDELINES_OPEN = "<RESPONSE_GUIDELINES>"
    RESP_GUIDELINES_CLOSE = "</RESPONSE_GUIDELINES>"
    THINKING_OPEN = "<THINKING>"
    THINKING_CLOSE = "</THINKING>"
    REFLECTION_OPEN = "<REFLECTION>"
    REFLECTION_CLOSE = "</REFLECTION>"
    CONCLUSION_OPEN = "<CONCLUSION>"
    CONCLUSION_CLOSE = "</CONCLUSION>"
    OUTPUT_OPEN = "<FINAL_OUTPUT>"
    OUTPUT_CLOSE = "</FINAL_OUTPUT>"
    COT_OPEN = "<COT>"
    COT_CLOSE = "</COT>"
```

Figura 14 – Enum utilizado nos prompts.
Fonte: Autoria Própria.

estão a montagem do *prompt*, a inserção de exemplos, a execução da pergunta e a extração da consulta. Isso permite maior rastreabilidade durante os testes e facilita a análise de erros.

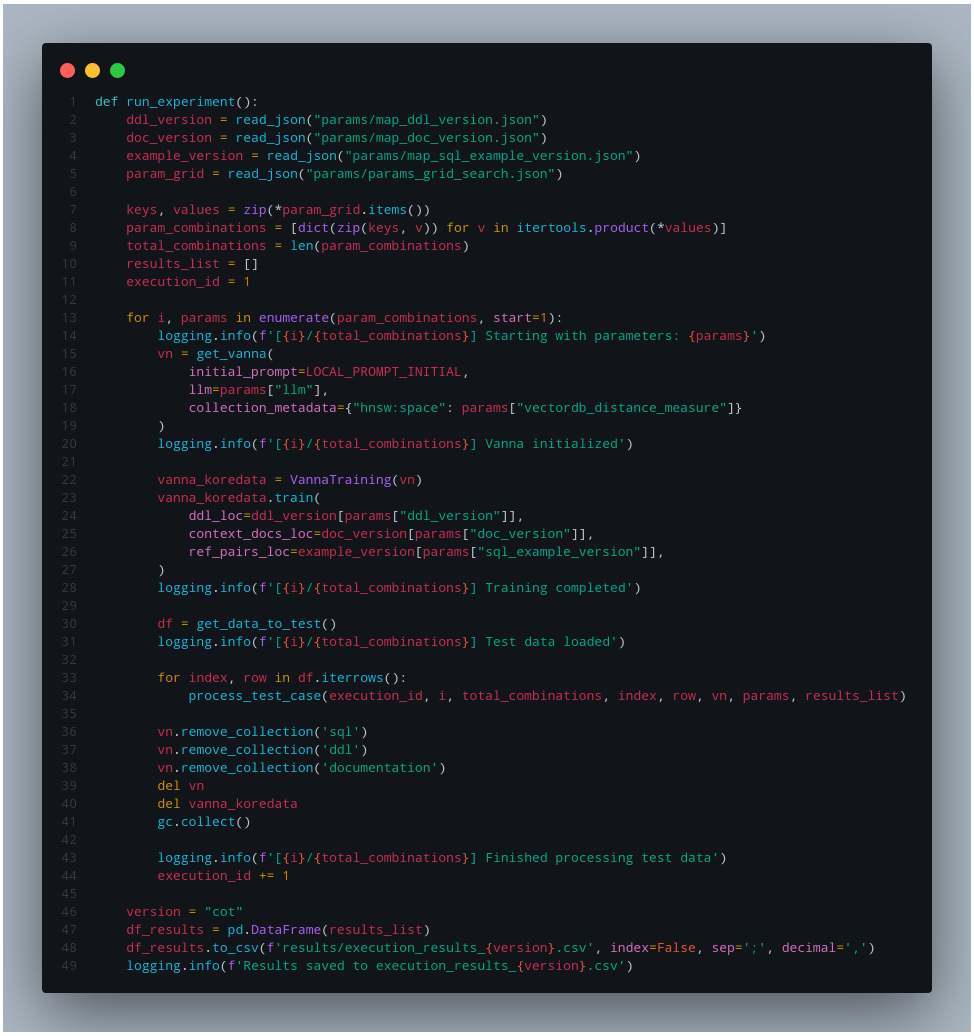
3.4 Análise das técnicas de *prompts*

Para a análise das técnicas de *prompting* utilizadas neste trabalho, foi desenvolvido um algoritmo de busca em grade (grid search) que automatiza a execução de testes combinando diferentes variações de *prompts* com distintas métricas de similaridade vetorial. O objetivo principal dessa abordagem foi explorar de maneira sistemática o impacto de cada configuração nos resultados gerados pelo modelo LLM, permitindo comparações diretas com base em métricas padronizadas de avaliação Text-to-SQL.

O experimento é orquestrado pela função `run_experiment()`, que atua como ponto de entrada do pipeline. Nessa função, são carregados os arquivos de configuração contendo os caminhos das versões dos DDLs, documentação e exemplos de SQL (via arquivos JSON). Também é carregado o *grid* de parâmetros (`params_grid_search.json`) contendo todas as combinações de variáveis experimentais, como o tipo de *prompt* e

a métrica de distância vetorial utilizada. Cada combinação é iterada utilizando itertools.product(), permitindo gerar um conjunto exaustivo de cenários de teste.

Para cada combinação de parâmetros, é inicializada uma instância do VannaAI (`get_vanna()`), que recebe o modelo LLM e os metadados da coleção vetorial, como o tipo de distância (ex.: cosine, euclidean, dot). Em seguida, a função `VannaTraining.train()` é responsável por realizar o treinamento da instância com base nos DDLs, documentos de contexto e exemplos anotados. Essa etapa garante que o modelo receba as informações necessárias do domínio antes de responder às perguntas.

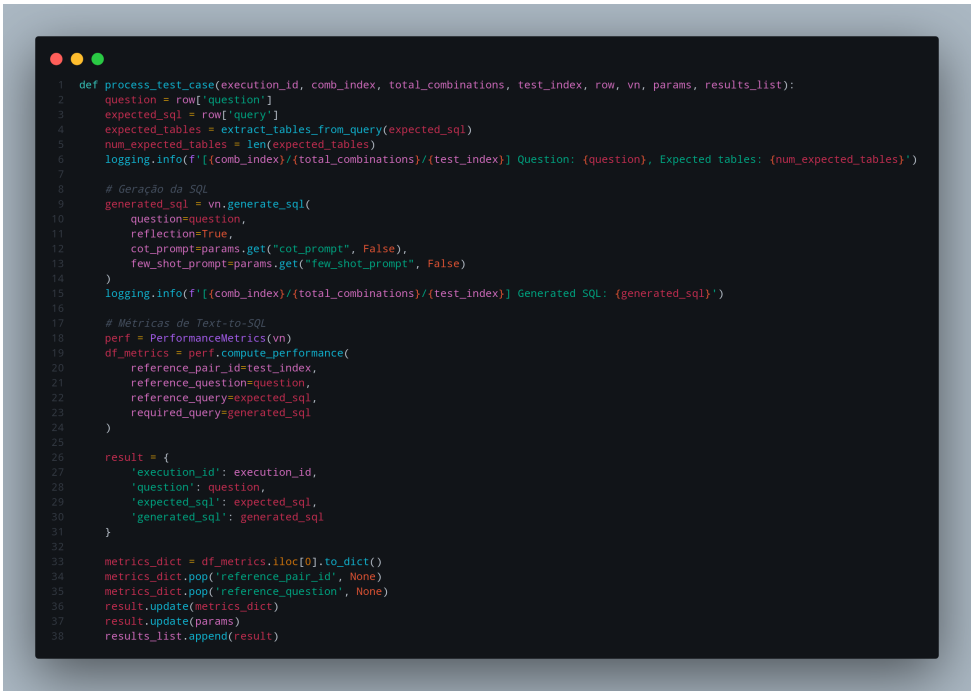


```python
def run_experiment():
    ddl_version = read_json("params/map_ddl_version.json")
    doc_version = read_json("params/map_doc_version.json")
    example_version = read_json("params/map_sql_example_version.json")
    param_grid = read_json("params/params_grid_search.json")

    keys, values = zip(*param_grid.items())
    param_combinations = [dict(zip(keys, v)) for v in itertools.product(*values)]
    total_combinations = len(param_combinations)
    results_list = []
    execution_id = 1

    for i, params in enumerate(param_combinations, start=1):
        logging.info(f'[{i}/{total_combinations}] Starting with parameters: {params}')
        vn = get_vanna(
            initial_prompt=LOCAL_PROMPT_INITIAL,
            llm=params["llm"],
            collection_metadata={"hnsw:space": params["vectordb_distance_measure"]}
        )
        logging.info(f'[{i}/{total_combinations}] Vanna initialized')

        vanna_koredata = VannaTraining(vn)
        vanna_koredata.train(
            ddl_loc=ddl_version[params["ddl_version"]],
            context_docs_loc=doc_version[params["doc_version"]],
            ref_pairs_loc=example_version[params["sql_example_version"]],
        )
        logging.info(f'[{i}/{total_combinations}] Training completed')

        df = get_data_to_test()
        logging.info(f'[{i}/{total_combinations}] Test data loaded')

        for index, row in df.iterrows():
            process_test_case(execution_id, i, total_combinations, index, row, vn, params, results_list)

        vn.remove_collection('sql')
        vn.remove_collection('ddl')
        vn.remove_collection('documentation')
        del vn
        del vanna_koredata
        gc.collect()

        logging.info(f'[{i}/{total_combinations}] Finished processing test data')
        execution_id += 1

    version = "cot"
    df_results = pd.DataFrame(results_list)
    df_results.to_csv(f'results/execution_results_{version}.csv', index=False, sep=';', decimal=',')
    logging.info(f'Results saved to execution_results_{version}.csv')
```

Figura 15 – Função para processamento geral do Grid search.
Fonte: Autoria própria (2025).

Após o treinamento, é carregado o conjunto de dados de teste por meio da função `get_data_to_test()`. Em seguida, cada linha do conjunto de testes é processada pela função `process_test_case()`, que está na figura 16, que realiza a execução do experimento individual para cada pergunta SQL. Essa função executa as seguintes etapas:

1. Leitura da pergunta e da query esperada;
2. Geração da SQL pelo modelo, com base na pergunta e nos parâmetros ativos de prompting (`few_shot_prompt, cot_prompt`);
3. Cálculo das métricas de avaliação por meio da classe PerformanceMetrics, incluindo: EM, EX, BLEU Score;
4. Armazenamento dos resultados em um dicionário, incluindo métricas, parâmetros e as queries envolvidas.



```python
def process_test_case(execution_id, comb_index, total_combinations, test_index, row, vn, params, results_list):
    question = row['question']
    expected_sql = row['query']
    expected_tables = extract_tables_from_query(expected_sql)
    num_expected_tables = len(expected_tables)
    logging.info(f'[{comb_index}/{total_combinations}/{test_index}] Question: {question}, Expected tables: {num_expected_tables}')

    # Geração da SQL
    generated_sql = vn.generate_sql(
        question=question,
        reflection=True,
        cot_prompt=params.get("cot_prompt", False),
        few_shot_prompt=params.get("few_shot_prompt", False)
    )
    logging.info(f'[{comb_index}/{total_combinations}/{test_index}] Generated SQL: {generated_sql}')

    # Métricas de Text-to-SQL
    perf = PerformanceMetrics(vn)
    df_metrics = perf.compute_performance(
        reference_pair_id=test_index,
        reference_question=question,
        reference_query=expected_sql,
        required_query=generated_sql
    )

    result = {
        'execution_id': execution_id,
        'question': question,
        'expected_sql': expected_sql,
        'generated_sql': generated_sql
    }

    metrics_dict = df_metrics.iloc[0].to_dict()
    metrics_dict.pop('reference_pair_id', None)
    metrics_dict.pop('reference_question', None)
    result.update(metrics_dict)
    result.update(params)
    results_list.append(result)
```

Figura 16 – Função para o processamento de cada teste.
Fonte: Autoria própria (2025).

A avaliação automatizada dos resultados foi realizada por meio da classe `PerformanceMetrics`, desenvolvida especificamente para este projeto. Essa classe encapsula os métodos de cálculo das principais métricas de avaliação.

O método `exact_match_score` verifica se a SQL gerada é idêntica à de referência após a normalização. A `execution_accuracy_score` executa ambas as *queries* em um banco de dados MySQL e compara os conjuntos retornados.

Por fim, o `bleu_score` aplica uma métrica baseada em n-gramas, originalmente usada em tradução automática, para medir a semelhança textual entre as *queries*. O encapsulamento dessas funções facilitou a reutilização do código e a integração com o *pipeline*.

Os resultados de cada execução experimental foram armazenados em dicionários com os campos: `execution_id`, `question`, `expected_sql` e `generated_sql`.

A esses dados, foram adicionadas as métricas computadas, e tudo foi estruturado em um *DataFrame* com as colunas: `reference_pair_id`, `reference_question`, `reference_query`, `required_query`, `exact_match_score`, `execution_accuracy_score` e `bleu_score`.

Ao final da execução de todas as combinações previstas no *grid search*, o conteúdo do *DataFrame* foi exportado para um arquivo CSV. Isso permitiu a consolidação dos resultados e a realização de análises estatísticas posteriores, que subsidiaram a avaliação comparativa entre os diferentes métodos de *prompting*.

Os tipos de *prompt* avaliados no *grid search* foram três: o *prompt* padrão do VannaAI, o Few-Shot — que inclui exemplos de entrada e saída — e o Chain-of-Thought (CoT), que orienta o modelo a desenvolver um raciocínio antes da geração.

Cada um desses foi testado com as métricas de similaridade suportadas pelo Qdrant: distância Euclidiana, distância de Cosseno, distância Manhattan e produto escalar. Essa estratégia automatizada permitiu realizar testes controlados, reprodutíveis e comparáveis, garantindo maior confiabilidade aos resultados apresentados na seção de avaliação.

3.5 Análise dos algoritmos de distância

A etapa de avaliação dos algoritmos de distância foi fundamental para entender como a escolha da métrica vetorial influencia o desempenho da geração de consultas SQL. Esta análise foi conduzida após a execução do *grid search* descrito na Seção 3.4, que combinou diferentes variações de *prompting* com distintas métricas de similaridade.

Os algoritmos de distância utilizados foram: Distância Euclidiana, Distância de Cosseno, Produto Escalar e Distância Manhattan, pois esses são os algoritmos disponibilizados pelo Qdrant. Esses algoritmos são comumente adotados em bancos de dados vetoriais por apresentarem características distintas na forma como medem a similaridade entre vetores de *embeddings*.

Durante os testes, cada consulta foi processada por meio da função `process_test_case()`, sendo avaliada conforme três métricas principais: EM, EX e *BLEU Score*. Essas métricas foram implementadas na classe `PerformanceMetrics`, responsável por padronizar a análise e garantir uniformidade na comparação dos resultados.

As respostas geradas, bem como suas respectivas métricas de avaliação e a distância utilizada em cada execução, foram agregadas automaticamente no arquivo `.csv` final gerado ao término do *grid search*. Isso possibilitou uma análise unificada dos resultados, facilitando a identificação de padrões e variações de desempenho entre os algoritmos testados.

Por fim, essa abordagem possibilitou não apenas a comparação entre estratégias de *prompting*, mas também a identificação das métricas vetoriais mais eficazes para tarefas de recuperação semântica no contexto de sistemas Texto-para-SQL.

3.6 Implementação da Interface

Como parte da metodologia desenvolvida, foi criada uma interface em notebook Jupyter para realizar a integração com o restante da aplicação. Essa interface permite explorar o funcionamento da geração de consultas SQL a partir de linguagem natural utilizando o modelo LLaMA 3.1 8B.

Inicialmente, foram definidos os caminhos para os arquivos necessários, incluindo o DDL do banco de dados, descrições textuais das tabelas, pares de pergunta e SQL de referência, além do prompt inicial de interação com o modelo. Esses caminhos são carregados dinamicamente por meio da biblioteca *dotenv*, permitindo a parametrização do ambiente de forma flexível.

Em seguida, configura-se a instância do modelo via servidor Ollama, especificando-se parâmetros como o nome do modelo (*llama3.1:8b*), temperatura de geração (*0.2*) e o prompt inicial. Essa configuração é utilizada para inicializar uma instância da classe *VannaTCC*, responsável pela comunicação com o modelo de linguagem.

O treinamento da instância é realizado por meio da classe *VannaTraining*, que incorpora os dados de estrutura (DDL), contexto documental e perguntas com SQL de referência. Esse processo habilita o sistema a contextualizar corretamente a base de dados-alvo durante a geração das queries.

Na etapa de inferência, o usuário insere uma pergunta em linguagem natural. A função `generate_sql` é então chamada com a opção `few_shot=True`, ativando exemplos anteriores como guia para a geração. A *query* gerada é exibida ao usuário e, em seguida, executada diretamente no banco de dados, com tratamento de exceções para falhas na execução.

Por fim, destaca-se que a execução foi realizada em ambiente local com suporte a GPU, necessário para o processamento eficiente do modelo LLaMA 3.1 8B. A imagem do Ollama utilizada foi manualmente configurada para ativar o uso da GPU por meio da interface Docker.

4 RESULTADOS

Esta seção apresenta os resultados obtidos a partir dos experimentos realizados com diferentes estratégias de *prompting* aplicadas à tarefa de conversão de linguagem natural para SQL. Os testes envolveram três tipos de *prompt*: o padrão do VannaAI, Few-Shot e CoT, os combinando com três métricas de distância vetorial fornecidas pelo banco Qdrant: distância Euclidiana, distância de Cosseno e produto escalar. Ao todo, foram analisadas 32 perguntas por configuração, totalizando centenas de consultas geradas e avaliadas.

A análise foi conduzida com base em três métricas amplamente utilizadas em tarefas Text-to-SQL: EM, que mede a igualdade literal entre a query gerada e a de referência; EX, que compara os resultados da execução das queries no banco de dados; e BLEU Score, que avalia a similaridade textual entre as consultas, mesmo que não sejam idênticas.

4.1 Análise por Prompt

A abordagem Few-Shot apresentou o melhor desempenho entre todas as testadas. Embora nenhuma das consultas geradas tenha sido idêntica à referência (EM = 0), a técnica obteve resultados positivos em EX, demonstrando que várias queries, mesmo com sintaxe diferente, retornaram os mesmos resultados esperados no banco. O BLEU Score médio foi de 0,567, o mais alto entre os três prompts, evidenciando uma boa similaridade textual entre as consultas geradas e as de referência.

Na Tabela 1, observa-se que, no melhor exemplo, a única diferença entre a *query* gerada e a original foi a substituição de count(name) por count(*), variação que não comprometeu a execução correta. No pior caso, a consulta gerada foi irrelevante, utilizando um SELECT * genérico e sem filtros adequados. Ainda assim, essa técnica demonstrou mais consistência e proximidade com o comportamento esperado para aplicações reais.

Tabela 1 – Exemplo de resultados Few-Shot

Índice Pergunta	Pergunta	Query de Referência	Query Gerada	EM	EX	BLEU
Q01	What are the top 10 countries with most number of operational plants?	`select country from nuclear_power_plants where status = "operational" group by country order by count(name) desc limit 10`	`select country from nuclear_power_plants where status = "operational" group by country order by count(*) desc limit 10;`	FALSE	1	0,93
Q30	What is the main source of the information for this table?	`select source from nuclear_power_plants group by source order by count(*) desc limit 1`	`select * from nuclear_power_plants;`	FALSE	0	0,01

O few-shot obteve aproximadamente 20 perguntas das 32, onde houve uma pequena diferença entre o SQL de referência e o SQL gerado pelo modelo. Podemos observar essas diferenças na Tabela .

Tabela 2 – Comparação

Índice Pergunta	Pergunta	Query de Referência	Query Esperada
Q06	How many kinds of nuclear reactor model in the world?	`select count(distinct reactormodel) from nuclear_power_plants`	`select count(distinct reactor_model) from nuclear_power_plants;`
Q19	What nuclear reactor model is the most popular?	`select reactormodel from nuclear_power_plants group by reactormodel order by count(*) desc limit 1`	`select reactor_model from nuclear_power_plants group by reactor_model order by count(*) desc limit 1;`

Já a técnica Chain-of-Thought obteve resultados inferiores. Assim como no Few-Shot, nenhuma correspondência exata foi registrada (EM = 0), mas diferentemente do Few-Shot, nenhuma das consultas geradas teve sucesso na execução (Execution Accuracy também foi 0). O BLEU Score médio foi de apenas 0,086, indicando que a maioria das queries estava distante, inclusive em termos textuais, das esperadas.

Como evidenciado na Tabela 3, os exemplos mostram que o modelo frequentemente produziu respostas em linguagem natural, ao invés de SQL, ou utilizou filtros e cláusulas incorretas. A técnica, que visa encorajar o modelo a “pensar em voz alta”, não se mostrou eficaz neste domínio, e seu uso sem ajustes adicionais pode comprometer a geração de respostas válidas para consultas SQL.

Tabela 3 – Exemplo de resultados COT

Índice Pergunta	Pergunta	Query de Referência	Query Gerada	EM	EX	BLEU
Q23	How many nuclear power plants are in preparation to be used in Japan?	`select count(*) from nuclear_power_plants where country = "japan" and Status = "under construction"`	`select count(*) from nuclear_power_plants where country = 'japan' and status = 'in_preparation';`	FALSE	0	0,58
Q21	What are operational nuclear power plants in Japan called?	`select name from nuclear_power_plants where status = "operational" and country = "japan"`	`nuclear power station`	FALSE	0	0

Também podemos ver na Tabela 4 alguns exemplos de alucinações que ocorreram durante os testes com o COT. Em diversos casos, é possível observar que o modelo tenta responder à pergunta baseado no seu conhecimento, em vez de utilizar o contexto fornecido.

Tabela 4 – Comparação de alucinações do COT

Índice Pergunta	Pergunta	Query de Referência	Query Esperada
Q04	Which country is Kaiga-4 built in?	`select country from nuclear_power_plants where name = "kaiga-4"`	`india`
Q05	How many PHWR are there today?	`select count(*) from nuclear_power_plants where reactortype = "phwr"`	`de acordo com o iaea, existem cerca de 30 reatores phwr em operação em todo o mundo. além disso, há outros projetos e planos para construir mais reatores phwr nos próximos anos.`
Q17	How many operating nuclear station in France?	`select count(*) from nuclear_power_plants where country = "france" and status = "operational"`	`de acordo com as informações disponíveis até 2023, a frança tem cerca de 58 reatores nucleares em operação. a maioria desses reatores é operada pela edf e está localizada em usinas como fessenheim, paluel, penly, cattenom, flamanville e gravelines.`

Por fim, o prompt padrão do VannaAI apresentou o pior desempenho entre os três avaliados. Nenhuma das queries geradas obteve sucesso em EM (EM = 0) ou em EX (EX = 0). Além disso, o BLEU Score médio foi de apenas 0,0012, valor extremamente baixo, o que indica uma grande divergência textual entre as queries

geradas e as queries de referência. Como podemos ver na Tabela 5, no seu melhor caso, o *prompt* padrão se saiu muito inferior se comparado com os demais *prompts*.

Em todas as perguntas testadas, o *prompt* padrão não gerou nenhuma query, apenas tentou responder à pergunta baseado nos dados que o modelo foi treinado.

Tabela 5 – Exemplo de resultados padrão

Índice Pergunta	Pergunta	Query de Referência	Query Gerada	EM	EX	BLEU
Q10	Which country has only one nuclear power plants?	`select country from nuclear_power_plants group by country having count(name) = 1`	`I don't have information about which country has only one nuclear power plant.`	FALSE	0	0,015
Q04	Which country is Kaiga-4 built in?	`select country from nuclear_power_plants where name = "Kaiga-4"`	`Kaiga-4 is a nuclear power plant located in Karnataka, India.`	FALSE	0	0

Esse resultado demonstra que a estrutura básica do prompt padrão, por si só, não fornece contexto suficiente para que o modelo compreenda corretamente a estrutura esperada da consulta SQL. As respostas geradas, em muitos casos, estavam incompletas, mal formadas ou sequer representavam uma consulta executável. Dessa forma, o uso desse prompt, sem nenhum tipo de otimização ou exemplo contextual, mostrou-se ineficaz para a tarefa proposta.

Ao comparar os melhores e piores casos de cada técnica, observa-se uma diferença significativa de desempenho. As Tabelas 6 e 7 apresentam essa comparação de forma consolidada, permitindo visualizar o contraste entre as estratégias testadas.

No pior cenário, o Few-Shot ainda conseguiu gerar uma query SQL completa, enquanto o COT apenas retornou uma resposta textual. Já o prompt padrão falhou completamente em ambas as tentativas.

Nos melhores exemplos, o Few-Shot demonstrou maior precisão, enquanto COT e Padrão apresentaram baixo alinhamento estrutural e semântico. Dessa forma, conclui-se que, para este caso de conversão de texto para SQL, a técnica Few-Shot apresenta desempenho superior.

Tabela 6 – Comparação dos melhores resultados

prompt	EM	EX	BLEU
Few-shot	FALSE	1	0,93
COT	FALSE	0	0,58
Padrão	FALSE	0	0,015

Tabela 7 – Comparação dos piores resultados

prompt	EM	EX	BLEU
Few-shot	FALSE	0	0,01
COT	FALSE	0	0
Padrão	FALSE	0	0

A Tabela 8 apresenta uma comparação direta entre os resultados obtidos por cada tipo de prompt (Few-Shot, Chain-of-Thought e Padrão) em um conjunto específico de perguntas do experimento. Essa estrutura permite observar como cada abordagem se comportou individualmente frente às mesmas entradas. Nota-se que o Few-Shot obteve, de forma geral, as maiores pontuações de BLEU e, em alguns casos, foi a única técnica a gerar consultas executáveis corretamente (EX = 1).

Em contraste, os prompts do tipo COT e Padrão mostraram desempenho inferior em todos os casos listados, com BLEU Score frequentemente muito baixos ou nulos. Essa tabela reforça a superioridade da técnica Few-Shot tanto em consistência quanto em proximidade semântica com as queries de referência, consolidando sua eficácia mesmo em um recorte mais restrito e detalhado da avaliação.

Tabela 8 – Comparação entre as perguntas em cada prompt

Índice Pergunta	Prompt	EM	EX	BLEU
Q01	Few-Shot	FALSE	1	0,935
	COT	FALSE	0	0,263
	Padrão	FALSE	0	0,001
Q04	Few-Shot	FALSE	0	0,840
	COT	FALSE	0	0
	Padrão	FALSE	0	0
Q10	Few-Shot	FALSE	0	0,698
	COT	FALSE	0	0,025
	Padrão	FALSE	0	0,015
Q21	Few-Shot	FALSE	0	0,477
	COT	FALSE	0	0
	Padrão	FALSE	0	0
Q23	Few-Shot	FALSE	0	0,670
	COT	FALSE	0	0,583
	Padrão	FALSE	0	0,001
Q30	Few-Shot	FALSE	0	0,010
	COT	FALSE	0	0,003
	Padrão	FALSE	0	0,002

4.2 Análise por Distância Vetorial

Além da comparação entre os tipos de *prompt*, foi realizada uma análise para verificar o impacto das métricas de distância vetorial na qualidade da recuperação do contexto e, consequentemente, na geração das queries.

A Figura 17 apresenta a variação do BLEU Score com quatro métricas distintas de distância: cosseno, euclidiana, manhattan e produto escalar. Cada subfigura mostra a distribuição de pontuações para a técnica Few-Shot, de acordo com a métrica utilizada no Qdrant.

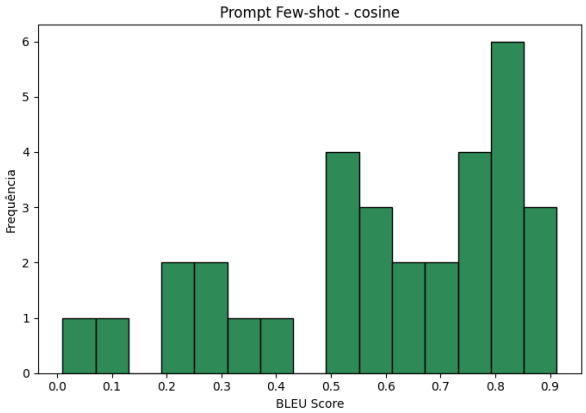
Na subfigura 17(a), observa-se que a métrica de cosseno teve uma distribuição concentrada entre 0,5 e 0,9. A maior frequência ocorreu nas faixas superiores, com destaque para a faixa entre 0,8 e 0,9. Isso sugere alta similaridade textual entre as queries geradas e as de referência, indicando que a métrica de cosseno favorece boas recuperações semânticas.

A subfigura 17(b), que mostra a distância euclidiana, apresenta uma distribuição relativamente uniforme, com leve concentração entre 0,7 e 0,9. Há menos ocorrências nas faixas inferiores, o que indica desempenho intermediário que é melhor do que o Manhattan, porem menos consistente que cosseno.

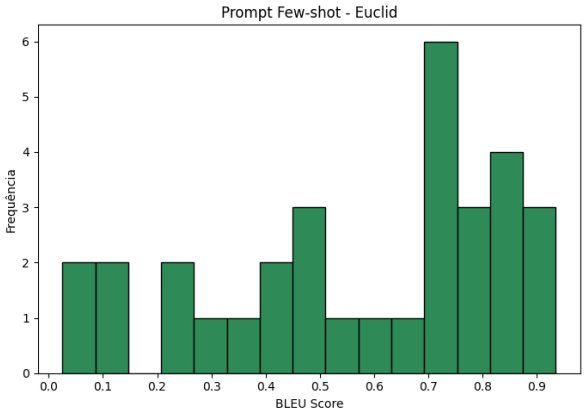
A subfigura 17(c), referente à distância de Manhattan, revela uma distribuição mais dispersa. Embora também haja alta concentração nas faixas intermediária para superior (0,5 a 0,9), observa-se maior frequência de valores baixos, incluindo uma quantidade significativa de casos com BLEU próximo de zero. Isso evidencia instabilidade na qualidade da recuperação.

Já na subfigura 17(d), que representa o produto escalar, a distribuição é semelhante à do cosseno, com forte presença entre 0,5 e 0,9 e seis ocorrências no intervalo de 0,8–0,9. No entanto, há também alguns casos de valores baixos (próximos de 0), o que aponta para maior sensibilidade à variação dos vetores e à magnitude, afetando a precisão da seleção.

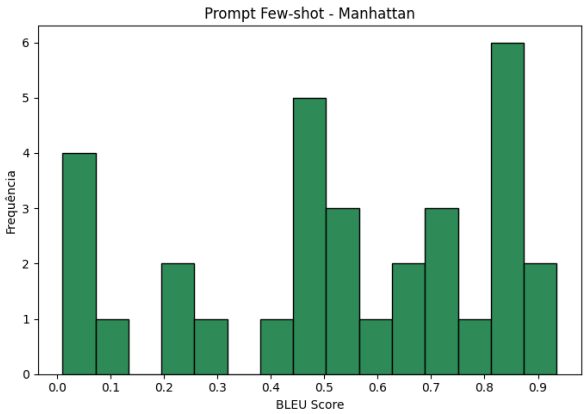
Os experimentos realizados mostraram que a combinação da técnica Few-Shot com distância de Cosseno é a mais promissora no contexto avaliado, obtendo as melhores médias e resultados individuais. Essa configuração demonstrou maior robustez para recuperar os trechos corretos do banco vetorial e gerar consultas que, mesmo sem serem idênticas, são semanticamente válidas e executáveis.



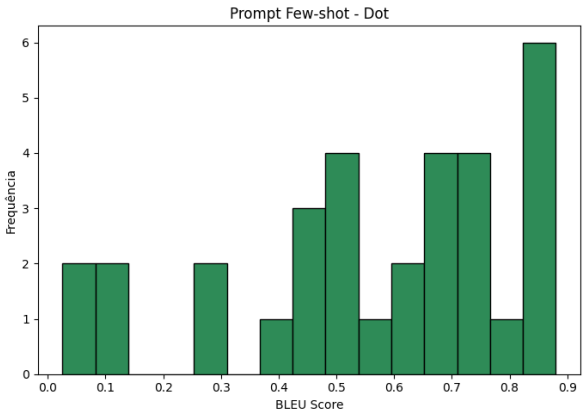
(a) Variação da pontuação com a distância cosseno.



(b) Variação da pontuação com a distância Euclidiana.



(c) Variação da pontuação com a distância Mangattan.



(d) Variação da pontuação com a distância Produto Escalar.

Figura 17 – Etapas do pipeline de conversão de texto para SQL.
Fonte: Autoria Própria.

Por outro lado, tanto o *prompt* padrão quanto o Chain-of-Thought apresentaram limitações severas. O padrão falhou na geração de consultas coerentes e válidas, enquanto o CoT não conseguiu alinhar sua estrutura de raciocínio à tarefa de geração SQL, resultando em respostas incoerentes ou em linguagem natural.

Os resultados reforçam a importância de avaliar cuidadosamente tanto a estratégia de *prompting* quanto a métrica de recuperação vetorial em sistemas Text-to-SQL. Pequenas mudanças nesses elementos podem impactar fortemente a qualidade das consultas geradas. O uso de técnicas como Few-Shot, combinadas com distâncias vetoriais mais eficazes como o cosseno, pode ser a chave para desenvolver sistemas mais precisos e úteis em aplicações reais.

5 CONCLUSÃO

O presente trabalho teve como objetivo geral avaliar diferentes técnicas de *prompting* e métodos de cálculo de distância em bancos de dados vetoriais, visando à implementação de um sistema de conversão de linguagem natural para SQL com apoio de modelos de linguagem de larga escala (LLMs). A proposta culminou na construção de uma aplicação funcional, capaz de gerar e executar consultas em banco de dados relacionais a partir de entradas em linguagem natural.

Inicialmente, foram investigadas as técnicas de prompting classificadas como melhores pelo artigo (SHI *et al.*, 2024): o prompt padrão da biblioteca VannaAI, o Few-Shot e o Chain-of-Thought (CoT). Em paralelo, avaliou-se o impacto de diferentes métricas de distância vetorial que são disponibilizadas pelo Qdrant, sendo as únicas possíveis de utilizar na ferramenta, na qualidade da recuperação do contexto: cosseno, euclidiana, manhattan e produto escalar. Os testes foram realizados com 32 perguntas de referência extraídas do artigo (LEE; POLOZOV; RICHARDSON, 2021), sendo as respostas avaliadas por métricas como Exact Match (EM), Execution Accuracy (EX) e BLEU Score.

Entre os métodos analisados, o prompt Few-Shot apresentou o desempenho mais consistente. Apesar da ausência de correspondência exata entre as queries geradas e as de referência, a técnica foi a única a gerar consultas SQL executáveis e semanticamente próximas das esperadas, com destaque para o BLEU Score médio de 0,567. Já as abordagens padrão e CoT apresentaram desempenho inferior, destacando-se negativamente pela geração de respostas incompletas, irrelevantes ou em linguagem natural.

Quanto às métricas de distância, observou-se que a distância de cosseno foi a mais eficaz na recuperação semântica de contexto, refletindo em melhores pontuações de BLEU. Essa métrica apresentou distribuição concentrada entre 0,5 e 0,9, enquanto outras métricas, como Manhattan e produto escalar, demonstraram maior instabilidade nos resultados.

Dessa forma, conclui-se que a combinação da técnica Few-Shot com a métrica de cosseno representa a melhor estratégia para a tarefa proposta, evidenciando o papel central do design de *prompts* e da escolha da métrica de similaridade no desempenho de sistemas Text-to-SQL baseados em LLMs.

Além da avaliação experimental, um dos objetivos específicos também foi atingido: a construção de uma ferramenta prática para consulta a bancos de dados relacionais. A aplicação desenvolvida integra o modelo de linguagem com um pipeline completo de recuperação vetorial e geração de SQL, permitindo que usuários realizem perguntas em linguagem natural e obtenham resultados diretamente do banco de dados. Esse sistema demonstra o potencial da proposta em aplicações reais, servindo como base para evoluções futuras.

Como trabalho futuro, sugere-se a realização de uma análise mais detalhada, com o intuito de identificar o desempenho das técnicas Few-Shot e, em especial, do CoT, que tendem a apresentar melhores resultados. Além disso, recomenda-se realizar testes com diferentes modelos de LLM, comparando seus desempenhos.

REFERÊNCIAS

ALTAIR, Junior. O processamento de linguagem natural aplicado à Arquivologia: uma análise das publicações científicas nacionais e internacionais. **Ufsc.br**, Florianópolis, SC., 2023. DOI: https://repositorio.ufsc.br/handle/123456789/254074. Disponível em: https://repositorio.ufsc.br/handle/123456789/254074.

BENTLEY, Jon Louis. Multidimensional binary search trees used for associative searching. **Communications of the ACM**, v. 18, n. 9, p. 509–517, set. 1975. DOI: https://doi.org/10.1145/361002.361007.

BISWAL, Asim *et al.* **Text2SQL is Not Enough: Unifying AI and Databases with TAG**. [*S. l.: s. n.*], 2024. arXiv: 2408.14717 [cs.DB]. Disponível em: https://arxiv.org/abs/2408.14717.

CIACCIA, Paolo; PATELLA, Marco; ZEZULA, Pavel. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. *In*: PROCEEDINGS of the 23rd International Conference on Very Large Data Bases. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997. (VLDB '97), p. 426–435. ISBN 1558604707.

DOLATSHAH, Mohamad; HADIAN, Ali; MINAEI-BIDGOLI, Behrouz. Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces. **CoRR**, abs/1511.00628, 2015. arXiv: 1511.00628. Disponível em: http://arxiv.org/abs/1511.00628.

GAO, Dawei *et al.* **Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation**. [*S. l.: s. n.*], 2023. arXiv: 2308.15363 [cs.DB]. Disponível em: https://arxiv.org/abs/2308.15363.

GAO, Yunfan *et al.* **Retrieval-Augmented Generation for Large Language Models: A Survey**. [*S. l.: s. n.*], 2024. arXiv: 2312.10997 [cs.CL]. Disponível em: https://arxiv.org/abs/2312.10997.

GUTTMAN, Antonin. R-trees. **ACM SIGMOD Record**, v. 14, n. 2, p. 47, jun. 1984. DOI: https://doi.org/10.1145/971697.602266.

HAN, Yikun; LIU, Chunjiang; WANG, Pengfei. **A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge**. [*S. l.: s. n.*], 2023a. arXiv: 2310.11703 [cs.DB]. Disponível em: https://arxiv.org/abs/2310.11703.

HAN, Yikun; LIU, Chunjiang; WANG, Pengfei. **A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge**. [*S. l.: s. n.*], 2023b. arXiv: 2310.11703 [cs.DB]. Disponível em: https://arxiv.org/abs/2310.11703.

KAMATH, Uday *et al.* Retrieval-Augmented Generation. *In*: LARGE Language Models: A Deep Dive: Bridging Theory and Practice. Cham: Springer Nature Switzerland, 2024. P. 275–313. ISBN 978-3-031-65647-7. DOI: 10.1007/978-3-031-65647-7_7. Disponível em: https://doi.org/10.1007/978-3-031-65647-7_7.

LEE, Chia-Hsuan; POLOZOV, Oleksandr; RICHARDSON, Matthew. **KaggleDBQA: Realistic Evaluation of Text-to-SQL Parsers**. [*S. l.: s. n.*], 2021. arXiv: 2106.11455 [cs.CL]. Disponível em: https://arxiv.org/abs/2106.11455.

LI, Jinyang *et al.* **Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs**. [*S. l.: s. n.*], 2023. arXiv: 2305.03111 [cs.CL]. Disponível em: https://arxiv.org/abs/2305.03111.

LIBERTI, Leo *et al.* **Euclidean distance geometry and applications**. [*S. l.: s. n.*], 2012. arXiv: 1205.0349 [q-bio.QM]. Disponível em: https://arxiv.org/abs/1205.0349.

LIU, Haomiao *et al.* Deep Supervised Hashing for Fast Image Retrieval. **Int. J. Comput. Vision**, Kluwer Academic Publishers, USA, v. 127, n. 9, p. 1217–1234, set. 2019. ISSN 0920-5691. DOI: 10.1007/s11263-019-01174-4. Disponível em: https://doi.org/10.1007/s11263-019-01174-4.

LUIZ VILLELA M. MIONI, José; RENATA S. C. DE BARBOSA, Cinthyan; FANTINELI C. DE OLIVEIRA, Bruno. Processamento de Linguagem Natural do Português Brasileiro para detecção de cyberbullying. **Computer on the Beach**, v. 15, p. 277–282, mai. 2024. DOI: https://doi.org/10.14210/cotb.v15.p277-282. Disponível em: https://periodicos.univali.br/index.php/acotb/article/view/20369.

MAJ, Michał *et al.* Optimizing customer support using Text2SQL to query natural language databases. **Um.edu.mt**, University of Piraeus. International Strategic Management Association, v. 27, n. 3, 2024. DOI: https://www.um.edu.mt/library/oar/handle/123456789/128634. Disponível em: https://www.um.edu.mt/library/oar/handle/123456789/128634.

MALKOV, Yury A.; YASHUNIN, Dmitry A. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. **CoRR**, abs/1603.09320, 2016. arXiv: 1603.09320. Disponível em: http://arxiv.org/abs/1603.09320.

META. **Introducing Llama 3.1: Our most capable models to date**. [*S. l.: s. n.*], 2024. Disponível em: https://ai.meta.com/blog/meta-llama-3-1/.

MIELKE, Sabrina J. *et al.* **Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP**. [*S. l.: s. n.*], 2021. arXiv: 2112.10508 [cs.CL]. Disponível em: https://arxiv.org/abs/2112.10508.

MORAES SILVA, Ergon Cugler de. **Asymptotic behavior of the Manhattan distance in n-dimensions: Estimating multidimensional scenarios in empirical experiments**. [*S. l.: s. n.*], 2024. arXiv: 2406.15441 [math.GM]. Disponível em: https://arxiv.org/abs/2406.15441.

NAVEED, Humza *et al.* **A Comprehensive Overview of Large Language Models**. [*S. l.: s. n.*], 2024. arXiv: 2307.06435 [cs.CL]. Disponível em: https://arxiv.org/abs/2307.06435.

PAN, James Jie; WANG, Jianguo; LI, Guoliang. **Survey of Vector Database Management Systems**. [*S. l.: s. n.*], 2023. arXiv: 2310.14021 [cs.DB]. Disponível em: https://arxiv.org/abs/2310.14021.

POURREZA, Mohammadreza; RAFIEI, Davood. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. *In*: OH, A. *et al.* (Ed.). **Advances in Neural Information Processing Systems**. [*S. l.*]: Curran Associates, Inc., 2023. v. 36, p. 36339–36348. Disponível em: https://proceedings.neurips.cc/paper_files/paper/2023/file/72223cc66f63ca1aa59edaec1b3670e6-Paper-Conference.pdf.

QDRANT. **Qdrant - Vector Database**. [*S. l.: s. n.*]. Disponível em: https://qdrant.tech.

QIN, Bowen *et al.* **A Survey on Text-to-SQL Parsing: Concepts, Methods, and Future Directions**. [*S. l.: s. n.*], 2022. arXiv: 2208.13629 [cs.CL]. Disponível em: https://arxiv.org/abs/2208.13629.

RODRIGUES, Keli. Processamento de linguagem natural e a Ciência da Informação: inter-relações e contribuições. **Utfpr.edu.br**, Universidade Estadual de Londrina, 2023. DOI: http://repositorio.utfpr.edu.br/jspui/handle/1/32098. Disponível em: https://repositorio.utfpr.edu.br/jspui/handle/1/32098.

SHANAHAN, Murray. Talking about Large Language Models. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 67, n. 2, p. 68–79, jan. 2024. ISSN 0001-0782. DOI: 10.1145/3624724. Disponível em: https://doi.org/10.1145/3624724.

SHI, Liang *et al.* **A Survey on Employing Large Language Models for Text-to-SQL Tasks**. [*S. l.: s. n.*], 2024. arXiv: 2407.15186 [cs.CL]. Disponível em: https://arxiv.org/abs/2407.15186.

SONG, Yewei *et al.* **Enhancing Text-to-SQL Translation for Financial System Design**. [*S. l.: s. n.*], 2024. arXiv: 2312.14725 [cs.SE]. Disponível em: https://arxiv.org/abs/2312.14725.

VANNA.AI. **Vanna.AI: AI-powered SQL generation and data analysis**. [*S. l.: s. n.*], 2025. Acesso em: 9 jun. 2025. Disponível em: https://vanna.ai/.

VASWANI, Ashish *et al.* **Attention Is All You Need**. [*S. l.: s. n.*], 2023. arXiv: 1706.03762 [cs.CL]. Disponível em: https://arxiv.org/abs/1706.03762.

YANG, Jiaxi *et al.* Synthesizing Text-to-SQL Data from Weak and Strong LLMs. *In*: KU, Lun-Wei; MARTINS, Andre; SRIKUMAR, Vivek (Ed.). **Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**. Bangkok, Thailand: Association for Computational Linguistics, ago. 2024. P. 7864–7875. DOI: 10.18653/v1/2024.acl-long.425. Disponível em: https://aclanthology.org/2024.acl-long.425/.

ZHANG, Kun *et al.* ReFSQL: A Retrieval-Augmentation Framework for Text-to-SQL Generation. *In*: BOUAMOR, Houda; PINO, Juan; BALI, Kalika (Ed.). **Findings of the Association for Computational Linguistics: EMNLP 2023**. Singapore: Association for Computational Linguistics, dez. 2023. P. 664–673. DOI: 10.18653/v1/2023.findings-emnlp.48. Disponível em: https://aclanthology.org/2023.findings-emnlp.48/.