INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CAMPUS PIRIPIRI
TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

FABRÍCIO DE CARVALHO MOTA

BENCHMARK DE MODELOS DE LINGUAGEM DE CÓDIGO ABERTO PARA *TEXT-TO-SQL* EM DADOS ONCOLÓGICOS BRASILEIROS

PIRIPIRI/PI
2026

FABRÍCIO DE CARVALHO MOTA

BENCHMARK DE MODELOS DE LINGUAGEM DE CÓDIGO ABERTO PARA *TEXT-TO-SQL* EM DADOS ONCOLÓGICOS BRASILEIROS

Trabalho de Conclusão de Curso (artigo científico) apresentado como exigência parcial para obtenção do diploma do Curso de Tecnólogo em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Piripiri.

Orientador(a): Prof. Dr. Wanderson de Vasconcelos Rodrigues da Silva
Coorientador: Prof. Me. Jeferson Soares do Nascimento

PIRIPIRI/PI
2026

BENCHMARK DE MODELOS DE LINGUAGEM DE CÓDIGO ABERTO PARA *TEXT-TO-SQL* EM DADOS ONCOLÓGICOS BRASILEIROS

Fabrício de Carvalho Mota[1]
Prof. Dr. Wanderson de Vasconcelos Rodrigues da Silva[2]
Prof. Me. Jeferson Soares do Nascimento[3]

RESUMO

Este trabalho apresenta um *benchmark* experimental para avaliar modelos de linguagem de código aberto na conversão de perguntas em português brasileiro para consultas executáveis sobre dados públicos de atendimentos oncológicos. A fundamentação reúne estudos sobre geração de consultas a partir de linguagem natural, associação entre termos das perguntas e elementos do esquema relacional e avaliação baseada nos resultados obtidos pela execução. Foram comparados os modelos Qwen2.5-Coder 7B, DeepSeek-Coder 6.7B e Llama 3 8B, executados localmente por meio do Ollama, em quatro cenários definidos pela presença ou ausência do esquema do banco de dados e de variações linguísticas controladas nas perguntas. A avaliação considerou a acurácia por execução, as categorias de erro e as latências de geração e execução. O Qwen2.5-Coder 7B apresentou o maior desempenho geral, com acurá-cia de 80,0%, além da menor latência média de geração. O fornecimento do esquema foi o fator de maior impacto, elevando a acurácia média dos modelos de 53,9% para 93,3%. As variações linguísticas não reduziram o desempenho médio, sugerindo estabilidade diante das alterações superficiais de escrita adotadas. Conclui-se que mo-delos de linguagem de código aberto podem gerar consultas em cenários controlados, mas seu desempenho depende significativamente do contexto estrutural fornecido e da qualidade da formulação das perguntas.

Palavras-chave: text-to-SQL; modelos de linguagem; banco de dados; dados oncológicos; avaliação por execução.

ABSTRACT

This work presents an experimental *benchmark* to evaluate open-source language models in the conversion of questions written in Brazilian Portuguese into executable queries over public oncology care data. The theoretical foundation covers studies on query generation from natural language, the association between terms in questions and elements of relational schemas, and evaluation based on execution results. The Qwen2.5-Coder 7B, DeepSeek-Coder 6.7B, and Llama 3 8B models were compared locally through Ollama in four scenarios defined by the presence or absence of the database schema and controlled linguistic variations in the questions. The evaluation considered execution-based accuracy, error categories, and generation and execution latencies. Qwen2.5-Coder 7B achieved the highest overall performance, with an accuracy of 80.0%, as well as the lowest average generation latency. Providing the database schema was the most influential factor, increasing the average accuracy of

[1]Discente do curso de Análise e Desenvolvimento de Sistemas do IFPI – Campus Piripiri. E-mail: fabriciomota2612@gmail.com.
[2]Docente orientador do IFPI – Campus Piripiri. E-mail: wanderson.vasconcelos@ifpi.edu.br.
[3]Docente coorientador do IFPI – Campus Piripiri. E-mail: jeferson.soares@ifpi.edu.br.

the models from 53.9% to 93.3%. The linguistic variations did not reduce average performance, suggesting stability under the superficial writing changes adopted in the experiment. The findings indicate that open-source language models can generate queries in controlled scenarios, although their performance depends significantly on the structural context provided and on the quality of question formulation.

Keywords: text-to-SQL; language models; database; oncology data; execution-based evaluation.

Data de aprovação: 14/07/2026

1 INTRODUÇÃO

Bancos de dados relacionais são amplamente utilizados para armazenar dados estruturados e permitir consultas sobre essas informações. Entretanto, o acesso direto a esses dados costuma depender do conhecimento do esquema, das tabelas, dos atributos e da linguagem *Structured Query Language* (SQL), o que pode dificultar seu uso por pessoas que não dominam linguagens de consulta ou a estrutura interna do banco. Nesse contexto, a tarefa *Text-to-SQL* consiste em converter perguntas formuladas em linguagem natural em consultas SQL, permitindo a interação com bancos de dados relacionais sem a escrita manual desses comandos (Yu *et al.*, 2018; Shi *et al.*, 2025).

Essa abordagem tem sido investigada como forma de reduzir barreiras de acesso a dados estruturados, mas ainda apresenta desafios relevantes. A consulta gerada pode selecionar tabelas ou colunas incorretas, interpretar inadequadamente a pergunta ou retornar resultados diferentes do esperado (Li *et al.*, 2023; Shi *et al.*, 2025). Com a adoção dos Modelos de Linguagem de Grande Porte (*Large Language Models* – LLMs), ampliaram-se as possibilidades de geração de consultas a partir de instruções textuais fornecidas ao modelo. Ainda assim, persistem desafios clássicos da área, como o alinhamento entre os termos da pergunta e os elementos do banco, conhecido como *schema linking*, a seleção correta de tabelas e colunas e a geração de consultas semanticamente compatíveis com a intenção do usuário (Shi *et al.*, 2025).

Na literatura de avaliação, destacam-se *benchmarks* como o *Spider*, um conjunto de dados em larga escala com anotações humanas para consultas complexas e *crossdomain*, e o *BIRD*, voltado à avaliação de modelos em bases mais próximas de cenários reais de uso (Yu *et al.*, 2018; Li *et al.*, 2023). Nesse sentido, Yu *et al.* (2018) descrevem o *Spider* como um *benchmark* desafiador, com múltiplos domínios,

esquemas inéditos na avaliação e consultas SQL de maior complexidade estrutural. Apesar da relevância desses *benchmarks*, ainda permanece uma distância entre o desempenho observado em ambientes acadêmicos controlados e as exigências de aplicação em bases reais. Nessa direção, Li *et al.* (2023) argumentam que os *benchmarks* predominantes ainda não capturam plenamente características do uso prático, como maior heterogeneidade de esquemas, consultas mais realistas e maior diversidade de formulações linguísticas. Além disso, grande parte desses conjuntos de avaliação foi originalmente desenvolvida em inglês, o que limita a análise do desempenho em contextos multilíngues (Dou *et al.*, 2023; Pham *et al.*, 2026). No caso do português, estudos anteriores já analisaram adaptações da tarefa *Text-to-SQL* para esse idioma (José; Cozman, 2021), e trabalhos com dados públicos brasileiros indicam a pertinência de investigar consultas em linguagem natural nesse contexto (Fróes; Braghetto, 2025).

Diante disso, este trabalho avalia modelos de linguagem de código aberto na tarefa *Text-to-SQL*, utilizando perguntas em português brasileiro e dados públicos do domínio oncológico. Esse domínio é tratado apenas como base tabular para avaliação de consultas SQL, sem envolver validação clínica de diagnósticos, tratamentos ou decisões assistenciais. Foram analisados os modelos Qwen2.5-Coder 7B, DeepSeek-Coder 6.7B e Llama 3 8B, executados localmente via Ollama, em quatro cenários experimentais definidos pela presença ou ausência do esquema do banco no *prompt* e pela presença ou ausência de variações linguísticas superficiais controladas nas perguntas (Wretblad *et al.*, 2024).

A principal contribuição do estudo é a construção e análise de um *benchmark* experimental em português brasileiro, voltado à geração de consultas SQL sobre dados tabulares do domínio oncológico. Além do desempenho por *Execution Accuracy* (EX), são analisados o impacto do fornecimento do esquema, o comportamento dos modelos diante de variações linguísticas controladas, as categorias de erro e as latên-cias de geração e execução.

O restante do artigo está organizado da seguinte forma: a Seção 2 apresenta o referencial teórico; a Seção 3 descreve a metodologia; a Seção 4 apresenta os resultados e a discussão; e a Seção 5 reúne as conclusões.

2 REFERENCIAL TEÓRICO

Este referencial teórico apresenta os conceitos e estudos que fundamentam a tarefa *Text-to-SQL* e sua avaliação. São abordados: (i) a formulação do problema como mapeamento de linguagem natural para SQL, (ii) o uso de modelos de linguagem, engenharia de *prompt* e contexto do esquema, (iii) *benchmarks* utilizados na avaliação da tarefa, com destaque para Spider e BIRD, e (iv) aspectos relacionados à avaliação por execução, ao idioma das perguntas e às variações linguísticas controladas. Esses tópicos sustentam as decisões metodológicas adotadas no experimento e orientam a análise dos resultados.

2.1 TEXT-TO-SQL COMO MAPEAMENTO DE LINGUAGEM NATURAL PARA SQL

A tarefa *Text-to-SQL*, também denominada *Natural Language to SQL*, consiste em converter uma pergunta formulada em linguagem natural em uma consulta SQL executável sobre um banco de dados (Yu *et al.*, 2018). No escopo deste estudo, essa execução ocorre sobre um banco relacional em SQLite. Por exemplo, uma pergunta como “Quantos registros há por sexo?” pode ser transformada em uma consulta que agrupa os registros pelo atributo sexo e calcula a quantidade de ocorrências em cada grupo.

Formalmente, essa tarefa pode ser compreendida como um problema de *semantic parsing*, isto é, o mapeamento de sentenças em linguagem natural para representações formais executáveis. Nesse sentido, Yu *et al.* (2018, p. 3911, tradução nossa) descrevem a análise semântica como o “mapeamento dessas sentenças para consultas executáveis com significado”. No caso de *Text-to-SQL*, a representação executável é uma consulta SQL que deve refletir a intenção da pergunta e respeitar a estrutura do banco, incluindo tabelas, colunas, filtros, agregações e relações entre entidades.

A dificuldade da tarefa aumenta conforme a estrutura necessária para responder à pergunta. Consultas simples podem envolver apenas seleção, contagem ou filtragem direta; já consultas mais elaboradas podem exigir agrupamentos, agregações, ordenações, junções entre tabelas e subconsultas. Esses elementos tornam a geração mais complexa porque exigem identificar quais atributos respondem à pergunta, como eles se relacionam no banco e quais operações SQL devem ser combinadas para produzir o resultado esperado (Yu *et al.*, 2018).

Outro aspecto relevante é que consultas SQL diferentes podem ser semanticamente equivalentes, produzindo o mesmo resultado quando executadas. Por isso, avaliações baseadas apenas na comparação textual entre consulta gerada e consulta de referência podem ser insuficientes. Essa limitação fundamenta o uso de métricas baseadas em execução, discutidas na subseção 2.4.

2.2 MODELOS DE LINGUAGEM, ENGENHARIA DE PROMPT E CONTEXTO DO ESQUEMA

Em tarefas *Text-to-SQL*, LLMs têm sido utilizados para gerar consultas SQL a partir de instruções em linguagem natural (Shi *et al.*, 2025). Nessa aplicação, a qualidade da resposta não depende apenas da capacidade linguística do modelo, mas também da forma como a tarefa é apresentada e das informações fornecidas sobre a estrutura do banco de dados.

A engenharia de *prompt* corresponde à elaboração da entrada textual forne-cida ao modelo, com o objetivo de orientar a tarefa, delimitar restrições e indicar o formato esperado da resposta. Em *Text-to-SQL*, o *prompt* pode incluir a pergunta do usuário, regras de geração e informações adicionais sobre o banco, buscando reduzir ambiguidades e aproximar a consulta gerada da intenção expressa na pergunta.

Um desafio central nesse processo é o *schema linking*, que consiste em as-sociar corretamente os termos da pergunta aos elementos do esquema do banco de dados, como tabelas, colunas e relações (Shi *et al.*, 2025; Li *et al.*, 2023). Por exemplo, uma pergunta pode mencionar “município” ou “ano de diagnóstico”, enquanto essas informações podem estar representadas no banco por nomes técnicos, códigos ou tabelas auxiliares. Quando esse alinhamento falha, a consulta gerada pode utilizar atributos inexistentes, selecionar colunas inadequadas ou retornar resultados semanticamente incorretos.

Por esse motivo, muitos métodos incluem o contexto do esquema no próprio *prompt*, isto é, uma descrição resumida das tabelas, colunas e relações disponíveis no banco (Shi *et al.*, 2025). Essa estratégia fornece ao modelo uma referência ex-plícita sobre quais elementos podem ser utilizados na consulta, sendo especialmente relevante em bancos com múltiplas tabelas, nomes técnicos de atributos e consultas que exigem filtros, agregações ou junções.

Essa discussão fundamenta a comparação adotada no experimento: em um cenário, os modelos recebem apenas a pergunta e instruções gerais de geração; em

outro, recebem também a descrição do esquema do banco. Com isso, torna-se possível avaliar em que medida o fornecimento do contexto estrutural influencia a qualidade das consultas SQL geradas.

*2.*3 BENCHMARKS E DESAFIOS EM TEXT-TO-SQL: SPIDER E BIRD

A avaliação comparativa de métodos *Text-to-SQL* depende de *benchmarks*, isto é, conjuntos padronizados de teste que reúnem bancos de dados, esquemas, perguntas em linguagem natural e consultas SQL de referência (Shi *et al.*, 2025). Esses conjuntos favorecem a reprodutibilidade porque permitem que diferentes métodos se-jam avaliados sob as mesmas condições, executando consultas sobre os mesmos bancos e comparando os resultados obtidos com referências previamente definidas.

O Spider foi proposto como um *benchmark* para avaliar *Text-to-SQL* em múltiplos domínios e esquemas de banco de dados. Portanto, não se trata de um único banco, mas de um conjunto de avaliação composto por diferentes bases, perguntas e consultas SQL de referência. Sua proposta enfatiza a generalização, isto é, a capacidade de lidar com consultas e esquemas não vistos durante o treinamento (Yu *et al.*, 2018).

Apesar da importância de conjuntos como o Spider, trabalhos posteriores apontam que bons resultados em *benchmarks* tradicionais nem sempre se traduzem diretamente em desempenho adequado em cenários aplicados. Em bases reais, os esquemas podem ser maiores, os valores armazenados podem influenciar a interpretação da pergunta e o domínio pode exigir conhecimento contextual adicional (Li *et al.*, 2023; Shi *et al.*, 2025).

Nesse sentido, o BIRD (*BIg Bench for LaRge-Scale Database Grounded Text-to-SQLs*) foi proposto para aproximar a avaliação acadêmica de condições mais próxi-mas de aplicações reais. Ao discutir essa lacuna, Li *et al.* (2023, p. 1, tradução nossa) observam que *benchmarks* predominantes “deixam uma lacuna entre o estudo acadêmico e as aplicações do mundo real”. O BIRD enfatiza bancos de dados em maior escala, conteúdo mais rico, presença de valores ruidosos e necessidade de conhecimento externo para interpretar perguntas e gerar consultas corretas (Li *et al.*, 2023).

A partir dessas referências, a construção de avaliações em *Text-to-SQL* requer a definição de perguntas, consultas SQL de referência, bancos de dados executáveis e critérios de comparação. Esses princípios orientam o protocolo adotado neste traba-lho, em escala menor e em domínio específico, com perguntas em português brasileiro e banco derivado de dados públicos oncológicos, permitindo comparar os modelos sob condições

padronizadas.

2.4 AVALIAÇÃO POR EXECUÇÃO E LIMITAÇÕES DA EX

A definição de métricas é um aspecto central em *Text-to-SQL*, pois avaliações baseadas apenas na forma textual da consulta podem penalizar consultas semanticamente equivalentes, mas sintaticamente distintas (Shi *et al.*, 2025; Kim *et al.*, 2025). Por esse motivo, a literatura adota métricas baseadas em execução, nas quais o resul-tado retornado pela consulta gerada é comparado ao resultado obtido pela consulta SQL de referência.

A *Execution Accuracy* (EX) mede a proporção de consultas geradas que retornam exatamente o resultado esperado após a execução no banco de dados. Embora seja recorrente em avaliações de *Text-to-SQL*, essa métrica apresenta limitações. Kim *et al.* (2025) apontam que a EX, embora amplamente utilizada, pode produzir falsos positivos e falsos negativos na avaliação de consultas *Text-to-SQL*. Assim, uma con-sulta incorreta pode retornar o resultado esperado por coincidência, enquanto uma consulta semanticamente adequada pode ser penalizada quando há ambiguidades na pergunta, ausência de critérios explícitos de ordenação ou problemas na anotação das referências.

Essas limitações indicam que a EX deve ser interpretada com cautela, mas não inviabilizam seu uso em avaliações controladas, desde que perguntas, consultas SQL de referência e resultados esperados sejam previamente definidos. No protocolo experimental adotado, a comparação entre os modelos é operacionalizada por meio da EX, calculada pela execução das consultas geradas e pela comparação dos resultados retornados com os resultados de referência.

Na literatura recente, propostas como o *FLEX* (*False-Less EXecution*) buscam reduzir limitações da avaliação por execução tradicional, aproximando a avaliação automática do julgamento de especialistas humanos por meio de modelos de linguagem e critérios semânticos mais abrangentes (Kim *et al.*, 2025). Neste trabalho, entretanto, o FLEX é tratado apenas como referência crítica sobre as limitações da EX, sem com-por a etapa avaliativa do experimento

2.5 IDIOMA E VARIAÇÕES LINGUÍSTICAS EM *TEXT-TO-SQL*

Parte relevante dos recursos de avaliação em *Text-to-SQL* foi originalmente proposta em inglês, o que motiva investigações sobre o desempenho da tarefa em outros idiomas. Nesse contexto, MultiSpider e MultiSpider 2.0 são propostas de avaliação multilíngue que expandem conjuntos de teste para diferentes línguas, permitindo examinar em que medida métodos avaliados em inglês mantêm desempenho em outros contextos linguísticos (Dou *et al.*, 2023; Pham *et al.*, 2026). Essa literatura não é ado-tada aqui para comparar idiomas, mas para situar a escolha metodológica de elaborar o conjunto avaliativo em português brasileiro.

No caso do português, estudos anteriores já investigaram adaptações da tarefa *Text-to-SQL* para esse idioma, incluindo a tradução de conjuntos de dados e a adaptação de modelos originalmente avaliados em inglês (José; Cozman, 2021). Neste estudo, porém, o português brasileiro não é tratado como variável de comparação multilíngue, mas como idioma das perguntas utilizadas no experimento. Essa delimita-ção permite concentrar a análise no comportamento dos modelos diante de consultas formuladas em português brasileiro sobre o domínio considerado.

Além do idioma, a forma de escrita das perguntas também pode influenciar a geração da consulta. Em situações de uso, entradas em linguagem natural podem conter erros de digitação, abreviações, informalidades e reformulações superficiais, afetando a interpretação da pergunta e o alinhamento entre seus termos e os ele-mentos do esquema do banco (Wretblad *et al.*, 2024). Por esse motivo, as variações linguísticas analisadas no experimento foram produzidas de forma controlada, preservando a intenção semântica original das perguntas. Assim, torna-se possível observar o comportamento dos modelos diante de alterações superficiais na escrita, sem intro-duzir novas exigências de consulta ou ambiguidades adicionais.

3 METODOLOGIA

Esta seção descreve o delineamento metodológico adotado neste estudo, apresentando de forma sistemática como o experimento foi concebido, implementado e avaliado. O objetivo é explicitar cada etapa do processo, desde a preparação dos dados até a definição das métricas finais, de modo a garantir transparência e reprodutibilidade.

3.1 FONTE DE DADOS E CONSTRUÇÃO DO BANCO RELACIONAL

A base utilizada no experimento foi o conjunto público *Oncology Treatment DataSUS – Brazil (2013–2023)*, derivado de registros de atendimentos oncológicos no Brasil e extraído da plataforma Kaggle (LHUCATENORIO, 2025). O conjunto foi adotado como base tabular para avaliação da tarefa *Text-to-SQL*; portanto, o domínio oncológico é tratado apenas como contexto temático e estrutural das consultas, sem finalidade de validação clínica de diagnósticos, tratamentos ou decisões assistenciais.

Para viabilizar a execução das consultas SQL geradas pelos modelos, o arquivo original em formato CSV foi transformado em um banco de dados relacional implementado em SQLite. A escolha do SQLite justifica-se por seu caráter local, autocontido e sem necessidade de servidor, o que favorece a portabilidade e a reprodutibilidade do experimento (SQLite Consortium, 2025). Além disso, sua integração direta ao *pipe-line* em Python facilitou a geração, validação e execução automatizada das consultas (Python Software Foundation, 2026).

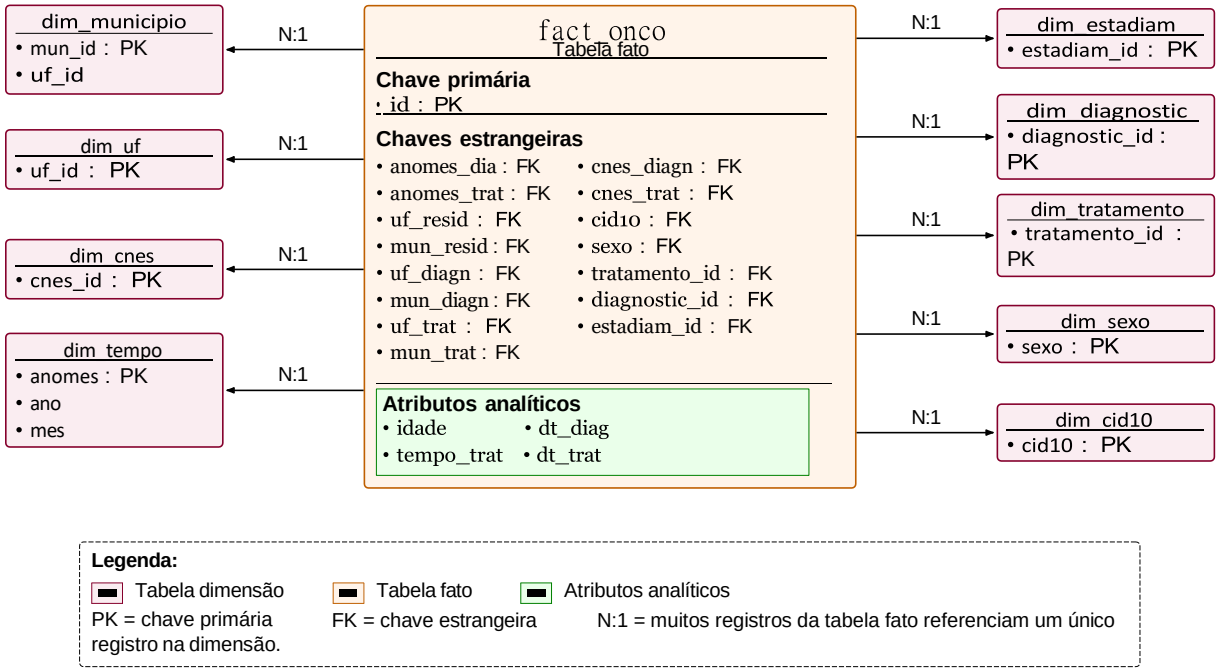
A construção do banco seguiu um fluxo de ETL, do inglês *Extract, Transform, Load*, organizado em extração, transformação e carga. Na extração, o arquivo CSV foi lido e carregado inicialmente na tabela raw_onco, que funcionou como camada intermediária de carga. Essa tabela preservou a estrutura tabular original após a leitura do arquivo, permitindo inspecionar os dados de entrada, rastrear as transformações realizadas e separar a etapa de carregamento inicial da modelagem relacional final. Na transformação, os campos da raw_onco foram reorganizados em uma estrutura dimensional simplificada, com padronização de nomes, tipos de dados e chaves de relacionamento. Na carga, as tabelas finais foram inseridas no banco SQLite utilizado nas etapas posteriores de geração, validação e execução das consultas.

O banco BR-onco foi organizado em um esquema dimensional do tipo estrela, com a tabela fato fact_onco no centro e dimensões auxiliares. Essa modelagem foi adotada por três razões principais: (i) preservar a rastreabilidade dos registros após o ETL; (ii) permitir consultas analíticas típicas de *Text-to-SQL*, sobretudo agregações, filtros e junções com dimensões; e (iii) manter um nível de complexidade controlado, adequado para isolar o efeito das variações de *prompt*, do uso de esquema e das diferenças entre modelos. Por isso, o esquema apresenta predominantemente relações muitos-para-um entre a tabela fato e as dimensões, sendo mais simples que bases transacionais

extensas, mas suficiente para o objetivo experimental do estudo.

A modelagem resultante é sintetizada na Figura 1, que apresenta a organização dimensional simplificada do banco. Essa estrutura é composta por uma tabela fato, (fact_onco), e nove tabelas dimensão. A tabela fato concentra os registros centrais de atendimento e atributos analíticos, como idade, tempo até tratamento, data do diagnóstico e data do tratamento, enquanto as dimensões armazenam atributos descritivos relacionados a sexo, diagnóstico, tratamento, estadiamento, unidade federativa, município, estabelecimento e tempo. Essa separação permite representar os dados em formato relacional e avaliar se os modelos conseguem associar correta-mente perguntas, tabelas, colunas e relações.

Figura 1 – Modelo dimensional simplificado do banco relacional utilizado.



Fonte: Elaboração própria (2026).

Na Figura 1, a tabela fact_onco aparece como elemento central do esquema, pois armazena os registros analisados e as chaves estrangeiras que a conectam às dimensões. As setas indicam relações do tipo N:1, nas quais muitos registros da tabela fato podem referenciar um único registro em uma tabela dimensão. As relações com dim_tempo ocorrem por meio dos campos anomes_dia e anomes_trat, ambos associados à chave anomes; com dim_uf, por meio de uf_resid, uf_diagn e uf_trat, associados à chave uf_id; com dim_municipio, por meio de mun_resid, mun_diagn e mun_trat, associados à chave mun_id; e com dim_cnes, por meio de cnes_diagn e cnes_trat, associados à chave cnes_id. As demais dimensões são associadas

diretamente pelos campos cid10, sexo, tratamento_id, diagnostic_id e estadiam_id. Essa organização favorece consultas com filtros, agrupamentos, agregações e junções, características relevantes para a avaliação de modelos *Text-to-SQL*.

A Tabela 1 apresenta a caracterização quantitativa do banco SQLite utilizado no experimento, incluindo o número de colunas, registros, chaves primárias e relações declaradas por tabela.

Tabela 1 – Caracterização quantitativa do banco SQLite.

Tabela	Papel	Colunas	Registros	Chave primária	Relações
raw_onco	Carga	21	995.374	–	0
fact_onco	Fato	20	995.374	id	15
dim_cid10	Dimensão	1	109	cid10	0
dim_cnes	Dimensão	1	2.013	cnes_id	0
dim_diagnostic	Dimensão	1	4	diagnostic_id	0
dim_estadiam	Dimensão	1	7	estadiam_id	0
dim_municipio	Dimensão	2	5.568	mun_id	0
dim_sexo	Dimensão	1	2	sexo	0
dim_tempo	Dimensão	3	150	anomes	0
dim_tratamento	Dimensão	1	5	tratamento_id	0
dim_uf	Dimensão	1	28	uf_id	0

Fonte: Elaboração própria (2026).

No total, o banco contém 11 tabelas de usuário: uma tabela de carga, uma tabela fato e nove tabelas dimensão. A estrutura final possui 53 colunas, 10 chaves primárias, 15 chaves estrangeiras declaradas e 6 índices explícitos. A tabela fact_onco contém 995.374 registros analíticos, correspondentes aos registros utilizados nas consultas do experimento.

Após a carga, foram realizadas verificações de consistência, incluindo conferência da estrutura das tabelas, validação das relações declaradas e execução de consultas simples para confirmação dos dados carregados. Essa etapa teve como ob-jetivo garantir que o banco estivesse organizado de forma compatível com o protocolo experimental e com a avaliação por execução adotada.

3.2 CONSTRUÇÃO DO CONJUNTO DE PERGUNTAS, CONSULTAS DE REFERÊNCIA E VARIAÇÕES LINGUÍSTICAS

. A avaliação experimental foi estruturada a partir de um conjunto de 30 perguntas canônicas em português brasileiro, elaboradas com base no esquema relacional descrito na subseção 3.1. Cada pergunta foi associada a uma consulta SQL de re-ferência, utilizada para produzir o resultado esperado no banco SQLite. Esse vínculo entre

pergunta, consulta de referência e resultado esperado fundamenta a avaliação por execução adotada no experimento.

As perguntas foram formuladas para contemplar operações comuns em consultas analíticas, incluindo subconsulta, agregações, filtros, agrupamentos, ordenações, limites, junções entre a tabela fato e as dimensões. As consultas SQL de referência foram construídas de forma determinística, com cláusulas de ordenação quando necessário e tratamento explícito de valores nulos em casos que poderiam comprometer a comparação por execução. Antes da avaliação dos modelos, todas as consultas de referência foram executadas no banco para verificar a ausência de erros e a coerência dos resultados retornados.

Para caracterizar o conjunto avaliativo, as perguntas canônicas foram classificadas segundo o tipo predominante de consulta, o nível de dificuldade e as operações SQL presentes nas consultas de referência. A Tabela 2 sintetiza essa caracterização. A classificação por operações não é mutuamente exclusiva, pois uma mesma consulta pode combinar agregação, filtro, ordenação, junção, limite e subconsulta.

Tabela 2 – Caracterização das perguntas canônicas e consultas de referência.

Critério	Categoria/operação	n (%)
Tipo predominante	Agregação	21 (70,0)
	Agregação com junção	8 (26,7)
	Subconsulta	1 (3,3)
Dificuldade	Simples	1 (3,3)
	Intermediária	15 (50,0)
	Complexa	14 (46,7)
Operações presentes	Agregações	30 (100,0)
	Filtros	25 (83,3)
	Ordenações	23 (76,7)
	Junções	8 (26,7)
	Limites	8 (26,7)
	Subconsultas	1 (3,3)

Fonte: Elaboração própria (2026).

Como indicado na Tabela 2, o conjunto avaliativo prioriza consultas analíticas de agregação, mas inclui diferentes combinações de filtros, ordenações, junções, limites e subconsulta. A concentração nos níveis intermediário e complexo indica que o conjunto, embora reduzido em quantidade de perguntas, não se limita a consultas triviais.

Em relação à dificuldade, o conjunto é composto por 1 pergunta simples (3,3%), 15 perguntas intermediárias (50,0%) e 14 perguntas complexas (46,7%), totalizando as 30

perguntas canônicas utilizadas em cada cenário experimental.

As perguntas do *benchmark* não foram balanceadas artificialmente entre as categorias *Simples*, *Intermediária* e *Complexa*. Em vez disso, buscou-se refletir o perfil das consultas analíticas consideradas mais relevantes para o domínio estudado, com predominância de agregações, filtros, ordenações e, em menor número, junções e subconsultas. A dificuldade foi definida a partir de características estruturais da SQL de referência, considerando a presença de junções, agregações, filtros, orde-nações, limites e subconsultas. Consultas com subconsulta ou maior combinação desses elementos foram classificadas como *Complexas*; consultas com combinação intermediária desses operadores foram classificadas como *Intermediárias*; e consultas mais diretas foram classificadas como *Simples*. As versões com variações linguísticas controladas herdaram a mesma classe de dificuldade das perguntas canônicas correspondentes, pois preservam a mesma intenção semântica e a mesma consulta SQL de referência.

A partir das perguntas canônicas, foram produzidas versões com variações linguísticas superficiais controladas. Neste trabalho, optou-se por empregar a expressão *variações linguísticas controladas*, em vez de apenas *ruído*, para descrever reformulações que incluem abreviações, simplificações sintáticas, formas coloquiais e erros ortográficos leves. Essa escolha é metodologicamente mais precisa, pois tais alterações não constituem corrupção aleatória da entrada, mas perturbações linguísticas deliberadas para testar a robustez dos modelos diante de formulações alternativas semanticamente equivalentes.

Essas variações preservaram a intenção semântica da pergunta original e mantiveram a mesma consulta SQL de referência, permitindo comparar o desempenho dos modelos diante de alterações de escrita sem modificar a resposta esperada. O objetivo não foi introduzir ambiguidade semântica, omissão de informações essenciais ou ruído adversarial, mas observar o comportamento dos modelos diante de formas simples de variação linguística.

As variações foram organizadas em cinco categorias: abreviação lexical, simplificação sintática, erro ortográfico controlado, reformulação coloquial e substituição lexical superficial. A abreviação lexical inclui reduções como “qtd”, “diag” e “trat”; a a simplificação sintática remove termos funcionais sem alterar a intenção da pergunta; o erro ortográfico controlado introduz pequenas alterações gráficas, como “pacintes” e “tratamnto”; a reformulação coloquial aproxima a pergunta da linguagem cotidiana; e a

substituição lexical superficial troca termos por equivalentes contextuais, como “Top 10” em lugar de “quais são os 10”.

Como exemplo, a pergunta canônica “Quantos registros há por sexo?” foi associada a uma consulta de referência que agrupa os registros pelo atributo sexo, calcula a quantidade de ocorrências e ordena o resultado pela contagem. Uma variação linguística controlada dessa pergunta poderia ser “qtd de registros por sexo”, preservando a mesma intenção semântica e, portanto, a mesma consulta de referência. Esse exemplo ilustra a lógica adotada na construção do conjunto avaliativo: alterar superfi-cialmente a forma da pergunta sem modificar o resultado esperado da consulta.

3.3 DELINEAMENTO EXPERIMENTAL E *PIPELINE*

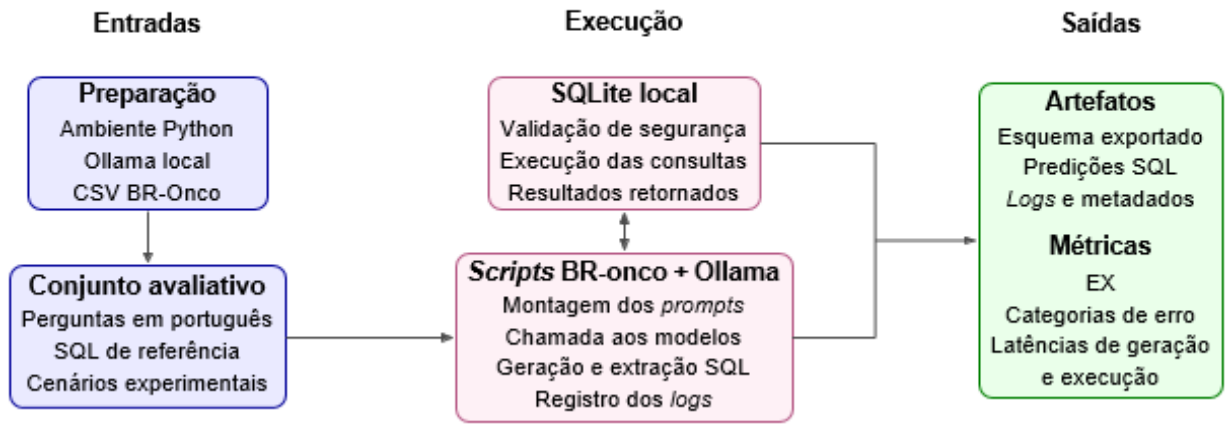
A metodologia foi definida a partir da finalidade, da abordagem e do procedi-mento adotado. Quanto à finalidade, a pesquisa é aplicada, pois avalia modelos de linguagem em uma tarefa computacional específica: a conversão de perguntas em português brasileiro para consultas SQL executáveis. Quanto à abordagem, é quantitativa, uma vez que a comparação entre os modelos ocorre por meio de medidas objetivas de desempenho, como *Execution Accuracy* (EX), categorias de erro e latências de geração e execução (Marconi; Lakatos, 2003).

Quanto ao procedimento, o estudo adota um delineamento experimental computacional. Essa escolha decorre da necessidade de comparar diferentes configurações sob condições controladas, mantendo fixos os elementos que não devem variar durante as execuções, como o banco SQLite, o conjunto avaliativo, as consultas SQL de referência, os parâmetros de geração e o protocolo de avaliação (Wohlin *et al.*, 2012). Com isso, busca-se reduzir interferências externas e tornar comparáveis os resultados obtidos em cada cenário.

O experimento compara três modelos executados localmente via Ollama. Os modelos avaliados são Qwen2.5-Coder 7B, DeepSeek-Coder 6.7B e Llama 3 8B. Além do modelo avaliado, foram considerados dois fatores experimentais: a presença ou ausência do esquema do banco no *prompt* e a presença ou ausência de variações linguísticas controladas nas perguntas. A combinação desses fatores originou quatro cenários: CSE, com perguntas canônicas sem esquema; CCE, com perguntas canônicas e esquema; VSE, com perguntas com variação linguística sem esquema; e VCE, com perguntas com variação linguística e esquema.

A Figura 2 apresenta a visão geral do *pipeline* experimental. Na etapa de entradas, o bloco Preparação reúne os recursos utilizados na execução, incluindo o ambiente Python, o Ollama local e o CSV BR-Onco utilizado na construção do banco. O bloco Conjunto avaliativo reúne as perguntas em português brasileiro, as consultas SQL de referência e os cenários experimentais.

Figura 2 – Visão geral do *pipeline* experimental de geração, execução e avaliação das consultas SQL.



Fonte: Elaboração própria (2026).

Na etapa de execução, os *Scripts* BR-onco + Ollama coordenam o fluxo experimental: montam os *prompts*, enviam as perguntas aos modelos, recebem as respostas, extraem a consulta SQL gerada e registram os *logs*. O bloco SQLite local representa a validação de segurança, a execução das consultas aceitas e o retorno dos resultados obtidos no banco, separando a geração da consulta de sua validação e execução.

Como saída, o *pipeline* produz artefatos e métricas. Os artefatos incluem o esquema exportado, as predições SQL, os *logs* e os metadados das execuções; as métricas incluem EX, categorias de erro e latências de geração e execução. Para favorecer a rastreabilidade e a reprodução do experimento, a estrutura do *pipeline*, incluindo *scripts*, organização dos diretórios, consultas SQL de referência e documentação de execução, foi disponibilizada em repositório público. O material é indicado no Apêndice A. Esses registros permitem rastrear o comportamento de cada modelo em cada cenário e sustentam a análise comparativa apresentada na seção de resultados.

3.4 AMBIENTE COMPUTACIONAL E CONFIGURAÇÃO EXPERIMENTAL

O experimento foi conduzido em ambiente computacional local, com etapas automatizadas por *scripts* em Python. A escolha da linguagem Python deve-se à sua adequação à automação de fluxos experimentais, especialmente em tarefas que envolvem leitura de arquivos, manipulação de dados tabulares, integração com bancos SQLite, chamadas a modelos locais e geração de relatórios (Python Software Foundation, 2026). No contexto deste trabalho, a linguagem permitiu organizar em um mesmo *pipeline* as etapas de preparação das entradas, interação com os modelos, validação das consultas, execução no banco e consolidação das métricas. A execução dos modelos ocorreu por meio do Ollama, versão 0.13.5, ferramenta utilizada para disponibilizar modelos de linguagem em ambiente local (OLLAMA, 2025).

O ambiente físico utilizado foi um *notebook* Samsung Galaxy Book4, com Windows 11 Home Single Language em sistema de 64 bits, processador 13th Gen Intel(R) Core(TM) i7-1355U, 16 GB de memória RAM, SSD de 512 GB e GPU integrada Intel(R) Iris(R) Xe Graphics. Embora o sistema operacional hospedeiro tenha sido o Windows, o *pipeline* foi executado no Windows Subsystem for Linux (WSL), com distribuição Ubuntu, onde foram configurados os *scripts*, o banco SQLite e a execução dos modelos via Ollama.

Foram avaliados três modelos de linguagem com perfis distintos: Qwen2.5 Coder 7B, DeepSeek-Coder 6.7B e Llama 3 8B. A seleção buscou contemplar modelos voltados à geração de código e um modelo geral de instrução, todos disponíveis para execução local via Ollama. O Qwen2.5-Coder 7B e o DeepSeek-Coder 6.7B foram incluídos por seu foco em tarefas de programação e geração de código (Hui *et al.*, 2024; Guo *et al.*, 2024). O Llama 3 8B foi adotado como modelo geral de instrução, permitindo contrastar seu desempenho com o de modelos mais especializados na tarefa *Text-to-SQL* (Llama Team, 2024).

Tabela 3 – Modelos avaliados no ambiente experimental.

Modelo	ID no Ollama	Parâm.	Janela de contex	Quant.
ven2.5-Coder 7B	qwen2.5-coder:7b	7,6B	32.768 *tokens*	Q4_K_M
pSeek-Coder 6.7B	deepseek-coder:6	7B	16.384 *tokens*	Q4_0
Llama 3 8B	llama3:latest	8B	8.192 *tokens*	Q4_0

Fonte: Elaboração própria (2026).

Nota: Parâm. = quantidade aproximada de parâmetros; Quant. = formato de quantização.

Na Tabela 3, o ID no Ollama corresponde ao identificador utilizado para executar cada modelo. A janela de contexto representa o limite de *tokens* considerados na entrada, enquanto a quantização corresponde ao formato de compressão numérica empregado para viabilizar a execução local dos modelos, reduzindo principalmente o consumo de memória.

As execuções foram realizadas com parâmetros padronizados para todos os modelos. O parâmetro temperature foi fixado em 0,0, reduzindo a aleatoriedade e favorecendo respostas mais determinísticas. O parâmetro num_predict=512 definiu o limite máximo de *tokens* gerados por resposta. Já top_p e top_k, que controlam a amostragem dos *tokens* candidatos durante a geração, não foram configurados explicitamente e permaneceram com os valores padrão do ambiente de execução. Da mesma forma, seed, parâmetro usado para controlar a reprodutibilidade em gerações amostradas, não foi especificado manualmente.

O parâmetro request_timeout=300 estabeleceu o tempo limite, em segundos, para cada chamada ao modelo, enquanto max_retries=0 indicou a ausência de novas tentativas automáticas após falhas ou excedimento de tempo. No banco SQLite, a execução ocorreu em modo somente leitura, com timeout=1.

Essas definições mantêm constantes as condições de inferência e execução. Assim, as diferenças observadas entre os resultados decorrem dos próprios modelos e dos cenários experimentais avaliados, e não de alterações nos parâmetros de geração ou no ambiente computacional.

3.5 GERAÇÃO DAS CONSULTAS PELOS MODELOS

A geração das consultas SQL foi realizada a partir das perguntas em português brasileiro definidas no conjunto avaliativo. Cada pergunta foi enviada aos modelos por

meio de um *prompt* construído automaticamente pelos *scripts* do experimento, produzindo uma consulta SQL candidata para cada combinação entre modelo e cenário.

Foram utilizados dois formatos de *prompt*. Nos cenários sem esquema, o modelo recebeu a pergunta e instruções gerais para retornar uma consulta SQL compatí-vel com SQLite. Nos cenários com esquema, o *prompt* incluiu também uma descrição textual do banco, contendo tabelas, colunas e relações disponíveis. Essa diferença permitiu avaliar se o fornecimento explícito da estrutura do banco favorece a geração de consultas mais aderentes ao esquema relacional.

O Quadro 1 ilustra, de forma reduzida, como os *prompts* foram estruturados nos quatro cenários experimentais. O exemplo não reproduz integralmente os *prompts* utilizados, mas apresenta a lógica de interação entre instrução, esquema do banco e pergunta enviada ao modelo.

Quadro 1 – Exemplo reduzido de *prompt* por cenário experimental.

Cenário	**Exemplo reduzido de *prompt* enviado ao modelo**
CSE	Gere apenas uma consulta SQL compatível com SQLite. Use somente SELECT. Não explique a resposta. Pergunta: “Quantos registros há por sexo?”
CCE	Gere apenas uma consulta SQL compatível com SQLite. Use somente SELECT. Não explique a resposta. Esquema: fact_onco(id, sexo, …); dim_sexo(sexo). Relação: fact_onco.sexo → dim_sexo.sexo. Pergunta: “Quantos registros há por sexo?”
VSE	Gere apenas uma consulta SQL compatível com SQLite. Use somente SELECT. Não explique a resposta. Pergunta: “qtd de registros por sexo”
VCE	Gere apenas uma consulta SQL compatível com SQLite. Use somente SELECT. Não explique a resposta. Esquema: fact_onco(id, sexo, …); dim_sexo(sexo). Relação: fact_onco.sexo → dim_sexo.sexo. Pergunta: “qtd de registros por sexo”

Fonte: Elaboração própria (2026).

Nota: CSE = canônica sem esquema; CCE = canônica com esquema; VSE = variação linguística sem esquema; VCE = variação linguística com esquema.

Em todos os cenários, os *prompts* mantiveram instruções comuns para reduzir respostas fora do padrão esperado: retornar somente uma consulta SQL, sem explicações, comentários ou formatação adicional, compatível com SQLite e restrita a instruções de leitura. Essa padronização facilitou a extração automática da consulta gerada e sua posterior validação.

Após a chamada ao modelo, a resposta textual foi processada pelos *scripts* para extração da consulta SQL candidata, removendo elementos externos, como marcadores de bloco de código, textos explicativos e espaços excedentes, sem corrigir semanticamente a saída gerada. Para cada combinação entre modelo, pergunta e

cenário, foram registrados a resposta bruta, a consulta extraída, o tempo de geração e os metadados da execução. Ao todo, foram produzidas 360 predições SQL, resultantes da combinação entre três modelos, 30 perguntas e quatro cenários, posteriormente submetidas às etapas de validação, execução e cálculo das métricas.

3.6 POLÍTICA DE SEGURANÇA E VALIDAÇÃO DAS CONSULTAS

Após a extração da consulta SQL candidata, cada predição passou por uma etapa de validação antes da execução no banco. A política adotada foi *SELECT-only*, isto é, apenas consultas de leitura iniciadas por SELECT foram consideradas válidas. Essa restrição buscou impedir que respostas geradas pelos modelos executassem comandos de modificação de dados ou alteração estrutural, como INSERT, UPDATE, DELETE, DROP, ALTER e CREATE. A validação também rejeitou consultas vazias e respostas com múltiplas instruções SQL.

As consultas aprovadas nessa filtragem foram executadas no banco SQLite em modo somente leitura, reforçando a proteção contra alterações indevidas. A validação não teve a finalidade de corrigir semanticamente a resposta do modelo, mas apenas de definir se a consulta poderia ser executada conforme o protocolo experimental.

As falhas foram classificadas em cinco categorias: violação de política, quando a resposta não seguia a regra *SELECT-only* ; erro sintático, quando a consulta apresentava SQL malformado; erro de execução, quando a consulta era sintaticamente reconhecida, mas falhava no banco, por exemplo, ao referenciar tabela ou coluna inexistente; *timeout*, quando a execução excedia o limite de tempo definido; e resultado incorreto, quando a consulta executava sem erro, mas retornava resultado diferente daquele produzido pela consulta SQL de referência.

Essa classificação permitiu diferenciar falhas de segurança, problemas estruturais da SQL gerada e erros semânticos de resposta. A mesma política foi aplicada a todos os modelos e cenários, preservando a comparabilidade entre as predições avaliadas.

3.7 PROTOCOLO DE AVALIAÇÃO

A avaliação foi realizada por execução, comparando o resultado retornado por cada consulta SQL gerada com o resultado produzido pela consulta SQL de referência correspondente. Essa escolha está alinhada à métrica *Execution Accuracy* (EX),

recorrente em avaliações de *Text-to-SQL* por considerar o resultado obtido no banco, e não apenas a semelhança textual entre a consulta prevista e a consulta de referência (Yu *et al.*, 2018; Shi *et al.*, 2025).

Para cada pergunta, a consulta SQL de referência foi executada previamente no banco SQLite, produzindo o resultado esperado. Quando a predição gerada pelo modelo passava pela validação, ela era executada no mesmo banco e comparada ao resultado de referência. A predição foi considerada correta quando ambos os resulta-dos coincidiam após a normalização adotada no experimento, incluindo padronização da estrutura retornada e comparação das tuplas resultantes. A EX foi calculada pela razão entre o número de consultas corretas e o total de consultas avaliadas em cada recorte de análise, conforme a Equação 1.

$$EX = \frac{número\ de\ predições\ corretas}{número\ total\ de\ predições\ no\ recorte} x100 \qquad (1)$$

As predições rejeitadas na validação, bem como as consultas com erro sintático, erro de execução, *timeout* ou resultado incorreto, foram contabilizadas como não corretas no cálculo da EX. Cada falha também foi registrada em sua respectiva categoria, conforme a política descrita na subseção anterior, permitindo analisar a taxa de acerto e os principais tipos de erro produzidos por cada modelo.

Além da EX, foram registradas duas medidas de tempo: a latência de geração e a latência de execução. A latência de geração corresponde ao tempo necessário para o modelo produzir uma resposta a partir do *prompt* recebido. A latência de execução corresponde ao tempo gasto para executar, no SQLite, a consulta extraída e validada. Essas medidas foram utilizadas de forma complementar, com o objetivo de observar diferenças de tempo de processamento entre modelos e cenários.

Os resultados foram consolidados por modelo e por cenário experimental, permitindo comparar o desempenho geral dos modelos, o efeito do fornecimento do esquema no *prompt* e o impacto das variações linguísticas superficiais nas perguntas. A combinação entre EX, categorias de erro e latências forneceu a base quantitativa para a análise apresentada na seção de resultados.

4 RESULTADOS E DISCUSSÃO

Esta seção apresenta os resultados obtidos a partir da execução do protocolo experimental descrito na metodologia. São analisados o desempenho geral dos modelos avaliados, a variação da EX nos quatro cenários experimentais, o impacto do fornecimento do esquema do banco de dados, o comportamento diante de variações linguísticas controladas, as categorias de erro observadas e as medidas de latência de geração e execução.

A análise é conduzida de forma comparativa entre os modelos Qwen2.5-Coder 7B, DeepSeek-Coder 6.7B e Llama 3 8B, considerando a métrica principal *Execution Accuracy* (EX). Além dos valores quantitativos, são discutidos os principais padrões observados nos resultados, buscando relacionar o desempenho dos modelos às condições experimentais avaliadas.

4.1 VISÃO GERAL DAS EXECUÇÕES EXPERIMENTAIS

Foram avaliados três modelos de linguagem em quatro cenários experimentais, totalizando 12 execuções. Cada condição foi executada uma única vez, com os mes-mos parâmetros de geração e avaliação. Assim, os resultados desta seção devem ser interpretados de forma descritiva e comparativa, e não inferencial, uma vez que o estudo não incluiu replicações suficientes para estimar variabilidade experimental entre execuções. Cada execução considerou 30 perguntas, resultando em 360 pre-dições SQL avaliadas. Os cenários foram definidos pela combinação de dois fatores: presença ou ausência do esquema do banco de dados no *prompt* e presença ou au-sência de variação linguística controlada nas perguntas.

Assim, o desenho experimental permitiu observar o desempenho dos modelos em quatro condições: perguntas canônicas sem esquema, perguntas canônicas com esquema, perguntas com variação linguística sem esquema e perguntas com variação linguística com esquema. Essa organização possibilita analisar tanto o desempenho geral dos modelos quanto o efeito dos fatores esquema e variação linguística.

A métrica principal de desempenho foi a *Execution Accuracy* (EX), complementada pela análise das categorias de erro e das medidas de latência de geração e execução. A Tabela 4 sintetiza a distribuição das execuções por modelo, indicando também o perfil de cada modelo avaliado. O perfil “Código” refere-se a modelos volta-dos à geração de código, enquanto o perfil “Geral” indica um modelo de instrução de propósito mais amplo.

A partir dessa configuração, as subseções seguintes apresentam a comparação do desempenho geral e por cenário, seguida da análise do impacto do esquema, do comportamento diante das variações linguísticas controladas, das categorias de erro e das medidas de latência

Tabela 4 – Síntese quantitativa das execuções experimentais por modelo.

Modelo	Perfil do modelo	Cenários	Perguntas/cenário	Predições SQL
M1	Código	4	30	120
M2	Código	4	30	120
M3	Geral	4	30	120
Total	–	4	30	360

Fonte: Elaboração própria (2026).

Nota: M1 = Qwen2.5-Coder 7B; M2 = DeepSeek-Coder 6.7B; M3 = Llama 3 8B.

4.2 DESEMPENHO GERAL E POR CENÁRIO DOS MODELOS

A Tabela 5 apresenta a EX obtida por cada modelo nos quatro cenários experimentais. Para melhor organização visual, os cenários são identificados como CSE, CCE, VSE e VCE, conforme nota apresentada abaixo da tabela. Em termos descri-tivos, o Qwen2.5-Coder 7B apresentou o maior valor agregado de EX, com 80,0%, seguido pelo DeepSeek-Coder 6.7B, com 71,7%, e pelo Llama 3 8B, com 69,2%.

Entretanto, a análise por cenário revela diferenças importantes. Nos cenários com fornecimento do esquema, todos os modelos alcançaram EX igual ou superior a 86,7%. Por outro lado, nos cenários sem esquema, a EX foi menor, especialmente para DeepSeek-Coder 6.7B e Llama 3 8B. Esse comportamento indica que a com-paração entre modelos deve considerar não apenas a EX agregada, mas também as condições de entrada fornecidas no *prompt*.

Tabela 5 – *Execution Accuracy* por modelo e cenário.

Modelo	CSE	CCE	VSE	VCE	EX geral
Qwen2.5-Coder 7B	66,7%	90,0%	66,7%	96,7%	80,0%
DeepSeek-Coder 6.7B	43,3%	93,3%	50,0%	100,0%	71,7%
Llama 3 8B	46,7%	86,7%	50,0%	93,3%	69,2%

Fonte: Elaboração própria (2026).

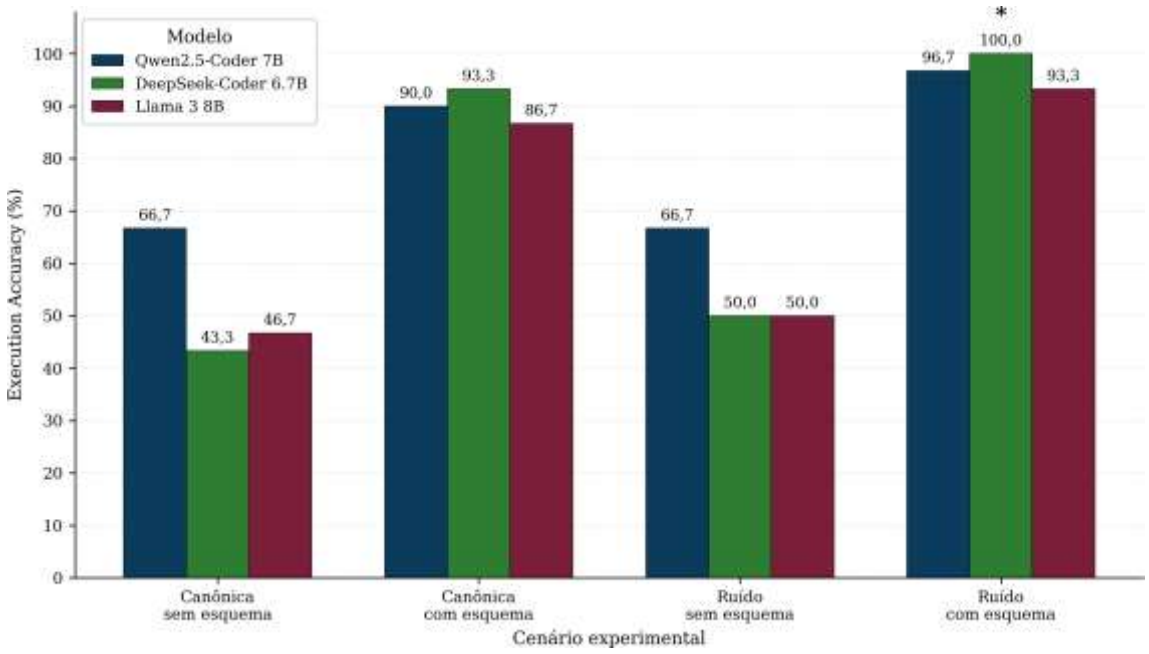
Nota: CSE = canônica sem esquema; CCE = canônica com esquema; VSE = variação linguística sem esquema; VCE = variação linguística com esquema.

Como cada cenário avaliou 30 perguntas, as diferenças pontuais entre modelos devem ser interpretadas de forma descritiva, especialmente quando os valores de EX são próximos. Ainda assim, os resultados mostram que a especialização em có-digo não garantiu superioridade em todos os cenários. O DeepSeek-Coder 6.7B, por exemplo, apresentou desempenho inferior ao Llama 3 8B no cenário canônico sem esquema, embora tenha alcançado os maiores valores nos cenários com esquema. Já o Qwen2.5-Coder 7B apresentou comportamento mais estável, mantendo 66,7% nos cenários sem esquema e atingindo 96,7% no cenário com variação linguística e com esquema.

Esse resultado evidencia que o perfil do modelo não explica isoladamente o desempenho. A disponibilidade do esquema no *prompt* aparece como um fator rele vante para a produção de consultas mais corretas, especialmente em bancos relacionais com múltiplas tabelas e relações, conforme discutido na literatura sobre *schema linking* (Shi *et al.*, 2025; Li *et al.*, 2023).

A Figura 3 apresenta a EX por meio de barras agrupadas, permitindo comparar simultaneamente os modelos em cada cenário experimental. No gráfico, a altura de cada barra representa a EX obtida pelo modelo, facilitando a visualização das diferen-ças entre cenários com e sem fornecimento do esquema.

Figura 3 – *Execution Accuracy* dos modelos nos quatro cenários experimentais



Fonte: Elaboração própria (2026).

Nota: O asterisco indica o único resultado perfeito do experimento, obtido pelo DeepSeek-Coder 6.7B no cenário com variação linguística e com esquema.

A figura também mostra que esse efeito não ocorreu de forma homogênea entre os modelos. O DeepSeek-Coder 6.7B apresentou maior sensibilidade à presença do esquema, saindo de desempenho baixo nos cenários sem contexto estrutural para o maior resultado isolado do estudo. O Qwen2.5-Coder 7B, embora não tenha atingido o maior valor isolado, apresentou menor variação entre os cenários, indicando comportamento mais estável nas condições sem apoio estrutural explícito. Observou-se, em algumas condições, aumento da EX nas variações linguísticas controladas em relação às perguntas canônicas. Esse resultado não deve ser interpretado como evidência de que “mais ruído melhora o modelo”, mas como indício de que determinadas reformulações mais curtas ou mais diretas podem ter reduzido ambiguidades presentes na formulação canônica. Por essa razão, a comparação foi interpretada como teste de sensibilidade a formulações alternativas, e não como experimento de degradação linguística aleatória.

4.3 IMPACTO DO FORNECIMENTO DO ESQUEMA

Nos resultados o fornecimento do esquema do banco de dados no *prompt* foi o Fator experimental mais associado ao aumento da EX dos modelos. Para essa análise,

foram agrupados os dois cenários sem esquema (CSE e VSE) e os dois cenários com esquema (CCE e VCE), permitindo comparar o efeito da disponibilização das tabelas, colunas e relações do banco.

A Tabela 6 reúne duas leituras complementares do efeito do esquema: a comparação da EX média por modelo e as faixas de desempenho segundo a dificuldade das perguntas. Todos os modelos apresentaram ganho expressivo de EX com o fornecimento do esquema. O maior aumento foi observado no DeepSeek-Coder 6.7B, cuja EX média passou de 46,7% para 96,7%, um acréscimo de 50,0 pontos percentuais. O Llama 3 8B também apresentou ganho elevado, passando de 48,4% para 90,0%. Já o Qwen2.5-Coder 7B apresentou menor ganho, mas manteve o melhor desempenho médio sem esquema.

Tabela 6 – Impacto do fornecimento do esquema na EX por modelo e dificuldade.

Comparação por modelo				
Modelo	SE	CE	Dif.	Var. rel.
Qwen2.5-Coder 7B	66,7%	93,4%	+26,7	+40,0%
DeepSeek-Coder 6.7B	46,7%	96,7%	+50,0	+107,2%
Llama 3 8B	48,4%	90,0%	+41,7	+86,1%
Faixas de EX por dificuldade				
Dificuldade	Sem esquema		Com esquema	
Intermediária (*n* = 15)	40,0%–73,3%		86,7%–100,0%	
Complexa (*n* = 14)	42,9%–57,1%		85,7%–100,0%	

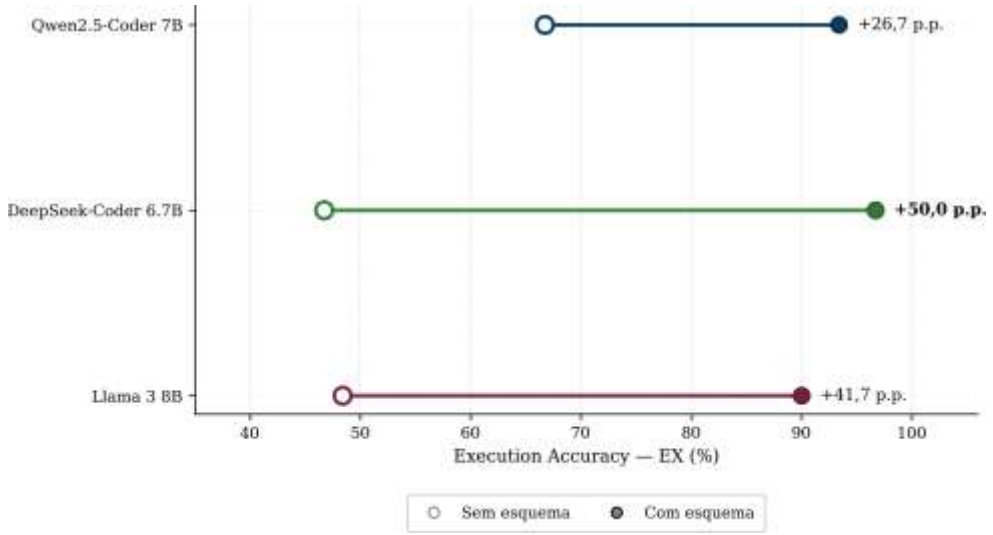
Fonte: Elaboração própria (2026).

Nota: SE = sem esquema; CE = com esquema; Dif. = diferença em pontos percentuais; Var. rel. = variação relativa. As faixas por dificuldade representam os menores e maiores valores de EX observados entre os modelos e cenários agrupados. A classe simples (n = 1) não foi apresentada porque uma única pergunta não permite uma faixa de desempenho.

Esse resultado reforça a relevância do contexto estrutural na tarefa *Text-to-SQL*. Ao receber explicitamente o esquema, o modelo passa a operar com informações sobre as tabelas e colunas disponíveis, reduzindo a necessidade de inferir a estrutura do banco apenas a partir da pergunta. Esse comportamento é coerente com a literatura sobre *schema linking*, que aponta o alinhamento entre linguagem natural e esquema relacional como um dos principais desafios da tarefa (Shi *et al.*, 2025; Li *et al.*, 2023). Do ponto de vista comparativo, o Qwen2.5-Coder 7B manteve o melhor desempenho médio sem esquema, enquanto o DeepSeek-Coder 6.7B apresentou o maior ganho com a inclusão desse recurso. Esse contraste indica que o fornecimento do esquema não apenas elevou a EX geral, mas também alterou a relação de desem-penho entre os modelos avaliados. A Figura 4 utiliza um gráfico do tipo *dumbbell*, no qual cada linha conecta a

EX média do modelo sem esquema e com esquema. A distância entre os pontos representa o ganho associado ao fornecimento do esquema no *prompt*.

Figura 4 – Variação da *Execution Accuracy* com e sem fornecimento do esquema



Fonte: Elaboração própria (2026).

Nota: Os valores sem esquema correspondem à média de CSE e VSE; os valores com esquema correspondem à média de CCE e VCE.

Todos os modelos apresentam deslocamento para valores mais altos de EX quando o esquema é incluído no *prompt*, mas a magnitude do ganho varia entre eles. O DeepSeek-Coder 6.7B apresentou o maior deslocamento, enquanto o Qwen2.5-Coder 7B partiu de um desempenho sem esquema mais elevado e teve ganho menor. O segundo painel da Tabela 6 evidencia que o ganho não se restringiu à EX agregada dos modelos. Tanto nas perguntas intermediárias quanto nas complexas, a faixa de desempenho dos cenários com esquema ficou integralmente acima da faixa observada nos cenários sem esquema. O contraste foi particularmente relevante nas consultas complexas, indicando que o contexto estrutural contribuiu para tarefas que exigem maior precisão na seleção de colunas, no uso de junções e na composição da SQL.

4.4 COMPORTAMENTO DIANTE DE VARIAÇÕES LINGUÍSTICAS CONTROLADAS

A análise do comportamento diante das variações linguísticas considerou a comparação entre as perguntas canônicas e suas versões com alterações superficiais de escrita. Para isso, foram agrupados os cenários com perguntas canônicas (CCE e CSE) e os cenários com variação linguística controlada (VSE e VCE), preservando em ambos

os grupos a presença de condições com e sem esquema.

A Tabela 7 mostra que as variações linguísticas controladas não reduziram a EX média dos modelos avaliados. Em termos descritivos, observou-se aumento leve em todos os casos: Qwen2.5-Coder 7B passou de 78,4% para 81,7%, DeepSeek-Coder 6.7B passou de 68,3% para 75,0%, e Llama 3 8B passou de 66,7% para 71,7%.

Esse resultado deve ser interpretado com cautela. Os valores observados não indicam que as variações linguísticas melhoram, por si só, a tarefa de *Text-to-SQL*. Eles sugerem apenas que as alterações introduzidas no conjunto avaliativo foram suficientemente leves para não degradar o desempenho médio dos modelos. Assim, o resultado aponta estabilidade diante de alterações superficiais de escrita, como abreviações, simplificações lexicais, erros ortográficos controlados e formulações mais coloquiais.

Tabela 7 – Impacto das variações linguísticas controladas na EX.

Modelo	SV	CV	Dif.	Var. rel.
Qwen2.5-Coder 7B	78,4%	81,7%	+3,3	+4,3%
DeepSeek-Coder 6.7B	68,3%	75,0%	+6,7	+9,8%
Llama 3 8B	66,7%	71,7%	+5,0	+7,5%

Fonte: Elaboração própria (2026).

Nota: SV = sem variação linguística; CV = com variação linguística controlada; Dif. = diferença em pontos percentuais; Var. rel. = variação relativa.

Uma explicação plausível é que as perguntas com variação preservaram a intenção semântica central e mantiveram termos relevantes para o mapeamento entre a pergunta e o banco de dados. Nesse sentido, o efeito das variações linguísticas foi menos determinante do que o fornecimento do esquema, que se mostrou o fator de maior impacto nos resultados. Essa leitura é coerente com estudos que discutem a influência de perturbações linguísticas em sistemas *Text-to-SQL*, nos quais alterações superficiais tendem a produzir efeitos diferentes de ruídos semânticos ou estruturais mais severos (Wretblad *et al.*, 2024).

4.5 ANÁLISE DAS CATEGORIAS DE ERRO

Além da EX, foram analisadas as categorias de erro registradas durante a validação e avaliação das consultas. Essa análise permite diferenciar falhas estruturais, relacionadas à formação ou execução da consulta SQL, de falhas semânticas, nas quais

a consulta é executada, mas retorna resultado diferente da referência.

A Tabela 8 apresenta a distribuição das falhas por modelo. O Qwen2.5-Coder 7B apresentou o menor número total de erros, com 24 falhas em 120 consultas. O DeepSeek-Coder 6.7B registrou 34 falhas, com maior concentração em erros de execução. O Llama 3 8B apresentou 37 falhas, sendo o modelo com maior número de resultados incorretos.

Tabela 8 – Distribuição das categorias de erro por modelo.

Modelo	VP	ES	EE	TO	RI	Total
Qwen2.5-Coder 7B	0	0	11	0	13	24
DeepSeek-Coder 6.7B	2	1	20	0	11	34
Llama 3 8B	0	0	16	0	21	37
Total	2	1	47	0	45	95

Fonte: Elaboração própria (2026).

Nota: VP = violação de política; ES = erro sintático; EE = erro de execução; TO = *timeout*; RI = resultado incorreto.

Considerando o total de erros, observa-se que 50 falhas foram estruturais, somando violações de política, erros sintáticos e erros de execução, enquanto 45 foram classificadas como resultado incorreto. Esse equilíbrio indica que o problema não se limita à geração de consultas SQL executáveis. Mesmo quando a consulta não apresenta erro de sintaxe ou execução, ainda pode falhar no alinhamento entre a intenção da pergunta e o resultado esperado.

Além da distribuição por modelo, as falhas também foram analisadas segundo os cenários experimentais. Como cada cenário reuniu 90 predições SQL, a taxa de falha permite comparar diretamente as condições avaliadas. A Tabela 9 apresenta o total de falhas e a respectiva proporção de predições não corretas em cada cenário.

Os dados da Tabela 9 mostram que a maior parte das falhas ocorreu nos cenários sem fornecimento do esquema. Somados, CSE e VSE concentraram 83 das 95 falhas registradas, enquanto CCE e VCE concentraram apenas 12. Esse padrão reforça a interpretação de que a ausência de contexto estrutural dificultou a geração de consultas corretas, seja por problemas de execução, seja por desalinhamento se-mântico em relação à consulta de referência.

Tabela 9 – Distribuição das falhas por cenário experimental.

Cenário	Condição experimental	Falhas	Taxa de falha
CSE	Canônica sem esquema	43	47,8%
CCE	Canônica com esquema	9	10,0%
VSE	Variação linguística sem esquema	40	44,4%
VCE	Variação linguística com esquema	3	3,3%
Total	360 predições avaliadas	95	26,4%

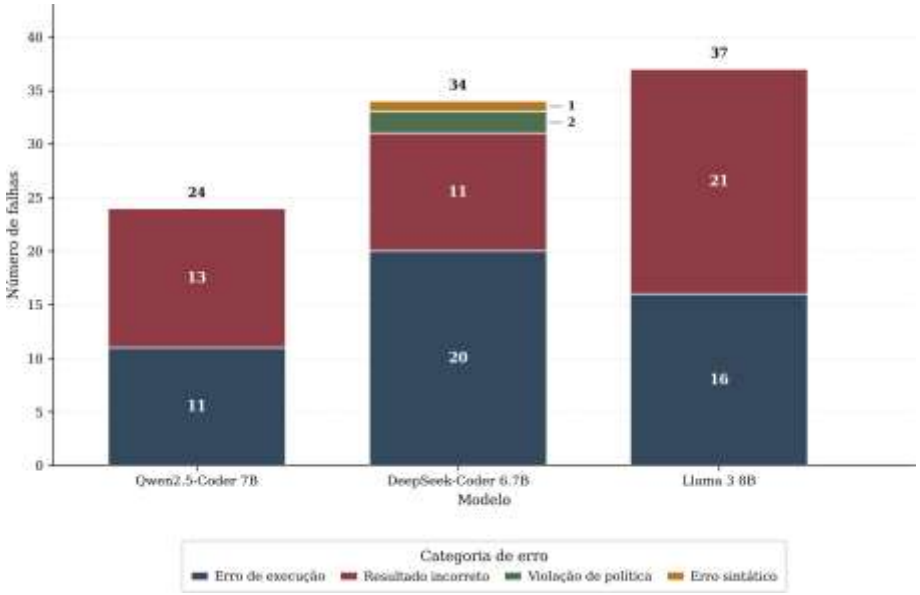
Fonte: Elaboração própria (2026).

As diferenças entre os modelos também são relevantes. No DeepSeek-Coder 6.7B, predominam falhas estruturais, especialmente erros de execução, o que é compatível com seu desempenho mais baixo nos cenários sem fornecimento do esquema. No Llama 3 8B, a maior parte dos erros concentra-se em resultados incorretos, sugerindo maior incidência de consultas válidas do ponto de vista operacional, mas semanticamente desalinhadas. O Qwen2.5-Coder 7B apresentou o menor volume total de falhas, com distribuição próxima entre erros de execução e resultados incorretos.

Essa distinção é importante porque, em *Text-to-SQL*, a validade sintática da consulta não garante sua correção semântica. Uma consulta pode ser executada sem erro e ainda assim responder incorretamente à pergunta formulada, limitação também discutida na literatura sobre avaliação por execução e confiabilidade de métricas como EX (Shi *et al.*, 2025; Kim *et al.*, 2025).

A Figura 5 apresenta barras percentuais empilhadas, nas quais cada barra representa 100% das falhas de um modelo. Assim, o gráfico permite comparar a composição dos erros, enquanto os totais absolutos permanecem indicados na Tabela 8 e pelo valor *n* acima de cada barra.

Figura 5 – Distribuição percentual das categorias de erro por modelo.



Fonte: Elaboração própria (2026).

Na Figura 5, observa-se que os modelos não diferem apenas no volume total de falhas, mas também no tipo de erro predominante. O DeepSeek-Coder 6.7B apresenta maior proporção de falhas estruturais, sobretudo erros de execução, enquanto o Llama 3 8B apresenta maior proporção de resultados incorretos. Esse contraste indica que modelos com desempenho geral relativamente próximo podem falhar por razões distintas.

Algumas predições analisadas ilustram essa diferença entre falhas estruturais e semânticas. Em ocorrências classificadas como erro de execução, foram observadas consultas que referenciavam atributos inexistentes no banco, como uf_residencia e uf_tratamento, em vez de campos presentes no esquema, como uf_resid e uf_trat. Já entre os resultados incorretos, houve casos em que a consulta foi executada, mas respondeu a uma operação diferente da esperada, como aplicar a função AVG ao campo tempo_trat em uma pergunta cuja referência envolvia a mediana do tempo até o tratamento.

4.6 ANÁLISE DE LATÊNCIA DE GERAÇÃO E EXECUÇÃO

A análise temporal do experimento considerou duas medidas: a latência de geração e a latência de execução. A latência de geração corresponde ao tempo necessário para que o modelo produza uma resposta a partir do *prompt*, enquanto a latência de execução corresponde ao tempo utilizado para executar a consulta no banco

SQLite.

A Tabela 10 apresenta as estatísticas agregadas por modelo. O Qwen2.5-Coder 7B apresentou a menor latência média de geração, com 12,26 segundos por consulta. O DeepSeek-Coder 6.7B apresentou tempo médio intermediário, com 29,62 segundos, enquanto o Llama 3 8B foi o modelo mais lento, com média de 41,96 segundos.

Tabela 10 – Latência agregada por modelo.

Modelo	GM	Gp50	Gp95	Gmax	EM
Qwen2.5-Coder 7B	12,26	8,62	19,03	188,29	0,147
DeepSeek-Coder 6.7B	29,62	20,77	68,28	477,41	0,261
Llama 3 8B	41,96	31,97	78,13	565,97	0,260

Fonte: Elaboração própria (2026).

Nota: GM = geração média (s); Gp50 = mediana da geração (s); Gp95 = percentil 95 da geração (s); Gmax = maior latência de geração (s); EM = execução média (s).

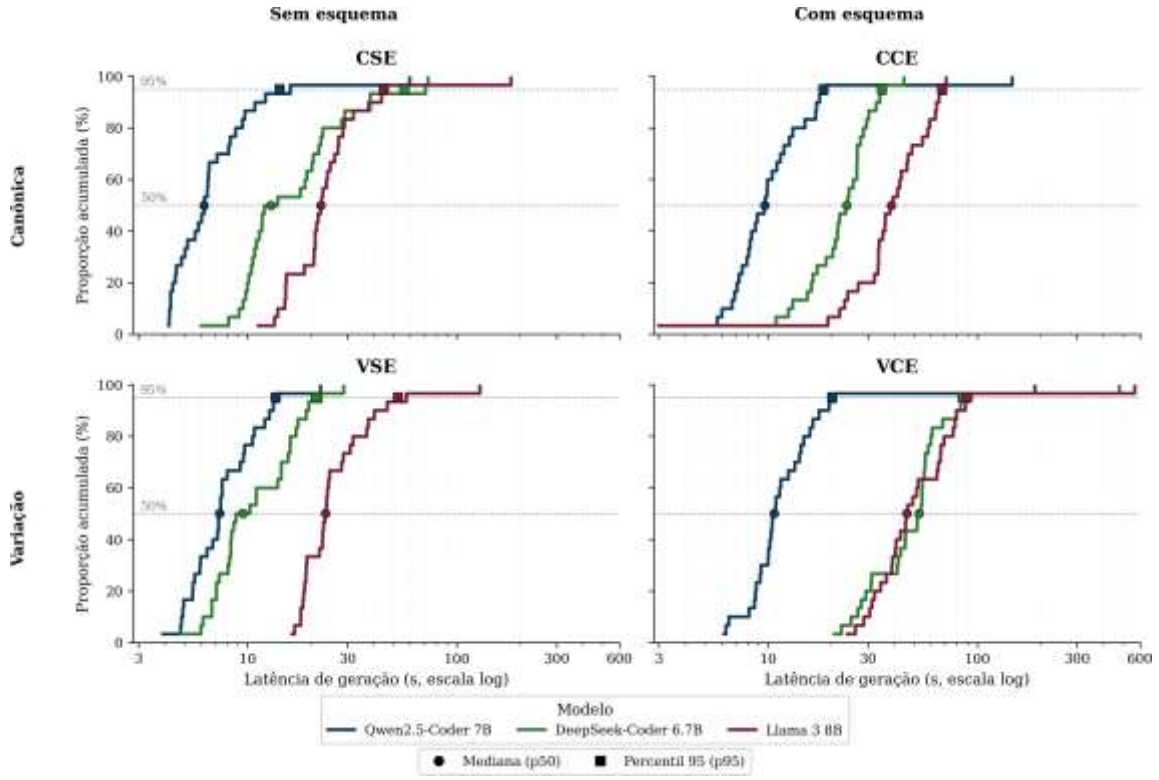
Os resultados indicam que o tempo do *pipeline* esteve concentrado principalmente na etapa de geração pelo modelo. Em todos os casos, a latência média de execução no SQLite permaneceu inferior a 0,27 segundo, indicando impacto reduzido da execução das consultas no tempo total observado.

A comparação entre EX e latência evidencia diferenças práticas entre os mode-los. O Qwen2.5-Coder 7B combinou a maior EX geral com a menor latência média de geração, apresentando a melhor relação entre desempenho e tempo de resposta no ambiente avaliado. Por outro lado, DeepSeek-Coder 6.7B e Llama 3 8B apresentaram maior variabilidade temporal, refletida nos valores de Gp95 e Gmax.

A latência de execução deve ser interpretada cuidadosamente, pois foi calculada apenas para consultas que chegaram à etapa de execução no banco. Assim, a latência de geração constitui a medida mais estável para comparação temporal entre os modelos. Também se observou que os cenários com fornecimento do esquema tenderam a apresentar maior tempo de geração, possivelmente devido ao aumento do conteúdo enviado no *prompt*, aspecto que passa a ser visualizado de forma mais explícita na Figura 6.

A Figura 6 apresenta as curvas ECDF da latência de geração por modelo, agora separadas nos quatro cenários experimentais: CSE, CCE, VSE e VCE. Esse formato permite observar não apenas o comportamento temporal global dos modelos, mas também como a distribuição das latências varia conforme a presença do esquema no *prompt* e conforme o tipo de formulação linguística adotada.

Figura 6 – Curvas ECDF da latência de geração por modelo, separadas por cenário experimental.



Fonte: Elaboração própria (2026).

Nota: Cada painel representa um cenário experimental: CSE (canônica sem esquema), CCE (canônica com esquema), VSE (variação linguística sem esquema) e VCE (variação linguística com esquema). O eixo horizontal está em escala logarítmica. Os marcadores indicam a mediana (p50) e o percentil 95 (p95) da latência de geração.

Na Figura 6, observa-se que o Qwen2.5-Coder 7B mantém, em todos os cenários, curvas mais deslocadas à esquerda, indicando menor latência na maior parte das consultas. O DeepSeek-Coder 6.7B ocupa posição intermediária na maior parte dos painéis, enquanto o Llama 3 8B tende a apresentar distribuições mais deslocadas à direita e maior dispersão. A separação por cenário também mostra que os cenários com esquema tendem a deslocar as curvas para latências mais altas, efeito mais visível em CCE e, sobretudo, em VCE. Assim, o novo arranjo evidencia que o custo temporal não depende apenas do modelo, mas também da quantidade de contexto estrutural incluída no *prompt*.

Esses resultados mostram que a comparação entre modelos não deve considerar apenas a EX final. O tempo de geração também é uma dimensão relevante para avaliar o uso prático de sistemas *Text-to-SQL*, especialmente quando se considera a execução local dos modelos.

4.7 SÍNTESE DOS RESULTADOS

Os resultados obtidos permitem destacar quatro achados principais. Primeiro, em termos descritivos, o Qwen2.5-Coder 7B apresentou o maior desempenho global, alcançando EX geral de 80,0%, além de registrar a menor latência média de geração entre os modelos avaliados. Esse resultado indica a melhor relação entre desempenho e tempo de resposta no ambiente experimental analisado.

Segundo, o fornecimento do esquema do banco de dados no *prompt* foi o fator experimental mais associado ao aumento da EX. Todos os modelos apresentaram ganhos nos cenários com esquema, com destaque para o DeepSeek-Coder 6.7B, cuja EX média aumentou 50,0 pontos percentuais. Esse comportamento é consistente com a literatura sobre *Text-to-SQL*, que aponta o alinhamento entre linguagem natural e esquema relacional como um dos principais desafios da tarefa, especialmente em bancos com múltiplas tabelas e relações (Shi *et al.*, 2025; Li *et al.*, 2023).

Terceiro, as variações linguísticas controladas introduzidas nas perguntas não degradaram o desempenho médio dos modelos. Esse resultado sugere estabilidade diante de alterações superficiais de escrita, como abreviações, simplificações lexicais e formulações mais coloquiais. No entanto, esse achado não deve ser generalizado para ruídos semânticos ou estruturais mais severos, uma vez que as variações utilizadas preservaram a intenção central das perguntas. Essa interpretação está alinhada a estudos que analisam o impacto de perturbações linguísticas em sistemas *Text-to-SQL* (Wretblad *et al.*, 2024).

Por fim, a análise das categorias de erro mostrou que as falhas se dividiram entre problemas estruturais e semânticos. Enquanto erros de execução foram mais frequentes no DeepSeek-Coder 6.7B, resultados incorretos predominaram no Llama 3 8B. Assim, a avaliação por execução evidenciou não apenas diferenças de EX en-tre modelos, mas também diferenças no tipo de falha produzido em cada caso. Essa distinção reforça que a validade operacional da consulta não garante, por si só, sua adequação semântica à pergunta, limitação também discutida na literatura sobre ava-liação por execução e métricas como EX (Shi *et al.*, 2025; Kim *et al.*, 2025).

Em conjunto, os resultados indicam que o desempenho em *Text-to-SQL* depende da interação entre modelo, contexto fornecido no *prompt* e formulação linguística da pergunta. No cenário avaliado, o Qwen2.5-Coder 7B apresentou a melhor relação entre EX e latência, enquanto o fornecimento explícito do esquema mostrou-se o fator mais relevante para elevar o desempenho dos três modelos.

5 CONCLUSÃO

Este trabalho apresentou um *benchmark* experimental para avaliar modelos de linguagem de código aberto na tarefa *Text-to-SQL*, utilizando perguntas em português brasileiro e dados públicos relacionados a atendimentos oncológicos no Brasil. O estudo comparou três modelos executados localmente via Ollama: Qwen2.5-Coder 7B, DeepSeek-Coder 6.7B e Llama 3 8B, em quatro cenários experimentais definidos pela presença ou ausência do esquema do banco no *prompt* e pela presença ou ausência de variações linguísticas controladas nas perguntas.

Os resultados indicaram que, no conjunto experimental avaliado, o Qwen2.5-Coder 7B apresentou a maior EX geral e a menor latência média de geração entre os modelos analisados. Esse resultado sugere melhor relação entre desempenho e tempo de resposta no ambiente considerado, embora diferenças entre modelos devam ser interpretadas de forma descritiva, em razão da escala reduzida do conjunto avaliativo.

O fornecimento do esquema do banco de dados no *prompt* foi o fator mais associado ao aumento da EX. Todos os modelos apresentaram ganhos nos cenários com esquema, o que reforça a importância do contexto estrutural para a geração de consultas SQL corretas. Esse efeito foi mais evidente nas perguntas intermediárias e complexas, nas quais o uso do esquema contribuiu para reduzir erros de referência a colunas, tabelas e relações do banco.

Em relação às variações linguísticas controladas, observou-se que elas não degradaram o desempenho médio dos modelos. Esse resultado sugere estabilidade diante de alterações superficiais de escrita, como abreviações, simplificações lexicais e formulações mais coloquiais. No entanto, as variações utilizadas preservaram a intenção semântica das perguntas e, portanto, não representam ruídos semânticos mais severos, ambíguos ou adversariais.

A análise das categorias de erro mostrou que as falhas não se restringiram à geração de consultas sintaticamente inválidas. Parte relevante dos erros ocorreu em consultas executáveis, mas semanticamente incorretas, evidenciando que produzir SQL válido não garante, por si só, alinhamento com a intenção da pergunta. Esse aspecto reforça a importância de combinar avaliação por execução com análise das categorias de falha.

Como limitações, destaca-se que o *benchmark* foi construído a partir de um único banco de dados de domínio específico, o que restringe a generalização dos resultados para outros contextos. Além disso, foi realizada apenas uma execução por condição

experimental, e a comparação ficou restrita a modelos locais executados via Ollama. O conjunto avaliativo é composto por 30 perguntas canônicas, acompanhadas de variações linguísticas controladas, o que permite uma comparação inicial entre modelos, mas ainda em escala reduzida quando comparado a *benchmarks* mais am-plos da literatura. Outra limitação refere-se ao tipo de variação linguística adotada, restrita a alterações superficiais de escrita, sem explorar ambiguidades semânticas mais complexas.

Como trabalhos futuros, sugere-se ampliar o conjunto de perguntas e diversifi-car os níveis de complexidade SQL, incorporando, por exemplo, subconsultas aninhadas, filtros temporais mais elaborados e combinações mais complexas entre tabelas. Também se recomenda aumentar o número de replicações experimentais, incluir modelos de referência acessados via API e explorar bases com maior heterogeneidade estrutural, de modo a aprofundar a análise de robustez e generalização. Adicionalmente, seria relevante avaliar outros modelos de linguagem de código aberto, incluindo versões mais recentes e modelos especificamente ajustados para geração de SQL, bem como explorar variações linguísticas mais desafiadoras e estratégias de engenharia de *prompt*, validação automática e correção iterativa de consultas.

REFERÊNCIAS

DOU, Longxu *et al.* MultiSpider: towards benchmarking multilingual text-to-SQL semantic parsing. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 37, n. 10, p. 12745–12753, 2023. Acesso em: 26 jan. 2026. DOI: 10.1609/aaai.v37i10.26367. Disponível em: https://doi.org/10.1609/aaai.v37i10.26367.

FRÓES, Karina; BRAGHETTO, Kelly Rosa. **Exploring Temporal Text-to-SQL Challenges in Brazilian Portuguese: Lessons from Educational Data**. *In:* ANAIS do XL Simpósio Brasileiro de Bancos de Dados. Fortaleza/CE: SBC, 2025. p. 963–969. DOI: 10.5753/sbbd.2025.247836. Disponível em: https://sol.sbc.org.br/index.php/sbbd/article/view/37310.

GUO, Daya *et al.* DeepSeek-Coder: when the large language model meets programming – the rise of code intelligence. **arXiv preprint arXiv:2401.14196**, 2024. Acesso em: 26 jan. 2026. DOI: 10.48550/arXiv.2401.14196. Disponível em: https://arxiv.org/abs/2401.14196.

HUI, Binyuan *et al.* Qwen2.5-Coder Technical Report. **arXiv preprint arXiv:2409.12186**, 2024. Acesso em: 26 jan. 2026. Disponível em:

https://arxiv.org/abs/2409.12186.

JOSÉ, Marcelo Archanjo; COZMAN, Fabio Gagliardi. mRAT-SQL+GAP: a Portuguese text-to-SQL transformer*. In:* BRITTO, Alceu; VALDIVIA DELGADO, Karin (ed.). **Intelligent Systems**. Cham: Springer International Publishing, 2021. p. 511–525. ISBN 978-3-030-91699-2.

KIM, Heegyu *et al.* FLEX: expert-level false-less EXecution metric for reliable text-to-SQL benchmark*. In:* CHIRUZZO, Luis; RITTER, Alan; WANG, Lu (ed.). **Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)**. Albuquerque: Association for Computational Linguistics, 2025. p. 4448–4475. Acesso em: 26 jan. 2026. ISBN 979-8-89176-189-6. DOI: 10.18653/v1/2025.naacl-long.228. Disponível em: https://aclanthology.org/2025.naacl-long.228.

LHUCATENORIO. **Oncology Treatment DataSUS – Brazil (2013–2023)**. [*S. l.: s. n.*], 2025. Kaggle. Disponível em: https://www.kaggle.com/datasets/lhucastenorio/oncology-treatment-datasus-brazil-20132023. Acesso em: 26 jan. 2026.

LI, Jinyang *et al.* Can LLM **Already Serve as a Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs***. In:* PROCEEDINGS of the Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track. New Orleans: [s. n.], 2023. Acesso em: 26 jan. 2026. Disponível em: https://openreview.net/forum?id=dI4wzAE6uV

.LLAMA TEAM. The Llama 3 Herd of Models. **arXiv preprint arXiv:2407.21783**, 2024. Acesso em: 26 jan. 2026. Disponível em: https://arxiv.org/abs/2407.21783.

MARCONI, Marina de Andrade; LAKATOS, Eva Maria. **Fundamentos de metodologia científica**. 5. ed. São Paulo: Atlas, 2003. ISBN 85-224-3397-6.

OLLAMA. **Ollama documentation**. [*S. l.: s. n.*], 2025. Disponível em: https://github.com/ollama/ollama. Acesso em: 15 jan. 2026.

PHAM, Khanh Trinh *et al.* **Multilingual text-to-SQL: benchmarking the limits of language models with collaborative language agents***. In:* BOROVICA-GAJIC, Renata

et al. (ed.). **Databases Theory and Applications**. Singapore: Springer Nature Singapore, 2026. p. 108–123. Acesso em: 26 jan. 2026. DOI: 10.1007/978-981-95-6196-4_8. Disponível em: https://doi.org/10.1007/978-981-95-6196-4_8.

PYTHON SOFTWARE FOUNDATION. **sqlite3 — DB-API 2.0 interface for SQLite databases**. [*S. l.: s. n.*], 2026. Acesso em: 05 jul. 2026. Disponível em: https://docs.python.org/3/library/sqlite3.html

SHI, Liang *et al.* A Survey on Employing Large Language Models for Text-to-SQL Tasks. **ACM Computing Surveys**, Association for Computing Machinery, New York, v. 58, n. 2, p. 1–37, 2025. Acesso em: 26 jan. 2026. ISSN 0360-0300. DOI: 10.1145/3737873. Disponível em: https://doi.org/10.1145/3737873.

SQLITE CONSORTIUM. **About SQLite**. [*S. l.: s. n.*], 2025. Acesso em: 05 jul. 2026. Disponível em: https://www.sqlite.org/about.html.

WOHLIN, Claes *et al.* **Experimentation in Software Engineering**. Berlin, Heidelberg: Springer, 2012. Acesso em: 26 jan. 2026. ISBN 978-3-642-29044-2. DOI: 10.1007/978-3-642-29044-2. Disponível em: https://doi.org/10.1007/978-3-642-29044-2.

WRETBLAD, Nils *et al.* Understanding the effects of noise in text-to-SQL: an examination of the BIRD-bench benchmark*. In:* KU, Lun-Wei; MARTINS, André; SRIKUMAR, Vivek (ed.). **Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)**. Bangkok: Association for Computational Linguistics, 2024. p. 356–369. Acesso em: 26 jan. 2026. DOI: 10.18653/v1/2024.acl-short.34. Disponível em: https://aclanthology.org/2024.acl-short.34.

YU, Tao *et al.* **Spider: a large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task***. In:* PROCEEDINGS of the 2018 Conference on Empirical Methods in Natural Language Processing. Brussels, Belgium: Association for Computational Linguistics, 2018. p. 3911–3921. Acesso em: 26 jan. 2026. DOI: 10.18653/v1/D18-1425. Disponível em: https://aclanthology.org/D18-1425/.

APÊNDICE A – REPOSITÓRIO E ARTEFATOS EXPERIMENTAIS

A estrutura do *pipeline* experimental utilizado neste trabalho está disponível em repositório público, incluindo *scripts*, organização dos diretórios, consultas SQL de referência, registros de execução e documentação necessária para reprodução do experimento.

A partir das instruções disponibilizadas, é possível replicar o processo experimental e gerar localmente os relatórios de execução e avaliação.

https://github.com/fabricio-c-mota/text2sql-benchmark.git