UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

FRANCIEL ALMEIDA SIMÃO

EXECUÇÃO LOCAL DE MODELOS DE LINGUAGEM NO NAVEGADOR: ANÁLISE COMPARATIVA E ESTUDO DE CASO

GUARAPUAVA

2026

FRANCIEL ALMEIDA SIMÃO

EXECUÇÃO LOCAL DE MODELOS DE LINGUAGEM NO NAVEGADOR: ANÁLISE COMPARATIVA E ESTUDO DE CASO

Local Execution of Language Models in the Browser: Comparative Analysis and Case Study

Projeto de Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Andres Jessé Porfirio

GUARAPUAVA

2026

4.0 Internacional

Esta licença permite compartilhamento, remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es). Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

RESUMO

Este trabalho propõe a integração do WebLLM, uma biblioteca que permite a execução de modelos de linguagem diretamente no navegador por meio da tecnologia WebGPU, ao DrawDB, uma aplicação web de código aberto voltada à diagramação e modelagem de bancos de dados. O objetivo é viabilizar a utilização desses modelos localmente, sem depender de serviços em nuvem ou do envio de dados a servidores externos. A metodologia compreende a catalogação dos modelos disponíveis no WebLLM, considerando suas características, parâmetros de quantização, consumo de memória e requisitos de execução; a implementação de uma interface de chat integrada ao DrawDB; e a definição de um protocolo de execução e avaliação. A interface desenvolvida permite selecionar e carregar modelos, enviar solicitações e visualizar as respostas geradas de forma incremental, por meio de streaming. A avaliação contempla métricas quantitativas, como tempo de carregamento, latência total, tempo até o primeiro token, velocidade de geração e número de tokens gerados, além de critérios qualitativos relacionados à correção, relevância e coerência das respostas. Espera-se, com este trabalho, avaliar a viabilidade da integração local de modelos de linguagem em aplicações web, considerando o equilíbrio entre desempenho e qualidade das respostas, bem como seu potencial para utilização em sistemas que priorizem a privacidade dos dados e a acessibilidade.

Palavras-chave: webllm; modelos de linguagem; drawdb; inteligência artificial; computação local.

ABSTRACT

This work proposes the integration of WebLLM, a library that allows the execution of language models directly in the browser through WebGPU technology, with DrawDB, an open-source web application aimed at database diagramming and modeling. The objective is to enable the use of these models locally, without relying on cloud services or sending data to external servers. The methodology includes cataloging the models available in WebLLM, considering their characteristics, quantization parameters, memory consumption, and execution requirements; implementing a chat interface integrated with DrawDB; and defining an execution and evaluation protocol. The developed interface allows users to select and load models, send requests, and view the generated responses incrementally through streaming. The evaluation includes quantitative metrics such as loading time, total latency, time to the first token, generation speed, and number of tokens generated, as well as qualitative criteria related to the correctness, relevance, and coherence of the responses. This work aims to evaluate the feasibility of local integration of language models in web applications, considering the balance between performance and response quality, as well as their potential for use in systems that prioritize data privacy and accessibility.

Keywords: webllm; language models; drawdb; artificial intelligence; local computing.

SUMÁRIO

1	**INTRODUÇÃO**	**4**
1.1	**Objetivos**	**5**
1.1.1	Objetivo geral	5
1.1.2	Objetivos específicos	5
2	**MATERIAIS E MÉTODOS**	**6**
2.1	**Materiais**	**6**
2.1.1	Infraestrutura e Ambiente de Desenvolvimento	6
2.1.2	Biblioteca WebLLM	6
2.1.3	DrawDB	6
2.2	**Métodos**	**7**
2.2.1	Identificação e catalogação dos modelos	7
2.2.2	Implementação da aplicação web	7
2.2.3	Protocolo de execução e avaliação	8
2.2.4	Definição do cenário experimental	8
2.2.5	Execução dos experimentos e coleta de dados	9
3	**RESULTADOS PARCIAIS**	**10**
3.1	**Identificação e catalogação dos modelos**	**10**
3.2	**Implementação da aplicação web**	**11**
3.3	**Protocolo de execução e avaliação**	**13**
3.3.1	Resultados dos prompts	13
3.3.2	Avaliação quantitativa	15
3.3.3	Avaliação qualitativa	16
3.3.4	Análise dos resultados	16
4	**CONSIDERAÇÕES FINAIS**	**17**
	REFERÊNCIAS	**18**

1 INTRODUÇÃO

Com a chegada da quarta revolução industrial, o avanço digital se demonstrou cada vez mais presente no cotidiano da população global. Segundo Klaus Schwab, fundador do Fórum Econômico Mundial, a nova revolução é caracterizada pela velocidade exponencial de tecnologias emergentes (SCHWAB, 2016). A Inteligência Artificial é apontada como cérebro e motor desta revolução, em que seu impacto é visto em três esferas: o físico, como integração a veículos autônomos; o biológico, como no monitoramento de biodiversidade; e especialmente no digital, com a chegada dos Modelos de Linguagem (LMs).

Tais modelos, popularizados por ferramentas como o GPT da OpenAI, Gemini do Google e o Claude da Anthropic, são pré-treinados utilizando imensas quantidades de dados, o que requer uma infraestrutura computacional de grande porte conhecida como data centers.

A proliferação de data centers produz lixo eletrônico e consome recursos hídricos em grande escala: um único data center de médio porte pode demandar até 5 milhões de galões de água por dia, equivalente ao consumo de uma cidade de 50 mil habitantes (GOREY, 2025). A ampla adoção de Modelos de Linguagem em diversos setores tem impulsionado o interesse na implantação desses serviços ao longo de toda a infraestrutura computacional, da nuvem à borda da rede (QU *et al.*, 2025), que gradualmente amplifica tais impactos.

Além disso, a centralização do processamento em nuvem acarreta, ainda, riscos à privacidade. Uma vez que os dados dos usuários são transmitidos a servidores remotos, eles podem ser utilizados para retreinamento dos modelos, conforme estabelecido nas políticas de uso de plataformas como OpenAI e Google (OpenAI, 2025; Google, 2025). Esse processo torna o usuário e corporações vulneráveis a vazamentos de dados sensíveis. Em 2025, um ataque à Mixpanel, fornecedora terceirizada de análises de dados da OpenAI, resultou no vazamento de informações de clientes da plataforma de API. O incidente, que teve origem em campanhas de *phishing* por SMS direcionadas aos funcionários da Mixpanel, permitiu que cibercriminosos acessassem e exportassem uma base de dados que a OpenAI enviava à parceira para fins de telemetria (CNN Brasil, 2025).

Nesse cenário, a viabilidade da execução local de Modelos de Linguagem revela-se como uma alternativa a ser explorada a fim de contribuir com objetivos sustentáveis, além de aprimorar a experiência e transparência com o usuário. O tratamento dos dados é feito localmente, não sendo necessário carregá-los para a nuvem, o que permite evitar os riscos associados ao processo de transmissão pela rede, garantindo assim a segurança dos dados. No caso de um ataque aos dados, este afeta apenas os dados locais e não a totalidade dos dados (CAO *et al.*, 2020).

A execução local de Modelos de Linguagem já é um tema difundido e em constante evolução, com diferentes abordagens de implantação. Os LLMs [1] locais podem variar em arquitetura (transformers, modelos híbridos), tamanho (de centenas de milhões a dezenas de bilhões

[1] Grandes Modelos de Linguagens, do inglês Large Language Models

de parâmetros) e nível de quantização, uma técnica de compressão que reduz a precisão numérica dos pesos do modelo para diminuir consumo de memória e acelerar inferência, em troca de uma redução na qualidade de resposta. Embora soluções nativas como Ollama e LM Studio já permitam essa execução em máquinas locais, o uso de LLMs diretamente em sistemas web por meio de WebLLMs representam uma abordagem ainda emergente (RUAN, 2026).

Deste modo, este projeto busca explorar e analisar os limites para uma integração local e eficiente de Modelos de Linguagens diretamente no navegador do cliente por meio de WebLLMs. Tal abordagem difere de soluções nativas que requerem a instalação e configuração de ambientes como Ollama ou LMStudio, promovendo maior acessibilidade para o uso. Ademais, a proposta busca contribuir com metas sustentáveis a fim de reduzir a dependência de data centers.

1.1 Objetivos

1.1.1 Objetivo geral

Avaliar as capacidades computacionais e limitações operacionais da integração de Modelos de Linguagem executados localmente no navegador via WebLLMs, assim como a integração prática da tecnologia por meio de uma aplicação web real.

1.1.2 Objetivos específicos

Para execução do objetivo geral, o projeto se desenvolverá seguindo as etapas:

- Identificar e catalogar os tipos, arquiteturas, tamanhos e níveis de quantização de Modelos de Linguagem compatíveis com execução via WebLLM;
- Implementar uma aplicação web com interface de chat que permita a interação direta do usuário com o modelo executado localmente no navegador;
- Estabelecer um protocolo padronizado para execução dos testes e avaliação quantitativa e qualitativa das respostas geradas pelos modelos locais;
- Elaborar um cenário experimental para condução dos testes e coleta de métricas;
- Executar os experimentos com os modelos WebLLM identificados, coletando dados de desempenho e qualidade das respostas.

2 MATERIAIS E MÉTODOS

Este capítulo apresenta as tecnologias utilizadas (Seção 2.1) e as etapas metodológicas do trabalho, cada uma associada a um objetivo específico (Seção 2.2).

2.1 Materiais

2.1.1 Infraestrutura e Ambiente de Desenvolvimento

O ambiente de desenvolvimento consiste no uso do sistema operacional **Linux Ubuntu**. O controle de versão do código-fonte é realizado com **Git**[1] e hospedagem remota no **GitHub**[2], que permitem versionamento, colaboração e documentação do projeto. O editor de código selecionado é o **Visual Studio Code**[3]. Por fim, o hardware utilizado para condução de testes consiste no processador **i7-1355U** com placa integrada **Intel Iris XeGraphics** e memória RAM de **16 GB**.

2.1.2 Biblioteca WebLLM

O **WebLLM**[4] é uma biblioteca JavaScript de código aberto que executa a inferência de Modelos de Linguagem diretamente no navegador por meio da API **WebGPU**, dispensando servidores externos. Tem como principal funcionalidade o carregamento de modelos pré-treinados, exigindo navegador com suporte a WebGPU (como Google Chrome, Firefox ou Microsoft Edge).

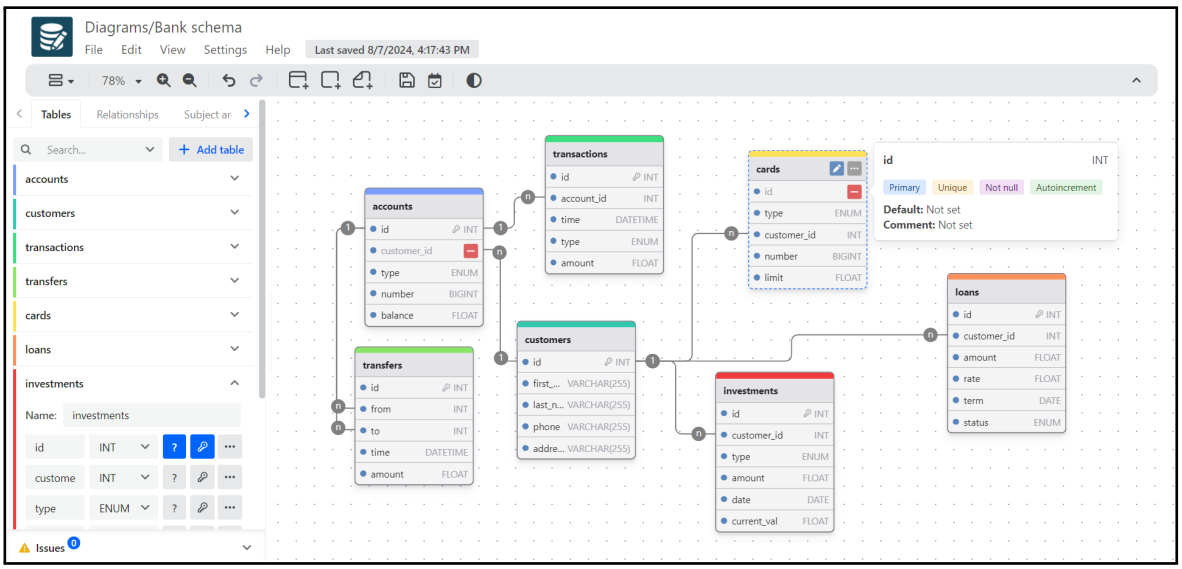
2.1.3 DrawDB

O **DrawDB**[5] é uma aplicação web de código aberto para diagramação de bancos de dados, executada inteiramente no navegador. Ele serve como um caso de uso real para a integração do WebLLM, por meio de uma interface de chat voltada para tarefas como geração de consultas SQL, explicação de diagramas e sugestões de modelagem. Suas principais tecnologias de desenvolvimento englobam **JavaScript**[6], **React**[7] e **TailwindCSS**[8].

A Figura 1 apresenta uma captura de tela da interface de diagramação da aplicação DrawDB onde é possível criar tabelas, atributos e organizar seus relacionamentos.

1 https://git-scm.com
2 https://github.com
3 https://code.visualstudio.com
4 https://webllm.mlc.ai
5 https://github.com/drawdb-io/drawdb
6 https://developer.mozilla.org/en-US/docs/Web/JavaScript
7 https://react.dev
8 https://tailwindcss.com

Figura 1 – Interface da aplicação DrawDB



Fonte: Adaptado de drawdb (2026).

2.2 Métodos

O trabalho compreende cinco etapas sequenciais, correspondentes aos objetivos específicos.

2.2.1 Identificação e catalogação dos modelos

Consulta da documentação oficial e o catálogo de modelos do WebLLM. Para cada modelo, são registrados nome, arquitetura, número de parâmetros, nível de **quantização** [9] e requisitos de execução, incluindo o uso de memória. Esse catálogo caracteriza as opções disponíveis e fundamenta a seleção dos modelos testados.

2.2.2 Implementação da aplicação web

Esta etapa consiste na modificação do DrawDB. Primeiramente, o repositório original da aplicação é clonado e hospedado na máquina local. Em seguida, o projeto é expandido, incluindo uma interface de chat integrada ao WebLLM. O propósito da modificação é permitir a seleção e o carregamento dos modelos escolhidos, além de enviar solicitações durante o uso da aplicação, com a inferência executada no navegador do usuário.

[9] A quantização, é uma técnica que reduz a precisão dos parâmetros de um modelo para um conjunto menor de valores (LANG; GUO; HUANG, 2024), tendo como objetivo economizar recursos computacionais e acelerar a inferência.

2.2.3 Protocolo de execução e avaliação

O protocolo tem como intuito a padronização dos testes, especificando os prompts, os parâmetros de geração, o procedimento de execução, as métricas e os critérios de avaliação. Esses critérios se dividem em quantitativos e qualitativos conforme expresso nas tabelas 1 e 2.

Tabela 1 – Critérios da avaliação quantitativa.

Métrica	**Definição**
Tempo de carregamento	Tempo necessário para o modelo WebLLM ficar disponível no navegador.
Latência total	Tempo entre o envio do prompt no chat e a conclusão da geração.
Tempo até o primeiro token (TTFT)	Tempo entre o envio do prompt e a exibição do primeiro token na interface.
Velocidade de geração	Número de tokens gerados por segundo.
Número de tokens gerados	Quantidade de tokens produzidos por resposta.

Fonte: Autoria própria (2026).

Tabela 2 – Critérios da avaliação qualitativa.

Critério	**Descrição**	**Escala**
Correção	Verifica se a resposta está tecnicamente correta, considerando SQL, estrutura do diagrama e regras de modelagem.	0 = incorreta; 1 = parcialmente correta; 2 = correta com omissões; 3 = correta e completa.
Relevância	Verifica se a resposta atende diretamente à solicitação realizada no chat.	0 = não responde; 1 = superficialmente; 2 = parcialmente relevante; 3 = totalmente relevante.
Coerência	Verifica organização, fluidez e consistência lógica da resposta.	0 = incoerente; 1 = pouco coerente; 2 = coerente com falhas; 3 = coerente e bem estruturada.

Fonte: Autoria própria (2026).

Os resultados de cada modelo são registrados em uma tabela padronizada, contendo as métricas de desempenho e os critérios de avaliação das respostas. Dessa forma, é possível comparar os resultados obtidos por diferentes modelos.

2.2.4 Definição do cenário experimental

Com base no protocolo, o cenário experimental define os modelos testados, as tarefas realizadas no DrawDB e seus prompts, o ambiente de execução (incluindo os recursos computacionais disponíveis) e as condições mantidas constantes entre os testes, além das métricas coletadas em cada execução.

2.2.5 Execução dos experimentos e coleta de dados

Os experimentos são executados no DrawDB com os modelos selecionados. Em cada execução, são registrados o modelo, as condições do teste, as métricas de desempenho e a resposta gerada. As respostas são avaliadas de acordo com os critérios do protocolo e organizadas em tabelas. Por fim, os resultados são analisados de forma descritiva, observando as diferenças de desempenho e de qualidade das respostas entre os modelos.

3 RESULTADOS PARCIAIS

Este capítulo apresenta os resultados parciais obtidos a partir da execução das etapas definidas no método. Os resultados são organizados em seções alinhadas às etapas do desenvolvimento do trabalho, compreendendo a catalogação dos modelos, a implementação da aplicação web e, posteriormente, a execução e avaliação dos testes.

3.1 Identificação e catalogação dos modelos

A primeira etapa consistiu na identificação e catalogação dos modelos disponíveis na biblioteca WebLLM. A consulta foi baseada na documentação oficial e na lista de modelos pré-compilados disponibilizada pela biblioteca (MLC AI, 2026).

Para cada modelo candidato, foram registrados o nome, a arquitetura, o número de parâmetros, o nível de quantização e o requisito estimado de memória. Quanto à quantização, foi adotada a variante **q4f16_1**[1] na qual os pesos são quantizados em 4 bits e as ativações e os cálculos são realizados em ponto flutuante de 16 bits.

A Tabela 3 apresenta o catálogo inicial dos modelos considerados para os testes. Os valores de memória referem-se aos requisitos estimados de VRAM informados na documentação e no catálogo do WebLLM, podendo variar conforme a versão do modelo, o contexto utilizado e as condições de execução.

Tabela 3 – Catálogo inicial de modelos candidatos do WebLLM.

Modelo	Arquitetura	Parâmetros	Quantização	VRAM
Llama-3.2-1B-Instruct-q4f16_1-MLC	Llama 3.2	1B	q4f16_1	∼0,9 GB
Qwen2.5-1.5B-Instruct-q4f16_1-MLC	Qwen2.5	1,5B	q4f16_1	∼1,6 GB
Llama-3.2-3B-Instruct-q4f16_1-MLC	Llama 3.2	3B	q4f16_1	∼2,2 GB
Qwen2.5-3B-Instruct-q4f16_1-MLC	Qwen2.5	3B	q4f16_1	∼2,5 GB
Phi-3.5-mini-instruct-q4f16_1-MLC	Phi-3.5	3,8B	q4f16_1	∼2,5 GB
Llama-3.1-8B-Instruct-q4f16_1-MLC	Llama 3.1	8B	q4f16_1	∼4,5 GB
Qwen2.5-7B-Instruct-q4f16_1-MLC	Qwen2.5	7B	q4f16_1	∼5,1 GB

Fonte: Elaborada pelo autor com base em (MLC AI, 2026).

A partir do catálogo, os modelos foram agrupados conforme o requisito de memória. Modelos com aproximadamente 1 GB de memória (Llama-3.2-1B e Qwen2.5-1.5B) representam alternativas mais leves, adequadas a dispositivos com recursos limitados. Modelos entre aproximadamente 2 GB e 2,5 GB (Llama-3.2-3B, Qwen2.5-3B e Phi-3.5-mini) representam um

[1] Essa configuração reduz o tamanho do modelo e o consumo de memória em relação a variantes de maior precisão, o que a torna adequada ao contexto de execução local no navegador.

ponto intermediário, enquanto modelos próximos a 4,5 GB (Qwen2.5-7B e Llama-3.1-8B) exigem maior capacidade computacional.

Essa catalogação fundamenta a seleção dos modelos que serão integrados e avaliados no DrawDB. A seleção final considerará, além do requisito de memória, a compatibilidade com o WebLLM, a capacidade de compreensão de instruções relacionadas a bancos de dados e o desempenho esperado nas tarefas de geração de SQL, explicação de diagramas e sugestão de modelagem.

3.2 Implementação da aplicação web

A implementação da aplicação web iniciou-se com a clonagem do repositório oficial do DrawDB. O projeto foi executado localmente por meio do ambiente de desenvolvimento do Vite, preservando as funcionalidades originais de diagramação de bancos de dados.

Como ferramenta de apoio para implementação foi utilizado o Claude AI que auxiliou na elaboração e revisão de trechos de código, na identificação de erros e na compreensão de funcionalidades das tecnologias utilizadas.

Para a integração com o WebLLM, foi criada a pasta `src/services/`, responsável por isolar o código relacionado à inferência local do restante da aplicação. A integração foi baseada na documentação oficial do WebLLM (MLC AI, 2023), adotando-se a execução do modelo em um *Web Worker*, a utilização de uma instância única do motor de inferência e a geração de respostas em *streaming*.

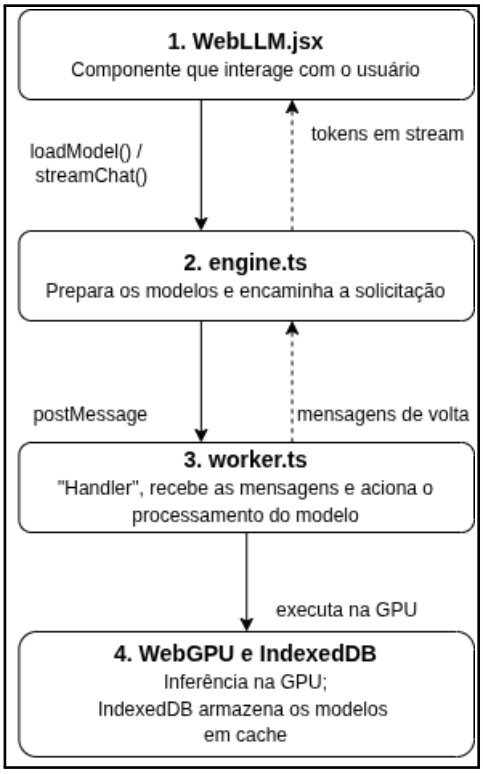
O serviço foi dividido em dois arquivos. No arquivo `worker.ts`, é criado o `WebWorkerMLCEngineHandler`, responsável por receber e responder às mensagens trocadas com o *worker*. Já o arquivo `engine.ts` reúne as funções que interagem com o WebLLM, como listar os modelos disponíveis, carregar o modelo escolhido, gerar respostas em *streaming* e liberar o modelo quando não for mais necessário.

O carregamento dos modelos é feito pela função `CreateWebWorkerMLCEngine`, que inicia o modelo dentro do *Web Worker* a partir do identificador do modelo selecionado. Os arquivos dos modelos são armazenados localmente no *IndexedDB*. Durante o carregamento, o progresso é enviado à interface por meio de uma função de retorno.

A geração de respostas foi implementada por meio da função `streamChat`, que utiliza a API de *chat completions* do WebLLM com a opção de *streaming* habilitada. Dessa forma, os tokens gerados são retornados à interface, permitindo a exibição da resposta conforme a inferência ocorre.

Em seguida, foi criado o componente `WebLLM.jsx`, responsável pela interface de chat integrada ao DrawDB. O componente é exibido como um painel flutuante na interface de edição da aplicação e exibe um seletor com os modelos pré-compilados do WebLLM, um botão para carregamento do modelo, um campo para envio de solicitações e uma área para exibição da resposta gerada. A Figura 2 representa o fluxo de execução por meio de um diagrama.

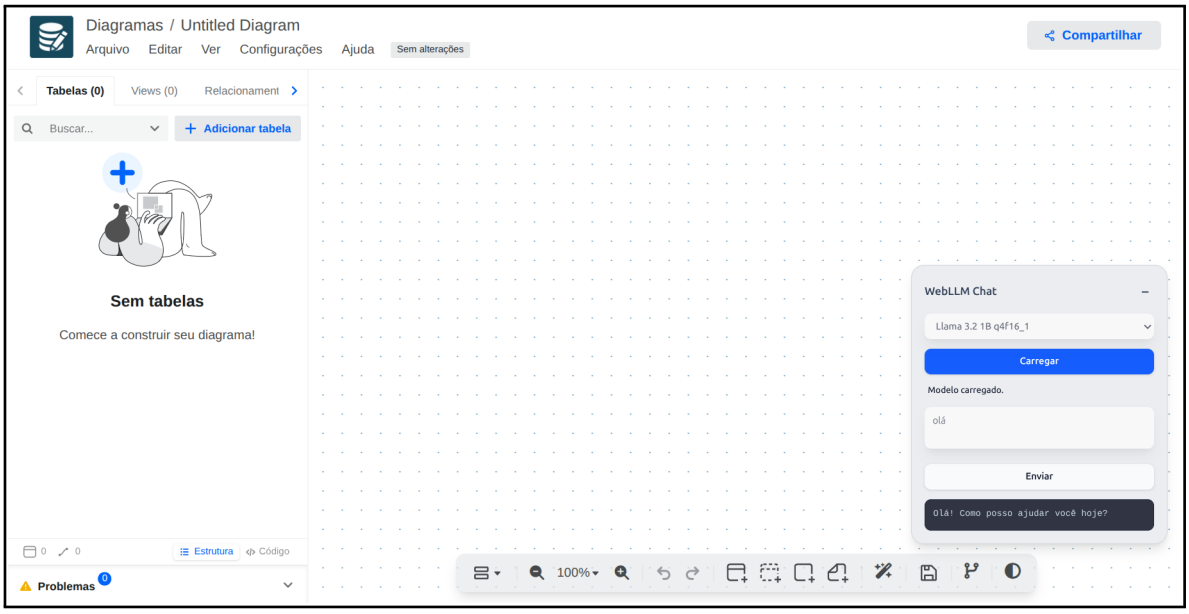
Figura 2 – Fluxo de carregamento e inferência WebLLM



Fonte: Autoria própria (2026).

Após a implementação, foi realizado um teste inicial de funcionamento. A aplicação foi executada localmente e acessada no navegador Google Chrome. O modelo carregado foi o Llama 3.2 1B. A Figura 3 apresenta a interface com a integração do componente `WebLLM.jsx`.

Figura 3 – Interface de edição do DrawDB



Fonte: Autoria própria (2026).

Dessa forma, a primeira versão da integração entre o DrawDB e o WebLLM foi concluída, permitindo a seleção, o carregamento e a utilização de modelos de linguagem diretamente na interface da aplicação.

3.3 Protocolo de execução e avaliação

Primeiramente foram definidos dois prompts, A e B, com o objetivo de avaliar o potencial explicativo e corretivo de cada modelo. Os testes iniciais foram realizados com os modelos *Llama 3.2 1B*, *Qwen 2.5 1,5B* e *Llama 3.2 3B*. Para cada modelo, foram executados os seguintes prompts:

Prompt A:

```
Em um parágrafo, explique a importância da diagramação
antes da modelagem dos dados.
```

Prompt B:

```
Ajuste a sintaxe do SQL a seguir:
    CREATE TABLE users (
      id INT PRIMARY KEY,
      name VARCHAR(50)

    CREATE TABLE posts (
      id INT PRIMARY KEY,
      user_id INT REFERENCES users(id),
      title VARCHAR(100)
    );
Retorne o SQL com a sintaxe corrigida e aponte
qual era o erro.
```

3.3.1 Resultados dos prompts

O **Prompt A** solicitou uma explicação, em um parágrafo, sobre a importância da diagramação antes da modelagem dos dados.

- **Llama 3.2 1B:** *A diagramação é um processo crítico na modelagem de dados, pois permite a visualização e a compreensão de como as diferentes partes de um sistema ou processo funcionam juntas. Antes de implementar a modelagem dos dados, a diagramação ajuda a identificar e clarificar os problemas, as relações entre os dados*

e as variáveis, e as necessidades de inovação. Essa etapa de diagramação também permite que os usuários ou stakeholders identifiquem as áreas de melhoria e as áreas que precisam ser melhoradas, o que pode influenciar a escolha de modelos de dados e a estratégia de inovação. Além disso, a diagramação ajuda a garantir a consistência e a qualidade dos dados, o que é fundamental para a precisão e a confiabilidade dos resultados.

- **Qwen 2.5 1,5B:** *A diagramação é uma fase crucial na engenharia de software, pois serve como um pré-visualizador do projeto final. Ela permite que os desenvolvedores identifiquem possíveis problemas e complexidades antes da modelagem dos dados. A diagramação ajuda a garantir que as estruturas de dados sejam organizadas de forma clara e eficiente, facilitando o entendimento para todos os envolvidos no projeto. Além disso, a diagramação pode ser usada para visualizar a hierarquia das classes e objetos, ajudando a evitar conflitos durante a modelagem dos dados. Em resumo, a diagramação é fundamental para garantir a qualidade e consistência do projeto de software.*

- **Llama 3.2 3B:** *A diagramação é um passo fundamental antes da modelagem dos dados, pois permite que os usuários visualizem e compreendam a estrutura e a relação entre os diferentes componentes de um sistema ou aplicativo. Ao criar diagramas, como fluxogramas, classificações, relações entre dados, entre outros, os usuários podem identificar problemas potenciais, otimizar a eficiência do sistema e garantir que os dados sejam coletados e armazenados de forma eficaz. Além disso, a diagramação ajuda a reduzir erros e ambiguidades, tornando mais fácil a modelagem dos dados e a implementação do sistema.*

O **Prompt B** solicitou a correção de um código SQL com erro de sintaxe e a identificação do problema encontrado.

- **Llama 3.2 1B:** *Aqui está a sintaxe corrigida do SQL:*

```
CREATE TABLE users (
  id INT PRIMARY KEY,
  name VARCHAR(50)
);

CREATE TABLE posts (
  id INT PRIMARY KEY,
  user_id INT,
  title VARCHAR(100)
```

```
10 );
```

O erro era que o `id` do `users` não era um `INT`, mas um `INT` que precisava ser convertido para `INT` antes de ser criado como uma coluna primária.

- **Qwen 2.5 1,5B:** *A sintaxe do SQL está correta. Não há nenhum erro.*
- **Llama 3.2 3B:** *O erro no código SQL fornecido é o faltamento do ponto e vírgula após a definição do campo `name` no primeiro comando SQL.*

```
1  CREATE TABLE users (
2    id INT PRIMARY KEY,
3    name VARCHAR(50)
4  );
5
6  CREATE TABLE posts (
7    id INT PRIMARY KEY,
8    user_id INT REFERENCES users(id),
9    title VARCHAR(100)
10 );
```

O erro original era o faltamento do ponto e vírgula após a definição do campo `name` no primeiro comando SQL. O ponto e vírgula é utilizado para separar comandos SQL em um script.

3.3.2 Avaliação quantitativa

A Tabela 4 apresenta os resultados quantitativos obtidos para cada modelo e prompt.

Tabela 4 – Resultados quantitativos obtidos nos testes iniciais.

Modelo	Prompt	Tokens	Velocidade (tokens/s)	TTFT (s)	Latência total (s)
Llama 3.2 1B	A	182	20,27	0,44	9,87
Llama 3.2 1B	B	102	20,38	0,59	5,73
Qwen 2.5 1,5B	A	151	14,06	3,26	14,24
Qwen 2.5 1,5B	B	19	13,95	0,61	1,93
Llama 3.2 3B	A	148	8,95	5,79	22,49
Llama 3.2 3B	B	138	9,47	1,13	15,82

Fonte: Autoria própria (2026).

Quanto ao tempo de carregamento inicial[2], o Llama 3.2 1B apresentou o menor valor, com 72,68 segundos, seguido pelo Qwen 2.5 1,5B, com 85,3 segundos, e pelo Llama 3.2 3B,

[2] O carregamento inicial refere-se a primeira execução do modelo, em que é feito o download do modelo selecionado. Após a conclusão, o modelo fica armazenado em cache.

com 178,05 segundos. Esses resultados sugerem que o aumento no número de parâmetros está associado a um maior tempo de carregamento e menor velocidade de geração.

3.3.3 Avaliação qualitativa

Baseado nos critérios qualitativos definidos na Tabela 2 da seção 2.2.3, que avaliam a correção técnica, a relevância em relação à solicitação e a coerência na organização das ideias, a Tabela 5 apresenta as pontuações atribuídas às respostas geradas por cada modelo.

Tabela 5 – Resultados da avaliação qualitativa das respostas.

Modelo	Prompt	Correção	Relevância	Coerência
Llama 3.2 1B	A	2	3	2
Llama 3.2 1B	B	1	2	2
Qwen 2.5 1,5B	A	2	3	3
Qwen 2.5 1,5B	B	0	1	2
Llama 3.2 3B	A	3	3	3
Llama 3.2 3B	B	2	3	2

Fonte: Autoria própria (2026).

No primeiro prompt, os modelos produziram respostas relevantes para a solicitação. O modelo *Llama 3.2 3B* apresentou melhor desempenho qualitativo, obtendo as maiores pontuações em correção, relevância e coerência. As respostas dos modelos *Llama 3.2 1B* e *Qwen 2.5 1,5B* foram consideradas parcialmente corretas, com algumas inconsistências conceituais e menor fluidez na organização das ideias.

No segundo prompt, o modelo *Llama 3.2 3B* identificou corretamente a ausência do ponto e vírgula após a definição do campo `name` na primeira tabela e apresentou o código SQL corrigido. Entretanto, a explicação do erro foi repetida, o que reduziu a pontuação de coerência. O modelo *Llama 3.2 1B* corrigiu parcialmente a sintaxe, mas apresentou uma explicação incorreta sobre o erro, resultando em pontuação inferior nos critérios de correção e relevância. Já o modelo *Qwen 2.5 1,5B* identificou, incorretamente, que não havia erros na sintaxe o que resultou na baixa pontuação nos critérios de correção e relevância.

3.3.4 Análise dos resultados

Os resultados iniciais indicam que o modelo Llama 3.2 3B apresentou melhor desempenho qualitativo, na compreensão de instruções e correção de SQL. O Llama 3.2 1B obteve bons resultados quantitativos, com menor latência e maior velocidade de geração. Já o Qwen 2.5 1,5B demonstrou boa capacidade explicativa, mas baixa precisão na correção de SQL.

4 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo integrar o WebLLM ao DrawDB, avaliando o uso de modelos de linguagem diretamente no navegador. Para isso, foram realizadas a identificação e catalogação dos modelos disponíveis, a implementação de uma interface de chat na aplicação e a definição de um protocolo de execução e avaliação.

A utilização do WebLLM permitiu executar modelos de linguagem localmente, por meio de WebGPU, sem a necessidade de serviços em nuvem. Essa característica representa uma vantagem relevante em que evita o envio de dados do usuário a servidores externos e diminui os custos associados ao uso de APIs. Por outro lado, a execução no navegador apresenta limitações relacionadas ao consumo de memória, à capacidade do dispositivo e ao desempenho da inferência, especialmente em modelos com menor número de parâmetros.

Os testes iniciais realizados com os modelos Llama 3.2 1B, Qwen 2.5 1,5B e Llama 3.2 3B indicaram que o aumento no número de parâmetros impacta o desempenho da inferência no navegador. Os modelos com aproximadamente 1B de parâmetros apresentaram maior velocidade de geração e menor latência, enquanto o modelo de 3B produziu respostas com melhor qualidade nas tarefas avaliadas. Esses resultados sugerem um compromisso entre eficiência e qualidade das respostas, que deve ser considerado na seleção dos modelos para uso na aplicação.

Como continuidade do projeto, a ser desenvolvida no TCC 2, pretende-se ampliar os testes para outros modelos, integrar o contexto do diagrama atual às solicitações enviadas e aprimorar o protocolo experimental. Para isso, serão elaborados prompts com diferentes níveis de complexidade, com ênfase para tarefas de geração e correção de consultas SQL, explicação de diagramas e sugestões de modelagem. Também serão revisados os critérios de avaliação qualitativa, como a possibilidade de uma análise comparativa entre modelos comerciais de grande porte com os modelos locais.

REFERÊNCIAS

CAO, K. *et al.* An overview on edge computing research. **IEEE Access**, v. 8, p. 85714–85728, 2020.

CNN Brasil. **OpenAI informa ataque a empresa parceira e potencial vazamento de dados**. 2025. Acessado em: 2026-10-01. Disponível em: https://www.cnnbrasil.com.br/economia/money/negocios/openai-informa-ataque-a-empresa-parceira-e-potencial-vazamento-de-dados/.

drawdb. **Online Database Schema Design & ER Diagram Tool**. 2026. Disponível em: https://www.drawdb.app/. Acesso em: 14 out. 2026.

Google. **Google Privacy Policy**. 2025. Accessed: 2026-10-01. Disponível em: https://policies.google.com/privacy.

GOREY, J. Data Drain: The land and water impacts of the AI boom. **Land Lines Magazine**, Lincoln Institute of Land Policy, oct 2025. Accessed: 2026-10-01. Disponível em: https://www.lincolninst.edu/publications/land-lines-magazine/articles/land-water-impacts-data-centers/.

LANG, J.; GUO, Z.; HUANG, S. A comprehensive study on quantization techniques for large language models. **arXiv preprint arXiv:2411.02530**, 2024. Acesso em: 29 out. 2026. Disponível em: https://arxiv.org/abs/2411.02530.

MLC AI. **WebLLM: Using Workers**. 2023. Disponível em: https://webllm.mlc.ai/docs/user/advanced_usage.html#using-workers.

MLC AI. **web-llm: High-performance in-browser LLM inference engine**. 2026. GitHub. Disponível em: https://github.com/mlc-ai/web-llm/blob/main/src/config.ts. Acesso em: 22 out. 2026.

OpenAI. **OpenAI Terms of Use**. 2025. Accessed: 2026-10-01. Disponível em: https://openai.com/policies/terms-of-use.

QU, G. *et al.* Mobile edge intelligence for Large Language Models: A contemporary survey. **IEEE Communications Surveys & Tutorials**, v. 27, n. 6, p. 3820–3860, dez. 2025.

RUAN, C. F. e. a. Webllm: A high-performance in-browser llm inference engine. **arXiv preprint arXiv:2412.15803**, 2026.

SCHWAB, K. **The Fourth Industrial Revolution**. New York: Crown Business, 2016.