



**Matheus Moraes Ferreira**

**Assistente Virtual utilizando Transformers  
Generativos Pré-treinados no Contexto de  
Gerenciamento de Reservatórios**

**Dissertação de Mestrado**

Dissertação apresentada como requisito parcial para a obtenção  
do grau de Mestre pelo Programa de Pós-graduação em Infor-  
mática da PUC-Rio.

Orientador : Prof. Alberto Barbosa Raposo  
Co-orientador: Dr. Paulo Roberto da Motta Pires

Rio de Janeiro  
Outubro de 2023



**Matheus Moraes Ferreira**

**Assistente Virtual utilizando Transformers  
Generativos Pré-treinados no Contexto de  
Gerenciamento de Reservatórios**

Dissertação apresentada como requisito parcial para a obtenção  
do grau de Mestre pelo Programa de Pós-graduação em In-  
formática da PUC-Rio. Aprovada pela Comissão Examinadora  
abaixo.

**Prof. Alberto Barbosa Raposo**  
Orientador  
Departamento de Informática – PUC-Rio

**Dr. Paulo Roberto da Motta Pires**  
Co-orientador  
Petróleo Brasileiro S.A. – Petrobras

**Prof. Hélio Côrtes Vieira Lopes**  
Departamento de Informática – PUC-Rio

**Profª. Melissa Lemos Cavaliéri**  
Departamento de Informática – PUC-Rio

Rio de Janeiro, 02 de Outubro de 2023

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador.

### **Matheus Moraes Ferreira**

Matheus Moraes Ferreira, bacharel em Ciência da Computação pela Universidade Federal Rural do Rio de Janeiro (2020).

#### **Ficha Catalográfica**

Moraes Ferreira, Matheus

Assistente Virtual utilizando Transformers Generativos Pré-treinados no Contexto de Gerenciamento de Reservatórios / Matheus Moraes Ferreira; orientador: Alberto Barbosa Raposo; co-orientador: Paulo Roberto da Motta Pires. – Rio de Janeiro: PUC-Rio, Departamento de Informática, 2023.

v., 53 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Processamento de linguagem natural;. 3. Aprendizagem de máquina;. 4. Large Language Models;. 5. GPT;. 6. Assistente virtual inteligente.. I. Barbosa Raposo, Alberto. II. Pires, Paulo Roberto da Motta. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

## **Agradecimentos**

Gostaria de agradecer primeiramente a Deus, por ter permitido que eu chegasse aqui e defendesse este trabalho. Em seguida gostaria de agradecer aos meus pais e minha família, por sempre ter me apoiado e nunca ter deixado faltar nada. Também gostaria de agradecer à minha companheira Helena Birschner, uma pessoa incrível, que sempre esteve comigo me apoiando e me ajudando, principalmente durante esse processo. Gostaria de agradecer ao meu orientador, Prof. Alberto Barbosa Raposo, pelo acompanhamento e atenção durante todo esse processo e a ajuda fornecida para conclusão desse trabalho. Gostaria de agradecer a banca por ter aceitado o convite e ter contribuído construtiva e positivamente com melhorias para esse trabalho. Gostaria de agradecer também ao Paulo Pires, meu co-orientador, por todo o apoio e confiança no projeto e por toda a ajuda, tanto profissional, técnica e pessoal, durante todo o processo, se mostrando uma pessoa incrível e um grande amigo. É muito importante agradecer a PUC-Rio e ao Tecgraf por toda as oportunidades de crescimento pessoal, profissional e acadêmico. Gostaria de agradecer também Luiz Elerati (meu líder de projeto), e Leandro Nazareth (meu gerente), por terem proporcionado todo apoio e confiança para início, desenvolvimento e finalização deste trabalho, forneceram toda a ajuda possível. Fernanda Gramorelli (Petrobras), Jacilene Torres (TIC/Petrobras), Rodrigo Vitória Lopes (TIC/Petrobras), Patricia Ferreira (CENPES/Petrobras) por também acreditarem no projeto e terem fornecido todos os meios para a conclusão do projeto e também a Petrobras por ter acreditado e viabilizado a execução de toda a pesquisa aqui realizada. Gostaria de agradecer também a Fabio Souza, Roberta Netto e Karin Santos por terem fornecido detalhes valiosos do sistema alvo e por terem ajudado a escrever o conjunto de treinamento, o que foi muito valioso para garantir a representatividade do conjunto. Gostaria de agradecer a equipe de desenvolvimento que garantiu um sistema bem construído e fácil de trabalhar. Por último e não menos importante, gostaria de agradecer aos meus amigos que sempre me apoiaram e de uma forma ou de outra tiveram alguma contribuição positiva na minha vida.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

## Resumo

Moraes Ferreira, Matheus; Barbosa Raposo, Alberto; Pires, Paulo Roberto da Motta. **Assistente Virtual utilizando Transformers Generativos Pré-treinados no Contexto de Gerenciamento de Reservatórios**. Rio de Janeiro, 2023. 53p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Com a crescente popularização das técnicas de Inteligência Artificial, principalmente voltadas ao processamento de linguagem natural, testemunhamos um notável avanço nos Large Language Models (modelos de linguagem avançados), dos quais o *Generative Pre-trained Transformer* (GPT) consiste no exemplo mais notável. Consequentemente, assistentes virtuais têm conquistado uma presença significativa em diversas áreas da vida contemporânea. Neste trabalho, é proposta uma metodologia para desenvolver uma assistente virtual inteligente, baseada em um modelo gerador, capaz de compreender a língua portuguesa do Brasil, bem como o domínio específico da Indústria de Óleo e Gás. Essa assistente tem a capacidade de interpretar comandos textuais fornecidos pelos usuários e executar ações correspondentes em um sistema corporativo. Essa metodologia é o resultado de uma cuidadosa análise de diferentes modelos generativos disponíveis, buscando identificar aquele que melhor se adequa aos requisitos da assistente virtual inteligente em português. Para treinamento é criado um *dataset* representativo com os conceitos necessários e específicos do sistema e da indústria do petróleo. É adotado um processo de refinamento que permite identificar eventuais falhas e aperfeiçoar a compreensão da assistente para garantir respostas precisas e direcionadas. Também são abordados neste trabalho os desafios e limitações inerentes aos modelos gerativos, bem como estratégias para superá-las a fim de obter gerações mais precisas e seguras.

## Palavras-chave

Processamento de linguagem natural; Aprendizagem de máquina; Large Language Models; GPT; Assistente virtual inteligente.

## Abstract

Moraes Ferreira, Matheus; Barbosa Raposo, Alberto (Advisor); Pires, Paulo Roberto da Motta (Co-Advisor). **Virtual assistant using pre-trained generative transformers in the context of reservoir management.** Rio de Janeiro, 2023. 53p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

With the growing popularity of Artificial Intelligence, specially related to Natural Language Processing, we notice a remarkable development of Large Language Models, which finds in the Generative Pre-Trained Transformers (GPT) their most outstanding example. As a result, virtual assistants have been gaining significant presence in various areas of modern life. In this work, we present the development of an intelligent virtual assistant, based on a generative model. The assistant understands Brazilian Portuguese and is trained on the specific jargon of the Oil and Gas Industry. This assistant has the ability to interpret textual commands provided by users and execute corresponding actions within a corporate system. This methodology is the result of a careful analysis of different available generative models, aiming to identify the one that best suited the requirements of an intelligent virtual assistant in Portuguese. Additionally, it involves the creation of a representative dataset, with concepts specific to the system and the Oil and Gas Industry, to effectively train the assistant. A refinement process allows the identification of potential flaws and the improvement of the assistant's understanding to ensure accurate and targeted responses. Furthermore, this work presents the challenges and the inherent limitations of generative models, and proposes strategies to overcome them in order to achieve more precise and secure generations.

## Keywords

Natural language processing; Machine Learning; Large Language Models; GPT; Virtual intelligent assistant.

# Sumário

1	Introdução	11
1.1	Motivação	12
1.2	Questões e objetivos de pesquisa	12
1.3	Estrutura do trabalho	13
2	Contextualização e trabalhos relacionados	14
3	Metodologia	21
3.1	Estrutura do sistema alvo	21
3.2	Modelo pré-treinado	22
3.3	Fine-tuning e training data	22
3.4	Restrição das respostas	25
3.5	Tokenização	25
3.6	Escopo e treinamento	27
4	Resultados	28
4.1	Treinamento	28
4.2	Caso 1	29
4.3	Caso 2	30
4.4	Caso 3	31
4.5	Caso 4	32
4.6	Caso 5	33
4.7	Resumo de resultados	34
4.8	Fuga do contexto	35
4.9	Comandos incompletos	37
4.10	Limites de compreensão do modelo	38
5	Conclusões e trabalhos futuros	43
5.1	Principais contribuições	44
5.2	Limitações	44
5.3	Trabalhos futuros	45
6	Referências bibliográficas	47
A	Anexos	50
A.1	Resultados por Caso	50

## **Lista de figuras**

Figura 2.1	Etapas principais de processamento dos <i>Transformers</i> . <b>Reprodução:</b> Hugging Face [24]	14
Figura 2.2	Funcionamento das cabeças de atenção em um processo de tradução. <b>Reprodução:</b> TensorFlow [25]	16
Figura 2.3	Derivações da tecnologia dos <i>Transformers</i> . <b>Reprodução:</b> Internet [25]	17
Figura 2.4	Ilustração do <i>fine-tuning</i> para tarefas específicas. Na esquerda é apresentado o modelo pré-treinado com um <i>dataset</i> genérico; na direita é representado o seu <i>fine-tuning</i> para três diferentes tarefas (MNLI, NER, SquAD). <b>Reprodução:</b> BERT [3]	18
Figura 3.1	Estrutura do sistema	21
Figura 4.1	<i>Loss</i> do treinamento	29
Figura 4.2	Caixa de seleção para complementação de comando requisitada pela assistente no sistema alvo.	38
Figura A.1	Resultado Caso 1	50
Figura A.2	Resultado Caso 2	51
Figura A.3	Resultado Caso 3	51
Figura A.4	Resultado Caso 4	52
Figura A.5	Resultado Caso 5	53

## **Lista de tabelas**

Tabela 3.1	Hiperparâmetros de treinamento	27
Tabela 4.1	Recursos computacionais e tempo para treinamento.	28
Tabela 4.2	Similaridade entre as respostas observadas e o <i>ground truth</i> .	35
Tabela 4.3	Ilustração de alguns comandos inválidos	36

## **Lista de siglas**

IA – Inteligência Artificial

PLN – Processamento de Linguagem Natural

IHM – Interação Humano-Máquina

E&P – Exploração e Produção

BERT – *Bidirectional Encoder Representations from Transformers*

GPT – *Generative Pre-Trained Transformer*

LLM – *Large Language Models*

RAIL – *Responsible AI Licenses*

BLOOM – *BigScience Large Open-science Open-access Multilingual*

API – *Application Programming Interface*

JSON – *JavaScript Object Notation*

ANP – Agência Nacional do Petróleo, Gás Natural e Biocombustíveis

# 1

## Introdução

Na Ciência de Dados e Inteligência Artificial (IA), a linguagem natural (ou humana) recebe representações matemáticas de modo a se tornar codificável e manipulável através de sistemas computacionais. Essas representações se dão através dos chamados **modelos de linguagem**. Esses modelos usam aprendizagem de máquina para recriar o comportamento da linguagem natural. Deste modo, assimilando a correlação entre palavras, tornam-se aptos a fazer previsões sobre a sequências que compõem frases, a traduzir um texto ou, até mesmo, a responder a perguntas de forma inteligente.

Atualmente, os modelos de linguagem vêm ganhando notoriedade devido ao rápido avanço na capacidade de lidar com ambiguidades da linguagem humana e na compreensão de contextos complexos. Tornam-se cada vez mais proficientes na interpretação de sutilezas emocionais, humor e sarcasmo, oferecendo respostas naturais e apropriadas ao contexto.

O acelerado desenvolvimento desses modelos abre importantes possibilidades para o Processamento de Linguagem Natural (PLN) e para a IA de modo geral. Com avanços em áreas como a compreensão de linguagem humana, geração de texto, tradução automática e reconhecimento de fala, os modelos de linguagem têm se mostrado cada vez mais capazes de imitar e até mesmo superar as habilidades humanas em algumas tarefas relacionadas à linguagem.

Isso tem implicações significativas para uma variedade de campos, desde recursos para Interação Humano-Computador (IHC) — por exemplo, assistentes virtuais e *chatbots* — até análise de sentimentos em redes sociais e na criação de conteúdo automatizado.

No entanto, os modelos de linguagem também apresentam limitações e impõe desafios [22]. Sua efetividade se mostra muito sensível à qualidade e à representatividade dos dados usados para o treinamento [23]. Da mesma forma, as respostas fornecidas pelo modelo dependem da precisão e confiabilidade das instruções fornecidas pelos usuários. Isso traz implicações relevantes. No aspecto técnico, os modelos podem encontrar dificuldades para adaptação a contextos específicos ou muito especializados [10, 11]. No aspecto ético e moral, podem acabar refletindo desigualdades ou preconceitos sociais presentes no conjunto de dados utilizado para treinamento [20].

## 1.1

### Motivação

A Indústria de Óleo e Gás apresenta uma complexa cadeia de valor, que abrange uma ampla gama de processos e operações. Desde o segmento de Exploração e Produção (E&P), também conhecido como *upstream*, até o Refino e Abastecimento, ou *downstream*, uma infinidade de dados e informações são coletadas das mais diversas fontes e precisam ser processadas e interpretadas.

Na Engenharia de Petróleo, em particular na disciplina de Reservatórios, há a necessidade de análise e interpretação de dados e informações de diferentes origens, formatos, representatividade e qualidade [27, 29]. *Datasets* extensos, compostos por dados sísmicos, informações de perfuração de poços, registros da produção e estudos de reservatórios, tornam a vida de geocientistas e engenheiros de petróleo desafiadora. Isso exige não apenas o conhecimento amplo acerca de diversos processos, mas também a familiaridade com sistemas computacionais especializados, com avançadas interfaces.

Inspirado pela popularização recente dos *chatbots* movidos por derivados do modelo *Generative Pre-trained Transformer* (GPT) [2], o desenvolvimento de uma assistente virtual capaz de realizar tarefas complexas e específicas em um sistema corporativo se apresenta como uma solução atrativa e eficiente para auxiliar as equipes de Reservatórios nas suas tarefas diárias.

## 1.2

### Questões e objetivos de pesquisa

A presente pesquisa tem por objetivo endereçar as seguintes questões:

1. É possível incorporar uma assistente virtual à interface complexa de um sistema corporativo pré-existente?
2. Caso afirmativo, como fazer isso e quais tecnologias usar?
3. Como contornar limitações das estratégias escolhidas?

Portanto, o objetivo principal deste trabalho é propor uma metodologia para a construção de uma assistente virtual inteligente capaz de receber instruções textuais em português do Brasil e de controlar um sistema corporativo, fornecendo instruções estruturadas a partir da entrada recebida. Além disso, são discutidas algumas limitações apresentadas pelos modelos de linguagem treinados em contextos genéricos, e sugeridas estratégias para contorná-las.

O sistema corporativo<sup>1</sup>, alvo para a assistente virtual, realiza a integração de múltiplas bases de dados especializadas, permitindo a visualização e análises avançadas de informações essenciais à rotina das equipes de Reservatórios.

O papel da assistente aqui proposta é aprimorar a experiência do usuário, aumentando sua produtividade. Para tanto, ela interpreta comandos em linguagem natural e, a partir deles, aciona as funcionalidades requeridas pelo usuário sem exigir dele o domínio completo da interface convencional do sistema.

Na literatura, não são comuns exemplos de assistentes virtuais baseados na família GPT. Modelos adaptados para o contexto da Indústria de Óleo e Gás, em português do Brasil, são mais escassos ainda. Essa, portanto, é uma importante contribuição deste trabalho.

### **1.3 Estrutura do trabalho**

Os capítulos subsequentes desta dissertação estão organizados da seguinte forma:

- O capítulo 2 traz uma revisão bibliográfica que contextualiza a técnica aqui pesquisada e expõe a lacuna bibliográfica que se pretende endereçar.
- O capítulo 3 descreve, em linhas gerais, o sistema corporativo alvo e a metodologia adotada para implementação e treinamento da assistente virtual.
- O capítulo 4 apresenta os resultados do treinamento e dos testes realizados com a assistente.
- O capítulo 5 encerra as conclusões e discussões acerca de trabalhos futuros a partir dos resultados aqui obtidos.

<sup>1</sup>Por questões de confidencialidade e proteção da propriedade intelectual, detalhes do sistema alvo são omitidos no texto.

## 2

### Contextualização e trabalhos relacionados

Desde sua introdução [1], a tecnologia dos *Transformers* tem permitido o desenvolvimento de modelos cada vez mais avançados de PLN, capazes de entender, gerar e manipular a linguagem humana com uma precisão e fluência progressivamente maiores.

Uma característica fundamental dos *Transformers* é sua independência de uma estrutura sequencial, o que lhes permite, diferente de outras arquiteturas, lidar com entradas de comprimentos variáveis.

O funcionamento de um modelo *Transformers* geralmente é dividido em duas etapas principais: a etapa de codificação (*encoding*) e a etapa de decodificação (*decoding*). A figura 2.1 ilustra tais etapas.

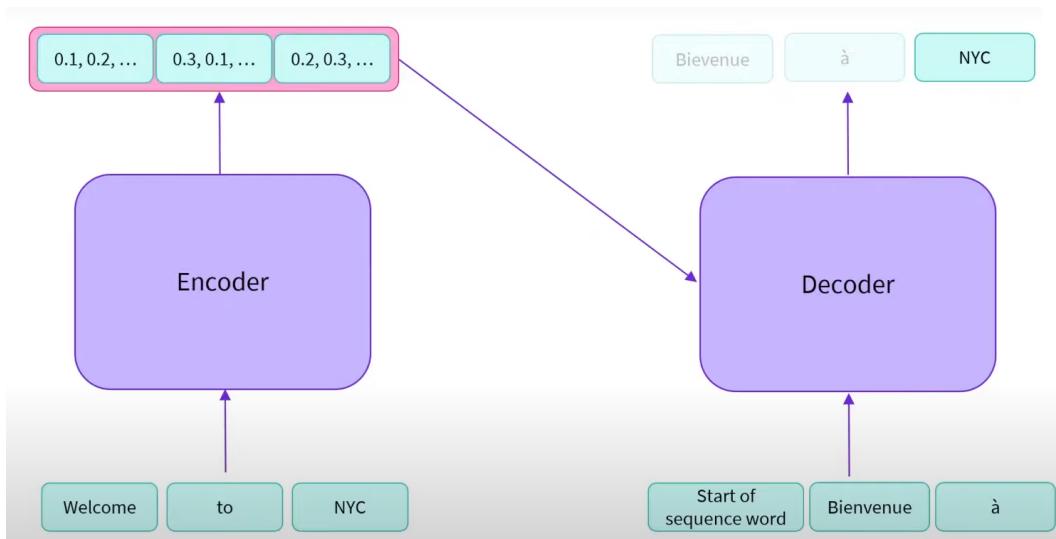


Figura 2.1: Etapas principais de processamento dos *Transformers*. **Reprodução:** Hugging Face [24]

Na codificação, o texto fornecido como entrada para o *Transformers* é transformado em um conjunto de vetores que representam as informações contidas nessa entrada. Esses vetores são chamados de vetores de codificação ou *embeddings*. Uma vez que codifica palavras, a técnica recebe o nome de *word embedding*. Para realizar a codificação, o *Transformers* utiliza vários estágios de **atenção**.

A função do mecanismo de atenção é focar o modelo nas partes relevantes da entrada. Isso possibilita ao modelo contextualizar cada palavra em relação às outras presentes, tornando possível o “entendimento” do seu significado no contexto em que ela se encontra.

Para isso, o mecanismo de atenção utiliza ‘cabeças de atenção’ (*attention heads*) que inferem individualmente o significado de cada palavra. Os significados sugeridos por cada cabeça são confrontados para se alcançar um consenso sobre a relação de cada palavra com as outras palavras presentes na entrada. Deste modo, maximizam a probabilidade da palavra ser interpretada no contexto correto.

É possível notar esse comportamento na figura 2.2. São apresentadas oito cabeças de atenção especializadas na tradução do inglês para o português. Cada uma delas corresponde a um quadro na figura. Nos quadros, são representadas as palavras de entrada em inglês e as palavras de saída esperada em português. Cada posição da matriz representa a magnitude da correlação entre as palavras de entrada e saída; magnitudes maiores apresentam uma coloração mais amarelada, magnitudes menores tendem ao azul escuro. Na prática, cada quadro representa uma matriz de correlação construída por cada cabeça de atenção a partir do conjunto de treinamento.

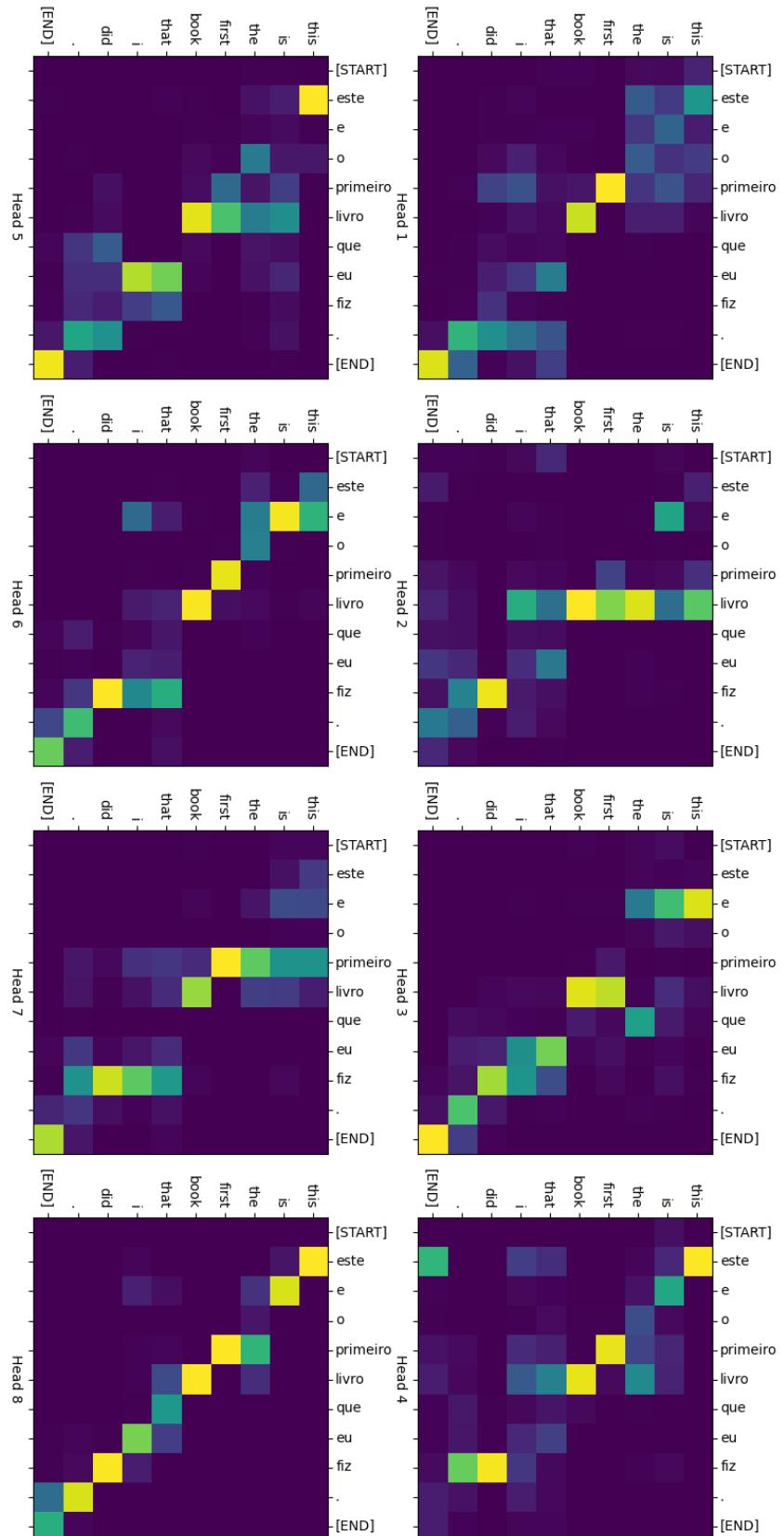


Figura 2.2: Funcionamento das cabeças de atenção em um processo de tradução. **Reprodução:** TensorFlow [25]

Na etapa de decodificação, o modelo constrói a saída palavra por palavra, usando um *token* especial de início de sequência como entrada inicial ([START], na figura 2.2). A cada estágio de geração, o modelo utiliza o mecanismo de atenção para identificar e salientar as partes relevantes da entrada e saída parcial. A saída parcial fornecida por cada estágio é representada por um conjunto de *word embeddings*, progressivamente atualizados pelos estágios anteriores.

Derivados da arquitetura dos *Transformers* surgiram os *Large Language Models* (LLMs), que estão entre as tecnologias de IA mais sofisticadas dos últimos anos. Trata-se de modelos treinados em grandes conjuntos de dados de texto — ou **corpus** (plural **corpora**) — e podem gerar texto coerente, responder perguntas e até mesmo realizar tarefas criativas, como compor poesia ou escrever histórias.

Dentre os LLMs mais famosos estão o GPT [2], da OpenAI<sup>1</sup>, e o BERT (*Bidirectional Encoder Representations from Transformers*) [3], da Google<sup>2</sup>. Desde então começaram a surgir variantes do GPT [5, 6, 7], que são consideradas o estado da arte na geração de texto. Por sua vez, as variantes do BERT [15, 16, 17] se revelam eficientes para identificação de entidades nomeadas, análise de sentimentos e preenchimento de máscaras. A figura 2.3 ilustra as principais variantes dos modelos BERT e GPT.

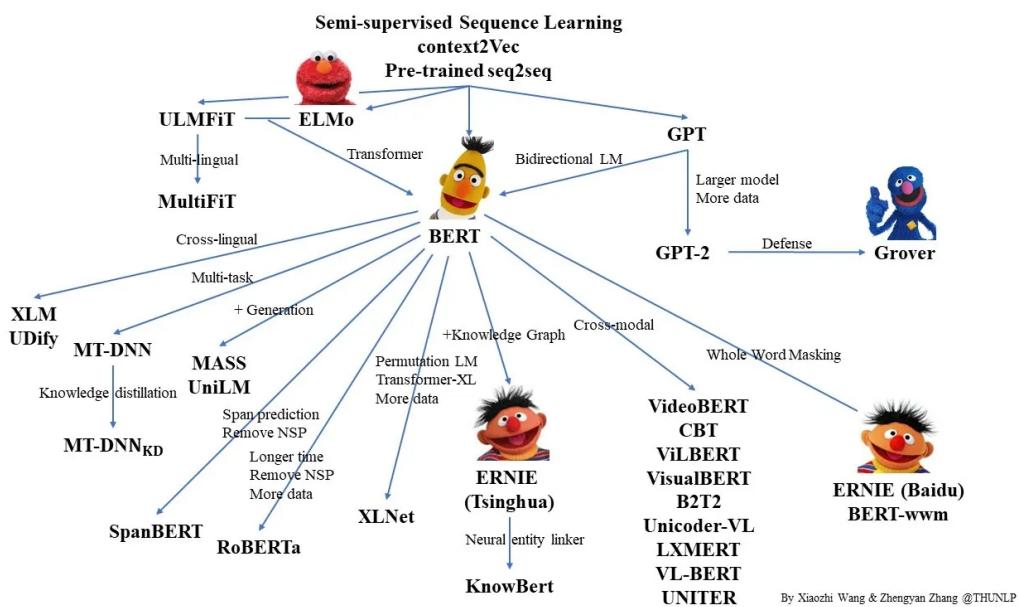


Figura 2.3: Derivações da tecnologia dos *Transformers*. **Reprodução:** Internet [25]

<sup>1</sup><https://openai.com/>

<sup>2</sup><https://www.google.com/>

Tanto o GPT quanto o BERT têm sido amplamente usados em aplicações como *chatbots*, geração textual e análise de sentimentos. Eles são capazes de entender a linguagem natural em um nível mais profundo do que os sistemas anteriores, baseados em redes recorrentes, e produzem saídas de texto que parecem ter sido escritas por um ser humano.

Embora sejam considerados o estado da arte atualmente, os LLMs também apresentam desafios. Por exemplo, eles são extremamente sensíveis e dependentes dos dados com os quais são treinados. Esses dados podem reproduzir viéses, suscitando inclusive questões éticas e morais [20]. Além disso, treinados com dados genéricos, os LLMs comumente encontram dificuldades em entender termos técnicos, jargões ou gírias [23]. Isso pode levar a respostas imprecisas caso o modelo seja usado em contextos específicos ou especializados.

Para contornar esses problemas, uma abordagem promissora é treinar modelos em conjuntos de dados mais amplos e diversos e, em seguida, aplicar uma técnica conhecida como *fine-tuning*, que permite que os modelos aprendam padrões mais gerais antes de serem ajustados para padrões mais específicos [10, 11]. Isso pode levar a modelos mais robustos, capazes de lidar com diferentes situações e de entender as nuances do contexto específico abordado no *fine-tuning* (figura 2.4).

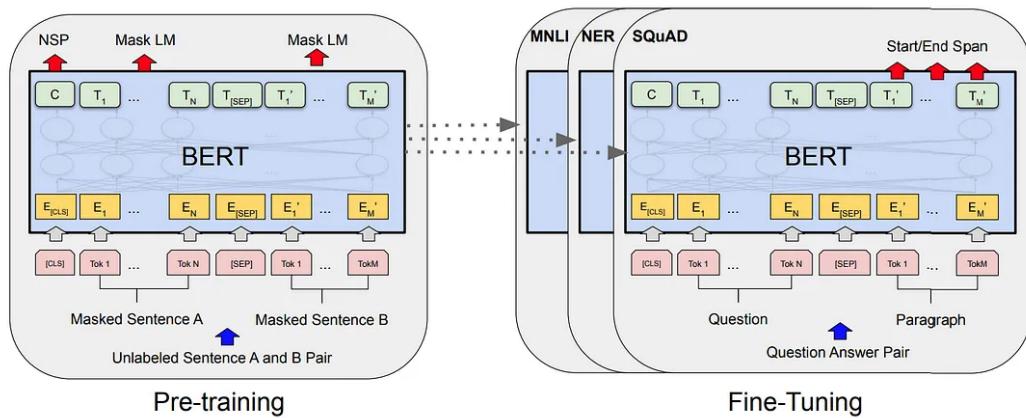


Figura 2.4: Ilustração do *fine-tuning* para tarefas específicas. Na esquerda é apresentado o modelo pré-treinado com um *dataset* genérico; na direita é representado o seu *fine-tuning* para três diferentes tarefas (MNLI, NER, SQuAD). **Reprodução:** BERT [3]

Com a crescente popularização dos LLMs, começaram a surgir diversos modelos que foram submetidos a um *fine-tuning* para o português do Brasil, permitindo que essa tecnologia de PLN seja aplicada de forma mais eficiente e precisa para os usuários brasileiros, reduzindo os efeitos dos efeitos dos viéses de treinamento (*training biases*).

Um deles é o BERTimbau [12], um modelo que a partir do BERT recebeu o *fine-tuning* para o português. Seu objetivo é melhorar o desempenho de tarefas específicas de PLN em português, como classificação de texto, análise de sentimentos e reconhecimento de entidades nomeadas (por exemplo, nomes de pessoas, objetos, lugares, datas, etc).

Por sua vez, o PetroBERT [13] é uma adaptação do BERTimbau feita para o contexto da Indústria de Óleo e Gás. Treinado a partir de dados específicos desse setor, como relatórios técnicos, especificações de equipamentos e documentos regulatórios, esse modelo se mostra mais preciso e eficaz na realização de tarefas relacionadas ao segmento.

O BERT e seus derivados se diferem substancialmente dos modelos geradores, como o GPT, em sua principal função e abordagem. Enquanto os modelos geradores são projetados para criar texto novo e fluente, o BERT tem como foco principal a compreensão contextual bidirecional das palavras em um texto. Em vez de gerar texto, o BERT codifica o texto de entrada em representações contextualizadas, tornando-se particularmente valioso para tarefas de processamento de linguagem natural como classificação de texto, preenchimento de lacunas e extração de informações.

O contexto de assistentes virtuais não é algo novo. A idéia de ter um *software* que consiga receber e interpretar comandos, sejam eles textuais ou por voz, existe há um tempo, muito antes da introdução da arquitetura dos *Transformers*. Destacam-se grandes exemplos de tecnologia como a *Siri* da Apple<sup>3</sup> e a *Alexa* da Amazon<sup>4</sup>. Esses assistentes virtuais revolucionaram a forma como interagimos com nossos dispositivos e ambientes domésticos.

Embora essa tecnologia já exista, ela ainda apresenta a limitação de se comunicar exclusivamente com o sistema para a qual ela foi projetada, o que dificulta o seu reúso em outros sistemas e aplicações. Por exemplo, ao tentar acoplar uma assistente dessas em um sistema muito específico, como o da Indústria de Óleo e Gás, e tentar solicitar a execução de um comando, é muito provável que a assistente não saiba responder pois ela não foi apresentada a esse contexto de forma suficientemente representativa para realizar interpretação e execução corretamente.

A lacuna endereçada pela presente pesquisa corresponde à criação de uma assistente virtual inteligente, em português do Brasil, construída a partir de modelos generativos, que trate algumas limitações inerentes a tecnologia dos *Transformers*, principalmente quando aplicada a um contexto específico, como a Engenharia e Gerenciamento de Reservatórios [27]. Tal assistente também

<sup>3</sup><https://www.apple.com/br/siri/>

<sup>4</sup><https://www.amazon.com.br/b?ie=UTF8&node=19949683011>

deve ser confiável para incorporação em um sistema corporativo especializado.

A pesquisa bibliográfica aqui realizada sugere que referências para o propósito aqui apresentado são escassas. Não foram encontrados trabalhos semelhantes que abordassem tal tema de pesquisa.

### 3

## Metodologia

Neste capítulo serão descritos o modelo proposto e o *dataset* utilizado para treinamento e validação.

### 3.1

#### Estrutura do sistema alvo

O sistema alvo corresponde a uma ferramenta avançada voltada a uma disciplina específica da engenharia de petróleo, a saber, gerenciamento de reservatórios. A figura 3.1 ilustra em linhas gerais a estrutura desse sistema. Os elementos da interface de usuário refletem em grande parte as regras de negócio e as técnicas de processamento e integração de dados que o sistema realiza. Assim, é necessário que o usuário conheça essas regras e técnicas para poder explorar toda a potencialidade do sistema.

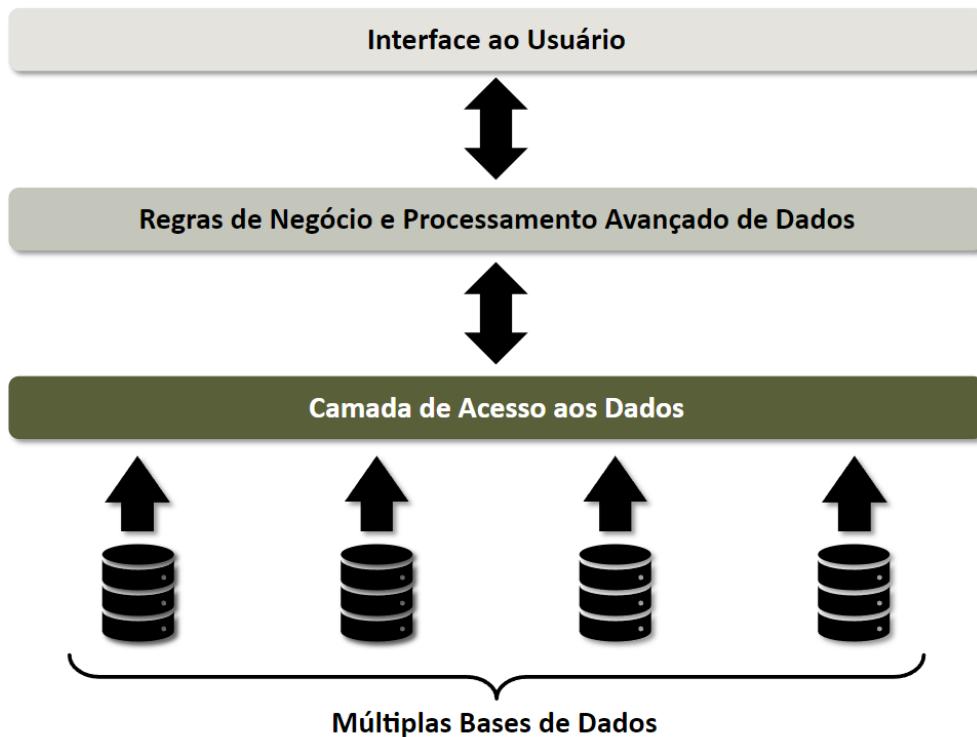


Figura 3.1: Estrutura do sistema

A assistente virtual é projetada para se integrar à interface do sistema, oferecendo uma camada adicional de interação e assistência. Através da sua capacidade de compreender comandos em linguagem natural, a assistente

interage com o sistema realizando as tarefas requeridas pelo usuário. Deste modo, ela livra o usuário dos detalhes de interface e da lógica de negócio, permitindo que ele se concentre somente na visualização e análise dos dados, abstraindo do passo a passo convencional necessário para chegar até eles.

### 3.2

#### Modelo pré-treinado

Como mencionado, os LLMs são extremamente sensíveis ao conjunto de dados com que são treinados. Por isso, para se ter um modelo que seja capaz de entender o português brasileiro, primeiro seria necessário expor tal modelo a um corpus bem construído em português. Essa tarefa, se realizada do zero demandaria enorme esforço, requerendo a criação de um *dataset* representativo com o maior número de palavras possível. Além disso, o tempo necessário para treinamento integral desse modelo seria impraticável no contexto do presente trabalho.

Portanto, com objetivo de evitar esses problemas, é utilizado aqui um modelo pré-treinado. O modelo BLOOM [8] é um LLM treinado em diversas linguagens naturais, incluindo o português do Brasil. Já foi validado e sua eficiência é comprovada [8, 9]. A distribuição desse modelo segue as *Responsible AI Licenses* (RAIL) [14], que garante o uso gratuito e consciente do modelo para fins comerciais e acadêmicos.

Esse modelo foi treinado nos moldes do GPT-3 [6] e possui 176 bilhões de parâmetros, o que assegura uma gigantesca capacidade de entendimento das palavras, seus respectivos contextos e significados. Um parâmetro consiste basicamente em um peso que será responsável por aprender alguma característica importante do problema que a rede tem como objectivo. No caso de LLMs, um parâmetro pode codificar o significado de uma palavra ou a relação de uma palavra com outra.

O modelo BLOOM possui algumas versões menores, como é o exemplo do BLOOM 1b1, que é o modelo mais leve, com 1,1 bilhão de parâmetros. Por motivos de limitação de poder computacional, a assistente aqui proposta utiliza essa versão, que apresenta uma significativa redução da quantidade dos parâmetros, mas em testes realizados, ainda consegue manter a capacidade de entender um grande número de sentenças.

### 3.3

#### Fine-tuning e training data

Sobre o modelo BLOOM 1b1, é aplicado um *fine-tuning*. O objetivo é adaptar e estender o modelo para uma tarefa específica e melhorar sua

capacidade de generalização para dados novos e desconhecidos. Além disso, o *fine-tuning* ajuda a reduzir o tempo e os recursos computacionais necessários, em comparação ao treinamento completo de um novo modelo a partir do zero.

Portanto, essa opção se revela rápida, prática e adequada para a prova de conceito que se busca nesta pesquisa. O *fine-tuning*, especificamente para este trabalho, é feito a partir dos termos e conceitos adotados pelo sistema alvo, ao qual a assistente virtual está acoplada. O processo de ajuste capacita o modelo a criar um mapeamento entre instruções expressas em linguagem natural e os comandos internos estruturados no sistema. Esses comandos representam instruções que existem em uma API (*Application Programming Interface*) dentro do sistema alvo, que a assistente utiliza para executar as instruções internamente.

Em PLN, problemas específicos podem esbarrar na indisponibilidade *datasets* prontos que atendam seus requisitos. Além disso, a própria natureza desses problemas demanda que sejam utilizados dados em formatos específicos, diferentes dos utilizados em *datasets* públicos. Portanto, para realizar o *fine-tuning* do modelo, é necessário criar um *dataset* contendo as características de uso e das regras de negócio do sistema alvo. A partir do *dataset* utilizado no *fine-tuning*, o treinamento habilita a assistente a efetuar a tradução de uma instrução em linguagem natural para um comando equivalente interno ao sistema.

Esse *dataset* é construído a partir de um conjunto de sentenças, compostas cada uma por:

1. Uma instrução textual indicando alguma tarefa desejada pelo usuário. Por exemplo,

```
Abrir relatório
Criar relatório
Criar um gráfico
Plotar gráfico
```

- 2.** Um comando do sistema alvo associado a instrução previamente descrita em **1**. Os comandos internos do sistema são modelados usando o formato JSON (*JavaScript Object Notation*). O objetivo disso é voltar o mecanismo de atenção para as combinações de palavras e assim ensinar o modelo a construir o JSON se baseando na instrução em linguagem natural fornecida como entrada.

```
{action: "open", target: "relatorio"}  
{action: "open", target: "relatorio"}  
{action: "open", target: "grafico"}  
{action: "open", target: "grafico"}
```

- 3.** Um *token* delimitador indicando o fim da sentença de instrução **1** e início do comando do sistema **2**. A assistente foi treinada para reconhecer o símbolo ‘:’ como delimitador, entre o texto fornecido pelo usuário e o comando correspondente.

<Texto (1)>: <Comando (2)>

A combinação dessas três regras, resulta em exemplos para treinamento com o seguinte formato.

```
Abrir relatório: {action: "open", target: "relatorio"}  
Criar relatório: {action: "open", target: "relatorio"}  
Criar um gráfico: {action: "open", target: "grafico"}  
Plotar gráfico: {action: "open", target: "grafico"}
```

Para a geração do *dataset* de treinamento, é utilizado *script* em Python para construir exemplos, seguindo essas regras de forma automatizada. Esse *script* se baseia nas regras de negócio do sistema alvo, combinando-as de forma a criar exemplos representativos para o escopo definido para a assistente.

Ao todo foram gerados 212 mil exemplos distintos para treinamento e a metade disso, 106 mil, para teste. Não há interseção entre os *datasets* para treinamento e para teste, ou seja, todos os exemplos do *dataset* de teste são distintos de todos os exemplos de treinamento.

### 3.4

#### Restrição das respostas

Uma vez que são muito sensíveis ao *dataset* com que são treinados, os LLMs podem gerar artefatos quando submetidos a contextos muito específicos ou não previstos no treinamento.

Essa feição dos LLMs pode resultar em comportamentos até mesmo polêmicos [20]. Por exemplo, se para o treinamento do modelo foi usado um conjunto de dados que contém mais exemplos de homens ocupando cargos de liderança do que mulheres, isso pode levar a resultados indesejados e enviesados quando o modelo é usado para gerar texto ou responder a perguntas sobre liderança. O modelo treinado tenderá a associar mais frequentemente a liderança com os homens do que com as mulheres.

Para contornar esse problema e aumentar a precisão nas respostas, o mapeamento entre as instruções em linguagem natural e comandos do sistema carregarão dois delimitadores distintos, um marcando o início ('#') e outro o final ('\$') da instrução:

```
# <sentença> $
```

Pelo próprio funcionamento interno dos modelos, a geração dos *tokens* — símbolos ou palavras que compõem o texto gerado — de saída é um processo contínuo que ocorre indefinidamente. Na sequência de saída a geração do *token* corrente depende apenas dos *tokens* previamente gerados. Para estabelecer um critério de parada para o processo, a saída é limitada a um tamanho predefinido, ou seja, um número máximo de *tokens*. Durante a geração sequencial de saída, caso seja insuficiente para conter um comando completo, este tamanho é dobrado. A presença de um comando completo na saída é indicada através dos delimitadores de início e final de sentença previamente descritos.

Ao delimitar um comando válido, esses *tokens* previnem que algum ruído seja inserido, além de evitar que perguntas fora do repertório de ações no sistema sejam produzidas, isto é, se algo fora do contexto do sistema for perguntado, os delimitadores não estarão presentes e portanto o texto gerado será invalidado.

### 3.5

#### Tokenização

Para que um LLM seja capaz de processar palavras de forma eficaz, é fundamental que ele esteja equipado com um tokenizador. O tokenizador é uma peça essencial do pré-processamento de texto, responsável por dividir o texto

em unidades menores chamadas *tokens*. Em um contexto de processamento de linguagem natural, um *token* pode representar uma palavra, um caractere, ou até mesmo uma parte significativa de uma palavra, como um sufixo ou um prefixo. Ao quebrar o texto em *tokens*, o tokenizador ajuda o LLM a entender a estrutura e a gramática da linguagem, tornando mais fácil para o modelo processar, analisar e gerar texto coerente.

Além de dividir o texto em *tokens*, o tokenizador tem a função de realizar a codificação desses *tokens* em uma representação matricial. Essa transformação é essencial para que os modelos de linguagem possam processar e manipular as informações de texto de maneira computacional. O texto precisa ser codificado em uma representação numérica pois a maioria dos modelos de aprendizado de máquina, incluindo os modelos de linguagem, trabalham com números, vetores e matrizes em vez de texto.

Cada *token* é associado a um vetor numérico, muitas vezes chamado de *embedding*, que reflete seu significado e contexto. Essa representação matricial permite que o modelo realize cálculos matemáticos, como operações de multiplicação de matrizes, que são fundamentais para o processamento de linguagem natural. Dessa forma, o tokenizador não apenas organiza o texto em unidades comprehensíveis, mas também habilita o LLM a trabalhar com essas unidades em um formato numérico, possibilitando a análise, a geração e a compreensão de texto de maneira eficaz e precisa.

O tokenizador escolhido foi o tokenizador pré-treinado disponibilizado pelos desenvolvedores do BLOOM. Esse tokenizador já contém *embeddings* relacionados ao português do Brasil.

### 3.6

#### Escopo e treinamento

Como descrito anteriormente, o sistema alvo para a assistente apresenta grande porte e complexidade. Trata-se de uma plataforma que acessa múltiplas bases de dados provendo recursos avançados para a análise integrada. Portanto, disponibiliza diversas funcionalidades. No entanto, o escopo da assistente limita-se apenas à criação e configuração de gráficos e relatórios, que se caracterizam como os recursos mais usados no sistema. Esse escopo se mostra adequado para a prova de conceito proposta no trabalho.

A tabela 3.1 lista os hiperparâmetros, que são configurações ajustáveis que não são aprendidas diretamente durante o treinamento do modelo, e influenciam e estão altamente ligados a facilidade de aprendizado do modelo, usados para *fine-tuning* do modelo.

Tabela 3.1: Hiperparâmetros de treinamento

Parâmetro	Valor
Número de exemplos distintos para treinamento	212 mil
Número de exemplos distintos para teste	106 mil
Número de <i>tokens</i>	6 milhões
Número de <i>batches</i> por <i>step</i>	8
Número de <i>batches</i> por avaliação	32
Número de exemplos por <i>batch</i>	256
Número total de <i>steps</i> de treinamento	50 mil
Número de <i>steps</i> entre avaliações	500
Otimizador	AdamW [18]

O treinamento é realizado através de *steps*. Em cada *step*, oito *batches* de dados são submetidos ao modelo. Cada *batch* contém 256 exemplos, amostrados randomicamente dos 212 mil exemplos disponíveis para treinamento.

Ao final de 500 *steps*, a taxa de erro (*loss*) do modelo é avaliada a partir de 32 *batches* extraídos do conjunto de teste. Para ajuste do modelo é adotado o otimizador AdamW [18]. Os parâmetros internos desse otimizador são escolhidos conforme o padrão documentado no *framework* utilizado, *Hugging Face Transformers* [26].

## 4

# Resultados

Para uma avaliação mais precisa, os resultados obtidos são comparados com os dados gerados por humanos especialistas no sistema, que são considerados como referência para determinar o que seria correto em cada caso de teste. Essa comparação permite identificar possíveis desvios ou limitações do modelo, buscando garantir que as respostas fornecidas sejam consistentes com o esperado. Cada dado gerado por humanos é necessariamente diferente em algum grau de todos os exemplos usados no treinamento, para poder garantir maior precisão na hora de avaliar o modelo.

Para realizar a avaliação, é utilizada a métrica de similaridades entre sentenças, que basicamente consiste em se verificar quão próximas duas sentenças estão entre si.

Na seção a seguir, por questões de confidencialidade, alguns termos podem ser omitidos ou alterados com o objetivo de preservar a parte da lógica de negócio do sistema.

## 4.1

### Treinamento

O treinamento foi realizando utilizando as configurações na tabela 4.1. Ao todo, até a conclusão dessa pesquisa, foram utilizadas 1500 horas de processamento para treinamento e ajustes finos.

Tabela 4.1: Recursos computacionais e tempo para treinamento.

Processador gráfico	NVIDIA V100™
Memória de vídeo	32 GB
RAM	125 GB
Processadores (CPU)	12
Tempo total de treinamento	17 horas

Ao concluir o processo de treinamento, conduzido com os parâmetros especificados na tabela 3.1, é possível avaliar de maneira rápida e objetiva se o modelo realmente aprendeu a partir dos dados a partir da curva de perda (*loss*), representada na figura 4.1. Esse processo de avaliação é essencial para determinar a eficácia do modelo e sua capacidade de generalização. A função de *loss* é uma medida da diferença entre as previsões do modelo e os valores reais dos dados de treinamento.

A análise gráfica da figura revela um padrão notável: a *loss* diminui progressivamente, aproximando-se de zero à medida que as épocas avançam. Esse comportamento evidencia a habilidade do modelo em aprimorar sua capacidade de generalização ao longo das épocas de treinamento.

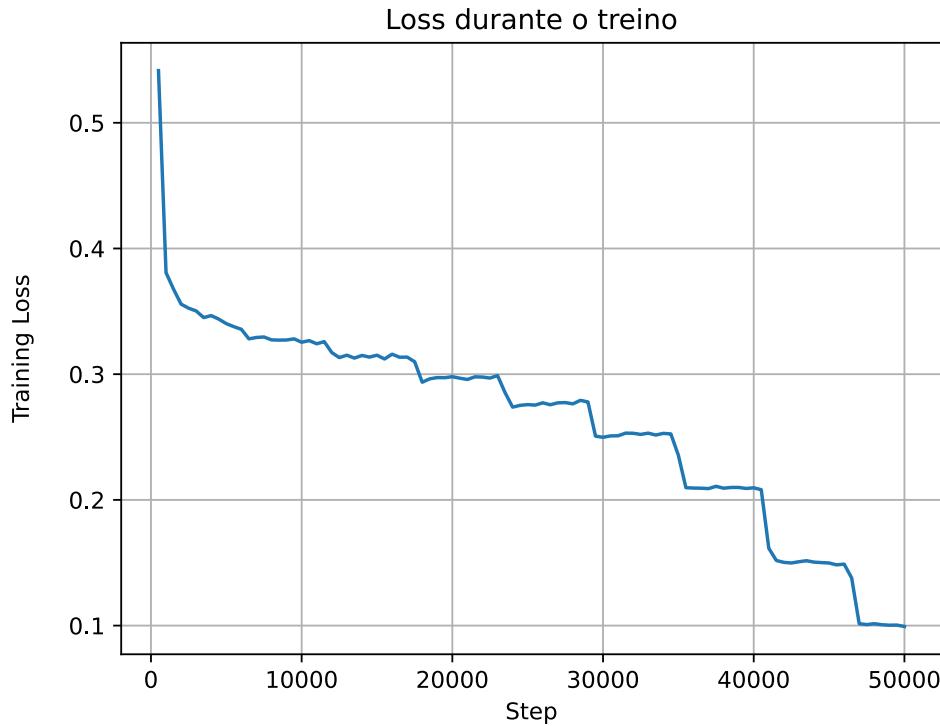


Figura 4.1: *Loss* do treinamento

## 4.2 Caso 1

No primeiro caso de teste, o usuário deseja construir um gráfico simples com a variável de vazão de óleo ( $q_o$ ). O comando em linguagem natural mais simples para executar essa tarefa é dado abaixo:

Comando: gerar um grafico com  $q_o$

Para esse caso a resposta esperada (*ground truth*) é dada no formato JSON abaixo:

```
Ground truth:
[
  {
    "alvo": 1,
    "metadados": {
      "funcionalidade": "grafico",
```

```

        "configuracao": { "variaveis": [{ "nome": "qo" }] }
    }
}
]
```

Para esse caso, a resposta fornecida pelo modelo é idêntica ao esperado, como mostrado abaixo, também no formato JSON:

```

Saída do modelo:
[
{
    "alvo": 1,
    "metadados": {
        "funcionalidade": "grafico",
        "configuracao": { "variaveis": [{ "nome": "qo" }] }
    }
}
]
```

### 4.3 Caso 2

No segundo caso de teste, o usuário deseja construir um relatório com a variável de vazão de óleo ( $q_o$ ), utilizando os dados de Marlim. O comando em linguagem natural mais simples para executar essa tarefa é dado abaixo:

Comando: abrir um relatorio com qo para o campo de marlim

Para esse caso a resposta esperada é dada no formato JSON abaixo:

Ground truth:

```

[
{
    "alvo": "2",
    "metadados": {
        "categorias": [{ "nome": "marlim", "tipoCategoria": "Campo" }]
    }
},
{
    "alvo": 1,
    "metadados": {
        "funcionalidade": "relatorio",
        "configuracao": { "variaveis": [{ "nome": "qo" }] }
    }
}
]
```

Para esse caso, a resposta fornecida pelo modelo também é idêntica ao esperado, como mostrado abaixo, também no formato JSON:

Saída do modelo:

```
[
  {
    "alvo": "2",
    "metadados": {
      "categorias": [{ "nome": "marlim", "tipoCategoria": "Campo" }]
    }
  },
  {
    "alvo": 1,
    "metadados": {
      "funcionalidade": "relatorio",
      "configuracao": { "variaveis": [{ "nome": "qo" }] }
    }
  }
]
```

#### 4.4

#### Caso 3

No terceiro caso de teste, o usuário deseja gerar um gráfico que apresente duas variáveis, ( $q_o$ ) e ( $q_g$ ), e associá-los a dois eixos separados, alocando ( $q_o$ ) ao primeiro eixo e ( $q_g$ ) ao segundo. O comando em linguagem natural mais simples para executar essa tarefa é dado abaixo:

Comando: gerar um grafico com qo, no eixo 1 e qg no eixo 2

Para esse caso a resposta esperada é dada no formato JSON abaixo:

Ground truth:

```
[
  {
    "alvo": 1,
    "metadados": {
      "funcionalidade": "grafico",
      "configuracao": {
        "variaveis": [
          { "nome": "qo", "eixo": 1 },
          { "nome": "qgi", "eixo": 2 }
        ]
      }
    }
  }
]
```

A resposta fornecida pelo modelo é idêntica ao esperado, como mostrado abaixo, também no formato JSON:

Saída do modelo:

```
[
  {
    "alvo": 1,
    "metadados": {
      "funcionalidade": "grafico",
      "configuracao": {
        "variaveis": [
          { "nome": "qo", "eixo": 1 },
          { "nome": "qgi", "eixo": 2 }
        ]
      }
    }
  }
]
```

## 4.5

### Caso 4

No quarto caso de teste, o usuário deseja construir um gráfico simples com a variável de vazão de óleo ( $q_o$ ), porém a variável está descrita por extenso e sem acentos. O comando em linguagem natural para executar essa tarefa é dado abaixo:

Comando: gerar um grafico com vazao de óleo

A resposta esperada é data no formato JSON abaixo:

Ground truth:

```
[
  {
    "alvo": 1,
    "metadados": {
      "funcionalidade": "grafico",
      "configuracao": { "variaveis": [ { "nome": "qo" } ] }
    }
  }
]
```

A resposta fornecida pelo modelo é idêntica ao esperado, como mostrado abaixo, também no formato JSON:

Saída do modelo:

```
[
  {
    "alvo": 1,
    "metadados": {
      "funcionalidade": "grafico",
      "configuracao": { "variaveis": [{ "nome": "qo" }] }
    }
  }
]
```

## 4.6 Caso 5

No último caso de teste, o usuário deseja construir um gráfico mais complexo, utilizando duas variáveis ( $q_o$ ) e ( $q_g$ ), configuradas em eixos separados, sendo a segunda especificada por extenso “vazão de gás”. A cada variável deve ser atribuída uma cor única. Além disso, o gráfico deve exibir os dados de todos os poços do campo Marlim agrupados (somados) em uma curva única. O comando em linguagem natural para executar essa tarefa é dado abaixo:

Comando: fazer um grafico multicolorido de qo no eixo 1 e vazao de gas no eixo 2, para todos os pocos do campo de marlim, com dados agrupados

A resposta esperada é dada no formato JSON abaixo:

Ground truth:

```
[
  {
    "alvo": "2",
    "metadados": {
      "tipoCategoriaFilha": "Poco",
      "categoriasPai": [
        { "nome": "marlim", "tipoCategoria": "Campo" }
      ]
    }
  },
  {
    "alvo": 1,
    "metadados": {
      "funcionalidade": "grafico",
      "configuracao": {
        "variaveis": [
          { "nome": "qo", "eixo": 1 },
          { "nome": "qg", "eixo": 2 }
        ],
        "agrupados": true
      }
    }
  }
]
```

```

        "dadosAgrupados": true,
        "multiplasCores": true
    }
}
]

```

A resposta fornecida pelo modelo é idêntica ao esperado, como mostrado abaixo, também no formato JSON:

Saída do modelo:

```

[
{
    "alvo": "2",
    "metadados": {
        "tipoCategoriaFilha": "Poco",
        "categoriasPai": [
            { "nome": "marlim", "tipoCategoria": "Campo" }
        ]
    }
},
{
    "alvo": 1,
    "metadados": {
        "funcionalidade": "grafico",
        "configuracao": {
            "variaveis": [
                { "nome": "qo", "eixo": 1 },
                { "nome": "qg", "eixo": 2 }
            ],
            "dadosAgrupados": true,
            "multiplasCores": true
        }
    }
}
]
```

## 4.7

### Resumo de resultados

Os exemplos acima ilustram o desempenho do modelo com *fine-tuning* para geração de comandos executáveis no sistema alvo. O grau de similaridade entre as respostas fornecidas pelo modelo e o *ground truth* é calculado pela semelhança caractere por caractere.

É possível observar que o modelo apresentou desempenho muito próximo de 100%, em todos os casos, inclusive para comandos complexos como o

apresentado na seção 4.6. Mesmo quando a similaridade caractere a caractere não é total, o modelo gera o comando exato do *ground truth*. As diferenças observadas se referem à presença ou não de espaços entre o nome e o valor das propriedades do JSON gerado. O exemplo abaixo mostra o detalhe da comparação entre uma diferença observada para o caso 4.6. No modelo gerado, não há o espaço separando a propriedade nome das variáveis  $q_o$  e  $q_g$ .

**Ground truth:**

```
...
{"nome": "qo", "eixo": "1"},  

{"nome": "qg", "eixo": "2"}  

...
```

**Saída do modelo:**

```
...
{"nome": "qo", "eixo": "1"},  

{"nome": "qg", "eixo": "2"}  

...
```

A tabela 4.2 sumariza o grau de similaridade observado para cada um dos casos acima. Os anexos ao final do presente documento trazem os resultados visuais (telas) dos comandos, em cada caso, executados no sistema alvo.

Tabela 4.2: Similaridade entre as respostas observadas e o *ground truth*.

Caso	Similaridade (%)
1	100,00
2	100,00
3	98,97
4	100,00
5	99,04

## 4.8

### Fuga do contexto

Os modelos gerativos podem resultar em textos indesejados a partir de determinadas entradas. Para impedir que isso aconteça, isto é, que o modelo fuja do contexto planejado, respondendo quaisquer perguntas não apenas relacionadas às funcionalidades escolhidas do sistema alvo, algumas mudanças foram propostas no treinamento:

1. Foram inseridas algumas instruções textuais para orientar o comportamento do modelo em algumas situações [28]. Como por exemplo:
  - *Sempre que receber um comando que não contenha algo relativo a gráfico e relatório sempre retornar "comando invalido".*
  - *Não deve responder perguntas que envolvam outros módulos.*
2. Os exemplos fornecidos foram definidos de tal forma que o mecanismo de atenção ajude na distinção entre perguntas válidas e inválidas. Por exemplo, o modelo consegue inferir que, se um comando não envolve gráfico ou relatório, muito provavelmente é inválido e não deve ser respondido.

Essa manipulação do mecanismo de atenção ajuda o modelo a identificar as palavras da entrada, e determinar onde e como elas devem ser encaixadas no comando resultante. Qualquer coisa que fuja desse contexto, por exemplo uma entrada não relacionada a criação ou manipulação de gráfico ou relatório, resulta em uma sequência de **comando invalido**, conforme apresentado na tabela 4.3. Essa sequência é passada para o sistema indicando que a assistente não consegue processar aquela entrada.

Comando	Resposta do modelo
<code>qual o melhor programa de tv</code>	<code>comando invalido</code>
<code>qual o melhor time de futebol</code>	<code>comando invalido</code>
<code>abrir funcionalidade_x com qo</code>	<code>comando invalido</code>
<code>qual o poço que mais produz no campo X</code>	<code>comando invalido</code>
<code>qual variavel que apresenta maior vazao para o poco X</code>	<code>comando invalido</code>
<code>quanto é 1 + 1?</code>	<code>comando invalido</code>

Tabela 4.3: Ilustração de alguns comandos inválidos

Pela ausência das marcações de início e final de sentença na resposta **comando invalido**, a assistente conseguirá precisamente notificar o sistema de que há algo errado com o comando que foi inserido e fazer com que o mesmo notifique o usuário que aquele determinado tipo de comando não é executável pela assistente.

## 4.9

### Comandos incompletos

Para criar um gráfico e relatório, o sistema alvo requer a especificação das variáveis com as quais essas funcionalidades são preenchidas. Por exemplo, ao criar um gráfico, o usuário deve escolher quais curvas ele quer visualizar.

Os comandos em linguagem natural tendem a refletir esse comportamento, ou seja, eles especificam tanto as funcionalidades quanto os seus complementos. Portanto, a assistente aprende a associar as funcionalidades sempre ao seu complemento. Assim, ela também é capaz de identificar quando um comando está incompleto, com o seu complemento faltante.

Um comando incompleto não configura erro. Para esses casos, o treinamento inclui uma ação adicional para requisitar um possível complemento da funcionalidade escolhida. O caso abaixo ilustra tal comportamento.

Comando: `abrir grafico`

Saída do modelo:

```
[  
  {  
    "alvo": 1,  
    "metadados": {  
      "funcionalidade": "grafico",  
      "configuracao": { "sugerirVariaveis": true }  
    }  
  }  
]
```

Na saída fornecida pelo modelo, uma configuração adicional é inserida nos `metadados`, a saber, a propriedade `sugerirVariaveis`. Recendo o valor `true`, essa propriedade ativa uma caixa de seleção na interface do sistema alvo para o usuário poder realizar a complementação do comando, conforme ilustrado na figura 4.2.

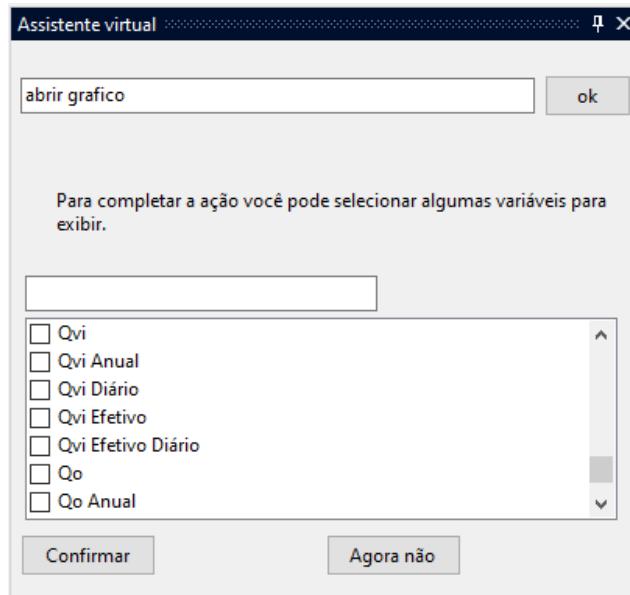


Figura 4.2: Caixa de seleção para complementação de comando requisitada pela assistente no sistema alvo.

## 4.10

### Limites de compreensão do modelo

Foram testadas algumas sentenças em linguagem natural fornecidas por um usuário que conhece o sistema alvo e sabe das limitações do modelo de linguagem. O objetivo aqui é testar os limites de compreensão do modelo exposto agora a um ambiente de menor controle, removendo assim um possível enviesamento presente nos testes demostrados anteriormente.

#### 4.10.1

##### Comandos inválidos

Alguns comandos livres fornecidos pelo usuário geram a sequência de comando `invalido`, conforme descrito na seção 4.8. Nessas situações, as sentenças se enquadram em três categorias:

1. Não apresentam sentido claro ou se mostram absurdas.
2. Não contêm explicitamente a referência a gráfico ou relatório.
3. Apresentam um texto significativamente mais elaborado do que os exemplos contidos no *dataset* de treinamento.

Isso dificulta o entendimento e geração do comando para o sistema alvo. Para essas sentenças, a acurácia pode ser considerada 100%, pois fogem do contexto estabelecido, ou se revelam demasiadamente ambíguas. As sentenças que geram comando inválido são:

- Exibir Razão Gás-Óleo de Marlim
- Crie, por favor, algo tipo um relatório e preencha as colunas com o BSW e com a RGO das plataformas
- Quero ver as vazões de água de Marlim
- Criar tabela com nome dos poços e plataforma
- Bom dia, Sistema!
- Preciso saber o quanto cada poço produziu ontem
- Selecione plataforma
- Escolha categoria poço
- Portar trafico com proteção de aguaia
- Criei um granito que olha pra agua
- Acai guardia zum de bes óleo
- Criar um gráfico com renda per capita países américa latina

#### 4.10.2

##### Sentenças resolvidas corretamente

A assistente consegue interpretar parcial ou completamente sentenças com maior grau de complexidade. Isso demonstra que o modelo identifica corretamente a intenção por trás do comando, o seu significado, e mapeia para a instrução adequada da API. Alguns exemplos podem ser observados abaixo.

- Comando: graf np  
Saída do modelo:  
[{"alvo": 1, "metadados": { "funcionalidade": "grafico", "configuracao": {"variaveis": [{"nome": "np"}]} }}]
- Comando: relat qo, qg, qw  
Saída do modelo:  
[{"alvo": 1, "metadados": { "funcionalidade": "relatorio", "configuracao": {"variaveis": [{"nome": "qo"}, {"nome": "qg"}, {"nome": "qw"}]} }}]
- Comando: Qo, BSW, Qw, Qg, relatório, poço  
Saída do modelo:  
[{"alvo": 1, "metadados": { "funcionalidade": "relatorio", "configuracao": {"variaveis": [{"nome": "Qo"}, {"nome": "BSW"}, {"nome": "Qw"}, {"nome": "Qg"}, {"nome": "poço"}]} }}]
- Comando: Relatório, poço, Qo, BSW  
Saída do modelo:  
[{"alvo": 1, "metadados": { "funcionalidade": "relatorio", "configuracao": {"variaveis": [{"nome": "BSW"}, {"nome": "poço"}, {"nome": "Qo"}]} }}]

- Comando: criar grafico com Razão Gás-Óleo de papa-terra  
 Saída do modelo:  
 [{"alvo": 1, "metadados": { "funcionalidade": "grafico", "configuracao": {"variaveis": [{"nome": "RGO"}]} }}]
- Comando: Mostrar curva de óleo  
 Saída do modelo:  
 [{"alvo": 1, "metadados": { "funcionalidade": "relatorio", "configuracao": {"variaveis": [{"nome": "7we"}, {"nome": "Wip"}, {"nome": "weo"}, {"nome": "wev"}]} }}]
- Comando: Fechar gráfico com Qo  
 Saída do modelo:  
 [{"alvo": 1, "metadados": { "funcionalidade": "grafico", "configuracao": {"variaveis": [{"nome": "Qo"}]} }}]
- Comando: Desenhar curva de Qw  
 Saída do modelo:  
 [{"alvo": 1, "metadados": { "funcionalidade": "relatorio", "configuracao": {"variaveis": [{"nome": "Qw"}]} }}]
- Comando: criar grafico com Razão Gás-Óleo de Mrl  
 Saída do modelo:  
 [{"alvo": 1, "metadados": { "funcionalidade": "grafico", "configuracao": {"variaveis": [{"nome": "RGO"}, {"nome": "SECr"}]} }}]

Em alguns comandos é possível notar que o modelo produz alguns artefatos no nome das variáveis, gerando sequências alfanuméricas sem sentido, como `7we`, `Wip`, `weo`, `wev` e `SECr`. Embora não representem variáveis reais do sistema alvo, essas sequências são coerentes com o *dataset* de treinamento.

Nesse *dataset* são inseridos sequências similares, que não representam variáveis reais, mas que tem o propósito de dar flexibilidade ao modelo. Essa estratégia permite que a assistente consiga identificar com mais precisão as variáveis e ao mesmo tempo não fique presa (“decore” ou “memorize”) somente as possibilidades das variáveis utilizadas no treinamento, visto que novas variáveis podem ser criadas a qualquer momento. Esse comportamento não afeta o resultado final executado no sistema, pois a API é capaz de descartar variáveis desconhecidas.

Alguns exemplos evidenciam que o modelo é capaz de compreender o contexto e não apenas palavras isoladas. Embora tenha sido treinado com instruções específicas (`abrir`, `gerar`, `fazer`, `criar`, `plotar` e `desenhar`), ele se mostra flexível para compreender o contexto completo do comando e fornecer a saída correta.

- Comando: fornecer um grafico de qo  
Saída do modelo:  
[{"alvo": 1, "metadados": { "funcionalidade": "grafico", "configuracao": {"variaveis": [{"nome": "qo"}]} }}]
- Comando: apresentar relatorio de np  
Saída do modelo:  
[{"alvo": 1, "metadados": { "funcionalidade": "relatorio", "configuracao": {"variaveis": [{"nome": "np"}]} }}]

#### 4.10.3

##### Comando não resolvidos corretamente

Em algumas situações, se observa não consegue captar a intenção do comando fornecido pelo usuário. Isso geralmente ocorre em comandos em linguagem natural que se desdobram mais de uma ação ou funcionalidade. O exemplo abaixo ilustra essa limitação:

Comando: Plotar gráfico e abrir relatório com qo  
Saída do modelo:  
[{"alvo": 1, "metadados": { "funcionalidade": "grafico", "configuracao": {"variaveis": [{"nome": "zoipi"}, {"nome": "acesrq"}]} }}]

O usuário expressa através de um único comando sua intenção tanto de abrir um gráfico quanto de construir um relatório, ambos preenchidos com a variável  $Q_o$ . Ainda que consiga interpretar parcialmente o comando fornecido, o modelo ignora a criação do relatório e não se mostra capaz de resolver o complemento para o gráfico.

Caso o complemento para o comando seja especificado de uma maneira mais explícita, como mostrado abaixo, o modelo consegue gerar uma instrução completa, porém ainda ignorando a parte do relatório.

Comando: Plotar gráfico com qo e abrir relatório com qw  
Saída do modelo:  
[{"alvo": 1, "metadados": { "funcionalidade": "grafico", "configuracao": {"variaveis": [{"nome": "qo"}, {"nome": "507"}], "dadosAgrupados": true } }}]

Essa limitação ocorre pois o modelo não foi exposto a exemplos que mostrem como se comportar quando duas funcionalidades são solicitadas ao mesmo tempo. Embora ele não consiga resolver esses comandos, isso não significa que ele não consegue extrair múltiplas instruções a partir de um mesmo comando, como observado abaixo:

Comando: abrir grafico de qo para o campo de marlim

Saída do modelo:

```
[{"alvo": "2", "metadados": {"categorias": [{"nome": "marlim", "tipoCategoria": "Campo"}]}}, {"alvo": 1, "metadados": {"funcionalidade": "grafico", "configuracao": {"variaveis": [{"nome": "qo"}]}}}
```

Nesse exemplo, duas instruções são geradas (alvos 1 e 2), orientando a API a executar a seleção das categorias escolhidas e a criação do gráfico.

Em algumas situações em que o comando fornecido pelo usuário apresenta incoerência, o modelo é capaz de propor uma solução próxima aquilo que ele conhece. O caso abaixo exemplifica uma situação em que, ao relatório, o usuário tenta associar eixos, que são configurações exclusivas da funcionalidade gráfico.

Comando: relatório qo qw dois eixos

Saída do modelo:

```
[{"alvo": 3, "metadados": {"funcionalidade": "relatorio", "nomeManipulacao": "converter", "metadados": {"funcionalidadeDestino": "grafico", "nomeAbaAberta": "qoqwhas2"}}}]
```

No sistema alvo é possível se criar um gráfico a partir do conteúdo de um relatório e vice-versa. Esse recurso é chamado de **conversão** e está descrito na API. O modelo entende esse recurso como a melhor solução para o comando fornecido pelo usuário. Essa situação é interessante porque ilustra a capacidade do modelo de “pensar sobre a pergunta” e propor a resposta mais coerente possível dentro do escopo em que foi treinado. Isso demonstra que o modelo não está preso a regras inflexíveis!

## 5

### Conclusões e trabalhos futuros

Os resultados gerados pelo modelo proposto não podem ser comparados diretamente com outros modelos, uma vez que muitos modelos convencionais são projetados com uma abordagem mais genérica e não possuem o conhecimento específico do sistema. Sendo assim, a comparação direta entre os resultados desses diferentes modelos pode levar a conclusões enganosas sobre o desempenho de cada um deles, a menos que sejam treinados com o mesmo *dataset*.

Compreender linguagem natural e fornecer respostas coerentes são desafios significativos enfrentados pelos modelos geradores atuais. Esta pesquisa procura explorar essas questões, fornecendo alguns detalhes sobre as limitações desses modelos e apresentando uma abordagem metodológica para superar essas dificuldades. Ao aprimorar o entendimento e a capacidade de resposta da assistente virtual, espera-se oferecer uma interação mais natural e eficiente, tornando-a uma ferramenta poderosa para auxiliar os usuários em suas necessidades diárias.

Endereçando as questões de pesquisa levantadas no capítulo 1:

1. *É possível incorporar uma assistente virtual à interface complexa de um sistema corporativo pré-existente?*

Sim, através do treinamento de um modelo de linguagem para o contexto específico do sistema alvo e o mapeamento de ações implementadas em uma API de comunicação é possível incorporar uma assistente virtual à interface complexa de um sistema corporativo pré-existente.

2. *Caso afirmativo, como fazer isso e quais tecnologias usar?*

Para implementar uma assistente para o sistema alvo, o primeiro passo é a escolha de um modelo adequado para o problema em questão. Para o modelo de linguagem foi escolhida a tecnologia dos *Transformers*, por se mostrar o estado da arte para dados textuais. A metodologia aqui proposta busca explorar os modelos geradores para interpretar um dado em linguagem natural e gerar uma resposta em formato estruturado, fornecida como entrada para API que executa os comandos solicitados. O segundo passo consiste na criação de um *dataset* para treinamento que seja representativo e suficiente em relação ao sistema alvo. Por fim, é necessário realizar o treinamento, com recursos computacionais compatíveis com a demanda de processamento exigida por tais modelos.

### 3. Como contornar limitações das estratégias escolhidas?

Tanto para acelerar o treinamento, quanto para reduzir seu custo computacional, foi escolhido um modelo pré-treinado para ser usado como base para o modelo final. Para prevenir os problemas de um modelo treinado em um contexto genérico e possibilitar que o modelo escolhido seja capaz de identificar um contexto específico, foi aplicado um *fine-tuning* com os termos e jargões do sistema. Por fim, foi aplicada sobre os dados de treinamento, uma manipulação que permite ao mecanismo de atenção ser capaz de filtrar o contexto possível de resposta, limitando-se a entradas ligadas ao contexto proposto na pesquisa. Isso elimina a possibilidade de respostas incoerentes, sensíveis ou enviesadas.

O modelo treinado, baseado na tecnologia dos *Transformers*, mostrou-se efetivo para a concepção de uma assistente virtual inteligente para o sistema alvo. Deste modo o modelo proposto oferece a possibilidade de tornar a interface do sistema mais natural, possibilitando que os usuários se concentrem na tarefa fim ao invés de se preocuparem com detalhes de interface ou de usabilidade do sistema.

#### 5.1

#### Principais contribuições

A principal contribuição desse estudo é a prova de conceito de uma assistente virtual inteligente baseada em modelos geradores, capaz de entender o português do Brasil e especializada no contexto de um sistema alvo pré-existente voltado para o Gerenciamento de Reservatórios. Ainda são escassos na literatura trabalhos que apresentem tais contribuições.

#### 5.2

#### Limitações

É importante ressaltar que a assistente atualmente está limitada a responder apenas a comandos relacionados ao sistema e às duas funcionalidades específicas — geração de gráficos e relatórios — para as quais foi treinada. Essa restrição é intencional e adequada, uma vez que visa evitar respostas inválidas ou potencialmente problemáticas, permitindo assim uma prova de conceito.

Embora essa limitação possa parecer restritiva em termos de variedade de interações possíveis, ela desempenha um papel crucial na segurança e na precisão das respostas fornecidas pela assistente, evitando mal-entendidos e fornecendo informações relevantes aos usuários.

### 5.3

#### Trabalhos futuros

Há um vasto leque de oportunidades para aprimorar ainda mais a assistente virtual e expandir suas capacidades. Com base nas conquistas alcançadas nessa pesquisa, existe um espaço promissor para testes e desenvolvimentos adicionais, visando fortalecer a inteligência, o alcance e a usabilidade da assistente em diversos aspectos, assim como melhorar o seu processo de treinamento com outros modelos base e conjuntos de hiperparâmetros.

##### 5.3.1

###### Treinamento contínuo

A assistente foi treinada para responder somente duas funcionalidades do sistema alvo. O próximo passo é disponibilizá-la em uma versão do sistema para os usuários reais, de modo a aprimorar continuamente a sua capacidade de resposta. Ao permitir que os usuários interajam com a assistente e solicitem novas ações que atualmente ela não consegue responder, torna-se possível treiná-la para responder a essas demandas específicas, por exemplo, manipular outras funcionalidades.

Essa abordagem interativa permitirá que a assistente expanda gradualmente sua cobertura e se torne cada vez mais completa, à medida que novos dados de treinamento são construídos a partir das interações dos usuários.

Assim, os próprios comandos e solicitações dos usuários desempenham um papel fundamental na melhoria da assistente. Cada interação fornece informações valiosas para aprimorar sua compreensão e resposta aos usuários, contribuindo para a criação de um conjunto de treinamento mais robusto. Ao aproveitar essa colaboração entre a assistente e os usuários, é possível impulsionar seu aprendizado e permitir um constante aperfeiçoamento das habilidades da assistente, atendendo cada vez mais às necessidades e expectativas dos usuários de forma abrangente, precisa e eficiente.

##### 5.3.2

###### Adaptação personalizada

Outro aspecto relevante a ser explorado é a personalização da assistente para se adaptar aos diferentes usuários. Isso pode envolver o desenvolvimento de mecanismos de aprendizado contínuo, nos quais a assistente seja capaz de se ajustar às preferências e necessidades individuais dos usuários, oferecendo respostas mais personalizadas e relevantes.

### 5.3.3

#### Expansão de linguagens e domínio

Além das perspectivas mencionadas anteriormente, uma área de pesquisa altamente relevante e promissora para o aprimoramento da assistente virtual é a sua expansão para além do português do Brasil, tornando-se verdadeiramente multilíngue. Ao tornar-se capaz de compreender e interagir em diversos idiomas, a assistente poderia atender a um público mais amplo e diversificado, facilitando a comunicação e oferecendo suporte em diferentes contextos linguísticos.

O mesmo princípio se aplica à expansão do escopo da assistente virtual, abrangendo uma variedade de domínios adicionais. Ao ampliar sua compreensão e conhecimento em diferentes áreas, a assistente seria capaz de fornecer informações e suporte especializados para além do setor de Óleo e Gás, abrangendo outros domínios relacionados à Indústria, para além da disciplina de Reservatórios, como a automação de processos. Com essa expansão de conhecimento e escopo, a assistente virtual se tornaria uma valiosa aliada na Indústria, ajudando a impulsionar a eficiência, a segurança e o desempenho geral dos processos automatizados.

### 5.3.4

#### Novos modelos e ajuste de hiperparâmetros

Com o avanço contínuo no campo de PLN, surgem constantemente modelos e técnicas aprimoradas que podem ser aplicados na construção da assistente. A investigação de novos modelos baseados em *Transformers*, ou outra tecnologia, pode conduzir a um passo interessante para a melhoria da assistente virtual. Um modelo promissor para a aplicação aqui apresentada é o modelo Llama [19], baseado na arquitetura dos *Transformers* com tamanhos que variam de 7 a 65 bilhões de parâmetros e que tem se mostrado um modelo eficiente para geração textual.

Além disso, os hiperparâmetros, que são configurações que controlam o comportamento dos modelos de aprendizado de máquina, também podem ser otimizados para melhorar o desempenho da assistente. Isso envolve a experimentação com diferentes valores para hiperparâmetros como taxa de aprendizado, tamanho da rede, otimizadores, entre outros, a fim de encontrar as combinações ideais que levem a um desempenho otimizado da assistente.

## 6

### Referências bibliográficas

- [1] VASWANI, A.; SHAZER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L. ; POLOSUKHIN, I.. **Attention is all you need**, 2017. 2
- [2] RADFORD, A.; NARASIMHAN, K.; SALIMANS, T. ; SUTSKEVER, I.. **Improving language understanding by generative pre-training**. 2018. 1.1, 2
- [3] DEVLIN, J.; CHANG, M.-W.; LEE, K. ; TOUTANOVA, K.. **Bert: Pre-training of deep bidirectional transformers for language understanding**, 2019. (document), 2, 2.4
- [5] RADFORD, A.; WU, J.; CHILD, R.; LUAN, D.; AMODEI, D. ; SUTSKEVER, I.. **Language models are unsupervised multitask learners**. 2019. 2
- [6] BROWN, T. B.; MANN, B.; RYDER, N.; SUBBIAH, M.; KAPLAN, J.; DHARIWAL, P.; NEELAKANTAN, A.; SHYAM, P.; SASTRY, G.; ASKELL, A.; AGARWAL, S.; HERBERT-VOSS, A.; KRUEGER, G.; HENIGHAN, T.; CHILD, R.; RAMESH, A.; ZIEGLER, D. M.; WU, J.; WINTER, C.; HESSE, C.; CHEN, M.; SIGLER, E.; LITWIN, M.; GRAY, S.; CHESS, B.; CLARK, J.; BERNER, C.; MCCANDLISH, S.; RADFORD, A.; SUTSKEVER, I. ; AMODEI, D.. **Language models are few-shot learners**, 2020. 2, 3.2
- [7] OPENAI. **Gpt-4 technical report**, 2023. 2
- [8] SCAO, T. L.; FAN, A.; AKIKI, C.; PAVLICK, E.; ILIĆ, S.; HESSLOW, D.; CASTAGNÉ, R.; LUCCIONI, A. S.; YVON, F.; GALLÉ, M. ; ET AL.. **Bloom: A 176b-parameter open-access multilingual language model**, 2022. 3.2
- [9] PIRES, R.; ABONIZIO, H.; ALMEIDA, T. S. ; NOGUEIRA, R.. **Sabiá: Portuguese large language models**, 2023. 3.2
- [10] LEWKOWYCZ, A.; ANDREASSEN, A.; DOHAN, D.; DYER, E.; MICHALEWSKI, H.; RAMASESH, V.; SLONE, A.; ANIL, C.; SCHLAG, I.; GUTMAN-SOLO, T.; WU, Y.; NEYSHABUR, B.; GUR-ARI, G. ; MISRA, V.. **Solving quantitative reasoning problems with language models**, 2022. 1, 2

- [11] TAYLOR, R.; KARDAS, M.; CUCURULL, G.; SCIALOM, T.; HARTSHORN, A.; SARAVIA, E.; POULTON, A.; KERKEZ, V. ; STOJNIC, R.. **Galactica: A large language model for science**, 2022. 1, 2
- [12] SOUZA, F.; NOGUEIRA, R. ; LOTUFO, R.. **Bertimbau: Pretrained bert models for brazilian portuguese**, 10 2020. 2
- [13] RODRIGUES, R.; PRIVATTO, P.; SOUSA, G.; MURARI, R.; AFONSO, L.; PAPA, J.; PEDRONETTE, D.; GUILHERME, I.; PERROUT, S. ; RIENTE, A.. **Petrobert: A domain adaptation language model for oil and gas applications in portuguese**, 03 2022. 2
- [14] **Bigscience rail license v1.0.** <https://huggingface.co/spaces/bigscience/license>. Acessado em: 01/05/2023. 3.2
- [15] LIU, Y.; OTT, M.; GOYAL, N.; DU, J.; JOSHI, M.; CHEN, D.; LEVY, O.; LEWIS, M.; ZETTLEMOYER, L. ; STOYANOV, V.. **Roberta: A robustly optimized bert pretraining approach**, 2019. 2
- [16] LAN, Z.; CHEN, M.; GOODMAN, S.; GIMPEL, K.; SHARMA, P. ; SORICUT, R.. **Albert: A lite bert for self-supervised learning of language representations**, 2020. 2
- [17] JOSHI, M.; CHEN, D.; LIU, Y.; WELD, D. S.; ZETTLEMOYER, L. ; LEVY, O.. **Spanbert: Improving pre-training by representing and predicting spans**, 2020. 2
- [18] LOSHCHILOV, I.; HUTTER, F.. **Decoupled weight decay regularization**, 2019. 3.1, 3.6
- [19] TOUVRON, H.; LAVRIL, T.; IZACARD, G.; MARTINET, X.; LACHAUX, M.-A.; LACROIX, T.; ROZIÈRE, B.; GOYAL, N.; HAMBRO, E.; AZHAR, F.; RODRIGUEZ, A.; JOULIN, A.; GRAVE, E. ; LAMPLE, G.. **Llama: Open and efficient foundation language models**, 2023. 5.3.4
- [20] WEIDINGER, L.; MELLOR, J.; RAUH, M.; GRIFFIN, C.; UESATO, J.; HUANG, P.-S.; CHENG, M.; GLAESE, M.; BALLE, B.; KASIRZADEH, A.; KENTON, Z.; BROWN, S.; HAWKINS, W.; STEPLETON, T.; BILES, C.; BIRHANE, A.; HAAS, J.; RIMELL, L.; HENDRICKS, L. A.; ISAAC, W.; LEGASSICK, S.; IRVING, G. ; GABRIEL, I.. **Ethical and social risks of harm from language models**, 2021. 1, 2, 3.4

- [22] ASHER, N.; BHAR, S.; CHATURVEDI, A.; HUNTER, J. ; PAUL, S.. **Limits for learning with language models**, 2023. 1
- [23] KANDPAL, N.; DENG, H.; ROBERTS, A.; WALLACE, E. ; RAFFEL, C.. **Large language models struggle to learn long-tail knowledge**, 2023. 1, 2
- [24] **Sequence-to-sequence models.** <https://huggingface.co/learn/nlp-course/chapter1/7?fw=pt>. Acessado em: 23/09/2023. (document), 2.1
- [25] **Modelo de transformador para compreensão da linguagem.** <https://www.tensorflow.org/text/tutorials/transformer?hl=pt-br>. Acessado em: 23/09/2023. (document), 2.2
- [25] **Must-read papers on pre-trained language models (plms).** <https://github.com/thunlp/PLMpapers>. Acessado em: 23/09/2023. (document), 2.3
- [26] **Hugging face – the ai community building the future.** <https://huggingface.co/docs/transformers/index>. Acessado em: 23/09/2023. 3.6
- [27] DAKE, L.. **Fundamentals of Reservoir Engineering**. ISSN. Elsevier Science, 1983. 1.1, 2
- [28] ZHANG, S.; DONG, L.; LI, X.; ZHANG, S.; SUN, X.; WANG, S.; LI, J.; HU, R.; ZHANG, T.; WU, F. ; WANG, G.. **Instruction tuning for large language models: A survey**, 2023. 1
- [29] ROSA, A.; DE SOUZA CARVALHO, R. ; XAVIER, J.. **Engenharia de reservatórios de petróleo**. Interciência, 2006. 1.1

## A Anexos

Aqui estão reunidas figuras que ilustram dos resultados obtidos após a execução dos comandos de teste no sistema alvo. Essas imagens correspondem a respostas produzidas na interface do sistema alvo, permitindo a compreensão abrangente do desempenho e funcionamento da assistente proposta nesse trabalho.

É importante destacar que, para preservar a propriedade intelectual da regra de negócio do sistema alguns termos e nomes de entidades foram descharacterizados ou ocultados. Todos os dados apresentados aqui, após a descharacterização e ocultação de nomes, refletem informações disponíveis publicamente no site da Agência Nacional do Petróleo, Gás Natural e Biocombustíveis (ANP), garantindo, assim, que os dados exibidos nesse trabalho são de domínio público.

### A.1 Resultados por Caso

#### A.1.1 Caso 1

A figura A.1 ilustra a execução do comando: `gerar um grafico com qo`. É possível observar que a assistente consegue identificar as entidades relavantes no comando e gerar a instrução correta para que o sistema alvo atenda a solicitação do usuário como esperado, isto é, resultando na criação de um gráfico como a variável  $q_o$ .

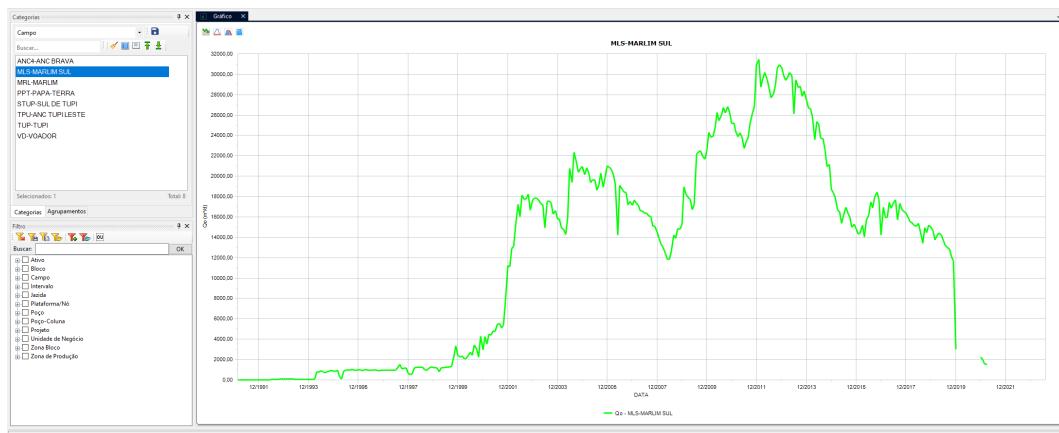


Figura A.1: Resultado Caso 1

### A.1.2

#### Caso 2

A figura A.2 ilustra a execução do comando: `abrir um relatorio com qo para o campo de marlim`. É possível observar que a assistente identifica as configurações necessárias para abrir o relatório com a variável solicitada e em seguida seleciona a categoria correta, nesse caso, o campo de Marlim, visível no painel chamado de ‘Categorias’, na lateral esquerda da tela.



Figura A.2: Resultado Caso 2

### A.1.3

#### Caso 3

A figura A.3 ilustra a execução do comando: `gerar um grafico com qo, no eixo 1 e qg no eixo 2`. A assistente identifica as configurações necessárias para abrir o gráfico com as variáveis solicitadas e em seguida as configurações em eixos separados, como observadas nas bordas esquerda e direita do gráfico.



Figura A.3: Resultado Caso 3

### A.1.4

#### Caso 4

A figura A.4 ilustra a execução do comando: `gerar um grafico com vazao de óleo`. A assistente identifica a variável, descrita por extenso, corretamente. Isso mostra que ela é capaz de identificar corretamente as entidades do sistema e escrever os comandos coerentes através da interpretação da entrada.

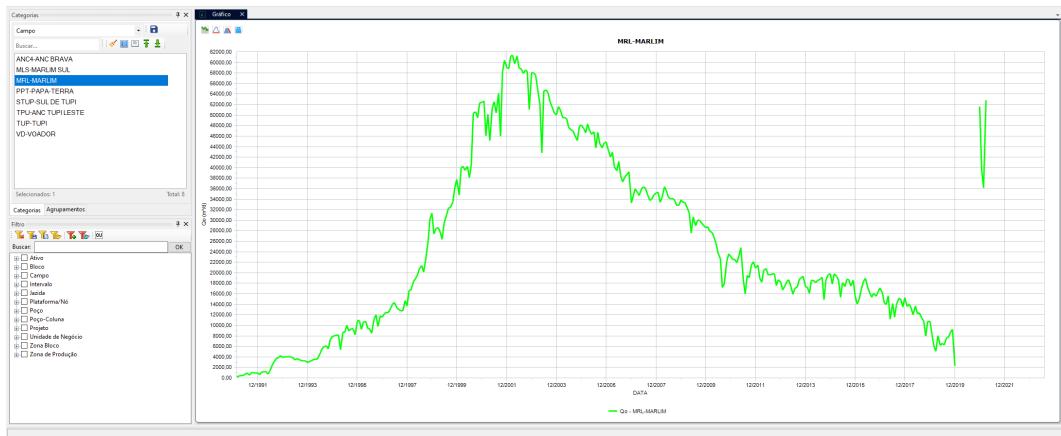


Figura A.4: Resultado Caso 4

### A.1.5

#### Caso 5

A figura A.5 ilustra a execução do comando: `fazer um grafico multicolorido de qo no eixo 1 e vazao de gas no eixo 2, para todos os poços do campo de marlim, com dados agrupados`.

Esse comando é mais complexo que os demais, dado que é composto de diversos detalhes e tarefas. Para processá-lo a assistente deve ser capaz de:

- Interpretar as variáveis descritas tanto na forma simbólica quanto na escrita por extenso,  $q_o$  e vazão de gás, respectivamente.
- Distribuir as variáveis interpretadas nos respectivos eixos solicitados.
- Identificar que a palavra ‘multicolorido’ remete a atribuição de cores distintas para as variáveis.
- Identificar que os dados de entrada do gráfico são provenientes de todos os poços do campo solicitado.
- Identificar que os dados devem ser agrupados, isto é, devem ser resgatados para cada poço e então somados para gerar a curva final.

Através da análise da figura A.5 é possível observar que a assistente consegue atingir a todos os objetivos mencionados anteriormente. É possível notar que, no painel de **Categorias**, todos os poços que terminam com o sufixo **MRL** estão selecionados. Isso mostra que a assistente entende que deve buscar todos os poços relacionados ao campo de Marlim e selecioná-los para compor as curvas finais exibidas no gráfico.

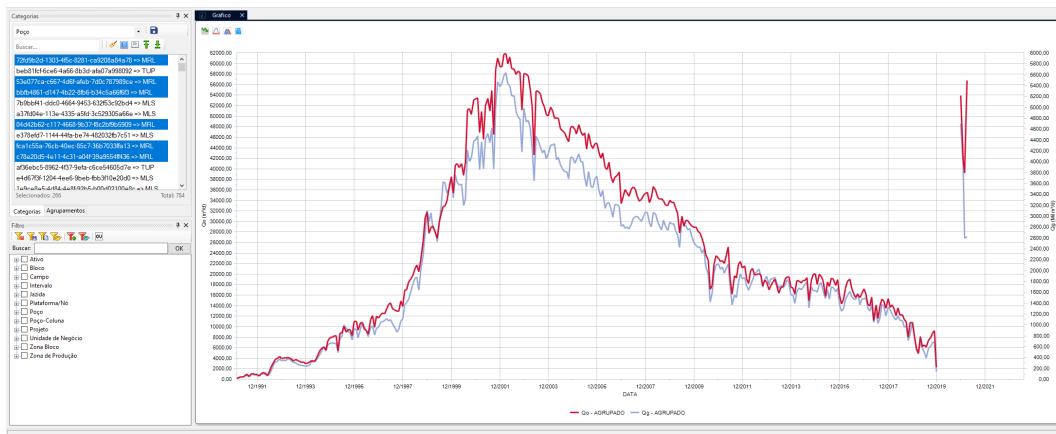


Figura A.5: Resultado Caso 5